



SN74ACT8800 Family

***32-Bit CMOS Processor
Building Blocks***

Data Manual

Data Manual

SN74ACT8800 Family

1989

1989

Overview

1

SN74ACT8818 16-Bit Microsequencer

2

SN74ACT8832 32-Bit Registered ALU

3

SN74ACT8836 32- × 32-Bit Parallel Multiplier

4

SN74ACT8837 64-Bit Floating Point Processor

5

SN74ACT8841 Digital Crossbar Switch

6

SN74ACT8847 64-Bit Floating Point/Integer Processor

7

Support

8

Mechanical Data

9

***SN74ACT8800 Family
32-Bit CMOS Processor
Building Blocks***

Data Manual



IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to or to discontinue any semiconductor product or service identified in this publication without notice. TI advises its customers to obtain the latest version of the relevant information to verify, before placing orders, that the information being relied upon is current.

TI warrants performance of its semiconductor products to current specifications in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Unless mandated by government requirements, specific testing of all parameters of each device is not necessarily performed.

TI assumes no liability for TI applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

Copyright © 1988, Texas Instruments Incorporated

First edition: March 1988

First revision: June 1988

Second revision: June 1989

INTRODUCTION

In this manual, Texas Instruments presents technical information on the TI SN74ACT8800 family of 32-bit processor "building block" circuits. The SN74ACT8800 family is composed of single-chip VLSI processor functions, all of which are designed for high-complexity processing applications.

This manual includes specifications and operational information on the following high-performance advanced-CMOS devices:

- SN74ACT8818 16-bit microsequencer
- SN74ACT8832 32-bit registered ALU
- SN74ACT8836 32- × 32-bit parallel multiplier
- SN74ACT8837 64-bit floating point processor
- SN74ACT8841 Digital crossbar switch
- SN74ACT8847 64-bit floating point/integer processor

These high-speed devices operate at or above 20 MHz, while providing the low power consumption of TI's advanced one-micron EPIC™ CMOS technology. The EPIC™ CMOS process combines twin-well structures for increased density with one-micron gate lengths for increased speed.

The *SN74ACT8800 Family Data Manual* contains design and specification data for all five devices previously listed and includes additional programming and operational information for the '8818, '8832, and '8837/'8847. Two application notes, "Chebyshev Routines for the SN74ACT8847" and "High-speed Vector Math and 3D Graphics Using the SN74ACT8837/8847 Floating Point Unit" are also included.

Introductory sections of the manual include an overview of the '8800 family and a summary of the software tools and design support TI offers for the chip-set. The general information section includes an explanation of the function tables, parameter measurement information, and typical characteristics related to the products listed in this volume.

Package dimensions are given in the Mechanical Data section of the book in metric measurement (and parenthetically in inches).

Complete technical data for any Texas Instruments semiconductor product is available from your nearest TI field sales office, local authorized TI distributor, or by calling Texas Instruments at 1-800-232-3200.

Overview

1

SN74ACT8818 16-Bit Microsequencer

2

SN74ACT8832 32-Bit Registered ALU

3

SN74ACT8836 32- × 32-Bit Parallel Multiplier

4

SN74ACT8837 64-Bit Floating Point Processor

5

SN74ACT8841 Digital Crossbar Switch

6

SN74ACT8847 64-Bit Floating Point/Integer Processor

7

Support

8

Mechanical Data

9

1

Overview

Overview

1

Overview

Introduction

Texas Instruments SN74ACT8800 family of 32-bit processor building blocks has been developed to allow the easy, custom design of functionally sophisticated, high-performance processor systems. The '8800 family is composed of single-chip, VLSI devices, each of which represents an element of a CPU.

Geared for computationally intensive applications, SN74ACT8800 devices include high-performance ALUs, multipliers, microsequencers, and floating point processors.

The '8800 chip set provides the performance, functionality, and flexibility to fill the most demanding processing needs and is structured to reduce system design cost and effort. Most of these high-speed processor functions operate at 20 MHz and above, and, at the same time, provide the power savings of TI's advanced, 1 μ m EPIC™ CMOS technology.

The family's building block approach allows the easy, "pick-and-choose" creation of customized processor systems, while the devices' high level of integration provides cost-effectiveness.

Designed especially for high-complexity processing, the devices in the '8800 family offer a range of functional options. Device features include three-port architecture, double-precision accuracy, optional pipelined operation, and built-in fault tolerance.

Array, digital signal, image, and graphics processing can be optimized with '8800 devices. Other applications are found in supermini and fault-tolerant computers, and I/O and network controllers.

In addition to the high-performance, CMOS processor functions featured in this data manual, the family includes several high-speed, low-power bipolar support chips. To reduce power dissipation and ensure reliability, these bipolar devices use TI's proprietary Schottky Transistor Logic (STL) internal circuitry.

At present, TI's '8800 32-bit processor building block family comprises the following functions:

- SN74ACT8818 16-bit microsequencer
- SN74ACT8832 32-bit registered ALU
- SN74ACT8836 32- × 32-bit parallel multiplier
- SN74ACT8837 64-bit floating point processor
- SN74ACT8841 digital crossbar switch
- SN74ACT8847 64-bit floating point and integer processor
- Bipolar Support Chips
 - SN74AS8838 32-bit barrel shifter
 - SN74AS8839 32-bit shuffle/exchange network
 - SN74AS8840 16 × 4 crossbar switch

20 MIPS and Low CMOS Power Consumption

With instruction cycle times of 50 ns or less and the low power consumption of EPIC™ CMOS, the '8800 chip set offers an unrivaled speed/power combination. Unlike traditional microprocessors, which require multiple cycles to perform an operation, the 'ACT8800 processors typically can complete instructions in a single cycle.

The 'ACT8832 registered ALU and 'ACT8818 microsequencer together create a powerful 20-MHz CPU. Because instructions can be performed in a single cycle, the 8832/8818 combination is capable of executing over 20 million instructions per second (MIPS).

For math-intensive applications, the 'ACT8836 fixed-point multiplier/accumulator (MAC), 'ACT8837 64-bit floating point processor, and 'ACT8847 64-bit floating point and integer processor offer unprecedented computational power.

The exceptional performance of the 'ACT8800 family is made possible by TI's EPIC™ CMOS technology. The EPIC™ CMOS process combines twin-well structures for increased density with one-micron gate lengths for increased speed.

Customized Solution

The '8800 family is designed with a variety of architectural and functional options to provide maximum design flexibility. These device features allow the creation of "customized" solutions with the '8800 chipset.

A **building block approach** to processing allows designers to match specialized hardware to their specific design needs. The '8818/8832 combination forms the basis of the system, a high-speed CPU. For applications requiring high-speed integer multiplication, the 'ACT8836 can be added. To provide the high precision and large dynamic range of floating point numbers, the 'ACT8837 or 'ACT8847 can be employed.

To ensure speed and flexibility, each component of the '8800 family has **three data ports**. Each data port accommodates 32 bits of data, plus four parity bits. This architecture eliminates many of the I/O bottlenecks associated with traditional single-I/O microprocessors.

The three-port architecture and functional partitioning of the '8800 chip-set opens the door to a variety of **parallel processing** applications. Placing the math and shifting functions in parallel with the ALU permits concurrent processing of data. Additional processors can be added when performance needs dictate.

The 'ACT8800 building block processors are microprogrammable, so that their instruction sets can be tailored to a specific application. This **high degree of programmability** offers greater speed and flexibility than a typical microprocessor and ensures the most efficient use of hardware.

A **separate control bus** eliminates the need for multiplexing instructions and data, further reducing processing bottlenecks. The microcode bus width is determined by the designer and the application.

Another source of design flexibility is provided by the **pipelined/flowthrough operation option**. Pipelining can dramatically reduce the time required to perform iterative, or sequential, calculations. On the other hand, random or nonsequential algorithms require fast flowthrough operations. The '8800 chip set allows the designer to select the mode (fully pipelined, partially pipelined, or nonpipelined) most suited to each design.

Scientific Accuracy

The '8800 family is designed to support applications which require **double-precision accuracy**. Many scientific applications, such as those in the areas of high-end graphics, digital signal processing, and array processing, require such accuracy to maintain data integrity. In general-purpose computing applications, floating point processors must often support double-precision data formats to maintain compatibility with existing software.

To ensure data integrity, '8800 devices (excluding the barrel shifter and microsequencer) support **parity checking and generation**, as well as **master/slave error detection**. Byte parity checking is performed on the input ports, and a parity generator and a master/slave comparator are provided at the output. **Fault tolerance** is built into the processors, ensuring correct device operation without extra logic or costly software.

The SN74ACT8800 Building Block Processor System

Some of the high-performance '8800 devices are described in the following paragraphs.

SN74ACT8818 16-Bit Microsequencer

In a high-performance microcoded system, a fast microcode controller is required to control the flow of instructions. The SN74ACT8818 is a high-speed, versatile 16-bit microsequencer capable of addressing 64K words of microcode memory. The 'ACT8818 can address the next instruction fast enough to support a 50-ns system cycle time.

The 'ACT8818 65-word-deep by 16-bit-wide stack is useful for storing subroutine return addresses, top of loop addresses, and loop counts. Addresses can be sourced from eight different sources: the three I/O ports, the two register counters, the microprogram counter, the stack, and the 16-way branch.

SN74ACT8832 Registered ALU

The SN74ACT8832 is a 32-bit registered ALU that operates at approximately 20 MHz. Because instructions can be performed in a single cycle, the 'ACT8832 is capable of executing 20 million microinstructions per second. An on-board 64-word register file is 36-bits-wide to permit the storage of parity bits. The 3-operand register file increases performance by enabling the creation of an instruction and the storage of the previous result in a single cycle. To facilitate data transfer, operands stored in the register file can be accessed externally, while the ALU is executing. To support the parallel processing of data, the 'ACT8832 can be configured to operate as four 8-bit ALUs, two 16-bit ALUs, or a single 32-bit ALU. The 'ACT8832 incorporates 32-bit shifters for double-precision shift operations.

SN74ACT8836 32- × 32-Bit Integer MAC

The SN74ACT8836 is a 32-bit integer multiplier/accumulator (MAC) that accepts two 32-bit inputs and computes a 64-bit product. The device can also operate as a 64-bit by 64-bit multiplier. An onboard adder is provided to add or subtract the product or the complement of the product from the accumulator.

When pipelined internally, the 1- μ m CMOS parallel MAC performs a full 32- × 32-bit multiply/accumulate in a single 36-ns clock cycle. In flowthrough mode (without any pipelining), the 'ACT8836 takes 60 ns to multiply two 32-bit numbers. The 'ACT8836 performs a 64- × 64-bit multiply/accumulate, outputting a 64-bit result, in 225 ns.

The 'ACT8836 can handle a wide variety of data types, including two's complement, signed, and mixed. Division is supported via the Newton-Raphson algorithm.

SN74ACT8837 64-Bit Floating Point Unit

The SN74ACT8837 is a high-speed floating point processor. This single-chip device performs 32- or 64-bit floating point operations.

More than just a coprocessor, the 'ACT8837 integrates on one chip a double-precision floating point ALU and multiplier. Integrating these functions on a single chip reduces data routing problems and processing overhead. In addition, three data ports and a 64-bit internal bus architecture allow for single-cycle operations.

The 'ACT8837 can be pipelined for iterative calculations or can operate with input registers disabled for low latency.

SN74ACT8841 Digital Crossbar Switch

The SN74ACT8841 is a single-chip digital crossbar switch. The high-performance device, cost-effectively eliminates bottlenecks to speed data through complex bus architecture.

The 'ACT8841 is ideal for multiprocessor applications, where memory bottlenecks tend to occur. The device has 64 bidirectional I/O ports that can be configured as 16 4-bit ports, 8 8-bit ports, or 4 16-bit ports. Each bidirectional port can be connected in any conceivable combination. Any single input port can be broadcast to any combination of output ports. The total time for data transfer is 20 ns.

The control sources for ten separate switching configurations are on-chip, including eight banks of programmable control flip-flops and two hard-wired control circuits.

The EPIC™ CMOS SN74ACT8841 and its predecessor, SN74AS8840, are based on the same architecture, differing in power consumption, number of control registers, and pin-out. Microcode written for the 'AS8840 can be run on the 'ACT8841.

SN74ACT8847 64-Bit Floating Point Unit

The SN74ACT8847 is a high-speed 64-bit floating point processor. The device is fully compatible with IEEE standard 754-1985 for addition, subtraction, multiplication, division, square root, and comparison. Division and square root operations are implemented via hardwired control.

The SN74ACT8847 FPU also performs integer arithmetic, logical operations, and logical shifts. Registers are provided at the inputs, outputs, and inside the ALU and multiplier to support multilevel pipelining. These registers can be bypassed for nonpipelined operations.

When fully pipelined, the 'ACT8847 can perform a double-precision floating point or 32-bit integer operation in under 40 ns. When in flowthrough mode, the 'ACT8847 takes less than 100 ns to perform an operation.

Bipolar Support Chips

1 The SN74AS8838 high-speed, 32-bit barrel shifter can shift up to 32 bits in a single instruction cycle of under 25 ns. Five basic shifts can be programmed: circular left, circular right, logical left, logical right, and arithmetic right. The 'AS8838 offloads the responsibility for shifting operations from the ALU, which increases shifter functionality and system throughput.

The SN74AS8839 is a 32-bit shuffle/exchange network. The high-speed device can perform data permutations on one 32-bit, two 16-bit, four 8-bit, or eight 4-bit data words in a single instruction cycle of under 25 ns. The shuffle/exchange network is designed primarily for use in digital signal processing applications.

Overview

1

SN74ACT8818 16-Bit Microsequencer

2

SN74ACT8832 32-Bit Registered ALU

3

SN74ACT8836 32- × 32-Bit Parallel Multiplier

4

SN74ACT8837 64-Bit Floating Point Processor

5

SN74ACT8841 Digital Crossbar Switch

6

SN74ACT8847 64-Bit Floating Point/Integer Processor

7

Support

8

Mechanical Data

9

2

SN74ACT8818

SN74ACT8818

16-Bit Microsequencer

2

SN74ACT8818

- Addresses Up to **64K** Locations of Microprogram Memory
- CLK-to-Y = **30 ns** (tpd)
- **Low-Power EPIC™ CMOS**
- Addresses Selected from **Eight** Different Sources
- Performs Multiway Branching, Conditional Subroutine Calls, and Nested Loops
- Large **65-Word by 16-bit** Stack
- **Cascadable**

Because they're microprogrammable, the ACT8800 building block processors provide greater speed and flexibility than does a typical microprocessor. In such a high-performance microcoded system, a fast microsequencer is required to control the flow of microinstructions.

The SN74ACT8818 is a high-speed, versatile 16-bit microsequencer capable of addressing 64K words of microcode memory. The 'ACT8818 can address the next instruction fast enough to support a 50-ns system cycle time.

The 'ACT8818 65-word-deep by 16-bit-wide stack is useful for storing subroutine return addresses, top-of-loop addresses, and loop counts. For added flexibility, addresses can be selected from eight different sources: the three I/O ports, the two register/counters, the microprogram counter, the stack, and the 16-way branch input.

EPIC is a trademark of Texas Instruments Incorporated.

2

SN74ACT8818

Contents

	<i>Page</i>
Introduction	2-11
Understanding the 'ACT8818 Microsequencer	2-11
Microprogramming the 'ACT8818	2-12
Design Support	2-12
Systems Expertise	2-13
'ACT8818 Pin Grid Allocation	2-14
'ACT8818 Specification Tables	2-21
Architecture	2-25
Y Output Multiplexer	2-28
Microprogram Counter	2-28
Register/Counters	2-28
Stack	2-29
Stack Pointer	2-29
Read Pointer	2-29
Stack Warning/Read Error Pin	2-29
Interrupt Return Register	2-30
Microprogramming the 'ACT8818	2-31
Address Selection	2-32
Stack Controls	2-32
Register Controls	2-33
Continue/Repeat Instructions	2-34
Branch Instructions	2-34
Conditional Branch Instructions	2-35
Loop Instructions	2-35
Subroutine Calls	2-37
Subroutine Returns	2-38
Reset	2-39
Clear Pointers	2-39
Read Stack	2-39
Interrupts	2-39

Contents (Continued)

	<i>Page</i>
Sample Microinstructions for the 'ACT8818	2-40
Continue	2-40
Continue and Pop	2-40
Continue and Push	2-40
Branch (Example 1)	2-42
Branch (Example 2)	2-42
Sixteen-Way Branch	2-42
Conditional Branch	2-44
Three-Way Branch	2-44
Thirty-Two-Way Branch	2-44
Repeat	2-46
Repeat on Stack	2-46
Repeat Until CC = H	2-48
Loop Until Zero	2-48
Conditional Loop Until Zero	2-50
Jump to Subroutine	2-52
Conditional Jump to Subroutine	2-52
Two-Way Jump to Subroutine	2-52
Return from Subroutine	2-54
Conditional Return from Subroutine	2-54
Clear Pointers	2-54
Reset	2-54

List of Illustrations

<i>Figure</i>	<i>Title</i>	<i>Page</i>
1	'ACT8818 GC Package	2-14
2	'ACT8818 FN Package	2-16
3	'ACT8818 Logic Symbol	2-17
4	'ACT8818 Functional Block Diagram	2-27
5	Continue	2-41
6	Continue and Pop	2-41
7	Continue and Push	2-41
8	Branch Example 1	2-43
9	Branch Example 2	2-43
10	Sixteen-Way Branch	2-43
11	Conditional Branch	2-45
12	Three-Way Branch	2-45
13	Thirty-Two Way Branch	2-45
14	Repeat	2-46
15	Repeat on Stack	2-47
16	Repeat Until $\overline{CC} = H$	2-49
17	Loop Until Zero	2-49
18	Conditional Loop Until Zero (Example 2)	2-51
19	Jump to Subroutine	2-53
20	Conditional Jump to Subroutine	2-53
21	Two-Way Jump to Subroutine	2-53
22	Return from Subroutine	2-55
23	Conditional Return from Subroutine	2-55
24	Clear Pointers	2-56

2

SN74ACT8818

List of Tables

<i>Table</i>	<i>Title</i>	<i>Page</i>
1	'ACT8818 Pin Grid Allocation	2-15
2	'ACT8818 Pin Functional Description	2-18
3	Response to Control Inputs	2-26
4	Y Output Controls (MUX2-MUX0)	2-32
5	Stack Controls (S2-S0)	2-33
6	Register Controls (RC2-RC0)	2-33
7	Decrement and Branch on Nonzero Encodings	2-36
8	Call Encodings without Register Decrements	2-37
9	Call Encodings with Register Decrements	2-38
10	Return Encodings without Register Decrements	2-38
11	Return Encodings with Register Decrements	2-39

SN74ACT8818

2

2

SN74ACT8818

Introduction

The SN74ACT8818 microsequencer is a low-power, high-performance microsequencer implemented in TI's EPIC™ Advanced CMOS technology. The 16-bit device addresses up to 64K locations of microprogram memory and is compatible with the SN74AS890 microsequencer.

The 'ACT8818 performs a range of sequencing operations in support of TI's family of building block devices and special-purpose processors such as the SN74ACT8847 Floating Point Unit (FPU).

Understanding the 'ACT8818 Microsequencer

The 'ACT8818 microsequencer is designed to control execution of microcode in a microprogrammed system. Basic architecture of such a system usually incorporates at least the microsequencer, one or more processing elements such as the 'ACT8847 FPU or the SN74ACT8832 Registered ALU, microprogram memory, microinstruction register, and status logic to monitor system states and provide status inputs to the microsequencer.

The 'ACT8818 combines flexibility and high speed in a microsequencer that performs multiway branching, conditional subroutine calls, nested loops, and a variety of other microprogrammable operations. The 'ACT8818 can also be cascaded for providing additional register/counters or addressing capability for more complex microcoded control functions.

In this microsequencer, several sources are available for microprogram address selection. The primary source is the 16-bit microprogram counter (MPC), although branch addresses may be input on the two 16-bit address buses, DRA and DRB. An address input on the DRA bus can be pushed on the stack for later selection. Register/counters RCA and RCB can store either branch addresses or loop counts as needed, either for branch operations or for looping on the stack.

The selection of address source can be based on external status from the device being controlled, so that three-way or multiway branching is supported. Once selected, the address which is output on the Y bus passes to the microprogram memory, and the microinstruction from the selected location is clocked into the pipeline register at the beginning of the next cycle.

It is also possible to interrupt the 'ACT8818 by placing the Y output bus in a high-impedance state and forcing an interrupt vector on the Y bus. External logic is required to place the bus in high impedance and load the interrupt vector. The first

microinstruction of the interrupt handler subroutine can push the address from the Interrupt Return register on the stack so that proper linkage is preserved for the return from subroutine.

Micropogramming the 'ACT8818

Microinstructions for the 'ACT8818 select the specific operations performed by the Y output multiplexer, the register/counters RCA and RCB, the stack, and the bidirectional DRA and DRB buses. Each set of inputs is represented as a separate field in the microinstructions, which control not only the microsequencer but also the ALU or other devices in the system.

The 3-port architecture of the 'ACT8818 facilitates both branch addressing and register/counter operations. Both register/counters can be used to hold either loop counts or branch addresses loaded from the DRA and DRB buses. Register/counter operations are selected by control inputs RC2-RC0.

Similarly, the 65-word by 16-bit stack can save addresses from the DRA bus, the microprogram counter (MPC), or the Interrupt Return register, depending on the settings of stack controls S2-S0 and related control inputs. Flexible instructions such as Branch DRA else Branch to Stack else Continue can be coded to take advantage of the conditional branching capability of the 'ACT8818.

Multiway branching (16- or 32-way) uses the B3-B0 inputs to set up a 16-way branch address on DRA or DRB by concatenating B3-B0 with the upper 12 bits of the DRA or DRB bus. The resulting branch addresses DRA' (DRA15-DRA4::B3-B0) and DRB' (DRB15-DRB4::B3-B0) are selected by the Y output multiplexer controls MUX2-MUX0. A Branch DRB' else Branch DRA' instruction can select up to 32 branch addresses, as determined by the settings of B3-B0.

Design Support

TI's '8818 16-bit microsequencer is supported by a variety of tools developed to aid in design evaluation and verification. These tools will streamline all stages of the design process, from assessing the operation and performance of the '8818 to evaluating a total system application. The tools include a functional model, behavioral model, and microcode development software and hardware. Section 8 of this manual provides specific information on the design tools supporting TI's SN74ACT8800 Family.

Systems Expertise

Texas Instruments VLSI Logic applications group is available to help designers analyze TI's high-performance VLSI products, such as the '8818 16-bit microsequencer. The group works directly with designers to provide ready answers to device-related questions and also prepares a variety of applications documentation.

The group may be reached in Dallas, at (214) 997-3970.

2

SN74ACT8818

2

SN74ACT8818

'ACT8818 Pin Grid Allocation

(TOP VIEW)

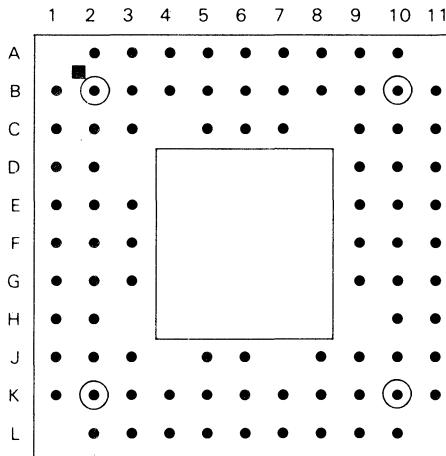


Figure 1. 'ACT8818 GC Package

Table 1. 'ACT8818 Pin Grid Allocation

PIN NO.		PIN NO.		PIN NO.		PIN NO.	
NAME		NAME		NAME		NAME	
A2	RC2	C2	RC0	F3	\overline{RBOE}	J10	S1
A3	Y1	C3	GND	F9	B0	J11	STKWRN/RER
A4	Y3	C5	GND	F10	B1	K1	DRBO
A5	Y5	C6	Y7	F11	MUX2	K2	SELDR
A6	Y6	C7	Y10	G1	DRB6	K3	DRA14
A7	Y8	C9	GND	G2	DRB5	K4	DRA12
A8	Y11	C10	VCC	G3	GND	K5	DRA10
A9	Y13	C11	\overline{RE}	G9	CLK	K6	DRA7
A10	NC	D1	DRB12	G10	MUX0	K7	DRA5
B1	DRB15	D2	DRB13	G11	MUX1	K8	DRA3
B2	RC1	D9	GND	H1	DRB4	K9	DRA0
B3	Y0	D10	COUT	H2	DRB3	K10	S0
B4	Y2	D11	INC	H10	\overline{CC}	K11	S2
B5	Y4	E1	DRB9	H11	ZEROOUT	L2	DRA15
B6	\overline{YOE}	E2	DRB10	J1	DRB2	L3	DRA13
B7	Y9	E3	DRB11	J2	DRB1	L4	DRA11
B8	Y12	E9	\overline{INT}	J3	VCC	L5	DRA9
B9	Y14	E10	B3	J5	GND	L6	DRA8
B10	Y15	E11	B2	J6	\overline{RAOE}	L7	DRA6
B11	ZEROIN	F1	DRB7	J8	DRA1	L8	DRA4
C1	DRB14	F2	DRB8	J9	GND	L9	DRA2
						L10	OSEL

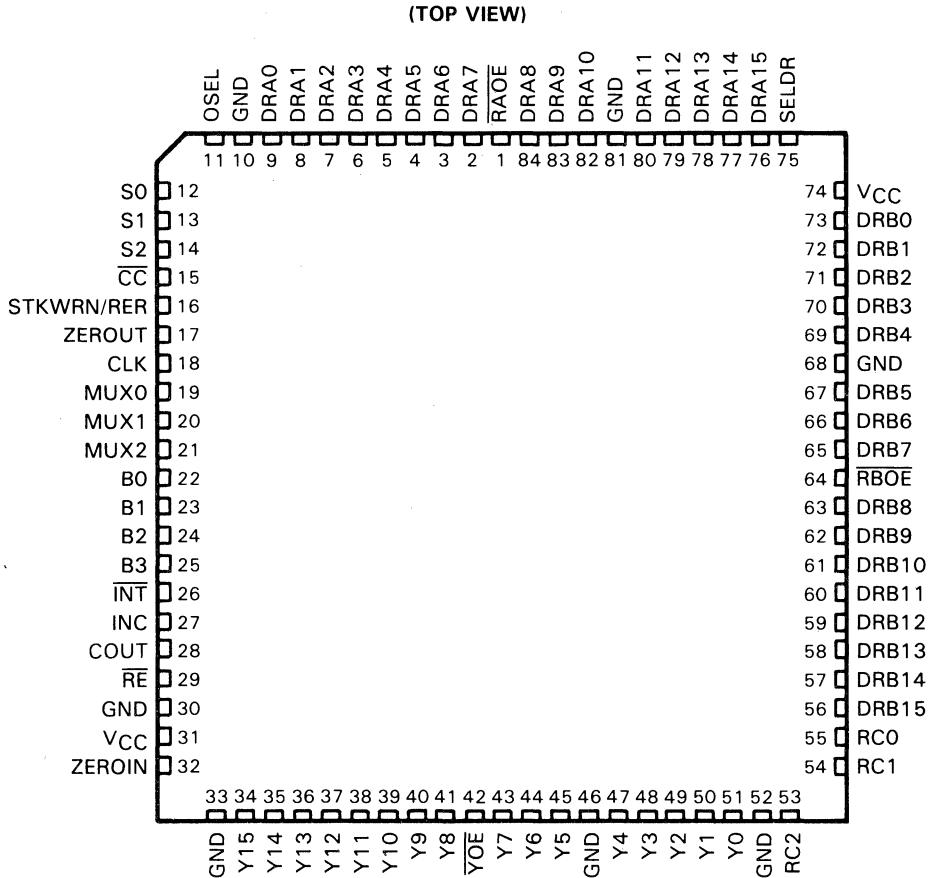


Figure 2. 'ACT8818 . . . FN Package

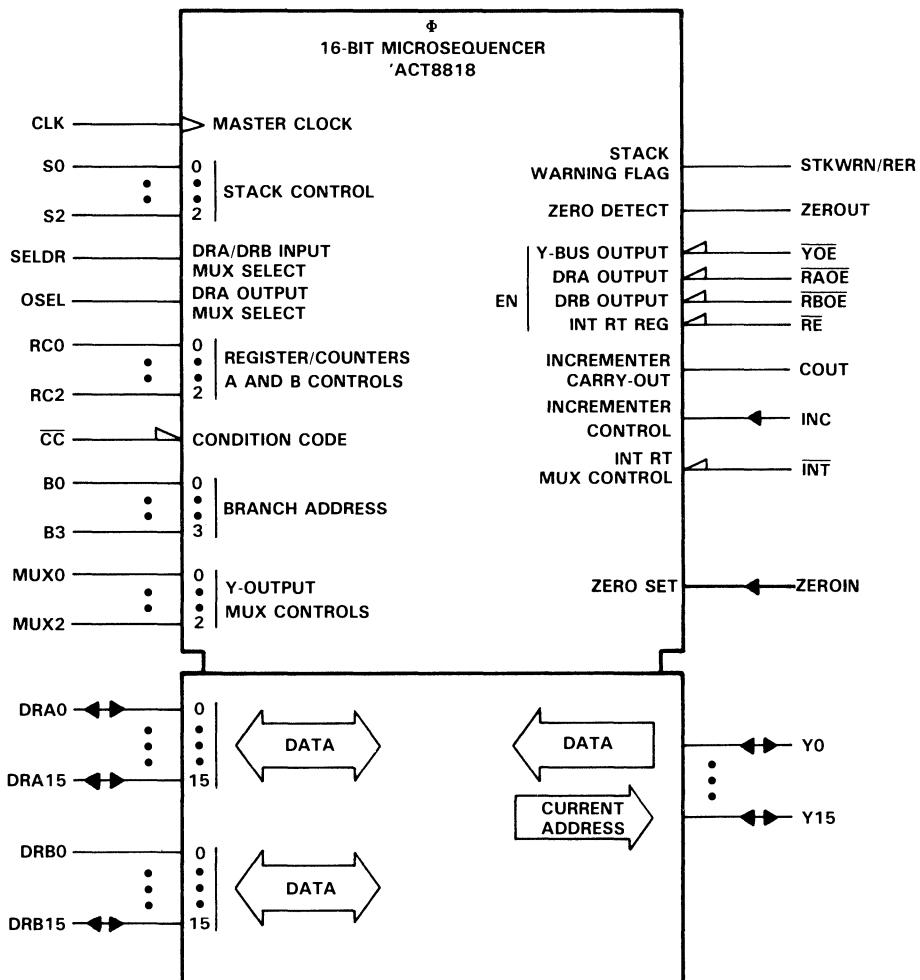


Figure 3. 'ACT8818 . . . Logic Symbol

Table 2. 'ACT8818 Pin Functional Description

PIN NAME	GC NO.	FN NO.	I/O	DESCRIPTION
B0	F9	22		
B1	F10	23		
B2	E11	24		
B3	E10	25		
CLK	G9	18		System clock
COUT	D10	28	O	Incrementer carry-out. Goes high when an attempt is made to increment microprogram counter beyond addressable micromemory.
CC	H10	15	I	Condition code
DRA0	K9	9		
DRA1	J8	8		
DRA2	L9	7		
DRA3	K8	6		
DRA4	L8	5		
DRA5	K7	4		
DRA6	L7	3		
DRA7	K6	2		
DRA8	L6	84		
DRA9	L5	83		
DRA10	K5	82		
DRA11	L4	80		
DRA12	K4	79		
DRA13	L3	78		
DRA14	K3	77		
DRA15	L2	76		
DRBO	K1	73		
DRB1	J2	72		
DRB2	J1	71		
DRB3	H2	70		
DRB4	H1	69		
DRB5	G2	67		
DRB6	G1	66		
DRB7	F1	65		
DRB8	F2	63		
DRB10	E2	61		
			I/O	Bidirectional DRA data port. Outputs data from stack or register/counter A ($\overline{RAOE} = 0$) or inputs external data ($\overline{RAOE} = 1$).
			I/O	Bidirectional DRB data port. Outputs data from register/counter B ($\overline{RBOE} = 0$) or inputs external data

Table 2. 'ACT8818 Pin Functional Description (Continued)

PIN NAME	GC NO.	FN NO.	I/O	DESCRIPTION
DRB11	E3	60		
DRB12	D1	59		
DRB13	D2	58	I/O	Bidirectional DRB data port. Outputs data from register/counter B ($\overline{RBOE} = 0$) or inputs external data ($\overline{RBOE} = 1$).
DRB14	C1	57		
DRB15	B1	56		
GND	C3	10		
GND	C5	30		
GND	C9	33		
GND	D9	46		Ground pins. All pins must be used.
GND	G3	52		
GND	J5	68		
GND	J9	81		
INC	D11	27	I	Incrementer control pin
INT	E9	26	I	Selects INT RT register to stack, active low (see Table 3)
MUX0	G10	19		
MUX1	G11	20	I	MUX control for Y output bus (see Table 4)
MUX2	F11	21		
OSEL	L10	11	I	DRA output MUX select. Low selects RCA, high selects stack.
RAOE	J6	1	I	DRA output enable, active low
RBOE	F3	64	I	DRB output enable, active low
RC0	C2	55		
RC1	B2	54	I	Controls for register/counters A and B
RC2	A2	53		
RE	C11	29	I	INT RT register enable, active low. A high input holds INT RT register while a low input passes Y to INT RT register (see Table 3).
S0	K10	12		
S1	J10	13	I	Stack controls
S2	K11	14		
SELDLDR	K2	75	I	Selects data source to DRA bus and DRB bus (See Table 3)
STKWRN/RER	J11	16	O	Stack warning signal flag
V _{CC}	C10	31		
V _{CC}	J3	74		Supply voltage (5 V)

Table 2. 'ACT8818 Pin Functional Description (Concluded)

PIN NAME	GC NO.	FN NO.	I/O	DESCRIPTION
Y0	B3	51		
Y1	A3	50		
Y2	B4	49		
Y3	A4	48		
Y4	B5	47		
Y5	A5	45		
Y6	A6	44		
Y7	C6	43	I/O	Bidirectional Y data port
Y8	A7	41		
Y9	B7	40		
Y10	C7	39		
Y11	A8	38		
Y12	B8	37		
Y13	A9	36		
Y14	B9	35		
Y15	B10	34		
YOĒ	B6	42	I	Y output enable, active low
ZEROIN	B11	32	I	Forces internal zero detect high
ZEROUT	H11	17	O	Outputs register/counter zero detect signal

'ACT8818 Specification Tables

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 = 0 to 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	MIN	NOM	MAX	UNIT
V _{CC}	Supply voltage	4.5	5	5.5	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
TA	Operating free-air temperature	0		70	°C

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

PARAMETER	TEST CONDITIONS	V _{CC}	TA = 25°C			MIN	TYP	MAX	UNIT
			MIN	TYP	MAX				
V _{OH}	I _{OH} = -20 μA	4.5 V	4.48						V
		5.5 V	5.46						
	I _{OH} = -8 mA	4.5 V	4.15		3.76				
		5.5 V	4.97		4.76				
V _{OL}	I _{OL} = 20 μA	4.5 V		0.014					V
		5.5 V		0.014					
	I _{OL} = 8 mA	4.5 V		0.15		0.45			
		5.5 V		0.13		0.45			
I _I	V _I = V _{CC} or 0	5.5 V				± 1			μA
I _{CC}	V _I = V _{CC} or 0	5.5 V		98		200			μA
C _i	V _I = V _{CC} or 0	5 V		3					pF
ΔI _{CC} [†]	One input at 3.4 V, other inputs at 0 or V _{CC}	5.5 V				1			mA

[†]This is the increase in supply current for each input that is at one of the specified TTL voltage levels rather than 0 V or V_{CC}.

maximum switching characteristics

PARAMETER	FROM (INPUT)	TO (OUTPUT)					UNIT
		Y	ZEROUT	DRA	DRB	STKWRN	
t_{pd}	<u>CC</u>	23					
	CLK	27		24	16	25	
		30 [†]	23 [†]				
	DRA15-DRA0	23					
	DRB15-DRB0	22					
	MUX2-MUX0	22					
	RC2-RC0	26	18				
	S2-S0	25		19			
	B3-B0	19					
	OSEL	25		20			
	ZEROIN	25					
	SELDRL	23					
t_{en}	<u>YOE</u>	16					
	<u>RAOE</u>		18				
	<u>RBOE</u>			17			
t_{dis}	<u>YOE</u>	14					
	<u>RAOE</u>		13				
	<u>RBOE</u>			14			

[†]Decrementing register/counter A or B and sensing a zero.

setup and hold times

PARAMETER	FROM (INPUT)	TO (OUTPUT)	MIN	MAX	UNIT
t_{SU}	\overline{CC}	Stack	15		ns
	DRA15-DRA0	Stack	9		
		RCA	6		
		INT RT	9		
	DRB15-DRBO	RCB	7		
		INT RT	11		
	INC	MPC	7		
	\overline{INT}	Stack	7		
	RC2-RCO	Stack	15		
		RCA, RCB	6		
		INT RT	16		
		Stack	13		
	S2-S0	INT RT	13		
		Stack	12		
	OSEL	INT RT	13		
		Stack	8		
	B3-B0	INT RT	14		
		Stack	10		
	SELDRL	INT RT	10		
		Stack	14		
	ZEROIN	INT RT	13		
		Y	MPC	6	
	\overline{RE}	INT RT (CLK)		7	
	MUX2-MUXO	INT RT	12		
t_H	Any Input	Any Destination	0		ns

clock requirements

PARAMETER		MIN	MAX	UNIT
t_{W1}	Pulse duration, clock low	7		ns
t_{W2}	Pulse duration, clock high	9		ns
t_C	Clock cycle time	33		ns

Architecture

The 'ACT8818 microsequencer is designed with a 3-port architecture similar to the bipolar SN74AS890 microsequencer. Figure 4 shows the architecture of the 'ACT8818. The device consists of the following principal functional groups:

1. A 16-bit microprogram counter (MPC) consisting of a register and incrementer which generates the next sequential microprogram address
2. Two register/counters (RCA and RCB) for counting loops and iterations, storing branch addresses, or driving external devices
3. A 65-word by 16-bit LIFO stack which allows subroutine calls and interrupts at the microprogram level and is expandable and readable by external hardware
4. An interrupt return register and Y output enable for interrupt processing at the microinstruction level
5. A Y output multiplexer by which the next address can be selected from MPC, RCA, RCB, external buses DRA and DRB, or the stack.

'ACT8818 control signals are summarized in Table 3. Those signals, which typically originate from the instruction register, are Y output multiplexer controls, MUX2-MUX0. These select the source of the next address; stack operation controls, S2-S0; register/counter operation controls, RC2-RC0; OSEL, which allows the stack to be read for diagnostics; input MUX select, SELDR; DRA and DRB output enables, RAOE and RBOE; and INT, used during the first cycle of interrupt service routines to push the address in the interrupt return register address onto the stack.

Control and data signals that commonly originate from the microinstruction and from other hardware sources include INC, which determines whether to increment the MPC; DRA and DRB, used to load or read loop counters and/or next addresses; and CC, the condition code input. The address being loaded into the MPC is not incremented if INC is low, allowing wait states and repeat until flag instructions to be implemented. If INC originates from status, repeat until flag instructions are possible.

The condition code input CC typically originates from ALU status to permit test and branch instructions. However, it must also be asserted under microprogram control to implement other instructions such as continue or loop. Therefore, CC will generally be controlled by the output of a status multiplexer. In this case, whether CC is to be forced high, forced low or taken from ALU status will be determined by a status MUX select field in the microinstruction.

Table 3. Response to Control Inputs

SIGNAL NAME	LOGIC LEVEL	
	HIGH	LOW
B0 [†]	Load stack pointer from 7 least significant bits of DRA	No effect
B1 [†]	Selects DRA contents as stack input (takes priority over <u>INT</u>)	No effect
<u>CC</u>	Condition code input. May be microcoded or selected from external status results.	Condition code input. For branch operations, low active.
INC	Increment address from Y bus and load into MPC	Pass address from Y bus to MPC unincremented.
<u>INT</u> [‡]	Selects MPC as input to stack	Selects interrupt return register as input to stack
OSEL	Selects stack as output from DRA output MUX	Selects RCA as output from DRA output MUX
MUX2-MUX0	See Table 4	See Table 4
<u>RAOE</u>	DRA output disabled (high-Z)	DRA output enabled
<u>RBOE</u>	DRB output disabled (high-Z)	DRB output enabled
RC2-RC0	See Table 6	See Table 6
<u>RE</u>	Hold interrupt return register contents	Load address on Y bus to interrupt return register
S2-S0	See Table 5	See Table 5
SELDRL	Selects DRA/DRB external data as inputs to DRA/DRB buses	Selects RCA (OSEL low) or stack (OSEL high) to DRA bus, RCB to DRB bus
<u>YOE</u>	Y output disabled (high-Z)	Y output enabled
ZEROIN	Sets ZEROOUT to a high externally to set up conditional branch	No effect

[†]No control effect when DRA' or DRB' selected (MUX2-MUX0) = HLH) because B3-B0 are address inputs.[‡]When B1 is low or B1 is not in control mode.

Control signals which may also originate from hardware are B3-B0, which can be used as a 4-bit status input to support 16- and 32-way branches, and YOE, which allows interrupt hardware to force an interrupt vector on the microaddress bus.

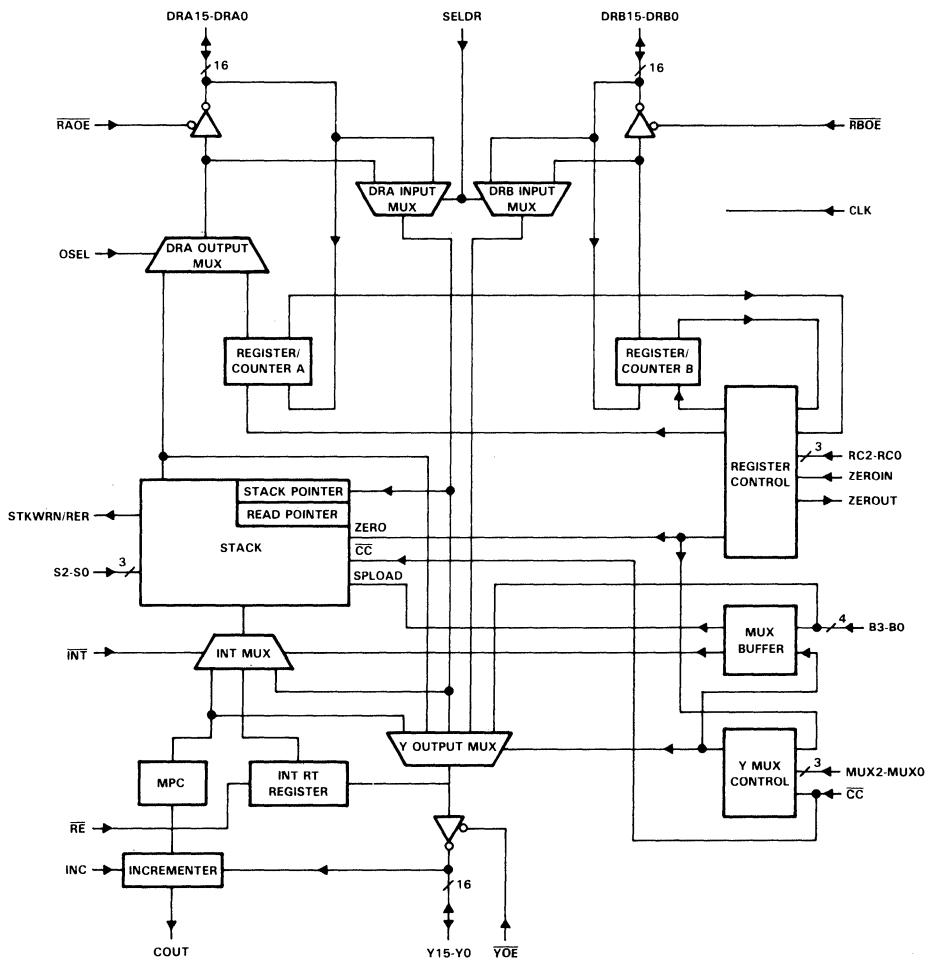


Figure 4. 'ACT8818 Functional Block Diagram

Status from the 'ACT8818 is provided by ZEROOUT, which is set at the beginning of a cycle in which either of the register/counters will decrement to zero, and STKWRN/RER, set at the beginning of the cycle in which the bottom of stack is read or in which the next to last location is written. In the latter case, STKWRN/RER remains high until the stack pointer is decremented from 64 to 63.

Y Output Multiplexer

Address selection is controlled by the Y output multiplexer and the $\overline{\text{RAOE}}$ and $\overline{\text{RBOE}}$ enables. Addresses can be selected from eight sources:

1. The microprogram counter register, used for repeat (INC off) and continue (INC on) instructions
2. The stack, which supports subroutine calls and returns as well as iterative loops and returns from interrupts
3. The DRA and DRB ports, which provide two additional paths from external hardware by which microprogram addresses can be generated
4. Register counters RCA and RCB, which can be used for additional address storage
5. B3-B0, whose contents can replace the four least significant bits of the DRA and DRB buses to support 16-way and 32-way branches
6. An external input onto the bidirectional Y port to support external interrupts.

Use of controls MUX2-MUX0 is explained further in the later section on microprogramming the 'ACT8818.

Micropogram Counter

Based on system status and the current instruction, the microsequencer outputs the next execution address in the microprogram. Usually the incrementer adds one to the address on the Y bus to compute next address plus one. Next address plus one is stored in the microprogram register at the beginning of the subsequent instruction cycle. During the next instruction, this 'continue' address will be ready at the Y output MUX for possible selection as the source of the subsequent instruction. The incrementer thus looks two addresses ahead of the address in the instruction register to set up a continue (increment by one) or repeat (no increment) address.

Selecting INC from status is a convenient means of implementing instructions that must repeat until some condition is satisfied; for example, Shift ALU Until MSB = 1, or Decrement ALU Until Zero. The MPC is also the standard path to the stack. The next address is pushed onto the stack during a subroutine call, so that the subroutine will return to the instruction following that from which it was called.

Register/Counters

Addresses or loop counts may be loaded directly into register/counters RCA and RCB through the direct data ports DRA15-DRA0 and DRB15-DRB0. The values stored in these registers may either be held, decremented, or read. Independent control of both the registers during a single cycle is supported with the exception of a simultaneous decrement of both registers.

Stack

The positive edge clocked 16-bit address stack allows multiple levels of nested calls or interrupts and can be used to support branching and looping. Seven stack operations are possible:

1. Reset, which pulls all Y outputs low and clears the stack pointer and read pointer
2. Clear, which sets the stack pointer and read pointer to zero
3. Pop, which causes the stack pointer to be decremented
4. Push, which puts the contents of the MPC, interrupt return register, or DRA bus onto the stack and increments the stack pointer
5. Read, which makes the address indicated by the read pointer available at the DRA port
6. Hold, which causes the address of the stack and read pointers to remain unchanged
7. Load stack pointer, which inputs the seven least significant bits of DRA to the stack pointer.

Stack Pointer

The stack pointer (SP) operates as an up/down counter; it increments whenever a push occurs and decrements whenever a pop occurs. Although push and pop are two event operations (store then increment SP, or decrement SP then read), the 'ACT8818 performs both events within a single cycle.

Read Pointer

The read pointer (RP) is provided as a tool for debugging microcoded systems. It permits a nondestructive, sequential read of the stack contents from the DRA port. This capability provides the user with a method of backtracking through the address sequence to determine the cause of overflow without affecting program flow, the status of the stack pointer, or the internal data of the stack.

Stack Warning/Read Error Pin

A high signal on the STKWRN/RER pin indicates a potential stack overflow or underflow condition. STKWRN/RER becomes active under two conditions. If 62 of the 65 stack locations (0-64) are full (the stack pointer is at 62) and a push occurs, the STKWRN/RER pin outputs a high signal to warn that the stack is approaching its capacity and will be full after two more pushes.

The STKWRN/RER signal will remain high if hold, push or pop instructions occur, until the stack pointer is decremented to 62. If a push instruction is attempted when the stack is full, the new address will be ignored and the old address in stack location 64 will be retained.

The STKWRN/RER pin will go high when the stack pointer is less than or equal to one and a pop or read from stack is coded on the S2-S0 pins. The pin will go high after reading the next to the bottom stack address (1). When the S2-S0 pins are set to pop or read the last address (0) or to pop or read an empty stack, the STKWRN/RER pin will go high. The pin depends only on the setting of the S2-S0 pins and the stack pointer, not on the clock.

2

Interrupt Return Register

Unlike the MPC register, which normally gets next address plus one, the interrupt return register simply gets next address. This permits interrupts to be serviced with zero latency, since the interrupt vector replaces the pending address.

The interrupting hardware disables the Y output and forces the vector onto the microaddress bus. This event must be synchronized with the system clock. The first address of the service routine must program INT low and perform a push to put the contents of the interrupt return register on the stack.

Micropogramming the 'ACT8818

Micropogramming is unlike programming monolithic processors for several reasons. First, the width of the microinstruction word is only partially constrained by the basic signals required to control the sequencer. Since the main advantage of a microprogrammed processor is speed, many operations are often supported by or carried out in special purpose hardware. Lookup tables, extra registers, address generators, elastic memories, and data acquisition circuits may also be controlled by the microinstruction.

The number of slices in a bit-slice ALU is user-defined, which makes the microinstruction width even more application dependent. Types of instructions resulting from manipulation of the sequencer controls are discussed below. Examples of some commonly used instructions can be found in the later section of microinstructions and flow diagrams. The following abbreviations are used in the tables in this section:

BR A	$Y \leftarrow DRA$
BR A'	$Y \leftarrow DRA'$
BR B	$Y \leftarrow DRB$
BR B'	$Y \leftarrow DRB'$
BR S	$Y \leftarrow STK$
CALL A	$Y \leftarrow DRA; STK \leftarrow MPC; SP \leftarrow SP + 1; RP \leftarrow RP + 1$
CALL B	$Y \leftarrow DRB; STK \leftarrow MPC; SP \leftarrow SP + 1; RP \leftarrow RP + 1$
CALL A'	$Y \leftarrow DRA'; STK \leftarrow MPC; SP \leftarrow SP + 1; RP \leftarrow RP + 1$
CALL B'	$Y \leftarrow DRB'; STK \leftarrow MPC; SP \leftarrow SP + 1; RP \leftarrow RP + 1$
CALL S	$Y \leftarrow STK; STK \leftarrow MPC; SP \leftarrow SP + 1; RP \leftarrow RP + 1$
CLR SP, RP	$SP \leftarrow 0; RP \leftarrow \text{points to TOS register}$
CONT/RPT	$Y \leftarrow MPC + 1 \text{ if INC} = H; Y \leftarrow MPC \text{ if INC} = L$
DRA	Bidirectional data port (can be loaded externally or from RCA)
DRA'	DRA15-DRA4::B3-B0
DRB	Bidirectional data port (can be loaded externally or from RCB)
DRB'	DRB15-DRB4::B3-B0
MPC	Microprogram counter
POP	$SP \leftarrow SP - 1; RP \leftarrow RP - 1$
PUSH	$STK \leftarrow \text{operand}; SP \leftarrow SP + 1; RP \leftarrow RP + 1$
RCA	Register/counter A
RCB	Register/counter B
READ	$DRA \leftarrow STK; RP \leftarrow RP - 1; SP \leftarrow SP - 1$
RESET	$Y \leftarrow 0; SP \leftarrow 0; RP \leftarrow \text{points to TOS register}$
RP	Read pointer
SP	Stack pointer
STK	Stack

Address Selection

Y-output multiplexer controls MUX2-MUX0 select one of eight 3-source branches as shown in Table 4. The states of \overline{CC} and ZERO determine which of the three sources is selected as the next address. ZERO is set at the beginning of any cycle in which a register/counter will decrement to zero. This applies to both internal ZERO and external ZEROOUT signals.

Table 4. Output Controls (MUX2-MUX0)

MUX2-MUX0	RESET	Y OUTPUT SOURCE			
		$\overline{CC} = L$		$\overline{CC} = H$	
		ZERO = L	ZERO = H		
XXX	Yes	All Low	All Low	All Low	
LLL	No	STK	MPC	DRA	
LLH	No	STK	MPC	DRB	
LHL	No	STK	DRA	MPC	
LHH	No	STK	DRB	MPC	
HLL	No	DRA	MPC	DRB	
HLH	No	DRA [†]	MPC	DRB [‡]	
HHL	No	DRA	STK	MPC	
HHH	No	DRB	STK	MPC	

[†]DRA15-DRA4::B3-B0

[‡]DRB15-DRB4::B3-B0

By programming \overline{CC} high or low without decrementing registers, only one outcome is possible; thus, unconditional branches or continuations can be implemented by forcing the condition code. Alternatively, \overline{CC} can be selected from status, in which case Branch A on Condition Code Else Branch B instructions are possible, where A and B are the address sources determined by MUX2-MUX0.

Decrement and Branch on Nonzero instructions, creating loops that repeat until a terminal count is reached, can be implemented by programming \overline{CC} low and decrementing a register/counter. If \overline{CC} is selected from status and registers are decremented, more complex instructions such as Exit on Condition Code or End or Loop are possible.

When MUX2-MUX0 = HLH, the B3-B0 inputs can replace the four least significant bits of DRA or DRB to create 16-Way branches or, when \overline{CC} is based on status, to create 32-way branches.

Stack Controls

As in the case of the MUX controls, each stack-control coding is a three-way choice based on \overline{CC} and ZERO (see Table 5). This allows push, pop, or hold stack operations to occur in parallel with the aforementioned branches. A subroutine call is accomplished by combining a branch and push, while returns result from coding a branch to stack with a pop.

Table 5. Stack Controls (S2-S0)

S2-S0	OSEL	STACK OPERATION			
		$\overline{CC} = L$		$\overline{CC} = H$	
		ZERO = L	ZERO = H		
LLL	X	Reset/Clear	Reset/Clear	Reset/Clear	
LLH	X	Clear SP/RP	Hold	Hold	
LHL	X	Hold	Pop	Pop	
LHH	X	Pop	Hold	Hold	
HLL	X	Hold	Push	Push	
HLH	X	Push	Hold	Hold	
HHL	X	Push	Hold	Push	
HHH	H	Read	Read	Read	
HHH	L	Hold	Hold	Hold	

Combining stack and MUX controls with status results and register decrements permits even greater complexity. For example: Return on Condition Code or End of Loop; Call A on Condition Code Else Branch to B; Decrement and Return on Nonzero; Call 16-Way.

Diagnostic stack dumps are possible using Read (S2-S0 = HHH) when OSEL is set high.

Register Controls

Unlike stack and MUX controls, register control is not dependent upon \overline{CC} and ZERO. Registers can be independently loaded, decremented, or held using register control inputs RC2-RC0 (see Table 6). All combinations are supported with the exception of simultaneous register decrements. The register control inputs can be set to store branch addresses and loop counts or to decrement loop counts, facilitating the complex branching instructions described above.

Table 6. Register Controls (RC2-RC0)

RC2-RC0	REGISTER OPERATIONS	
	REG A	REG B
LLL	Hold	Hold
LLH	Decrement	Hold
LHL	Load	Hold
LHH	Decrement	Load
HLL	Load	Load
HLH	Hold	Decrement
HHL	Hold	Load
HHH	Load	Decrement

The contents of RCA are accessible to the DRA port when OSEL is low and the output bus is enabled by \overline{RAOE} being low. Data from RCB is available when DRB is enabled by \overline{RBOE} being low.

Continue/Repeat Instructions

The most commonly used instruction is a continue, implemented by selecting MPC at the Y output MUX and setting INC high. If MPC is selected and INC is off, the current instruction will simply be repeated.

A repeat instruction can be implemented in two ways. A programmed repeat (INC forced low) may be useful in generating wait states, for example, wait for interrupt.

A conditional repeat (INC originates from status) may be useful in implementing Do While operations. Several bit patterns in the MUX control field of the microinstruction will place MPC on the microaddress bus.

Branch Instructions

A branch or jump to a given microaddress can also be coded several ways. RCA, DRA, RCB, DRB, and STK are possible sources for branch addresses (see Table 4). Branches to register or stack are useful whenever the branch address could be stored to reduce overhead.

The simplest branches are to DRA and DRB, since they require only one cycle and the branch address is supplied in the microinstruction. Use of registers or stack requires an initial load cycle (which may be combined with a preceding instruction), but may be more practical when an entry point is referenced over and over throughout the micropogram, for example, in error-handling routines. Branches to stack or register also enhance sequencing techniques in which a branch address is dynamically computed or multiple branches to a common entry point are used, but the entry point varies according to the system state. In this case, the state change might require reloading the stack or register.

In order to force a branch to DRA or DRB, \overline{CC} must be programmed high or low. A branch to stack is only possible when \overline{CC} is forced low (see Table 4).

When \overline{CC} is low, the ZERO flag is tested, and if a register decrements to zero the branch will be transformed into a Decrement and Branch on Nonzero instruction. Therefore, registers should not be decremented during branch instructions using $\overline{CC} = 0$ unless it is certain the register will not reach terminal count. Call (Branch and Push MPC) instructions and Return (Branch to Stack and Pop) instructions are discussed in later sections.

Conditional Branch Instructions

Perhaps the most useful of all branches is the conditional branch. The 'ACT8818 permits three modes of conditional branching: Branch on Condition Code; Branch 16-Way from DRA or DRB; and Branch on Condition Code 16-Way from DRA Else Branch 16-Way from DRB. This increases the versatility of the system and the speed of processing status tests because both single-bit and 4-bit status are allowed.

Testing single bit status is preferred when the status can be set up and selected through a status MUX prior to the conditional branch. Four-bit status allows the 'ACT8818 to process instructions based on Boolean status expressions, such as Branch if Overflow and Not Carry if Zero or if Negative. It also permits true n-way branches, such as If Negative then Branch to X, Else if Overflow, and Not Carry then Branch to Y. The tradeoff is speed versus program size. Since multiway branching occurs relatively infrequently in most programs, users will enjoy increased speed at a negligible cost. Call (Branch and Push MPC) instructions and Return (Branch to Stack and Pop) instructions are discussed in later sections.

Loop Instructions

Up to two levels of nested loops are possible when both counters are used simultaneously. Loop count and levels of nesting can be increased by adding external counters if desired. The simplest and most widely used of the loop instructions is Decrement and Branch on Nonzero, in which \overline{CC} is forced low while a register is decremented. As before, many forms are possible, since the top-of-loop address can originate from RCA, DRA, RCB, DRB, or the stack (see Table 4). Upon terminal count, instruction flow can either drop out of the bottom of the loop or branch elsewhere.

When loops are used in conjunction with \overline{CC} as status, B3-B0 as status and/or stack manipulation, many useful instructions are possible, including Decrement and Branch on Nonzero else Return, Decrement and Call on Nonzero, and Decrement and Branch 16-Way on Nonzero. Possible variations are summarized in Table 7. Call (Branch and Push MPC) instructions and Return (Branch to Stack and Pop) instructions are discussed in later sections.

Another level of complexity is possible if \overline{CC} is selected from status while looping. This type of loop will exit either because \overline{CC} is true or because a terminal count has been reached. This makes it possible, for example, to search the ALU for a bit string. If the string is found, the match forces \overline{CC} high. However, if no match is found, it is necessary to terminate the process when the entire word has been scanned. This complex process can then be implemented in a simple compact loop using Conditional Decrement and Branch on Nonzero.

Table 7. Decrement and Branch on Nonzero Encodings

MUX2- MUX0	SE-S0	OSEL	$\overline{CC} = L$		$\overline{CC} = H$
			ZERO = L	ZERO = H	
LLL	LLH	X	BR S: CLR SP/RP	CONT/RPT	BR A
LLL	LHL	X	BR S	CONT/RPT: POP	BR A: POP
LLL	HLL	X	BR S	CONT/RPT: PUSH	CALL A
LLL	HHH	0	BR S	CONT/RPT	BR A
LLL	HHH	1	BR S: READ	CONT/RPT: READ	BR A: READ
LLH	LLH	X	BR S: CLR SP/RP	CONT/RPT	BR B
LLH	LHL	X	BR S	CONT/RPT: POP	BR B: POP
LLH	HLL	X	BR S	CONT/RPT: PUSH	CALL B
LLH	HHH	0	BR S	CONT/RPT	BR B
LLH	HHH	1	BR S: READ	CONT/RPT: READ	BR B: READ
LHL	LLH	X	BR S: CLR SP/RP	BR A	CONT/RPT
LHL	LHL	X	BR S	BR A: POP	CONT/RPT: POP
LHL	HLL	X	BR S	CALL A	CONT/RPT: PUSH
LHL	HHH	0	BR S	BR A	CONT/RPT
LHL	HHH	1	BR S: READ	BR A: READ	CONT/RPT: READ
LHH	LLH	X	BR S: CLR SP/RP	BR B	CONT/RPT
LHH	LHL	X	BR S	BR B: POP	CONT/RPT: POP
LHH	HLL	X	BR S	CALL B	CONT/RPT: PUSH
LHH	HHH	0	BR S	BR B	CONT/RPT
LHH	HHH	1	BR S: READ	BR B: READ	CONT/RPT: READ
HLL	LLH	X	BR A: CLR SP/RP	CONT/RPT	BR B
HLL	LHL	X	BR A	CONT/RPT: POP	BR B: POP
HLL	LHH	X	BR A: POP	CONT/RPT	BR B
HLL	HLL	X	BR A	CONT/RPT: PUSH	CALL B
HLL	HHH	0	BR A	CONT/RPT	BR B
HLL	HHH	1	BR A: READ	CONT/RPT: READ	BR B: READ
HLH	LLH	X	BR A' (16-way): CLR SP/RP	CONT/RPT	BR B' (16-way)
HLH	LHL	X	BR A' (16-way)	CONT/RPT: POP	BR B' (16-way): POP
HLH	LHH	X	BR A' (16-way): POP	CONT/RPT	BR B' (16-way)
HLH	HLL	X	BR A' (16-way)	CONT/RPT: PUSH	CALL B'(16-way)
HLH	HHH	0	BR A' (16-way)	CONT/RPT	BR B' (16-way)
HLH	HHH	1	BR A' (16-way): READ	CONT/RPT: READ	BR B' (16-way): READ
HHL	LLH	X	BR A: CLR SP/RP	BR S	CONT/RPT
HHL	LHL	X	BR A	RET (BRS: POP)	CONT/RPT: POP
HHL	LHH	X	BR A: POP	BR S	CONT/RPT
HHL	HLL	X	BR A	CALL S	CONT/RPT: PUSH
HHL	HHH	0	BR A	BR S	CONT/RPT
HHL	HHH	1	BR A: READ	BR S: READ	CONT/RPT: READ

Table 7. Decrement and Branch on Nonzero Encodings (Continued)

MUX2- MUX0	SE-S0	OSEL	$\overline{CC} = L$		$\overline{CC} = H$
			ZERO = L	ZERO = H	
HHH	LLH	X	BR B: CLR SP/RP	BR S	CONT/RPT
HHH	LHL	X	BR B	RET	CONT/RPT: POP
HHH	LHH	X	BR B: POP	BR S	CONT/RPT
HHH	HLL	X	BR B	CALL S	CONT/RPT: PUSH
HHH	HHH	0	BR B	BR S	CONT/RPT
HHH	HHH	1	BR B: READ	BR S: READ	CONT/RPT: READ

Subroutine Calls

The various branch instructions described above can be merged with a push instruction to implement subroutine calls in a single cycle. Calls, conditional calls, and Decrement and Call on Nonzero are the most obvious.

Since a push is conditional on \overline{CC} and ZERO, many hybrid instructions are also possible, such as Call X on Condition Code Else Branch, or Decrement and Return on Nonzero Else Branch. Codes that cause subroutine calls are summarized in Tables 8 and 9.

Table 8. Call Encodings without Register Decrements

MUX2-MUX0	S2-S0	OSEL	$\overline{CC} = L$ (ZERO = L)	$\overline{CC} = H$
LLL	HLH	X	CALL S	BR A
LLL	HHL	X	CALL S	CALL A
LLH	HLH	X	CALL S	BR B
LLH	HHL	X	CALL S	CALL B
LHL	HLH	X	CALL S	CONT/RPT
LHL	HHL	X	CALL S	CONT/RPT: PUSH
LHH	HLH	X	CALL S	CONT/RPT
LHH	HHL	X	CALL S	CONT/RPT: PUSH
HLL	HLH	X	CALL A	BR B
HLL	HHL	X	CALL A	CALL B
HLH	HLH	X	CALL A' (16-way)	BR B' (16-way)
HLH	HHL	X	CALL A' (16-way)	CALL B' (16-way)
HHL	HLH	X	CALL A	CONT/RPT
HHL	HHL	X	CALL A	CONT/RPT: PUSH
HHH	HLH	X	CALL B	CONT/RPT
HHH	HHL	X	CALL B	CONT/RPT: PUSH

Table 9. Call Encodings with Register Decrements

MUX2-MUX0	S2-S0	OSEL	$\overline{CC} = L$		$\overline{CC} = H$
			ZERO = L	ZERO = H	
LLL	HLH	X	CALL S	CONT/RPT	BR A
LLL	HHL	X	CALL S	CONT/RPT	CALL A
LLH	HLH	X	CALL S	CONT/RPT	BR B
LLH	HHL	X	CALL S	CONT/RPT	CALL B
LHL	HLH	X	CALL S	BR A	CONT/RPT
LHL	HHL	X	CALL S	BR A	CONT/RPT: PUSH
LHH	HLH	X	CALL S	BR B	CONT/RPT
LHH	HHL	X	CALL S	BR B	CONT/RPT: PUSH
HLL	HLH	X	CALL A	CONT/RPT	BR B
HLL	HHL	X	CALL A	CONT/RPT	CALL B
HLH	HLH	X	CALL A' (16-way)	CONT/RPT	BR B' (16-way)
HLH	HHL	X	CALL A' (16-way)	CONT/RPT	CALL B' (16-way)
HHL	HLH	X	CALL A	BR S	CONT/RPT
HHL	HHL	X	CALL A	BR S	CONT/RPT: PUSH
HHH	HLH	X	CALL B	BR S	CONT/RPT
HHH	HHL	X	CALL B	BR S	CONT/RPT: PUSH

Subroutine Returns

A return from subroutine can be implemented by coding a branch to stack with a pop. Since pop is also conditional on \overline{CC} and ZERO, the complex forms discussed previously also apply to return instructions: Decrement and Return on Nonzero; Return on Condition Code; Branch on Condition Code Else Return. Return encodings are summarized in Tables 10 and 11.

Table 10. Return Encodings without Register Decrements

MUX2-MUX0	S2-S0	OSEL	$\overline{CC} = L$	$\overline{CC} = H$
LLL	LHH	X	RET	BR A
LLH	LHH	X	RET	BR B
LHL	LHH	X	RET	CONT/RPT
LHH	LHH	X	RET	CONT/RPT

Table 11. Return Encodings with Register Decrements

MUX2-MUX0	S2-S0	OSEL	$\overline{CC} = L$		$\overline{CC} = H$
			ZERO = L	ZERO = H	
LLL	LHH	X	RET	CONT/RPT	BR A
LLH	LHH	X	RET	CONT/RPT	BR B
LHL	LHH	X	RET	BR A	CONT/RPT
LHH	LHH	X	RET	BR B	CONT/RPT
HHL	LHL	X	BR A	RET	CONT/RPT: POP
HHH	LHL	X	BR B	RET	CONT/RPT: POP

Reset

Pulling the S2-S0 pins low clears the stack and read pointers, and zeroes the Y output multiplexer (See Table 5).

Clear Pointers

The stack and read pointers may be cleared without affecting the Y output multiplexer by setting S2-S0 to LLH and forcing \overline{CC} low (see Table 5).

Read Stack

Placing a high value on all of the stack inputs (S2-S0) and OSEL places the 'ACT8818 into the read mode. At each low-to-high clock transition, the address pointed to by the read pointer is available at the DRA port and the read pointer is decremented. The bottom of the stack is detected by monitoring the stack warning/read error pin (STKWRN/RER). A high appears on the STKWRN/RER output when the stack contains one word and a read instruction is applied to the S2-S0 pins. This signifies that the last address has been read.

The stack pointer and stack contents are unaffected by the read operation. Under normal push and pop operations, the read pointer is updated with the stack pointer and contains identical information.

Interrupts

Real-time vectored interrupt routines are supported for those applications where polling would impede system throughput. Any instruction, including pushes and pops, may be interrupted. To process an interrupt, the following procedure should be followed:

1. Place the bidirectional Y bus into a high-impedance state by forcing \overline{YOE} high.
2. Force the interrupt entry point vector onto the Y bus. INC should be high.
3. Push the current value in the Interrupt Return register on the stack as the execution address to return to when interrupt handling is complete.

The first instruction of the interrupt routine must push the address stored in the interrupt return register onto the stack so that proper return linkage is maintained. This is accomplished by setting \overline{INT} and B1 low and coding a push on the stack.

Sample Microinstructions for the 'ACT8818

Representative examples of instructions using the 'ACT8818 are given below. The examples assume a one-level pipeline system, in which the address and contents of the next instruction are being fetched while the current instruction is being executed, and an ALU status register contains the status results of the previous instruction.

Since the incrementer looks two addresses ahead of the address in the instruction register to set up some instructions such as continue or repeat, a set-up instruction has been included with each example. This shows the required state of both INC and \overline{CC} . \overline{CC} must be set up early because the status register on which Y-output selection is typically based contains the results of the previous instruction.

Flow diagrams and suggested code for the sample microinstructions are also given below. Numbers inside the circles are microword address locations expressed as hexadecimal numbers. Fields in microinstructions are binary numbers except for inputs on DRA or DRB, which are also in hexadecimal. For a discussion of sequencing instructions, see the preceding section on microprogramming.

Continue

To Continue (Instruction 10), INC and \overline{CC} must be programmed high one cycle ahead of instruction 10 for pipelining.

Address	Instruction	MUX2-MUX0	S2-S0	R2-R0	OSEL	\overline{CC}	INC	DRA	DRB
(Set-up)		XXX	XXX	XXX	X	1	1	XXXX	XXXX
10	Continue	110	111	XXX	0	X	X	XXXX	XXXX

Continue and Pop

To Continue and decrement the stack pointer (Pop), INC and \overline{CC} are forced high in the previous instruction.

Address	Instruction	MUX2-MUX0	S2-S0	R2-R0	OSEL	\overline{CC}	INC	DRA	DRB
(Set-up)		XXX	XXX	XXX	X	1	1	XXXX	XXXX
10	Continue/Pop	110	010	XXX	X	X	X	XXXX	XXXX

Continue and Push

To Continue and push the microprogram counter onto the stack (Push), INC and \overline{CC} are forced high one cycle ahead of Instruction 10 for pipelining.

Address	Instruction	MUX2-MUX0	S2-S0	R2-R0	OSEL	\overline{CC}	INC	DRA	DRB
(Set-up)		XXX	XXX	XXX	X	1	1	XXXX	XXXX
10	Continue/Push	110	100	XXX	0	X	X	XXXX	XXXX

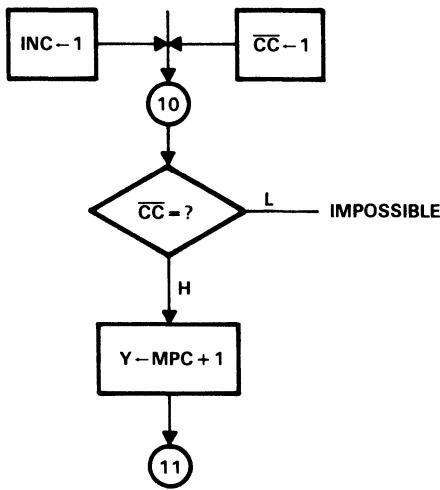


Figure 5. Continue

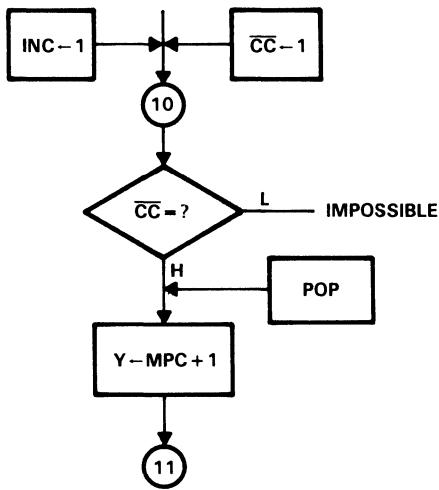


Figure 6. Continue and Pop

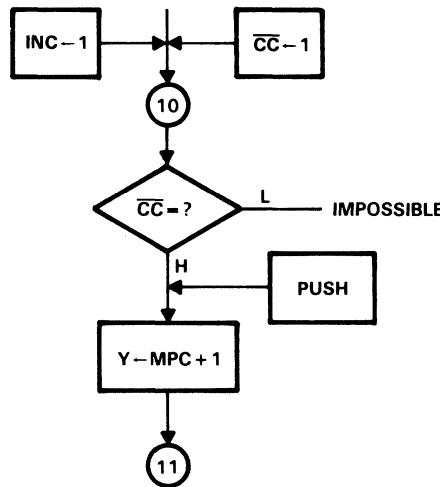


Figure 7. Continue and Push

Branch (Example 1)

To Branch from address 10 to address 20, \overline{CC} must be programmed high one cycle ahead of Instruction 10 for pipelining.

Address	Instruction	MUX2-MUX0	S2-S0	R2-R0	OSEL	\overline{CC}	INC	DRA	DRB
(Set-up)		XXX	XXX	XXX	X	1	X	XXXX	XXXX
10	BR A	000	111	XXX	0	X	X	0020	XXXX

Branch (Example 2)

To Branch from address 10 to address 20, \overline{CC} is programmed low in the previous instruction; as a result, a ZERO test follows the condition code test in Instruction 10. To ensure that a ZERO = H condition will not occur, registers should not be decremented during this instruction.

Address	Instruction	MUX2-MUX0	S2-S0	R2-R0	OSEL	\overline{CC}	INC	DRA	DRB
(Set-up)		XXX	XXX	XXX	X	0	X	XXXX	XXXX
10	BR A	110	111	000	0	X	X	0020	XXXX

Sixteen-Way Branch

To Branch 16-Way, \overline{CC} is programmed high in the previous instruction. The branch address is derived from the concatenation DRB15-DRB4::B3-B0.

Address	Instruction	MUX2-MUX0	S2-S0	R2-R0	OSEL	\overline{CC}	INC	DRA	DRB
(Set-up)		XXX	XXX	XXX	X	1	X	XXXX	XXXX
10	BR B'	101	111	XXX	0	X	X	XXXX	0040

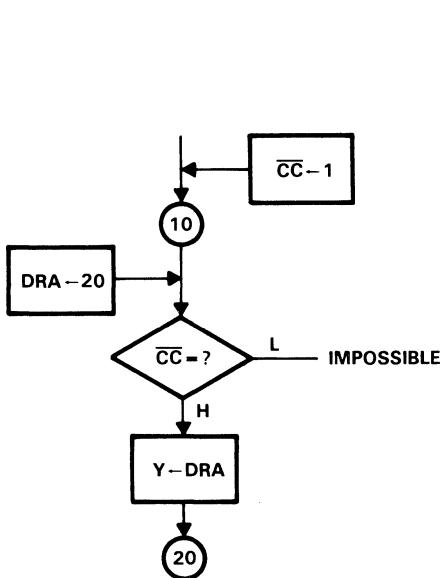


Figure 8. Branch Example 1

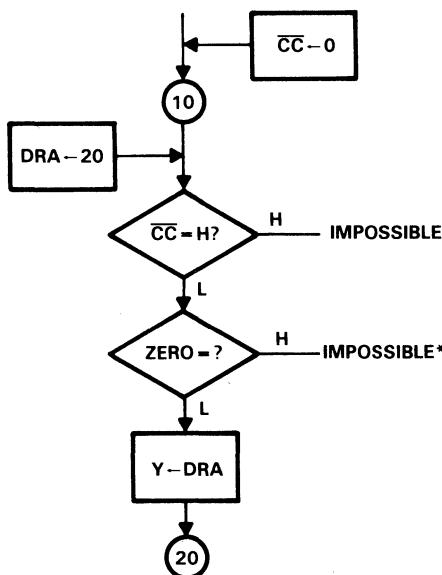


Figure 9. Branch Example 2

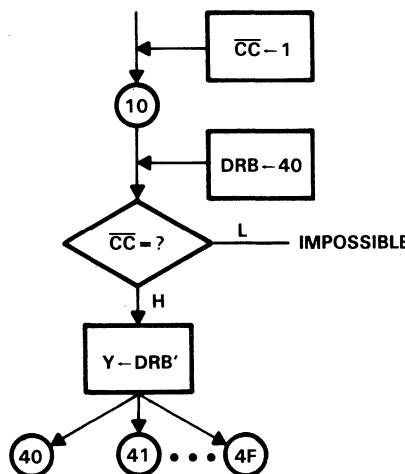


Figure 10. Sixteen-Way Branch

Conditional Branch

To Branch to address 20 Else Continue to address 11, INC is set high in the preceding instruction to set up the Continue.

Address	Instruction	MUX2-MUX0	S2-S0	R2-R0	OSEL	CC'	INC	DRA	DRB
(Set-up)		XXX	XXX	XXX	X	X	1	XXXX	XXXX
10	BR A else Continue	110	111	000	0	X	X	0020	XXXX

Three-Way Branch

To Branch 3-Way, this example uses an instruction from Table 7 with BR A in the ZERO = L column, CONT/RPT in the ZERO = H column and BR B in the \overline{CC} = H column. To enable the ZERO = H path, register A must decrement to zero during this instruction (see Table 6 for possible register operations). INC is programmed high in Instruction 10 to set up the Continue.

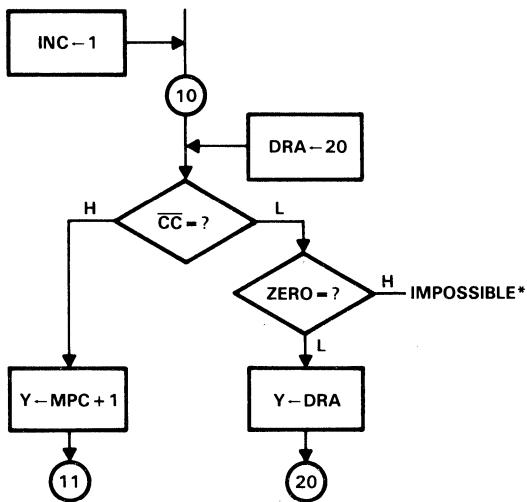
Address	Instruction	MUX2-MUX0	S2-S0	R2-R0	OSEL	\overline{CC}	INC	DRA	DRB
(Set-up)		XXX	XXX	XXX	X	1	1	XXXX	XXXX
10	Continue and Load Reg A	110	111	010	0	t	1	XXXX	XXXX
11	Decrement Reg A; Branch 3-Way	100	111	001	0	X	X	0020	0030

^tSelected from external status

Thirty-Two-Way Branch

To Branch 32-Way, the four least significant bits of the DRA' and DRB' addresses must be input at the B3-B0 port; these are concatenated with the 12 most significant bits of DRA and DRB to provide new addresses DRA' (DRA15-DRA4::B3-B0) and DRB' (DRB15-DRB4::B3-B0).

Address	Instruction	MUX2-MUX0	S2-S0	R2-R0	OSEL	\overline{CC}	INC	DRA	DRB
(Set-up)		XXX	XXX	XXX	X	X	1	XXXX	XXXX
10	32-way Branch	101	111	000	0	X	X	0040	0030



*no register decrement

Figure 11. Conditional Branch

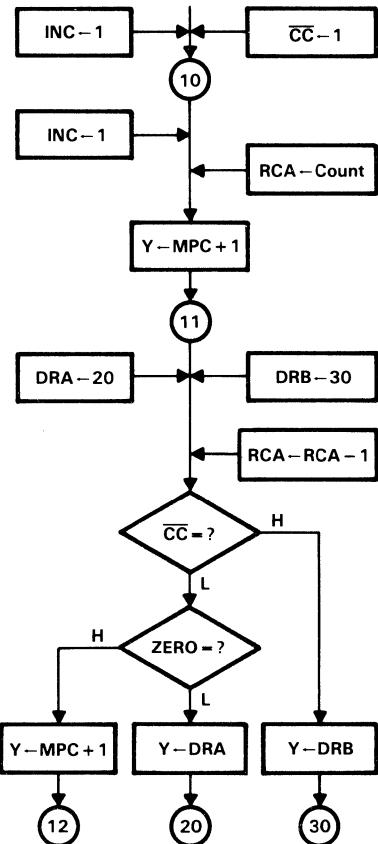
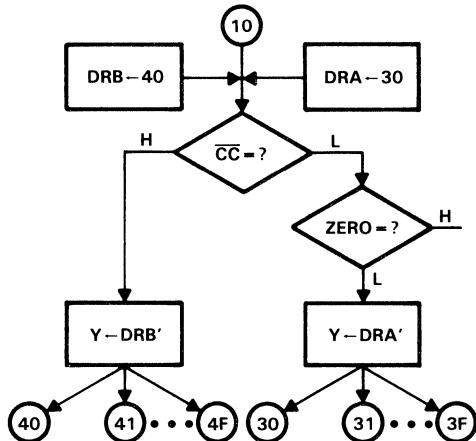


Figure 12. Three-Way Branch



*no register decrement

Figure 13. Thirty-Two-Way Branch

Repeat

To Repeat (Instruction 10), INC must be programmed low and \overline{CC} high one cycle ahead of Instruction 10 for pipelining.

Address	Instruction	MUX2-MUX0	S2-S0	R2-R0	OSEL	\overline{CC}	INC	DRA	DRB
(Set-up)		XXX	XXX	XXX	X	1	0	XXXX	XXXX
10	Continue	110	111	XXX	0	X	X	XXXX	XXXX

Repeat on Stack

To Continue and push the microprogram counter onto the stack (Push), INC and \overline{CC} must be forced high one cycle ahead for pipelining.

To Repeat (Instruction 12), an BR S instruction with ZERO = L is used. To avoid a ZERO = H condition, registers are not decremented during this instruction (see Table 6 for possible register operations. \overline{CC} and INC are programmed high in Instruction 12 to set up the Continue in Instruction 11.

Address	Instruction	MUX2-MUX0	S2-S0	R2-R0	OSEL	\overline{CC}	INC	DRA	DRB
(Set-up)		XXX	XXX	XXX	X	1	1	XXXX	XXXX
10	Continue/Push	110	100	XXX	X	1	1	XXXX	XXXX
11	Continue	110	111	XXX	0	0	0	X	XXXX
12	BR Stack	010	111	000	0	1	1	XXXX	XXXX

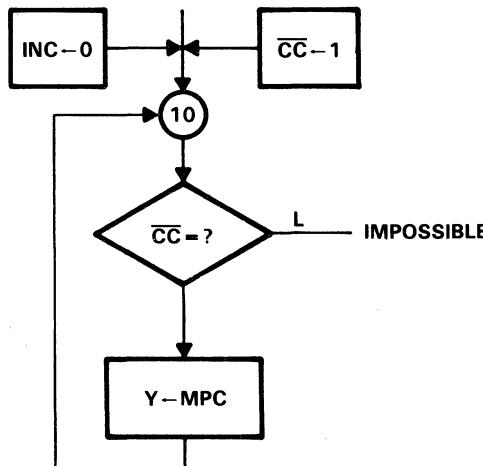
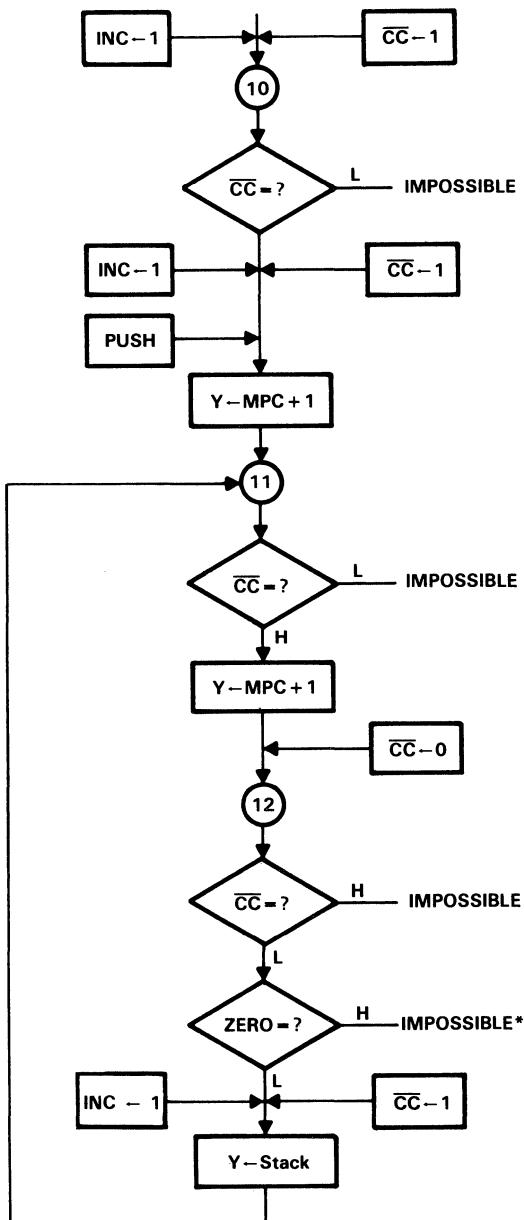


Figure 14. Repeat



*no register decrement

Figure 15. Repeat on Stack

Repeat Until $\overline{CC} = H$

To Continue and push the microprogram counter onto the stack (Push), INC and \overline{CC} must be forced high one cycle ahead for pipelining.

To Repeat Until $\overline{CC} = H$ (Instruction 12), use a BR S instruction with $\overline{CC} = L$ and CONT/RPT: POP instruction with $\overline{CC} = H$. To avoid a ZERO = H condition, registers are not decremented (See Table 6 for possible register operations). \overline{CC} and INC are programmed high in Instruction 12 to set up the Continue in Instruction 11. A consequence of this is that the instruction following 13 cannot be conditional.

Address	Instruction	MUX2-MUX0	S2-S0	R2-R0	OSEL	\overline{CC}	INC	DRA	DRB
(Set-up)		XXX	XXX	XXX	X	1	1	XXXX	XXXX
10	Continue/Push	110	100	XXX	X	1	1	XXXX	XXXX
11	Continue	110	111	XXX	0	†	1	XXXX	XXXX
12	BR Stack else Continue	010	010	000	X	1	1	XXXX	XXXX

†Selected from external status

Loop Until Zero

To Continue and push the microprogram counter onto the stack (Push), INC and \overline{CC} are forced high one cycle ahead for pipelining. Register A is loaded with the loop counter using a Load A instruction from Table 6.

To decrement the loop count, a decrement register A and hold register B instruction from Table 6 is used. To Repeat Else Continue and Pop (decrement the stack pointer), an instruction from Table 7 with BR S in the ZERO = L column and CONT/RPT: POP in the ZERO = H column is used. \overline{CC} is programmed low in Instruction 11 to force the ZERO test in Instruction 12; it is programmed high in Instruction 12 to set up the Continue in Instruction 11.

Address	Instruction	MUX2-MUX0	S2-S0	R2-R0	OSEL	\overline{CC}	INC	DRA	DRB
(Set-up)		XXX	XXX	XXX	X	1	1	XXXX	XXXX
10	Continue/Push	110	100	XXX	0	1	1	XXXX	XXXX
11	Continue/Load Reg A	110	111	010	0	0	1	XXXX	XXXX
12	Decrement Reg A; BR S else Continue: Pop	000	010	001	1	1	1	XXXX	XXXX

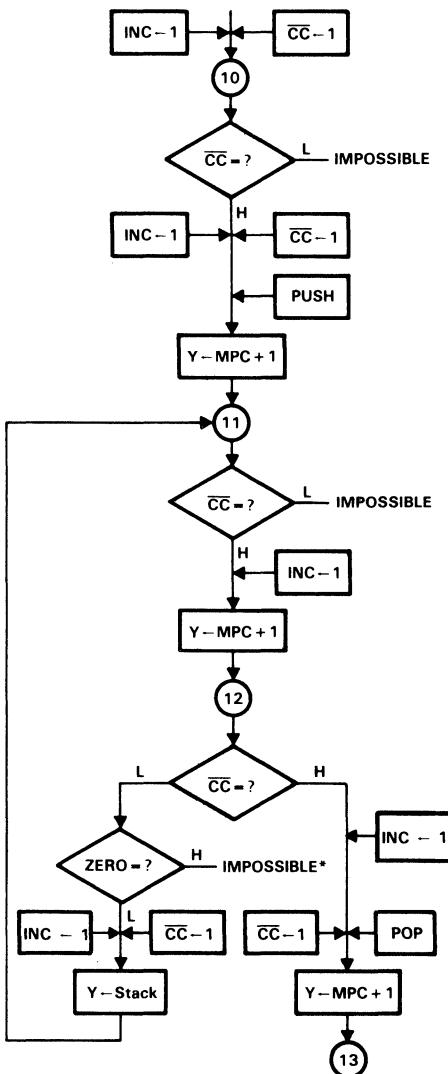
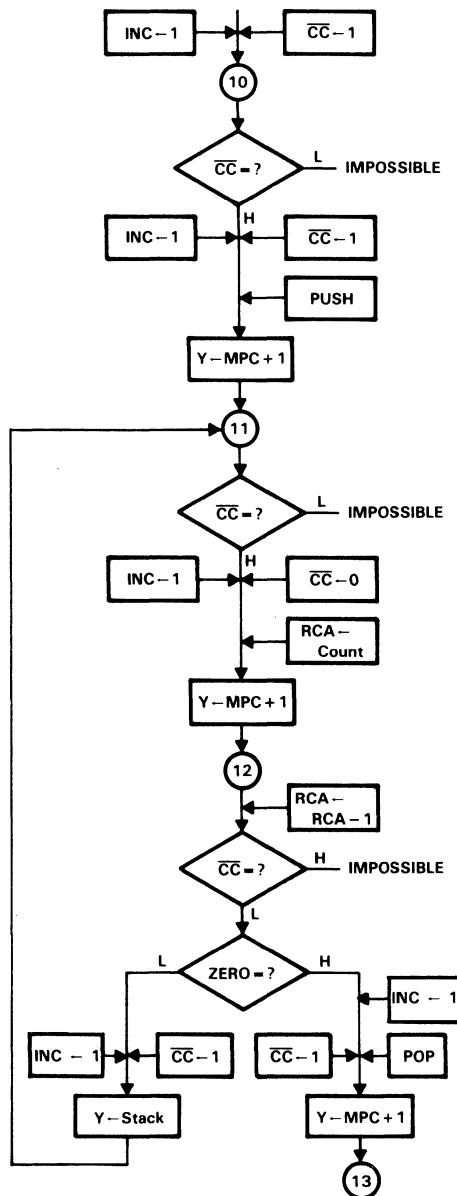
Figure 16. Repeat Until $\overline{CC} = H$ 

Figure 17. Loop Until Zero

Conditional Loop Until Zero

Two examples of a Conditional Loop on Stack with Exit are presented below. Both use the microcode shown below to branch to the stack on nonzero, continue and pop on zero, and branch to DRA with a pop if $\overline{CC} = H$. In the first example, the value on the DRA bus is the same as the value in the microprogram counter, making the exit destinations on the \overline{CC} and ZERO tests the same. In the second, the values are different, generating a two-way exit.

To Continue and push the microprogram counter onto the stack (Push), INC must be high. \overline{CC} is forced high in the preceding instruction for pipelining.

To Continue (Instruction 11), INC must be high. \overline{CC} must be programmed high in the previous instruction. INC is programmed high to set up the Continue in Instruction 12.

To Decrement and Branch else Exit (Instruction 12), an instruction from Table 7 with BR S in the ZERO = L column, CONT/RPT: POP in the ZERO = H column and BR A: POP in the $\overline{CC} = H$ column is used.

Example 1:

Address	Instruction	MUX2-MUX0	S2-S0	R2-R0	OSEL	\overline{CC}	INC	DRA	DRB
(Set-up)		XXX	XXX	XXX	X	1	1	XXXX	XXXX
10	Continue/Push Load Reg A	110	100	010	X	1	1	XXXX	XXXX
11	Continue	110	111	XXX	0	†	1	XXXX	XXXX
12	Decrement Reg A; BR S else Continue: Pop else BR A: Pop	000	010	001	X	1	1	0013	XXXX

† Selected from external status

Example 2:

Address	Instruction	MUX2-MUX0	S2-S0	R2-R0	OSEL	\overline{CC}	INC	DRA	DRB
(Set-up)		XXX	XXX	XXX	X	1	1	XXXX	XXXX
10	Continue/Push Load Reg A	110	100	010	X	1	1	XXXX	XXXX
11	Continue	110	111	XXX	0	†	1	XXXX	XXXX
12	Decrement Reg A; BR S else Continue: Pop else BR A: Pop	000	010	001	X	1	1	0025	XXXX

† Selected from external status

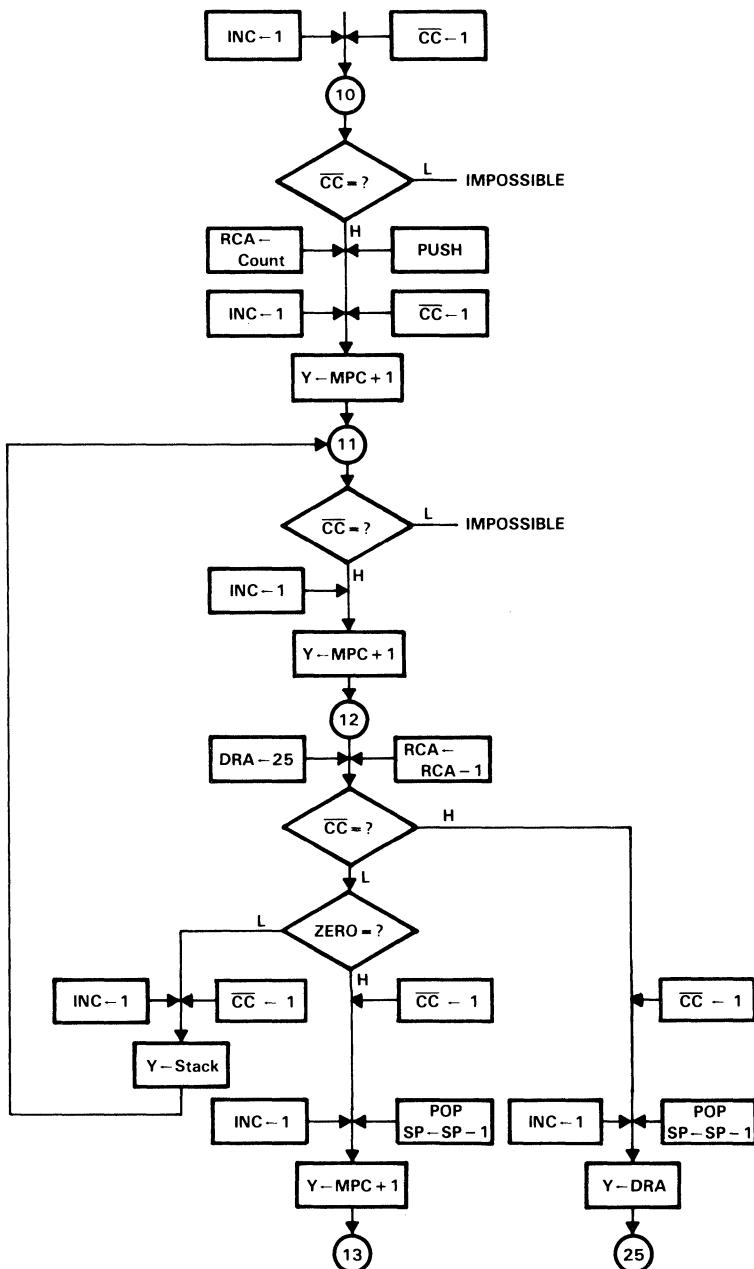


Figure 18. Conditional Loop Until Zero (Example 2)

Jump to Subroutine

To Call a Subroutine at address 30, this example uses the instruction from Table 8 with CALL A in the $\overline{CC} = H$ column. \overline{CC} is programmed high in the previous instruction.

Address	Instruction	MUX2-MUX0	S2-S0	R2-RO	OSEL	\overline{CC}	INC	DRA	DRB
(Set-up)		XXX	XXX	XXX	X	1	1	XXXX	XXXX
10	Call A	000	110	XXX	X	X	X	0030	XXXX

Conditional Jump to Subroutine

To conditionally Call a Subroutine at address 20, this example uses an instruction from Table 8 with CALL A in the $\overline{CC} = L$ column and CONT/RPT in the $\overline{CC} = H$ column. \overline{CC} is generated by external status during the preceding instruction. INC is programmed high in the preceding instruction to set up the Continue. To avoid a ZERO = H condition, registers should not be decremented during Instruction 10.

Address	Instruction	MUX2-MUX0	S2-S0	R2-RO	OSEL	\overline{CC}'	INC	DRA	DRB
(Set-up)		XXX	XXX	XXX	X	†	1	XXXX	XXXX
10	Call A else Continue	110	101	000	X	X	X	0020	XXXX

†Selected from external status

Two-Way Jump to Subroutine

To perform a Two-Way Call to Subroutine at address 20 or address 30, this example uses an instruction from Table 8 with CALL A in the $\overline{CC} = L$ column and CALL B in the $\overline{CC} = H$ column. In this example, \overline{CC} is generated by external status during the preceding (set-up) instruction. INC is programmed high in the preceding instruction to set up the Push. To avoid a ZERO = H condition, registers should not be decremented during Instruction 10.

Address	Instruction	MUX2-MUX0	S2-S0	R2-RO	OSEL	\overline{CC}	INC	DRA	DRB
(Set-up)		XXX	XXX	XXX	X	†	1	XXXX	XXXX
23	Call A else Call B	100	110	000	X	X	X	0020	0030

†Selected from external status

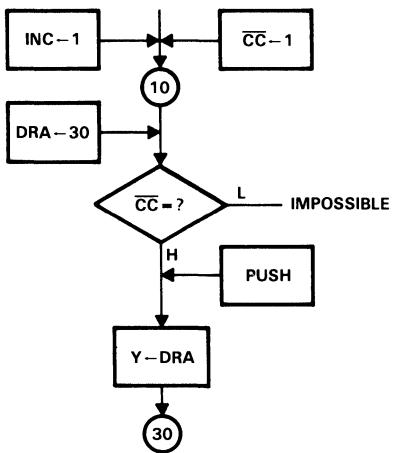
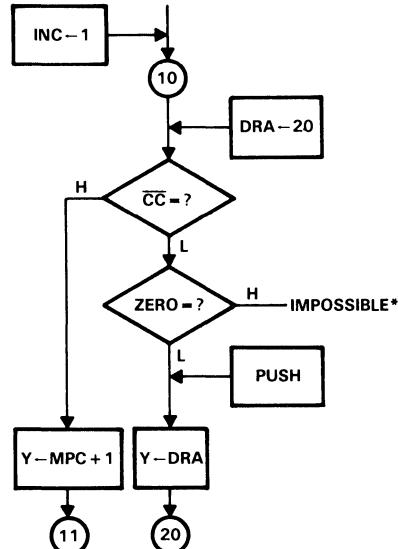
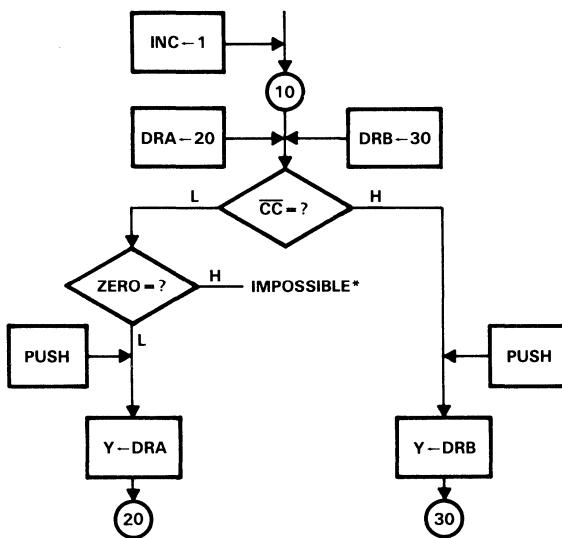


Figure 19. Jump to Subroutine



*no register decrement

Figure 20. Conditional Jump to Subroutine



*no register decrement

Figure 21. Two-Way Jump to Subroutine

Return from Subroutine

To Return from a subroutine, this example uses an instruction from Table 10 with RET in the $\overline{CC} = L$ column. \overline{CC} is programmed low in the previous instruction. To avoid a ZERO = H condition, registers are not decremented during Instruction 23.

Address	Instruction	MUX2-MUX0	S2-S0	R2-R0	OSEL	\overline{CC}	INC	DRA	DRB
(Set-up)		XXX	XXX	XXX	X	0	X	XXXX	XXXX
23	Return	010	011	000	X	X	X	XXXX	XXXX

Conditional Return from Subroutine

To conditionally Return from a Subroutine, this example uses an instruction from Table 10 with RET in the $\overline{CC} = L$ column and CONT/RPT in the $\overline{CC} = H$ column. \overline{CC} is selected from external status in the previous instruction. To avoid a ZERO = H condition, registers are not decremented during Instruction 23.

Address	Instruction	MUX2-MUX0	S2-S0	R2-R0	OSEL	\overline{CC}	INC	DRA	DRB
(Set-up)		XXX	XXX	XXX	X	†	1	XXXX	XXXX
23	Return else Continue	010	011	000	X	X	X	XXXX	XXXX

†Selected from external status

Clear Pointers

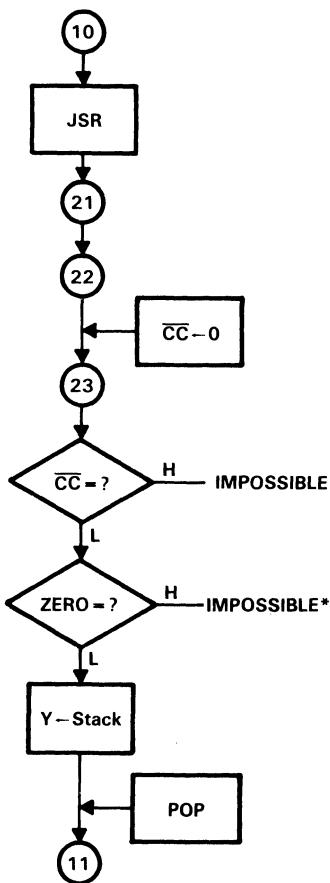
To Continue (Instruction 10), INC must be high; \overline{CC} must be programmed high in the previous instruction. To Clear the Stack and Read Pointers and Branch to address 20 (instruction 11), \overline{CC} is programmed low in instruction 10 to set up the Branch. To avoid a ZERO = H condition, registers are not decremented during Instruction 11.

Address	Instruction	MUX2-MUX0	S2-S0	R2-R0	OSEL	\overline{CC}	INC	DRA	DRB
(Set-up)		XXX	XXX	XXX	X	1	1	XXXX	XXXX
10	Continue	110	111	XXX	0	0	X	0020	XXXX
11	BR A and Clear SP/RP	110	001	000	X	X	X	XXXX	XXXX

Reset

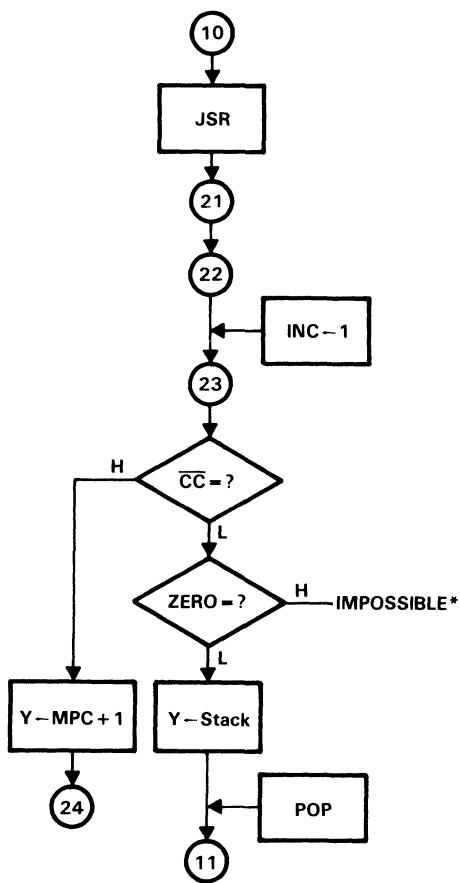
To Reset the 'ACT8818, pull the S2-S0 pins low. This clears the stack and read pointers and places the Y bus into a low state.

Address	Instruction	MUX2-MUX0	S2-S0	R2-R0	OSEL	\overline{CC}	INC	DRA	DRB
10	Reset	XXX	000	XXX	X	X	X	XXXX	XXXX



*no register decrement

Figure 22. Return from Subroutine

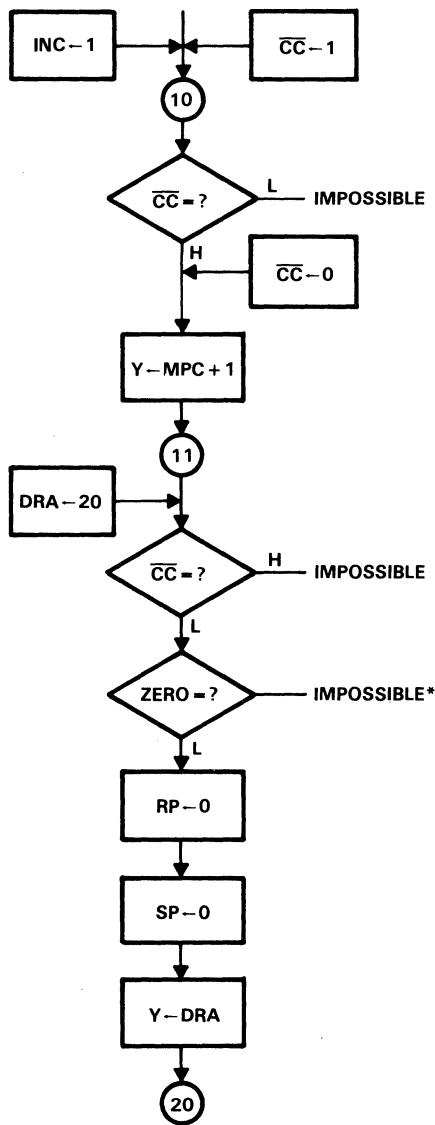


*no register decrement

Figure 23. Conditional
Return from Subroutine

2

SN74ACT8818



*no register decrement

Figure 24. Clear Pointers

Overview

1

SN74ACT8818 16-Bit Microsequencer

2

SN74ACT8832 32-Bit Registered ALU

3

SN74ACT8836 32- × 32-Bit Parallel Multiplier

4

SN74ACT8837 64-Bit Floating Point Processor

5

SN74ACT8841 Digital Crossbar Switch

6

SN74ACT8847 64-Bit Floating Point/Integer Processor

7

Support

8

Mechanical Data

9

3

SN74ACT8832

SN74ACT8832

CMOS 32-Bit Registered ALU

- 50-ns Cycle Time
- Low-Power EPIC™ CMOS
- Three-Port I/O Architecture
- 64-Word by 36-Bit Register File
- Simultaneous ALU and Register Operations
- Configurable as Quad 8-Bit or Dual 16-Bit Single Instruction, Multiple Data Machine
- Parity Generation/Checking

The SN74ACT8832 is a 32-bit registered ALU that can operate at 20 MHz and 20 MIPS (million instructions per second). Most instructions can be performed in a single cycle. The 'ACT8832 was designed for applications that require high-speed logical, arithmetic, and shift operations and bit/byte manipulations.

The 'ACT8832 can act as host CPU or can accelerate a host microprocessor. In high-performance graphics systems, the 'ACT8832 generates display-list memory addresses and controls the display buffer. In I/O controller applications, the 'ACT8832 performs high-speed comparisons to initialize and end data transfers.

A three-operand, 64-word by 36-bit register file allows the 'ACT8832 to create an instruction and store the previous result in a single cycle.

EPIC is a trademark of Texas Instruments Incorporated.

3

SN74ACT8832

Contents

	<i>Page</i>
Introduction	3-13
Understanding Microprogrammed Architecture	3-13
'ACT8832 Registered ALU	3-13
Support Tools	3-14
Design Support	3-15
Systems Expertise	3-15
'ACT8832 Pin Descriptions	3-16
'ACT8832 Specification Tables	3-25
'ACT8832 Registered ALU	3-28
Architecture	3-28
Data Flow	3-29
Architectural Elements	3-31
Three-Port Register File	3-31
R and S Multiplexers	3-32
Data Input and Output Ports	3-34
ALU	3-34
ALU and MQ Shifters	3-36
Bidirectional Serial I/O Pins	3-36
MQ Register	3-37
Conditional Shift Pin	3-37
Master/Slave Comparator	3-37
Divide/BCD Flip-Flops	3-37
Status	3-38
Input Data Parity Check	3-38
Test Pins	3-38
Instruction Set Overview	3-39
Arithmetic/Logic Instructions with Shifts	3-43
Other Arithmetic Instructions	3-46
Data Conversion Instructions	3-48
Bit and Byte Instructions	3-49
Other Instructions	3-49
Configuration Options	3-50
Masked 32-Bit Operation	3-50
Shift Instructions	3-50
Bit and Byte Instructions	3-51
Status Selection	3-51

Contents (Continued)

	<i>Page</i>
Instruction Set	3-52
ABS	3-53
ADD	3-55
ADDI	3-57
AND	3-59
ANDNR	3-61
BADD	3-63
BAND	3-65
BCDBIN	3-67
BINCNS	3-70
BINCS	3-72
BINEX3	3-74
BOR	3-76
BSUBR	3-78
BSUBS	3-80
BXOR	3-82
CLR	3-84
CRC	3-85
DIVRF	3-88
DNORM	3-90
DUMPFF	3-92
EX3BC	3-94
EX3C	3-96
INCNR	3-99
INCNS	3-101
INCR	3-103
INCS	3-105
LOADFF	3-107
LOADMQ	3-109
MQSLC	3-111
MQSLL	3-113
MQSRA	3-115
MQSRL	3-117
NAND	3-119
NOP	3-121
NOR	3-123
OR	3-125
PASS	3-127

Contents (Concluded)

	<i>Page</i>
SDIVI	3-129
SDIVIN	3-131
SDIVIS	3-133
SDIVIT	3-135
SDIVO	3-137
SDIVQF	3-139
SEL	3-141
SETO	3-143
SET1	3-145
SLA	3-147
SLAD	3-149
SLC	3-151
SLCD	3-153
SMTC	3-155
SMULI	3-157
SMULT	3-159
SNORM	3-161
SRA	3-163
SRAD	3-165
SRC	3-167
SRCD	3-169
SRL	3-171
SRLD	3-173
SUBI	3-175
SUBR	3-177
SUBS	3-179
TB0	3-181
TB1	3-183
UDIVI	3-185
UDIVIS	3-187
UDIVIT	3-189
UMULI	3-191
XOR	3-193

3

SN74ACT8832

List of Illustrations

<i>Figure</i>	<i>Title</i>	<i>Page</i>
1	Microprogrammed System Block Diagram	3-14
2	SN74ACT8832 GB Package	3-16
3	SN74ACT8832 Logic Symbol	3-17
4	'ACT8832 32-Bit Registered ALU	3-30
5	Data I/O	3-31
6	16-Bit Configuration	3-34
7	8-Bit Configuration	3-35
8	Shift Examples, 32-Bit Configuration	3-44
9	Shift Examples, 16-Bit Configuration	3-51
10	Shift Examples, 8-Bit Configuration	3-52

3**SN74ACT8832**

List of Tables

<i>Table</i>	<i>Title</i>	<i>Page</i>
1	SN74ACT8832 Pin Grid Allocation	3-18
2	SN74ACT8832 Pin Description	3-19
3	Recommended Operating Conditions	3-25
4	Electrical Characteristics	3-26
5	Register File Write Setup	3-26
6	Maximum Switching Characteristics	3-27
7	'ACT8832 Response to Control Inputs	3-29
8	RF MUX Select Inputs	3-32
9	ALU Source Operand Selects	3-32
10	Destination Operand Select/Enables	3-33
11	Configuration Mode Selects	3-36
12	Data Determining SIO Input	3-36
13	Data Determining BYOF Outputs	3-38
14	Test Pin Inputs	3-39
15	'ACT8832 Instruction Set	3-39
16	Shift Definitions	3-44
17	Bidirectional SIO Pin Functions	3-45
18	Signed Multiplication Algorithm	3-46
19	Unsigned Multiplication Algorithm	3-46
20	Mixed Multiplication Algorithm	3-46
21	Signed Division Algorithm	3-47
22	Unsigned Division Algorithm	3-47
23	BCD to Binary Algorithm	3-48
24	Binary to Excess-3 Algorithm	3-49
25	CRC Algorithm	3-50

3

SN74ACT8832

Introduction

The SN74ACT8832 Registered Arithmetic/Logic Unit (ALU) holds a primary position in the Texas Instruments family of innovative 32-bit LSI devices. Compatible with the SN74AS888 architecture and instruction set, the 'ACT8832 performs as a high-speed microprogrammable 32-bit registered ALU which can also be configured to operate as two 16-bit ALUs or four 8-bit ALUs in single-instruction, multiple-data (SIMD) mode.

Besides introducing the 'ACT8832, this section discusses basic concepts of microprogrammed architecture and the support tools available for system development. Details of the 'ACT8832 architecture and instruction set are presented. Pin descriptions and assignments for the 'ACT8832 are also presented.

Understanding Microprogrammed Architecture

Figure 1 shows a simple microprogrammed system. The three basic components are an arithmetic/logic unit, a microsequencer, and a memory. The program that resides in this memory is commonly called the microprogram, while the memory itself is referred to as a micromemory or control store. The ALU performs all the required operations on data brought in from the external environment (main memory or peripherals, for example). The sequencer is dedicated to generating the next micromemory address from which a microinstruction is to be fetched. The sequencer and the ALU operate in parallel so that data processing and next-address generation are carried out concurrently.

The microprogram instruction, or microinstruction, consists of control information to the ALU and the sequencer. The microinstruction consists of a number of fields of code that directly access and control the ALU, registers, bus transceivers, multiplexers, and other system components. This high degree of programmability in a parallel architecture offers greater speed and flexibility than a typical microprocessor, although the microinstruction serves the same purpose as a microprocessor opcode: it specifies control information by which the user is able to implement desired data processing operations in a specified sequence. The microinstruction cycle is synchronized to a system clock by latching the instruction in the microinstruction, or pipeline, register once for each clock cycle. Status results are collected in a status register which the sequencer samples to produce conditional branches within the microprogram.

'ACT8832 Registered ALU

This device comprises a 32-bit ALU, a 64-word by 36-bit register file, two shifters to support double-precision arithmetic, and three independent bidirectional data ports.

The 'ACT8832 is engineered to support high-speed, high-level operations. The ALU's 13 basic arithmetic and logic instructions can be combined with a single- or double-precision shift operation in one instruction cycle. Other instructions support data conversions, bit and byte operations, and other specialized functions.

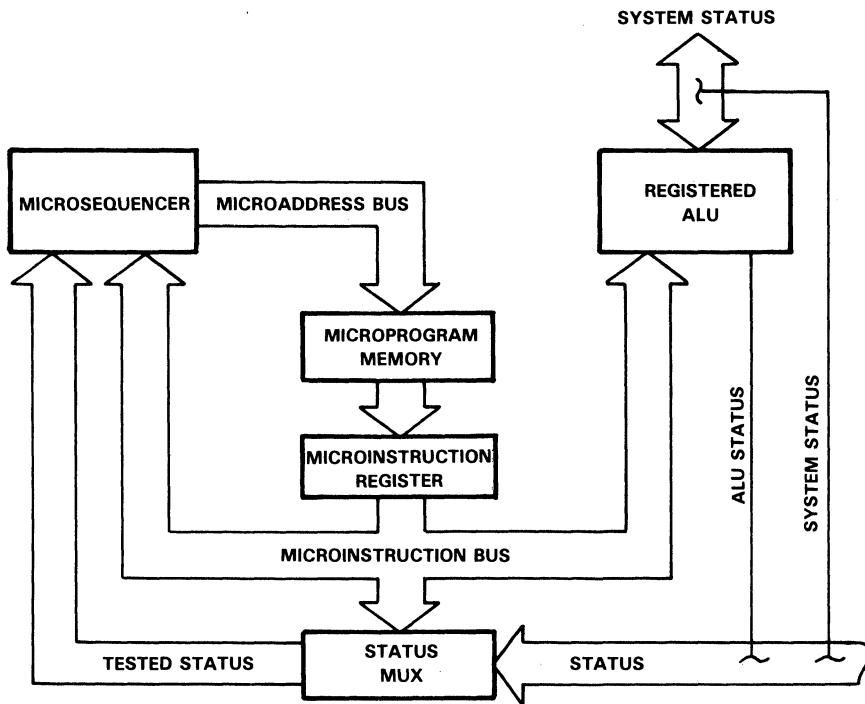


Figure 1. Microprogrammed System Block Diagram

The configuration of this processor enhances processing throughput in arithmetic and radix conversion. Internal generation and testing of status results in fast processing of division and multiplication algorithms. This decision logic is transparent to the user; the reduced overhead assures shorter microprograms, reduced hardware complexity, and shorter software development time.

Support Tools

Texas Instruments has designed a family of low-cost, real-time evaluation modules (EVM) to aid with initial hardware and microcode design. Each EVM is a small self-contained system which provides a convenient means to test and debug simple microcode, allowing software and hardware evaluation of components and their operation.

At present, the 74AS-EVM-8 Bit-Slice Evaluation Module has been completed, and 16- and 32-bit EVMs are in advanced stages of development. EVMs and support tools for other devices in the 'ACT8800 family are also planned for future development.

Design Support

TI's '8832 32-bit registered ALU is supported by a variety of tools developed to aid in design evaluation and verification. These tools will streamline all stages of the design process, from assessing the operation and performance of the '8832 to evaluating a total system application. The tools include a functional model, behavioral model, and microcode development software and hardware. Section 8 of this manual provides specific information on the design tools supporting TI's SN74ACT8800 Family.

Systems Expertise

Texas Instruments VLSI Logic applications group is available to help designers analyze TI's high-performance VLSI products, such as the '8832 32-bit registered ALU. The group works directly with designers to provide ready answers to device-related questions and also prepares a variety of applications documentation.

The group may be reached in Dallas, at (214) 997-3970.

'ACT8832 Pin Descriptions

Pin descriptions and grid allocations for the 'ACT8832 are given on the following pages.

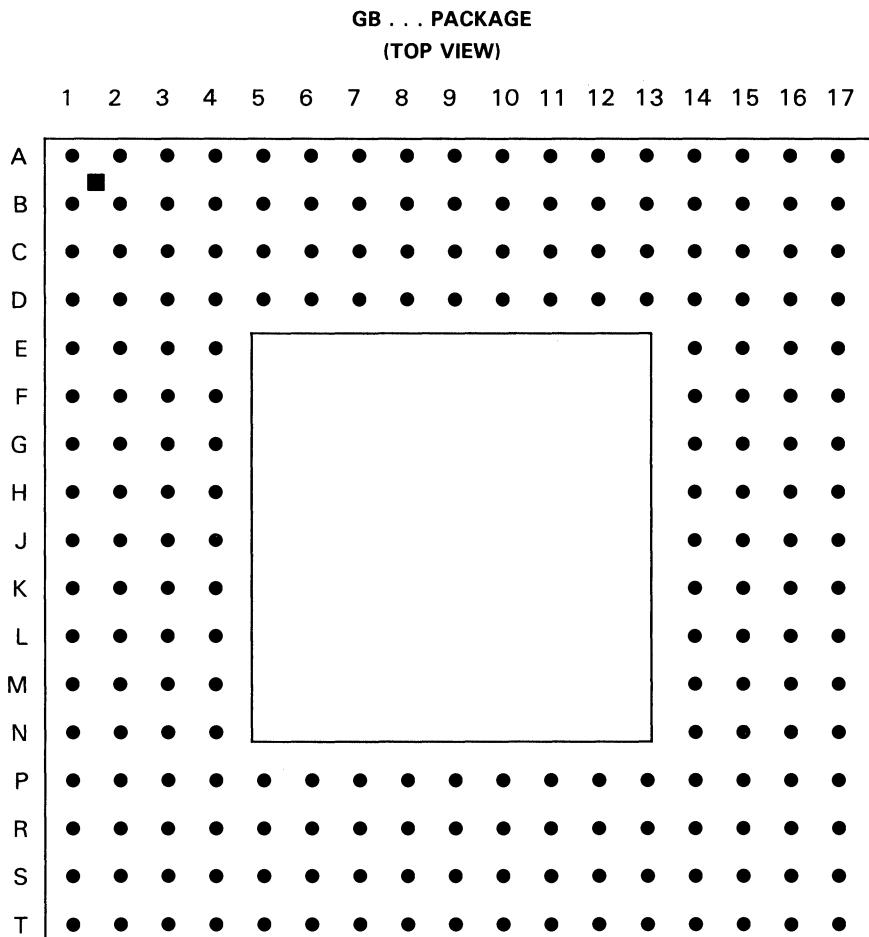


Figure 2. SN74ACT8832 . . . GB Package

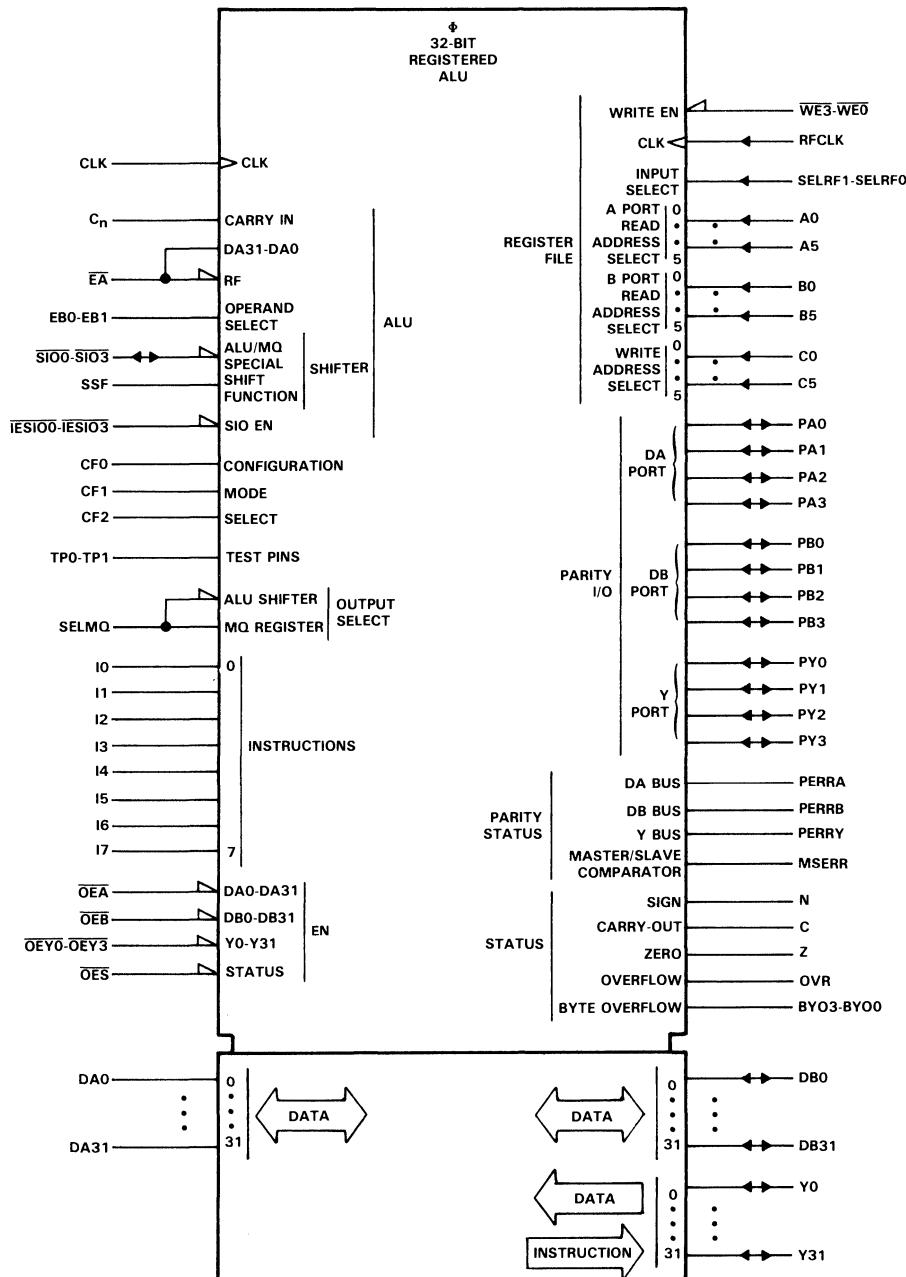


Figure 3. SN74ACT8832 . . . Logic Symbol

Table 1. SN74ACT8832 Pin Grid Allocation

PIN NO.		PIN NO.		PIN NO.		PIN NO.		PIN NO.		PIN NO.	
NAME	NAME	NAME	NAME	NAME	NAME	NAME	NAME	NAME	NAME	NAME	NAME
A1	Y7	C2	Y5	E3	Y0	J15	DA28	P1	DA5	S1	DB10
A2	Y13	C3	OEY0	E4	Y4	J16	DA27	P2	DB8	S2	DB15
A3	Y15	C4	Y9	E14	Y30	J17	DA29	P3	DB12	S3	DA10
A4	BYOF1	C5	Y11	E15	TP0	K1	DB6	P4	DA9	S4	DA13
A5	SIO3	C6	Y14	E16	I2	K2	DB7	P5	DA15	S5	PERRA
A6	SIO2	C7	OEY1	E17	I3	K3	DA0	P6	A5	S6	A3
A7	IESIO1	C8	GND	F1	EB1	K4	GND	P7	A1	S7	WE0
A8	IESIO0	C9	VCC	F2	Cn	K14	GND	P8	VCC	S8	WE3
A9	SIO0	C10	C	F3	CLK	K15	DA24	P9	GND	S9	RFCLK
A10	N	C11	PERRY	F4	CF2	K16	DA25	P10	C4	S10	B4
A11	OES	C12	Y17	F14	OEY3	K17	DA26	P11	PERRB	S11	B2
A12	SSF	C13	Y22	F15	I1	L1	PBO	P12	GND	S12	C3
A13	Y18	C14	OEY2	F16	I4	L2	DA2	P13	DB22	S13	CO
A14	Y20	C15	Y28	F17	I6	L3	VCC	P14	DA16	S14	DB17
A15	Y23	C16	PY3	G1	DB0	L4	GND	P15	DA18	S15	DB20
A16	Y24	C17	BYOF3	G2	EA	L14	GND	P16	DA22	S16	DB23
A17	Y25	D1	CF1	G3	EBO	L15	VCC	P17	DB27	S17	DA21
B1	Y6	D2	Y1	G4	GND	L16	DB30	R1	PA0	T1	DB14
B2	BYOFO	D3	Y3	G14	GND	L17	PB3	R2	DB11	T2	DA8
B3	Y10	D4	PY0	G15	I5	M1	DA1	R3	PB1	T3	DA12
B4	Y12	D5	Y8	G16	I7	M2	DA4	R4	DA11	T4	DA14
B5	PY1	D6	GND	G17	PA3	M3	DA7	R5	PA1	T5	OE4
B6	IESIO3	D7	GND	H1	DB2	M4	GND	R6	A4	T6	A2
B7	IESIO2	D8	GND	H2	DB1	M14	PA2	R7	A0	T7	WE1
B8	SIO1	D9	VCC	H3	VCC	M15	DB26	R8	WE2	T8	SELRF1
B9	Z	D10	GND	H4	GND	M16	DB28	R9	VCC	T9	SELRF0
B10	OVR	D11	GND	H14	GND	M17	DB31	R10	B1	T10	B5
B11	MSERR	D12	GND	H15	VCC	N1	DA3	R11	C2	T11	B3
B12	Y16	D13	BYOF2	H16	DA31	N2	DA6	R12	OEB	T12	BO
B13	Y19	D14	Y27	H17	DA30	N3	DB9	R13	DB18	T13	C5
B14	Y21	D15	Y31	J1	DB3	N4	DB13	R14	DB21	T14	C1
B15	PY2	D16	TP1	J2	DB4	N14	DA19	R15	PB2	T15	DB16
B16	Y26	D17	IO	J3	DB5	N15	DA23	R16	DA20	T16	DB19
B17	Y29	E1	SELMQ	J4	VCC	N16	DB25	R17	DB24	T17	DA17
C1	Y2	E2	CFO	J14	VCC	N17	DB29				

Table 2. SN74ACT8832 Pin Description

PIN NAME	NO.	I/O	DESCRIPTION
A0	R7		
A1	P7		
A2	T6		
A3	S6		
A4	R6		
A5	P6		
B0	T12		
B1	R10		
B2	S11		
B3	T11		
B4	S10		
B5	T10		
BYOF0	B2		
BYOF1	A4		
BYOF2	D13		
BYOF3	C17		
C	C10	O	Status signal representing carry out condition
C0	S13		
C1	T14		
C2	R11		
C3	S12		
C4	P10		
C5	T13		
CF0	E2		
CF1	D1		
CF2	F4		
Cn	F2	I	ALU carry input
CLK	F3	I	Clocks synchronous registers on positive edge
DA0	K3		
DA1	M1		
DA2	L2		
DA3	N1		
DA4	M2		
DA5	P1		
DA6	N2		
DA7	M3		
DA8	T2		
DA9	P4		
		I/O	A port data bus. Outputs register data ($\overline{OEA} = 0$) or inputs external data ($\overline{OEA} = 1$).

Table 2. SN74ACT8832 Pin Description (Continued)

PIN NAME	NO.	I/O	DESCRIPTION
DA10	S3		
DA11	R4		
DA12	T3		
DA13	S4		
DA14	T4		
DA15	P5		
DA16	P14		
DA17	T17		
DA18	P15		
DA19	N14		
DA20	R16	I/O	A port data bus. Outputs register data ($\overline{OEA} = 0$) or inputs external data ($\overline{OEA} = 1$).
DA21	S17		
DA22	P16		
DA23	N15		
DA24	K15		
DA25	K16		
DA26	K17		
DA27	J16		
DA28	J15		
DA29	J17		
DA30	H17		
DA31	H16		
DB0	G1		
DB1	H2		
DB2	H1		
DB3	J1		
DB4	J2		
DB5	J3		
DB6	K1		
DB7	K2		
DB8	P2	I/O	B port data bus. Outputs register data ($\overline{OEB} = 0$) or used to input external data ($\overline{OEB} = 1$)
DB9	N3		
DB10	S1		
DB11	R2		
DB12	P3		
DB13	N4		
DB14	T1		
DB15	S2		

Table 2. SN74ACT8832 Pin Description (Continued)

PIN NAME	NO.	I/O	DESCRIPTION
DB16	T15		
DB17	S14		
DB18	R13		
DB19	T16		
DB20	S15		
DB21	R14		
DB22	P13		
DB23	S16	I/O	B port data bus. Outputs register data ($\overline{OEB} = 0$) or used to input external data ($\overline{OEB} = 1$)
DB24	R17		
DB25	N16		
DB26	M15		
DB27	P17		
DB28	M16		
DB29	N17		
DB30	L16		
DB31	M17		
\overline{EA}	G2	I	ALU input operand select. High state selects external DA bus and low state selects register file
EBO	G3	I	ALU input operand select. Selects between register file, external DB port and MQ register
EB1	F1	I	
GND	C8		
GND	D6		
GND	D7		
GND	D8		
GND	D10		
GND	D11		
GND	D12		
GND	G4		
GND	G14		
GND	H4		
GND	H14		
GND	K4		
GND	K14		
GND	L4		
GND	L14		
GND	M4		
GND	P9		
GND	P12		Ground pins. All ground pins must be used.

Table 2. SN74ACT8832 Pin Description (Continued)

PIN NAME	PIN NO.	I/O	DESCRIPTION
I0	D17		
I1	F15		
I2	E16		
I3	E17		
I4	F16	I	Instruction input
I5	G15		
I6	F17		
I7	G16		
<u>I_{ESIO0}</u>	A8		
<u>I_{ESIO1}</u>	A7		
<u>I_{ESIO2}</u>	B7	I	Shift pin enables, increases system speed and reduces bus conflict, active low
<u>I_{ESIO3}</u>	B6		
MSERR	B11	O	Master Slave Error pin, indicates error between data at Y output MUX and external Y port
N	A10	O	Output status signal representing sign condition
<u>OEA</u>	T5	I	DA bus enable, active low
<u>OEB</u>	R12	I	DB bus enable, active low
<u>OES</u>	A11	I	Status enable, active low
<u>OEY0</u>	C3		
<u>OEY1</u>	C7		
<u>OEY2</u>	C14	I	Y bus output enable, active low
<u>OEY3</u>	F14		
OVR	B10	O	Output status signal represents overflow condition
PA0	R1		
PA1	R5		
PA2	M14	I/O	Parity bits port for DA data
PA3	G17		
PB0	L1		
PB1	R3		
PB2	R15	I/O	Parity bits port for DB data
PB3	L17		
PERRA	S5	O	DA data parity error, signals error if an even parity check fails for any byte
PERRB	P11	O	DB data parity error, signals error if an even parity check fails for any byte
PERRY	C11	O	Y data parity error, signals error if an even parity check fails for any byte

Table 2. SN74ACT8832 Pin Description (Continued)

PIN NAME	NO.	I/O	DESCRIPTION
PY0	D4		
PY1	B5		
PY2	B15		
PY3	C16		
RFCLK	S9	I	Register File Clock, allows multiple writes to be performed in one master clock cycle
SELMQ	E1	I	MQ register select, selects output of ALU shifter or MQ register to be placed on Y bus
SELRF0	T9	I	Register File select. Controls selection of the Register File(RF) inputs by the RF MUX
SELRF1	T8	I	
SIO0	A9		
SIO1	B8		
SIO2	A6		
SIO3	A5		
SSF	A12	I	Special Shift Function, implements conditional shift algorithms
TPO	E15		
TP1	D16	I	Test pins, supports system testing
V _{CC}	C9		
V _{CC}	D9		
V _{CC}	H3		
V _{CC}	H15		
V _{CC}	J4		
V _{CC}	J14		
V _{CC}	L3		
V _{CC}	L15		
V _{CC}	P8		
V _{CC}	R9		
WE0	S7		
WE1	T7	I	Register File WRITE ENABLE. Data is written into RF when write enables are low and a low to high Register File Clock (RFCLK) transition occurs.
WE2	R8		
WE3	S8		Active low.

Table 2. SN74ACT8832 Pin Description (Concluded)

PIN NAME	NO.	I/O	DESCRIPTION
Y0	E3		
Y1	D2		
Y2	C1		
Y3	D3		
Y4	E4		
Y5	C2		
Y6	B1		
Y7	A1		
Y8	D5		
Y9	C4		
Y10	B3		
Y11	C5		
Y12	B4		
Y13	A2		
Y14	C6		
Y15	A3	I/O	Y port data bus
Y16	B12		
Y17	C12		
Y18	A13		
Y19	B13		
Y20	A14		
Y21	B14		
Y22	C13		
Y23	A15		
Y24	A16		
Y25	A17		
Y26	B16		
Y27	D14		
Y28	C15		
Y29	B17		
Y30	E14		
Y31	D15		
Z	B9	O	Output status signal represents zero condition

'ACT8832 Specification Tables

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 ($V_O = 0$ to 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.

Table 3. Recommended Operating Conditions

PARAMETER	MIN	NOM	MAX	UNIT
V_{CC} Supply voltage	4.5	5.0	5.5	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

Table 4. Electrical Characteristics

PARAMETER	TEST CONDITIONS	V _{CC}	T _A = 25°C			UNIT
			MIN	TYP	MAX	
V _{OH}	I _{OH} = -20 μA	4.5 V	4.49	4.3		V
		5.5 V	5.49	5.3		
	I _{OH} = -8 mA	4.5 V		3.76		
		5.5 V		4.76		
V _{OL}	I _{OL} = 20 μA	4.5 V	0.01		0.10	V
		5.5 V	0.01		0.10	
	I _{OL} = 8 mA	4.5 V			0.45	
		5.5 V			0.45	
I _I	V _I = V _{CC} or 0	5.5 V			±1	μA
I _{CCQ}	V _I = V _{CC} or 0, I _O	5.5 V			200	μA
C _i	V _I = V _{CC} or 0	5 V			15	pF
ΔI _{CC} [†]	One input at 3.4 V, other inputs at 0 or V _{CC}	5.5 V			1	mA

Table 5. Register File Write Setup

	PARAMETER	MIN	MAX	UNIT
t _{su}	C5-C0	4		ns
	DA/B32-DA/B0, PA/B3-PA/B0	7		
	I7-I4	13		
	OEY3-OEY0	7		
	Y31-Y0	4		
	WE3-WEO	4		
	SELRF(DA,DB,PA,PB)	5		
	SELRF(Y)	9		
	SIO	10		
	SELMQ	9		
	IESIO3-IESIO0	10		
	t _h ALL	0		

[†]This is the increase in supply current for each input that is at one of the specified TTL voltage levels rather than 0 V to V_{CC}.

Table 6. Maximum Switching Characteristics

PARAMETER	FROM (INPUT)	TO (OUTPUT)										UNIT	
		Y	C	Z	SIO	PERRA/B	N	OVR	PA/B	DA/B	PY	PERRY	
t_{pd}	A5-A0,B5-B0	36	30	37	28		30	37	16	37			ns
	DA31-DAO,PA3-PA0	36	25	37	25	20	28	37		37			
	DB31-DB0,PB3-PB0												
	C_n	30	22	31	24		28	28		32			
	\overline{EA}	37	28	37	25		31	37		37			
	EB1-EBO	37	28	37	25		31	37		37			
	I7-IO	37	30	37	28		32	37		37			
	CF2-CFO	37	30	37	28		32	37		37			
	$\overline{OEB},\overline{OEA}$								15				
	$\overline{OEY3}-\overline{OEY0}$	20								20			
	SELMQ	15								20			
	SIO3-SIO0	15		25			25			27			
	CLK	21								28			
	CLKMQ	37								37			
	RCLK	37	32	37	24		32	37		37			
	$\overline{IESIO3}-\overline{IESIO0}$	15		25			25			27			
	SSF	25		30	22		30	22		30			
	Y									15	15		

3

SN74ACT8832

'ACT8832 Registered ALU

The SN74ACT8832 is a 32-bit registered ALU that can be configured to operate as four 8-bit ALUs, two 16-bit ALUs, or a single 32-bit ALU. The processor instruction set is 100 percent upwardly compatible with the 'AS888 and includes 13 arithmetic and logical functions with 8 conditional shifts, multiplication, division, normalization, add and subtract immediate, bit and byte operations, and data conversions such as BCD, excess-3, and sign magnitude. New instructions permit internal flip-flops controlling BCD and divide operations to be loaded or read.

Additional functions added to the 'ACT8832 include byte parity and master/slave operation. Parity is checked at the three data input ports and generated at the Y output port. The 64-word register file is 36 bits wide to permit storage of the parity bits. Master/slave comparator circuitry is provided at the Y port.

The DA and DB ports can simultaneously input data to the ALU and the 64-word by 36-bit register file. Data and parity from the register file can be output on the DA and DB ports. Results of ALU and shift operations are output at the bidirectional Y port. The Y port can also be used in an input mode to furnish external data to the register file or during master/slave operation as an input to the master/slave comparator.

Three 6-bit address ports allow a two-operand fetch and an operand write to be performed at the register file simultaneously. An MQ shifter and MQ register can also be configured to function independently to implement double-precision 8-bit, 16-bit, and 32-bit shift operations. An internal ALU bypass path increases the speeds of multiply, divide and normalize instructions. The path is also used by 'ACT8832 instructions that permit bits and bytes to be manipulated.

Architecture

Figure 4 is a functional block diagram of the 'ACT8832. Control input signals are summarized in Table 7. Data flow and details of the functional elements are presented in the following paragraphs.

Table 7. 'ACT8832 Response to Control Inputs

SIGNAL	HIGH	LOW
CF2-CFO	See Table 11	See Table 11
EA	Selects external DA bus	Selects register file
EB1-EBO	See Table 9	See Table 9
IESIO3-IESIO0	Normal operation	Force corresponding SIO inputs to high impedance
I7-I0	See Table 15	See Table 15
MQSEL	Selects MQ register	Selects ALU
OEA	Inhibits DA and PA output	Enables DA and PA output
OEB	Inhibits DB and PB output	Enables DB and PB output
OEY3-OEYO	Inhibits Y and PY outputs	Enables Y and PY outputs
SELRF1-SELRF0	See Table 8	See Table 8
SSF	Selects shifted ALU output	Selects ALU (unshifted) output
TP1-TPO	See Table 14	See Table 14
WE3-WEO	Inhibits register file write	Byte enables for register file write (0 = LSB)

Data Flow

As shown in Figure 5, data enters the 'ACT8832 from three primary sources: the bidirectional Y port, which is used in an input mode to pass data to the register file; and the bidirectional DA and DB ports, used to input data to the register file or the R and S buses serving the ALU. Three associated I/O ports (PY, PA, and PB) are provided for associated parity data input and output.

Data is input to the ALU through two multiplexers: R MUX, which selects the R bus operand from the DA port or the register file addressed by A5-A0; and S MUX, which selects data from the DB port, the register file addressed by B5-B0, or the multiplier-quotient (MQ) register.

The result of the ALU operation is passed to the ALU shifter, where it is shifted or passed without shift to the Y bus for possible output from the 'ACT8832 and to the feedback MUX for possible storage in the internal register file. The MQ shifter, which operates in parallel with the ALU shifter, can be loaded from the ALU or the MQ register. The MQ shift result is passed to the MQ register, where it can be routed through the S MUX to the ALU or to the Y MUX for output from the chip.

An internal bypass path allows data from the S MUX to be loaded directly into the ALU shifter or the divide/BCD flip-flops. Data from the divide/BCD flip-flops can be output via the MQ register.

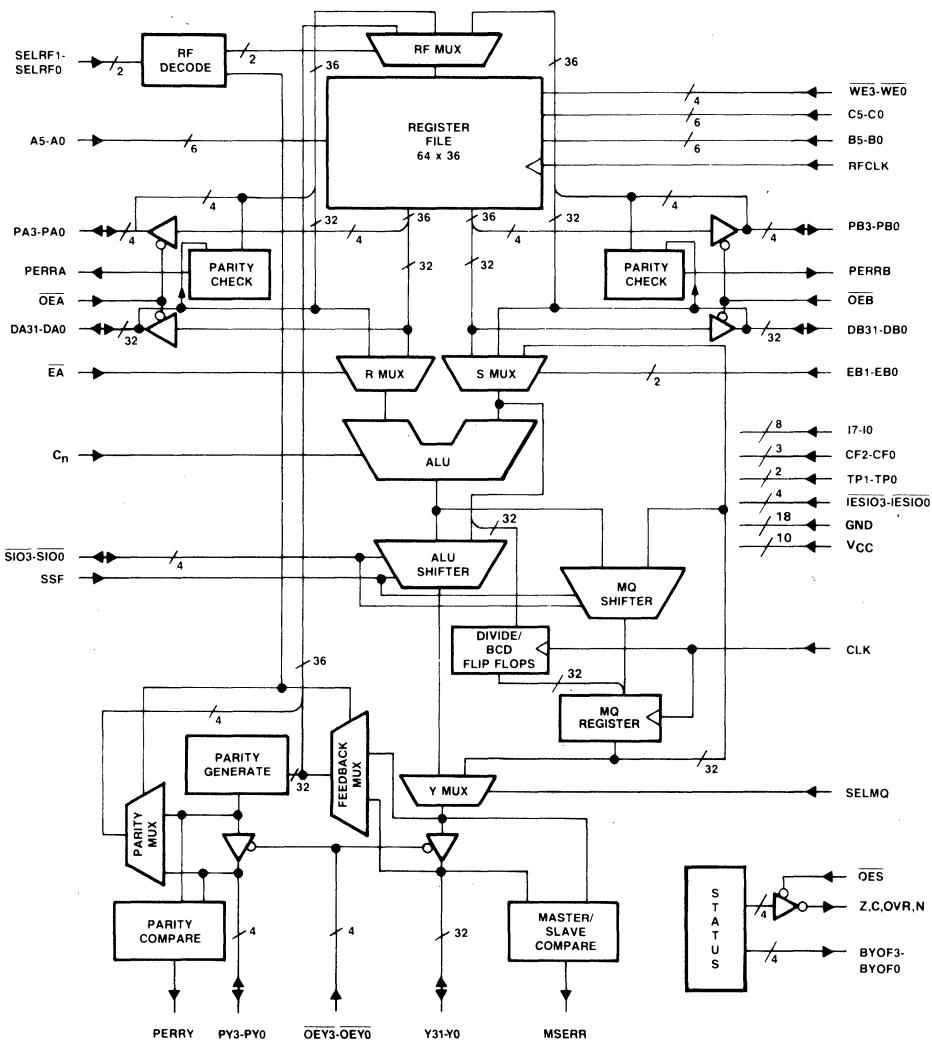


Figure 4. 'ACT8832 32-Bit Registered ALU

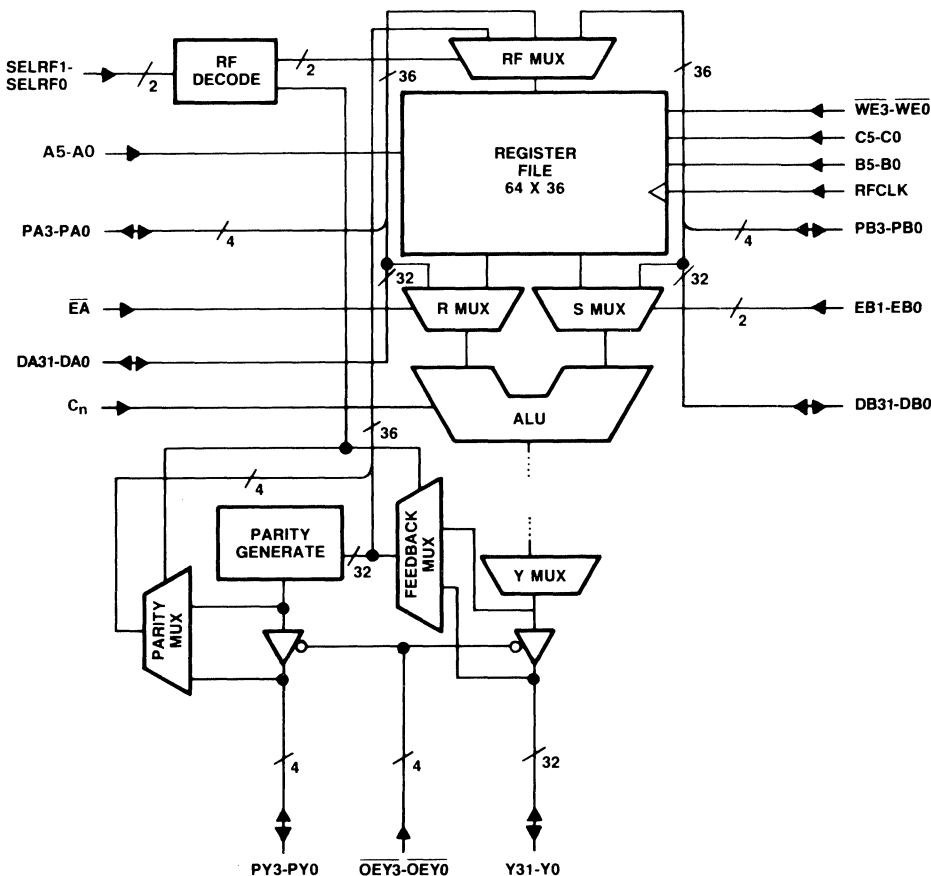


Figure 5. Data I/O

Data can be output from the three bidirectional ports, Y, DA, and DB, and their associated parity ports, PY, PA, and PB. DA and DB can also be used to read ALU input data on the R and S buses for debug or other special purposes.

Architectural Elements

Three-Port Register File

The register file is 36 bits wide, permitting storage of a 32-bit data word with its associated parity bits. The 64 registers are accessed by three address ports. C5-C0 address the destination register during write operations; A5-A0 and B5-B0 address any two registers during read operations. The address buses are also used to furnish

immediate data to the ALU: A3-A0 to provide constant data for the add and subtract immediate instructions; C3-C0 and A3-A0 to provide masks for set, reset, and test bit operations.

Data is written into the register file when the write enable is low and a low-to-high register file clock (RFCLK) transition occurs. The separate register file clock allows multiple writes to be performed in one master clock cycle, allowing processors in multi-processor environments to update one another's internal register files during a single cycle.

Four write enable inputs are provided to allow separate control of data inputs in a byte-oriented system. $\overline{WE3}$ is the write enable for the most significant byte.

Register file inputs are selected by the RF MUX under the control of two register file select signals, SELRF1 and SELRFO, shown in Table 8 (see also Table 10).

Table 8. RF MUX Select Inputs

SELRF1	SELRFO	SOURCE
0	0	External DA input
0	1	External DB input
1	0	Y-output MUX
1	1	External Y port

R and S Multiplexers

ALU inputs are selected by the R and S multiplexers. Controls which affect operand selection for instructions other than those using constants or masks are shown in Table 9.

Table 9. ALU Source Operand Selects

R-BUS OPERAND SELECT \overline{EA}	S-BUS OPERAND SELECT EB1-EBO	RESULT DESTINATION	←SOURCE OPERAND
0		R bus	←Register file addressed by A5-A0
1		R bus	←DA port
	0 0	S bus	←Register file addressed by B5-B0
	1 0	S bus	←DB port
	X 1	S bus	←MQ register

Table 10. Destination Operand Select/Enables

REGISTER FILE WRITE ENABLE <u>WE</u>	Y BUS OUTPUT ENABLE <u>OEY</u>	Y MUS SELECT MQSEL	REGISTER FILE SELECT RFSEL1-RFSEL0	DA PORT OUTPUT <u>OEA</u>	DB PORT OUTPUT <u>OEB</u>	RESULT DESTINATION	← SOURCE
1	0	0	X	X		Y/PY	← ALU shifter/parity generate
1	0	1	X	X		Y/PY	← MQ register/parity generate
0	0	0	1	0		Y/PY, RF	← ALU shifter/parity generate
0	0	1	1	0		Y/PY, RF	← MQ register/parity generate
0	1	X	1	1		RF	← External Y/PY
0	X	X	0	0	1	RF	← External DA/PA
0	X	X	0	1	X	RF	← External DB/PB
				0		DA/PA	← R bus register file output
				1		DA/PA	Hi-Z
					0	DB/PB	← S bus register file output
					1	DB/PB	Hi-Z

Data Input and Output Ports

The DA and DB ports can be used to load the S and/or R multiplexers from an external source or to read S or R bus outputs from the register file. The Y port can be used to load the register file and to output the next address selected by the Y output multiplexer. Tables 9 and 10 describe the MUX and output controls which affect DA, DB, and Y.

ALU

The ALU can perform seven arithmetic and six logical instructions on the two 32-bit operands selected by the R and S multiplexers. It also supports multiplication, division, normalization, bit and byte operations and data conversion, including excess-3 BCD arithmetic. The 'ACT8832 instruction set is summarized in Table 15.

The 'ACT8832 can be configured to operate as a single 32-bit ALU, two 16-bit ALUs, or four 8-bit ALUs (see Figures 6 and 7). It can also be configured to operate on a 32-bit word formed by adding leading zeros to the 12 least significant bits of R bus data. This is useful in certain IBM relative addressing schemes.

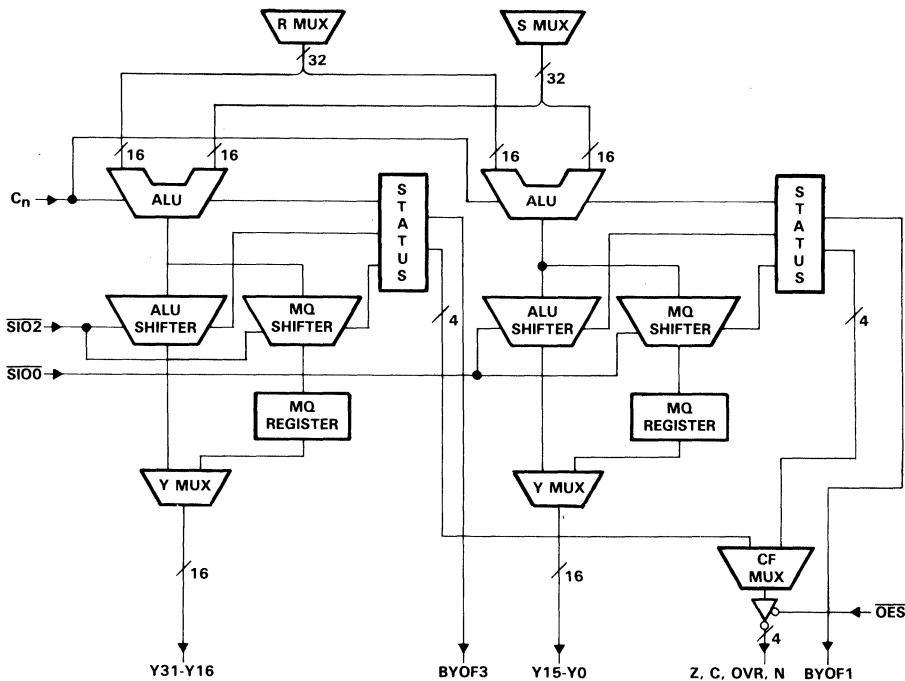


Figure 6. 16-Bit Configuration

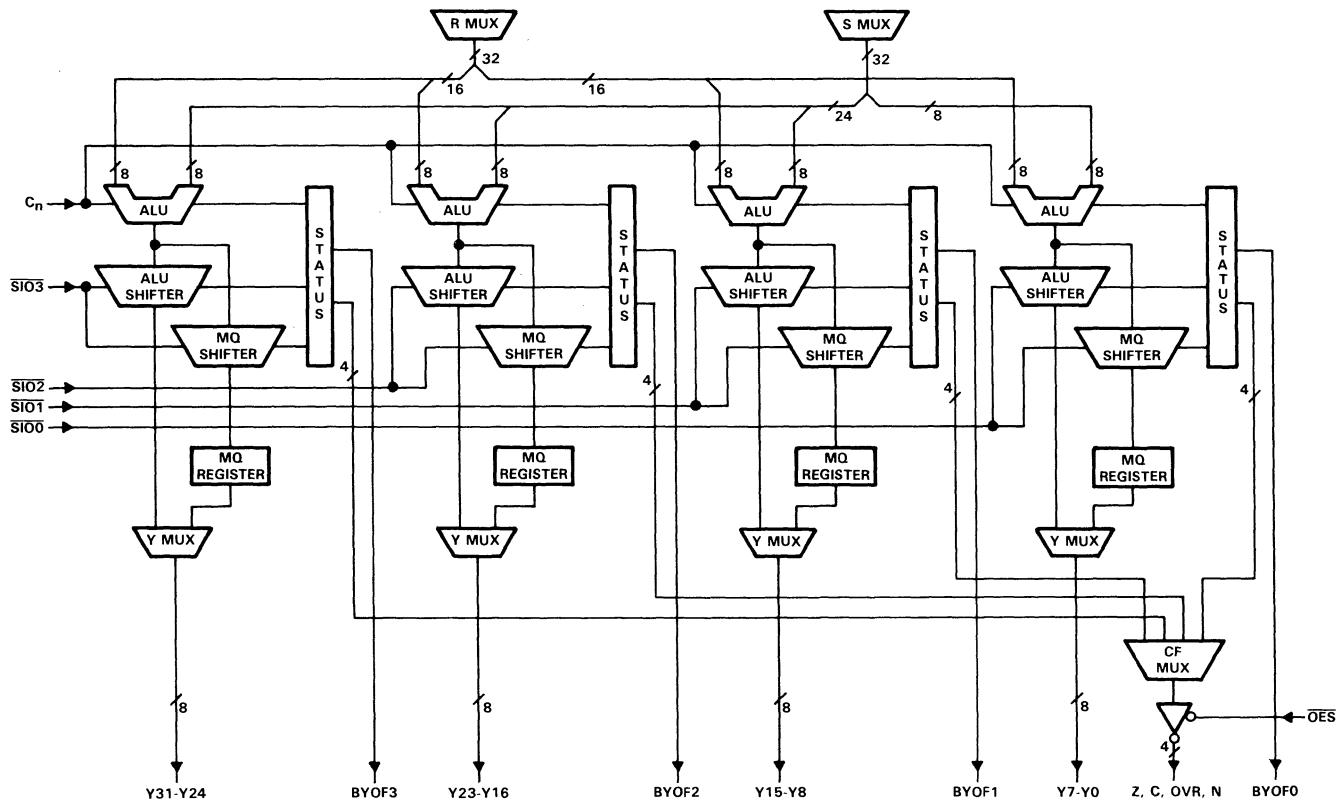


Figure 7. 8-Bit Configuration

Configuration modes are controlled by three CF inputs as shown in Table 11. These signals also select the data from which status signals other than byte overflow will be generated.

Table 11. Configuration Mode Selects

CONTROL INPUTS			MODE SELECTED	DATA FROM WHICH STATUS OTHER THAN BYOF WILL BE GENERATED
CF2	CF1	CF0		
0	0	0	Four 8-bit	Byte 0
0	0	1	Four 8-bit	Byte 1
0	1	0	Four 8-bit	Byte 2
0	1	1	Four 8-bit	Byte 3
1	0	0	Two 16-bit	Least significant 16-bit word
1	0	1	Two 16-bit	Most significant 16-bit word
1	1	0	One 32-bit	32-bit word
1	1	1	Masked 32-bit	32-bit word

ALU and MQ Shifters

The ALU and MQ shifters are used in all of the shift, multiply, divide and normalize functions. They can be used independently for single precision or concurrently for double precision shifts. Shifts can be made conditional, using the Special Shift Function (SSF) pin.

Bidirectional Serial I/O Pins

Four bidirectional \overline{SIO} pins are provided to supply an end fill bit for certain shift instructions. These pins may also be used to read bits that are shifted out of the ALU or MQ shifters during certain instructions. Use of the \overline{SIO} pins as inputs or outputs is summarized in Table 17.

The four pins allow separate control of end fill inputs in configurations other than 32-bit mode (see Table 12 and Figure 4).

Table 12. Data Determining \overline{SIO} Input

SIGNAL	CORRESPONDING WORD, PARTIAL WORD OR BYTE		
	32-BIT MODE	16-BIT MODE	8-BIT MODE
$\overline{SIO3}$	—	—	Byte 3
$\overline{SIO2}$	—	most significant word	Byte 2
$\overline{SIO1}$	—	—	Byte 1
$\overline{SIO0}$	32-bit word	least significant word	Byte 0

To increase system speed and reduce bus conflict, four SIO input enables (IESI03-IESI00) are provided. A low on these enables will override internal pull-up resistor logic and force the corresponding SIO pins to the high impedance state required before an input signal can appear on the signal line. If the SIO enables are not used, this condition is generated internally in the chip. Use of the enables allow internal decoding to be bypassed, resulting in faster speeds.

The IESIs are defaulted to a high because of internal pull-up resistors. When an SIO pin is used as an output, a low on its corresponding IESI pin would force SIO to a high impedance state. The output would then be lost, but the internal operation of the chip would not be affected.

MQ Register

Data from the MQ shifter is written into the MQ register when a low-to-high transition occurs on clock CLK. The register has specific functions in double precision shifts, multiplication, division and data conversion algorithms and can also be used as a temporary storage register. Data from the register file and the DA and DB buses can be passed to the MQ register through the ALU.

The Y bus contains the output of the ALU shifter if SELMQ is low and the output of the MQ register if SELMQ is high. If OEY is low, ALU or MQ shifter output will be passed to the Y port; if OEY is high, the Y port becomes an input to the feedback MUX.

Conditional Shift Pin

Conditional shifting algorithms may be implemented using the SSF pin under hardware or firmware control. If the SSF pin is high or floating, the shifted ALU output will be sent to the output buffers. If the SSF pin is pulled low externally, the ALU result will be passed directly to the output buffers, and MQ shifts will be inhibited. Conditional shifting is useful for scaling inputs in data arrays or in signal processing algorithms.

Master/Slave Comparator

A master/slave comparator is provided to compare data bytes from the Y output MUX with data bytes on the external Y port when OEY is high. If the data are not equal, a high signal is generated on the master slave error output pin (MSERR). A similar comparator is provided for the Y parity bits.

Divide/BCD Flip-Flops

Internal multiply/divide flip-flops are used by certain multiply and divide instructions to maintain status between instructions. Internal excess-3 BCD flip-flops preserve the carry from each nibble in excess-3 BCD operations. The BCD flip-flops are affected by all instructions except NOP and are cleared when a CLR instruction is executed. The flip-flops can be loaded and read externally using instructions LOADFF and DUMPFF

(see Table 15). This feature permits an iterative arithmetic operation such as multiplication or division to be interrupted immediately so that an external interrupt can be processed.

Status

Eight status output signals are generated by the 'ACT8832. Four signals (BYOF3-BYOF0) indicate overflow conditions in certain data bytes (see Table 13). The others represent sign (N), zero (ZERO), carry-out (Cout) and overflow (OVR). N, ZERO, Cout, and OVR are generated from data selected by the mode configuration controls (CF2-CF0) as shown in Table 11.

3

Carry-out is evaluated after each ALU operation. Sign and zero status are evaluated after ALU shift operation. Overflow (OVR) is determined by ORing the overflow result from the ALU with the overflow result from the ALU shifter.

SN74ACT8832

Table 13. Data Determining BYOF Outputs

SIGNAL	CORRESPONDING WORD, PARTIAL WORD OR BYTE		
	32-BIT MODE	16-BIT MODE	8-BIT MODE
BYOF3	32-bit word	most significant word	Byte 3
BYOF2	—	—	Byte 2
BYOF1	—	least significant word	Byte 1
BYOF0	—	—	Byte 0

Input Data Parity Check

An even parity check is performed on each byte of input data at the DA, DB and Y ports. The check is performed by counting the number of ones in each byte and its corresponding parity bit. Parity bits are input on PA for DA data, PB for DB data and PYF or Y data. PA0, PB0 and PY0 are the parity bits for the least significant bytes of DA, DB and Y, respectively. If the result of the parity count is odd for any byte, a high appears at the parity error output pin (PERRA for DA data, PERRB for DB data, PERRY for Y data).

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 14).

Table 14. Test Pin Inputs

TP1	TP0	RESULT
0	0	All outputs and I/Os forced low
0	1	All outputs and I/Os forced high
1	0	All outputs and I/Os placed in a high impedance state
1	1	Normal operation (default state)

Instruction Set Overview

Bits I7-I0 are used as instruction inputs to the 'ACT8832. Table 15 lists all instructions, divided into five groups, with their opcodes and mnemonics.

Table 15. 'ACT8832 Instruction Set

GROUP 1 INSTRUCTIONS		
INSTRUCTION BITS (HEX)	MNEMONIC	FUNCTION
0		Used to access Group 4 instructions
1	ADD	R + S + Cn
2	SUBR	$\bar{R} + S + Cn$
3	SUBS	$R + \bar{S} + Cn$
4	INCS	S + Cn
5	INCNS	$\bar{S} + Cn$
6	INCR	R + Cn
7	INCNR	$\bar{R} + Cn$
8		Used to access Group 3 instructions
9	XOR	R XOR S
A	AND	R AND S
B	OR	R OR S
C	NAND	R NAND S
D	NOR	R NOR S
E	ANDNR	\bar{R} AND S
F		Used to access Group 5 instructions

Table 15. 'ACT8832 Instruction Set (Continued)

GROUP 2 INSTRUCTIONS		
INSTRUCTION BITS I7-I4 (HEX)	MNEMONIC	FUNCTION
0	SRA	Arithmetic right single precision shift
1	SRAD	Arithmetic right double precision shift
2	SRL	Logical right single precision shift
3	SRD	Logical right double precision shift
4	SLA	Arithmetic left single precision shift
5	SLAD	Arithmetic left double precision shift
6	SLC	Circular left single precision shift
7	SLCD	Circular left double precision shift
8	SRC	Circular right single precision shift
9	SRCD	Circular right double precision shift
A	MQSRA	Arithmetic right shift MQ register
B	MQSRL	Logical right shift MQ register
C	MQSLL	Logical left shift MQ register
D	MQSLC	Circular left shift MQ register
E	LOADMQ	Load MQ register
F	PASS	Pass ALU to Y

Table 15. 'ACT8832 Instruction Set (Continued)

GROUP 3 INSTRUCTIONS		
INSTRUCTION BITS I7-I0 (HEX)	MNEMONIC	FUNCTION
08	SET1	Set bit 1
18	SETO	Set bit 0
28	TB1	Test bit (one)
38	TBO	Test bit (zero)
48	ABS	Absolute value
58	SMTC	Sign magnitude/two's complement
68	ADDI	Add immediate
78	SUBI	Subtract immediate
88	BADD	Byte add R to S
98	BSUBS	Byte subtract S from R
A8	BSUBR	Byte subtract R from S
B8	BINCS	Byte increment S
C8	BINCNS	Byte increment negative S
D8	BXOR	Byte XOR R and S
E8	BAND	Byte AND R and S
F8	BOR	Byte OR R and S

3

SN74ACT8832

Table 15. 'ACT8832 Instruction Set (Continued)

GROUP 4 INSTRUCTIONS		
INSTRUCTION BITS I7-I0 (HEX)	MNEMONIC	FUNCTION
00	CRC	Cyclic redundancy character accumulation
10	SEL	Select S or R
20	SNORM	Single length normalize
30	DNORM	Double length normalize
40	DIVRF	Divide remainder fix
50	SDIVQF	Signed divide quotient fix
60	SMULI	Signed multiply iterate
70	SMULT	Signed multiply terminate
80	SDIVIN	Signed divide initialize
90	SDIVIS	Signed divide start
A0	SDIVI	Signed divide iterate
B0	UDIVIS	Unsigned divide start
C0	UDIVI	Unsigned divide iterate
D0	UMULI	Unsigned multiply iterate
E0	SDIVIT	Signed divide terminate
F0	UDIVIT	Unsigned divide terminate

Table 15. 'ACT8832 Instruction Set (Continued)

GROUP 5 INSTRUCTIONS		
INSTRUCTION BITS I7-I0 (HEX)	MNEMONIC	FUNCTION
OF	LOADFF	Load divide/BCD flip-flops
1F	CLR	Clear
2F	CLR	Clear
3F	CLR	Clear
4F	CLR	Clear
5F	DUMPFF	Output divide/BCD flip-flops
6F	CLR	Clear
7F	BCDBIN	BCD to binary
8F	EX3BC	Excess-3 byte correction
9F	EX3C	Excess-3 word correction
AF	SDIVO	Signed divide overflow test
BF	CLR	Clear
CF	CLR	Clear
DF	BINEX3	Binary to excess-3
EF	CLR	Clear
FF	NOP	No operation

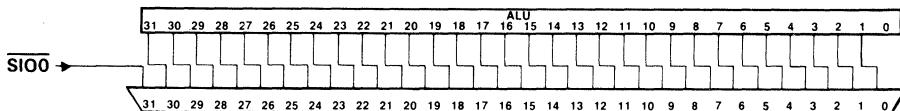
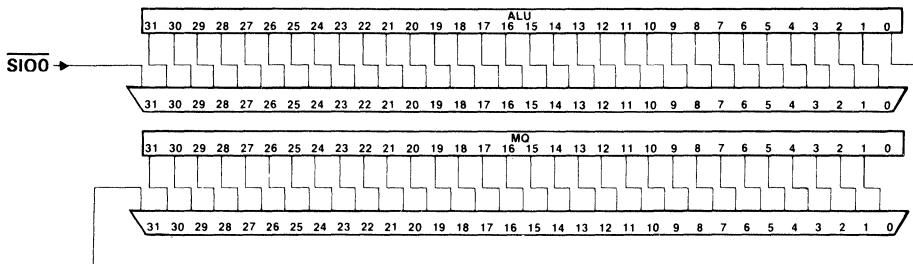
Group 1, a set of ALU arithmetic and logic operations, can be combined with the user-selected shift operations in Group 2 in one instruction cycle. The other groups contain instructions for bit and byte operations, division and multiplication, data conversion, and other functions such as sorting, normalization and polynomial code accumulation.

Arithmetic/Logic Instructions with Shifts

The seven Group 1 arithmetic instructions operate on data from the R and/or S multiplexers and the carry-in. Carry-out is evaluated after ALU operation; other status pins are evaluated after the accompanying shift operation, when applicable. Group 1 logic instructions do not use carry-in; carry-out is forced to zero.

Possible shift instructions are listed in Group 2. Fourteen single and double precision shifts can be specified, or the ALU result can be passed unshifted to the MQ register or to the specified output destination by using the LOADMQ or PASS instructions. Table 16 lists shift definitions.

When using the shift registers for double precision operations, the least significant half should be placed in the MQ register and the most significant half in the ALU for passage to the ALU shifter. An example of a double-precision shift using the ALU and MQ shifters is given in Figure 8.

**SERIAL DATA
INPUT SIGNALS**

Single Precision Logical Right Single Shift, 32-Bit Configuration
**SERIAL DATA
INPUT SIGNALS**

Double Precision Logical Right Single Shift, 32-Bit Configuration
Figure 8. Shift Examples, 32-Bit Configuration

All Group 2 shifts can be made conditional using the conditional shift pin (SSF). If the SSF pin is high or floating, the shifted ALU output will be sent to the output buffers, MQ register, or both. If the SSF pin is pulled low, the ALU result will be passed directly to the output buffers and any MQ shifts will be inhibited.

Table 16. Shift Definitions

SHIFT TYPE	NOTES
Left	Moves a bit one position towards the most significant bit
Right	Moves a bit one position towards the least significant bit
Arithmetic right	Retains the sign unless an overflow occurs, in which case, the sign would be inverted
Arithmetic left	May lose the sign bit if an overflow occurs. Zero is filled into the least significant bit unless the bit is set externally
Circular right	Fills the least significant bit in the most significant bit position
Circular left	Fills the most significant bit in the least significant bit position
Logical right	Fills a zero in the most significant bit position unless the bit is forced to one by placing a zero on an \overline{SIO} pin
Logical left	Fills a zero in the least significant bit position unless the bit is forced to one by placing a zero on an \overline{SIO} pin

The bidirectional $\overline{\text{SIO}}$ pins can be used to supply external end fill bits for certain Group 2 shift instructions. When $\overline{\text{SIO}}$ is high or floating, a zero is filled, otherwise a 1 is filled. Table 17 lists instructions that make use of the $\overline{\text{SIO}}$ inputs and identifies input and output functions.

Table 17. Bidirectional SIO Pin Functions

INSTRUCTION BITS I7-I0 (HEX)	SIO		
	MNEMONIC	I/O	DATA
0*	SRA	O	Shift out
1*	SRAD	O	Shift out
2*	SRL	I	Most significant bit
3*	SRLD	I	Most significant bit
4*	SLA	I	Least significant bit
5*	SLAD	I	Least significant bit
6*	SLC	O	Shifted input to MQ shifter
7*	SLCD	O	Shifted input to MQ shifter
8*	SRC	O	Shifted input to ALU shifter
9*	SRCD	O	Shifted input to ALU shifter
A*	MQSRA	O	Shift out
B*	MQSRL	I	Most significant bit
C*	MQSLL	I	Least significant bit
D*	MQSLC	O	Shifted input to MQ shifter
00	CRC	O	Internally generated end fill bit
20	SNORM	I	Least significant bit
30	DNORM	I	Least significant bit
60	SMULI	O	ALUO
70	SMULT	O	ALUO
80	SDIVIN	O	Internally generated end fill bit
90	SDIVIS	O	Internally generated end fill bit
A0	SDIVI	O	Internally generated end fill bit
B0	UDIVIS	O	Internally generated end fill bit
C0	UDIVI	O	Internally generated end fill bit
D0	UMULI	O	Internal input
E0	SDIVT	O	Internally generated end fill bit
F0	UDIVIT	O	Internally generated end fill bit
7F	BCDBIN	I	Least significant bit
DF	BINEX3	O	Shifted input to MQ register

Other Arithmetic Instructions

The 'ACT8832 supports two immediate arithmetic operations. ADDI and SUBI (Group 3) add or subtract a constant between the values of 0 and 15 from an operand on the S bus. The constant value is specified in bits A3-A0.

Twelve Group 4 instructions support serial division and multiplication. Signed, unsigned and mixed multiplication are implemented using three instructions: SMULI, which performs a signed times unsigned iteration; SMULT, which provides negative weighting of the sign bit of a negative multiplier in signed multiplication; and UMULI, which performs an unsigned multiplication iteration. Algorithms using these instructions are given in Tables 18, 19, and 20. These include: signed multiplication, which performs a two's complement multiplication; unsigned multiplication, which produces an unsigned times unsigned product; and mixed multiplication which multiplies a signed multiplicand by an unsigned multiplier to produce a signed result.

Table 18. Signed Multiplication Algorithm

OP CODE	MNEMONIC	CLOCK CYCLES	INPUT S PORT	INPUT R PORT	OUTPUT Y PORT
E4	LOADMQ	1	Multiplier	—	Multiplier
60	SMULI	N-1 [†]	Accumulator	Multiplicand	Partial product
70	SMULT	1	Accumulator	Multiplicand	Product (MSH) [‡]

Table 19. Unsigned Multiplication Algorithm

OP CODE	MNEMONIC	CLOCK CYCLES	INPUT S PORT	INPUT R PORT	OUTPUT Y PORT
E4	LOADMQ	1	Multiplier	—	Multiplier
D0	UMULI	N-1 [†]	Accumulator	Multiplicand	Partial product
D0	UMULI	1	Accumulator	Multiplicand	Product (MSH) [‡]

Table 20. Mixed Multiplication Algorithm

OP CODE	MNEMONIC	CLOCK CYCLES	INPUT S PORT	INPUT R PORT	OUTPUT Y PORT
E4	LOADMQ	1	Multiplier	—	Multiplier
60	SMULI	N-1 [†]	Accumulator	Multiplicand	Partial product
60	SMULI	1	Accumulator	Multiplicand	Product (MSH) [‡]

[†]N = 8 for quad 8-bit mode, 16 for dual 16-bit mode, 32 for 32-bit mode.

[‡]The least significant half of the product is in the MQ register.

Instructions that support division include start, iterate and terminate instructions for unsigned division routines (UDIVIS, UDIVI and UDIVIT); initialize, start, iterate and terminate instructions for signed division routines (SDIVIN, SDIVIS, SDIVI and SDIVIT); and correction instructions for these routines (DIVRF and SDIVQF). A Group 5 instruction, SDIVO, is available for optional overflow testing. Algorithms for signed and unsigned division are given in Tables 21 and 22. These use a nonrestoring technique to divide a 16 N-bit integer dividend by an 8 N-bit integer divisor to produce an 8 N-bit integer quotient and remainder, where N = 1 for quad 8-bit mode, N = 2 for dual 16-bit mode, and N = 4 for 32-bit mode.

Table 21. Signed Division Algorithm

OP CODE	MNEMONIC	CLOCK CYCLES	INPUT S PORT	INPUT R PORT	OUTPUT Y PORT
E4	LOADMQ	1	Dividend (LSH)	—	Dividend (LSH)
80	SDIVIN	1	Dividend (MSH)	Divisor	Remainder (N)
AF	SDIVO	1	Remainder (N)	Divisor	Overflow Test Result
90	SDIVIS	1	Remainder (N)	Divisor	Remainder (N)
A0	SDIVI	N-2 [†]	Remainder (N)	Divisor	Remainder (N)
E0	SDIVIT	1	Remainder (N)	Divisor	Remainder [§]
40	DIVRF	1	Remainder [‡]	Divisor	Remainder [¶]
50	SDIVQF	1	MQ register	Divisor	Quotient [#]

[†]N = 8 for quad 8-bit mode, 16 for dual 16-bit mode, 32 for 32-bit mode.

[‡]The least significant half of the product is in the MQ register.

[§]Unfixed

[¶]Fixed (corrected)

[#]The quotient is stored in the MQ register. Remainder can be output at the Y port or stored in the register file accumulator.

Table 22. Unsigned Division Algorithm

OP CODE	MNEMONIC	CLOCK CYCLES	INPUT S PORT	INPUT R PORT	OUTPUT Y PORT
E4	LOADMQ	1	Dividend (LSH)	—	Dividend (LSH)
B0	UDIVIS	1	Dividend (MSH)	Divisor	Remainder (N)
C0	UDIVI	N-1 [†]	Remainder (N)	Divisor	Remainder (N)
F0	UDIVIT	1	Remainder (N)	Divisor	Remainder [‡]
40	DIVRF	1	Remainder [§]	Divisor	Remainder [§]

[†]N = 8 in quad 8-bit mode, 16 in dual 16-bit mode, 32 in 32-bit mode

[‡]Unfixed

[§]Fixed (corrected)

Data Conversion Instructions

Conversion of binary data to one's and two's complement can be implemented using the INCNR instruction (Group 1). SMTc (Group 3) permits conversion from two's complement representation to sign magnitude representation, or vice versa. Two's complement numbers can be converted to their positive value, using ABS (Group 3).

SNORM and DNORM (Group 4) provide for normalization of signed, single- and double-precision data. The operand is placed in the MQ register and shifted toward the most significant bit until the two most significant bits are of opposite value. Zeroes are shifted into the least significant bit, provided SIO is high or floating. (A low on SIO will shift a one into the least significant bit.) SNORM allows the number of shifts to be counted and stored in one of the register files to provide the exponent.

Data stored in binary-coded decimal form can be converted to binary using BCDBIN (Group 5). A routine for this conversion, given in Table 23, allows the user to convert an N-digit BCD number to a 4N-bit binary number in $4N + 8$ clock cycles.

Table 23. BCD to Binary Algorithm

OP CODE	MNEMONIC	CLOCK CYCLES	INPUT S PORT	INPUT R PORT	OUTPUT DESTINATION
E4	LOADMQ	1	BCD operand	—	MQ reg.
D2	SUBR/MQSLC	1	Accumulator	Accumulator	Accumulator/MQ reg.
D2	SUBR/MQSLC	1	Mask reg.	Mask reg.	Mask reg/MQ reg.
D1	MQSLC	2	Don't care	Don't care	MQ reg.
68	ADDI (15)	1	Accumulator	Decimal 15	Mask reg.
REPEAT N-1 TIMES [†]					
DA	AND/MQSLC	1	MQ reg.	Mask reg.	Interim reg/MQ reg.
D1	ADD/MQSLC	1	Accumulator	Interim reg.	Interim reg/MQ reg.
7F	BCDBIN	1	Interim reg.	Interim res.	Accumulator/MQ reg.
7F	BCDBIN	1	Accumulator	Interim reg.	Accumulator/MQ reg.
END REPEAT					
FA	AND	1	MQ reg.	Mask reg.	Interim reg.
D1	ADD MQSLC	1	Accumulator	Interim reg.	Accumulator

[†]N = Number of BCD digits

BINEX3, EX3BC, and EX3C assist binary to excess-3 conversion. Using BINEX3, an N-bit binary number can be converted to an $N/4$ - digit excess-3 number. For an algorithm, see Table 24.

Table 24. BCD to Binary Algorithm

OP CODE	MNEMONIC	CLOCK CYCLES	INPUT S PORT	INPUT R PORT	OUTPUT DESTINATION
E4	LOADMQ	1	Binary number	—	MQ reg.
D2	SUBR	1	Accumulator	Accumulator	Accumulator
D2	SET1 (33)16	1	Accumulator	Mask (33)16	Accumulator
REPEAT N TIMES†					
DF	BINEX3	1	Accumulator	Accumulator	Accumulator/MQ reg
9F	EX3C	1	Accumulator	Internal data	Accumulator
END REPEAT					

†N = Number of bits in binary number

Bit and Byte Instructions

Four Group 3 instructions allow the user to test or set selected bits within a byte. SET1 and SET0 force selected bits of a selected byte (or bytes) to one and zero, respectively. TB1 and TBO test selected bits of a selected byte (or bytes) for ones and zeros. The bits to be set or tested are specified by an 8-bit mask formed by the concatenation of register file address inputs C3-C0 and A3-A0. The register file addressed by B5-B0 is used as the destination operand for the set bit instructions. Register writes are inhibited for test bit instructions. Bytes to be operated on are selected by forcing SION low, where n represents the byte position and 0 represents the least significant byte. A high on the zero output pin signifies that the test data matches the mask; a low on the zero output indicates that the test has failed.

Individual bytes of data can also be manipulated using eight Group 3 byte arithmetic/logic instructions. Bytes can be added, subtracted, incremented, ORed, ANDed and exclusive ORed. Like the bit instructions, bytes are selected by forcing SION low, but multiple bytes can be operated on only if they are adjacent to one another; at least one byte must be nonselected.

Other Instructions

SEL (Group 4) selects one of the ALU's two operands, S or R, depending on the state of the SSF pin. This instruction could be used in sort routines to select the larger or smaller of two operands by performing a subtraction and sending the status result to SSF. CRC (Group 4) is designed to verify serial binary data that has been transmitted over a channel using a cyclic redundancy check code. An algorithm using this instruction is given in Table 25.

Table 25. CRC Algorithm

OP CODE	MNEMONIC	CLOCK CYCLES	INPUT S PORT	INPUT R PORT	OUTPUT DESTINATION
E4	LOADMQ	1	Vector $c'(x)^\dagger$	—	MQ reg.
F6	INCR	1	—	Polynomial $g(x)$	Poly reg.
F2	SUBR	1	Accumulator	Accumulator	Accumulator
REPEAT	n/8N TIMES [†]				
00	CRC	1	Accumulator	Poly reg.	Accumulator
E4	LOADMQ	1	Vector $c'(x)^\dagger$	—	MQ reg.
END REPEAT					

[†]N = Number of bits in binary number

n = Length of the code vector

CLR forces the ALU output to zero and clears the internal BCD flip-flops used in excess-3 BCD operations. NOP forces the ALU output to zero, but does not affect the flip-flops.

Configuration Options

The 'ACT8832 can be configured to operate in 8-bit, 16-bit, or 32-bit modes, depending on the setting of the configuration mode selects (CF2-CFO). Table 11 shows the control inputs for the four operating modes. Selecting an operating configuration other than 32-bit mode affects ALU operation and status generation in several ways, depending on the mode selected.

Masked 32-Bit Operation

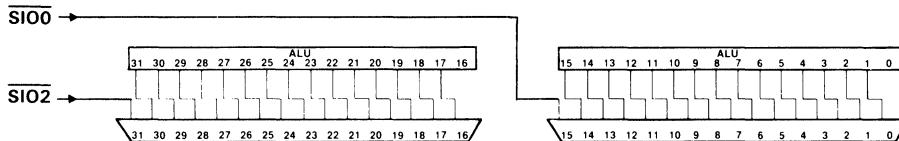
Masked 32-bit operation is selected to reset to zero the 20 most significant bits of the R Mux input. The 12 least significant bits are unaffected by the mask. Only Group 1 and Group 2 instructions can be used in this operating configuration. Status generation is similar to unmasked 32-bit operating mode.

Shift Instructions

Shift instructions operate similarly in 8-bit, 16-bit, and 32-bit modes. The serial I/O (SIO3'-SIO0') pins are used to select end-fill bits or to shift bits in or out, depending on the operation being performed. Table 12 shows the \overline{SIO} signals associated with each byte or word in the different modes, and Table 17 indicates the specific function performed by the \overline{SIO} pins during shift, multiply, and divide operations.

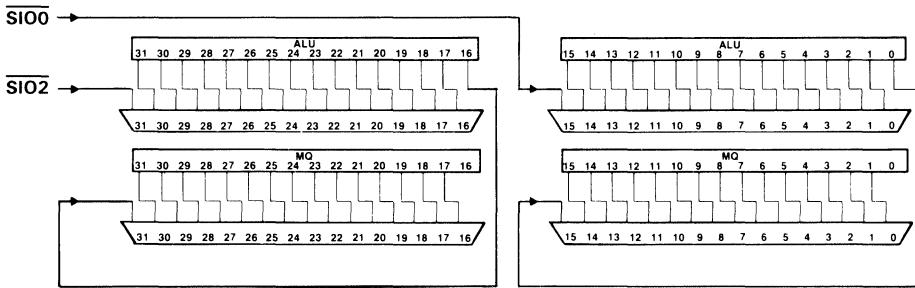
Figures 9 and 10 present examples of logical right shifts in 16-bit and 8-bit configurations.

**SERIAL DATA
INPUT SIGNALS**



Single Precision Logical Right Single Shift, 16-Bit Configuration

**SERIAL DATA
INPUT SIGNALS**



Double Precision Logical Right Single Shift, 16-Bit Configuration

Figure 9. Shift Examples, 16-Bit Configuration

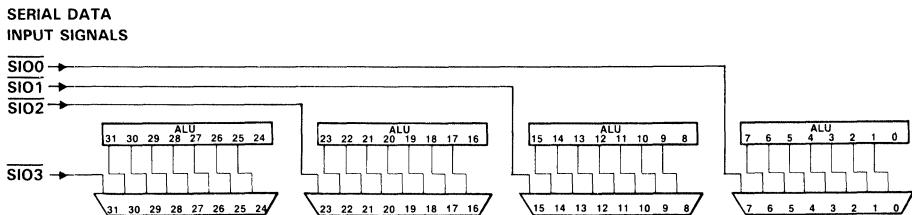
Bit and Byte Instructions

The 'ACT8832 performs bit operations similarly in 8-bit, 16-bit, and 32-bit modes. Masks are loaded into the R MUX on the A3-A0 and C3-C0 address inputs, and the bytes to be masked are selected by pulling their \overline{SIO} ' inputs low. Instructions which set, reset, or test bits are explained later.

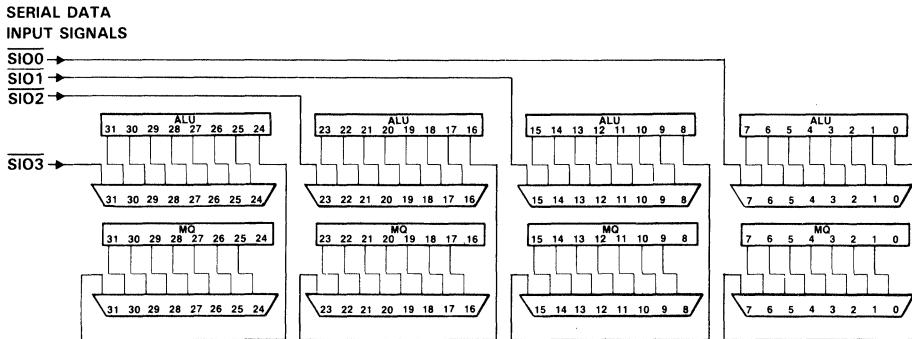
Byte operations should be performed in 32-bit mode to get the necessary status outputs. While byte overflow signals are provided for all four bytes (BYOF3-BYOF0), the other status signals (C, N, Z) are output only for the word selected with the configuration control signals (CF2-CF0).

Status Selection

Status results (C, N, Z, and overflow) are internally generated for all words in all modes, but only the overflow results (BYOF3-BYOF0) are available for all four bytes in 8-bit mode or for both words in 16-bit mode. If a specific application requires that the four status results are read for two or four words, it is possible to toggle the configuration



Single-Precision Logical Right Shift, 8-Bit Configuration



Double-Precision Logical Right Shift, 8-Bit Configuration

Figure 10. Shift Examples, 8-Bit Configuration

control signals (CF2-CF0) within the same clock cycle and read the additional status results. This assumes that the necessary external hardware is provided to toggle CF2-CF0 and collect the status for the individual words before the next clock signal is input.

Instruction Set

The 'ACT8832 instruction set is presented in alphabetical order on the following pages. The discussion of each instruction includes a functional description, list of possible operands, data flow diagram, and notes on status and control bits affected by the instruction. Microcoded examples are also shown.

Mnemonics and opcodes for instructions are given at the top of each page. Opcodes for instructions in Groups 1 and 2 are four bits long and are combined into eight-bit instructions which select combinations of arithmetic, logical, and shift operations. Opcodes for the other instruction groups are all eight bits long.

An asterisk in the left side of the opcode box for a Group 1 instruction indicates that a Group 2 opcode is needed to complete the instruction. An asterisk in the right side of a box indicates that a Group 1 opcode is required to combine with the Group 2 opcode in the left side of the box.

FUNCTION

Computes the absolute value of two's complement data on the S bus.

DESCRIPTION

Two's complement data on the S bus is converted to its absolute value. The carry must be set to one by the user for proper conversion. ABS causes $S' + C_n$ to be computed; the state of the sign bit determines whether S or $S' + C_n$ will be selected as the result. SSF is used to transmit the sign of S .

Available R Bus Source Operands

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
No	No	No	No

3

SN74ACT8832

Available S Bus Source Operands

RF (B5-B0)	DB-Port	MQ Register
Yes	Yes	Yes

Available Destination Operands Shift Operations

RF (C5-C0)	RF (B5-B0)	Y-Port	ALU	MQ
Yes	No	Yes	None	None

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inactive
<u>SIO0</u>	No	Inactive
<u>SIO1</u>	No	Inactive
<u>SIO2</u>	No	Inactive
<u>SIO3</u>	No	Inactive
Cn	Yes	Should be programmed high for proper conversion.

Status Signals

ZERO = 1 if result = 0

N = 1 if MSB (input) = 1

OVR = 1 if input of most significant byte is 80 (Hex) and inputs (if any) in all other bytes are 00 (Hex).

C = 1 if S = 0

EXAMPLES (assumes a 32-bit configuration)**3**

Convert the two's complement number in register 1 to its positive value and store the result in register 4.

SN74ACT8832

Instr Code 17-10	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel EB1- EA EBO	Dest Addr C5-C0	Destination Selects								Cn	CF2- CFO	
					WE3-	SELRF1-	OEY3				OEA	OEB	OEO	OES	
SELMO	WEO	SELRF0	OEA	OEB	OEO	OES									
0100 1000	XX XXXX	00 0001	X 00	00 0100	0	0000	10	X	X	XXXX	0	1	110		

Example 1: Assume register file 1 holds F6D81340 (Hex):

Source

1111 0110 1101 1000 0001 0011 0100 0000

 S \leftarrow RF(1)Destination

0000 1001 0010 0111 1110 1100 1100 0000

 RF(4) \leftarrow S + Cn

Example 2: Assume register file 1 holds 09D527C0 (Hex):

Source

0000 1001 1101 0101 0010 0111 1100 0000

 S \leftarrow RF(1)Destination

0000 1001 1101 0101 0010 0111 1100 0000

 RF(4) \leftarrow S

FUNCTION

Adds data on the R and S buses to the carry-in.

DESCRIPTION

Data on the R and S buses is added with carry. The sum appears at the ALU and MQ shifters.

*The result of this instruction can be shifted in the same microcycle by specifying a shift instruction in the upper nibble (I7-I4) of the instruction field. The result may also be passed without shift. Possible instructions are listed in Table 15.

3

SN74ACT8832

Available R Bus Source Operands

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
Yes	No	Yes	No

Available S Bus Source Operands

RF (B5-B0)	DB-Port	MQ Register
Yes	Yes	Yes

Available Destination Operands Shift Operations

RF (C5-C0)	RF (B5-B0)	Y-Port
Yes	No	Yes

ALU	MQ
Yes	Yes

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Affect shift instructions programmed in bits I7-I4 of instruction field.
SIO0	No	Inactive
SIO1	No	Inactive
SIO2	No	Inactive
SIO3	No	Inactive
Cn	Yes	Increments sum if set to one.

3

SN74ACT8832

Status Signals[†]

ZERO = 1 if result = 0

N = 1 if MSB = 1

OVR = 1 if signed arithmetic overflow

C = 1 if carry-out = 1

[†]C is ALU carry out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

EXAMPLES (assumes a 32-bit configuration)

Add data in register 1 to data on the DB bus with carry-in and pass the result to the MQ register.

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel EB1- EA EBO	Dest Addr C5-C0	Destination Selects								Cn	CF2- CFO	
					WE3-	SELRF1-	OEY3-			OEA	OEB	OEOY	OES		
SELMQ	WE0	SELRF0	OEA	OEB	OEOY	OES									
1110 0001	00 0001	XX XXXX	0 10	XX XXXX	0	1111	10	X	X	XXXX	0	0	0	110	

Assume register file 1 holds 0802C618 (Hex) and DB bus holds 1E007530 (Hex):

Source 0000 1000 0000 0010 1100 0110 0001 1000 R ← RF(1)

Source 0001 1110 0000 0000 0111 0101 0011 0000 S ← DB bus

Destination 0010 0110 0000 0011 0011 1011 0100 1000 MQ register ← R + S + Cn

FUNCTION

Adds four-bit immediate data on A3-A0 with carry to S-bus data.

DESCRIPTION

Immediate data in the range 0 to 15, supplied by the user at A3-A0, is added with carry to S.

Available R Bus Source Operands (Constant)

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: Mask
No	Yes	No	No

Available S Bus Source Operands

RF (B5-B0)	DB-Port	MQ Register
Yes	Yes	Yes

Available Destination Operands Shift Operations

RF (C5-C0)	RF (B5-B0)	Y-Port	ALU	MQ
Yes	No	Yes	None	None

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inactive
SIO0	No	Inactive
SIO1	No	Inactive
SIO2	No	Inactive
SIO3	No	Inactive
Cn	Yes	Increments sum if set to one.

Status Signals

ZERO	= 1 if result = 0
N	= 1 if MSB = 1
OVR	= 1 if signed arithmetic overflow
C	= 1 if carry-out = 1

EXAMPLES (assumes a 32-bit configuration)

Add the value 12 to data on the DB bus with carry-in and store the result in register file 1.

3

SN74ACT8832

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel EB1- <u>EA</u> EBO	Dest Addr C5-C0	Destination Selects								CF2- CFO Cn
					WE3- SELMQ	WE0 SELMO	SELRF1- SELRF0	OEA	OEB	OEY3 OEOY0	OES		
0110 1000	00 1100	XX XXXX	X 10	00 0001	0	0000	10	X	X	XXXX	0	0	110

Assume bits A5-A0 hold 0C (Hex) and DB bus holds 24000100 (Hex):

Source 0000 0000 0000 0000 0000 0000 0000 1100 R ← A5-A0

Source 0010 0100 0000 0000 0001 0000 0000 S ← DB bus

Destination 0010 0100 0000 0000 0000 0001 0000 1100 RF(1) ← R + S + Cn

FUNCTION

Evaluates the logical expression R AND S.

DESCRIPTION

Data on the R bus is ANDed with data on the S bus. The result appears at the ALU and MQ shifters.

*The result of this instruction can be shifted in the same microcycle by specifying a shift instruction in the upper nibble (I7-I4) of the instruction field. The result may also be passed without shift. Possible instructions are listed in Table 15.

Available R Bus Source Operands

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
Yes	No	Yes	No

Available S Bus Source Operands

RF (B5-B0)	DB-Port	MQ Register
Yes	Yes	Yes

Available Destination Operands Shift Operations

RF (C5-C0)	RF (B5-B0)	Y-Port	ALU	MQ
Yes	No	Yes	Yes	Yes

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Affect shift instructions programmed in bits I7-I4 of instruction field.
SIO0	No	Inactive
SIO1	No	Inactive
SIO2	No	Inactive
SIO3	No	Inactive
Cn	No	Inactive

Status Signals[†]

ZERO = 1 if result = 0

N = 1 if MSB = 1

OVR = 0

C = 0

[†]C is ALU carry out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

EXAMPLES (assumes a 32-bit configuration)

3

Logically AND the contents of register 3 and register 5 and store the result in register 5.

SN74ACT8832

Instr Code 17-10	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel <u>EB1-</u> <u>EA EA0</u>	Dest Addr C5-C0	Destination Selects									
					WE3-	SELRF1-			OEY3				Cn	CF2- CFO
					SELMQ	WE0	SELRF0	OE _A	OE _B	OEY0	OEY1	OES		
1111 1010	00 0011	00 0101	0 00	00 0101	0	0000	10	X	X	XXXX	0	X	110	

Assume register file 3 holds F617D840 (Hex) and register file 5 holds 15F6D842 (Hex):

Source 1111 0110 0001 0111 1101 1000 0100 0000 R ← RF(3)

Source 0001 0101 1111 0110 1101 1000 0100 0010 S ← RF(5)

Destination 0001 0100 0001 0110 1101 1000 0100 0000 RF(5) ← R AND S

FUNCTION

Computes the logical expression S AND NOT R.

DESCRIPTION

The logical expression S AND NOT R is computed. The result appears at the ALU and MQ shifters.

*The result of this instruction can be shifted in the same microcycle by specifying a shift instruction in the upper nibble (I7-I4) of the instruction field. The result may also be passed without shift. Possible instructions are listed in Table 15.

Available R Bus Source Operands

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
Yes	No	Yes	No

3

SN74ACT8832

Available S Bus Source Operands

RF (B5-B0)	DB-Port	MQ Register
Yes	Yes	Yes

Available Destination Operands Shift Operations

RF (C5-C0)	RF (B5-B0)	Y-Port	ALU	MQ
Yes	No	Yes	Yes	Yes

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Affect shift instructions programmed in bits I7-I4 of instruction field.
SIO0	No	Inactive
SIO1	No	Inactive
SIO2	No	Inactive
SIO3	No	Inactive
Cn	No	Inactive

Status Signals[†]

ZERO = 1 if result = 0

N = 0

OVR = 0

C = 0

[†]C is ALU carry out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

EXAMPLE (assumes a 32-bit configuration)**3**

Invert the contents of register 3, logically AND the result with data in register 5 and store the result in register 10.

SN74ACT8832

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel EB1- <u>EA</u> EBO	Dest Addr C5-C0	Destination Selects								Cn	CF2- CFO	
					WE3-	SELRF1-	OEY3-								
SELMQ	WEO	SELRF0	OEA	OEB	OEY0	OES									
1111 1110	00 0011	00 0101	0 00	00 1010	0	0000	10	X	X	XXXX	0	X	110		

Assume register file 3 holds 15F6D840 (Hex) and register file 5 hold F617D842 (Hex):

Source 0001 0101 1111 0110 1101 1000 0100 0000 R \leftarrow RF(3)

Source 1111 0110 0001 0111 1101 1000 0100 0010 S \leftarrow RF(5)

Destination 1110 0010 0000 0001 0000 0000 0000 0010 RF(10) $\leftarrow \bar{R} \text{ AND } S$

FUNCTION

Adds S with carry-in to a selected byte or selected adjacent bytes of R.

DESCRIPTION

SIO3-SIO0 are used to select bytes of R to be added to the corresponding bytes of S. A byte of R with SIO programmed low is selected for the computation of $R + S + C_n$. If the SIO signal for a byte of R is left high, the corresponding byte of S is passed unaltered. Multiple bytes can be selected only if they are adjacent to one another. At least one byte must be nonselected.

Available R Bus Source Operands

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
Yes	No	Yes	No

Available S Bus Source Operands

RF (B5-B0)	DB-Port	MQ Register
Yes	Yes	Yes

Available Destination Operands Shift Operations

RF (C5-C0)	RF (B5-B0)	Y-Port	ALU	MQ
Yes	No	Yes	None	None

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inactive
<u>SIO0</u>	Yes	Byte select
<u>SIO1</u>	Yes	Byte select
<u>SIO2</u>	Yes	Byte select
<u>SIO3</u>	Yes	Byte select
Cn	Yes	Propagates through nonselected bytes; increments selected byte(s) if programmed high.

Status Signals

ZERO	= 1 if result (selected bytes) = 0
N	= 0
OVR	= 1 if signed arithmetic overflow (selected bytes)
C	= 1 if carry-out (most significant selected byte) = 1

EXAMPLE (assumes a 32-bit configuration)

Add bytes 1 and 2 of register 3 with carry to the contents of register 1 and store the result in register 11.

3

SN74ACT8832

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel EB1- EA EBO	Dest Addr C5-C0	Destination Selects								CF2- CFO	SIO3- SIO0 IESI03- IESI00	
					WE3-	SELRF1-	SELMO	WEO	SELRFO	OEA	OEB	OEY3-	OEOY	OES	
0100 1000	00 0011	00 0001	0 00	00 1011	0	0000	10	X	X	XXXX	0	1	110	1001	0000

Assume register file 3 holds 2C018181 (Hex) and register file 1 holds 7A8FBE3E (Hex):

Source 0010 1100 0000 0001 1000 0001 1000 0001 Rn \leftarrow RF(3)n

Source 0111 1010 1000 1111 1011 1110 0011 1110 Sn \leftarrow RF(1)n

ALU 1010 0110 1001 0001 0100 0000 1100 0000 Fn \leftarrow Rn + Sn + Cn

Destination 0111 1010 1001 0001 0100 1111 0011 1110 RF(11)n \leftarrow Fn or Sn[†]

[†]F = ALU result

n = nth byte

Register file 11 gets F if byte selected, S if byte not selected.

BAND**Byte AND R AND S (Byte Logical AND R AND S)****E | 8****FUNCTION**

Evaluates the logical AND of selected bytes of R-bus and S-bus data.

DESCRIPTION

Bytes with their corresponding \overline{SIO} signals programmed low compute R AND S. Bytes with \overline{SIO} signals programmed high, pass S unaltered. Multiple bytes can be selected only if they are adjacent to one another. At least one byte must be nonselected.

Available R Bus Source Operands

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
Yes	No	Yes	No

3

Available S Bus Source Operands

RF (B5-B0)	DB-Port	MQ Register
Yes	Yes	Yes

SN74ACT8832

Available Destination Operands Shift Operations

RF (C5-C0)	RF (B5-B0)	Y-Port	ALU	MQ
Yes	No	Yes	None	None

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Forced low
$\overline{SIO0}$	Yes	Byte select
$\overline{SIO1}$	Yes	Byte select
$\overline{SIO2}$	Yes	Byte select
$\overline{SIO3}$	Yes	Byte select
Cn	No	Inactive

Status Signals

ZERO	= 1 if result (selected bytes) = 0
N	= 0
OVR	= 0
C	= 0

EXAMPLE (assumes a 32-bit configuration)

Logically AND bytes 1 and 2 of register 3 with input on the DB bus; store the result in register 3.

3

SN74ACT8832

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel EB1- EA EA EB0	Dest Addr C5-C0	Destination Selects								CF2- CFO	S103- S100	IESI03- IESI00
					WE3- SELMQ	SELRF1- WEO	SELRF0	OEA	OEB	OEY3- OEYO	OES	Cn			
1110 1000	00 0011	XX XXXX	0 10	00 0011	0	0000	10	X	X	XXXX	0	X	110	1001	0000

Assume register file 3 holds 398FBEBE (Hex) and input on the DB port is 4290BFBF (Hex):

Source 0011 1001 1000 1111 1011 1110 1011 1110 Rn \leftarrow RF(3)n

Source 0100 0010 1001 0000 1011 1111 1011 1111 Sn \leftarrow DBn

Destination 0100 0010 1000 0000 1011 1110 1011 1111 RF(3)n \leftarrow Fn or Sn[†]

[†]F = ALU result

n = nth byte

Register file 3 gets F if byte selected, S if byte not selected.

FUNCTION

Converts a BCD number to binary.

DESCRIPTION

This instruction allows the user to convert an N-digit BCD number to a 4N-bit binary number in 4(N-1) plus 8 clocks. The instruction sums the R and S buses with carry.

A one-bit arithmetic left shift is performed on the ALU output. A zero is filled into bit 0 of the least significant byte unless $\overline{S1O0}$ is set low, which would force bit 0 to one. Bit 7 of the most significant byte is dropped.

Simultaneously, the contents of the MQ register are rotated one bit to the left. Bit 7 of the most significant byte is rotated to bit 0 of the least significant byte.

3

SN74ACT8832

Recommended R Bus Source Operands

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
Yes	No	No	No

Recommended S Bus Source Operands

RF (B5-B0)	DB-Port	MQ Register
Yes	Yes	No

Recommended Destination Operands Shift Operations

RF (C5-C0)	RF (B5-B0)	Y-Port
Yes	No	No

ALU	MQ
Left	Left

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inactive
SIO0	Yes	If high or floating, fills a zero in LSB of ALU shifter; if low, fills a one in LSB of ALU shifter.
SIO1	No	Inactive in 32-bit configuration. Used in other configurations to select endfill in LSBs.
SIO2	No	
SIO3	No	
Cn	Yes	Should be programmed low for proper conversion.

3

Status Signals

ZERO = 1 if result = 0
N = 1 if MSB = 1
OVR = 1 if signed arithmetic overflow
C = 1 if carry-out = 1

SN74ACT8832

ALGORITHM

The following code converts an N-digit BCD number to a 4N-bit binary number in 4(N-1) plus 8 clocks. This is one possible user generated algorithm. It employs the standard conversion formula for a BCD number (shown here for 32 bits):

$$ABCD = [(A \times 10 + B) \times 10 + C] \times 10 + D.$$

The conversion begins with the most significant BCD digit. Addition is performed in radix 2.

PSEUDOCODE

LOADMQ	NUM	Load MQ with BCD number.
SUB	ACC, ACC, SLCMQ	Clear accumulator; Circular left shift MQ.
SUB	MSK, MSK, SLCMQ	Clear mask register; Circular left shift MQ.
SLCMQ		Circular left shift MQ.
SLCMQ		Circular left shift MQ.
ADDI	ACC, MSK, 15	Store 15 in mask register.

Repeat N-1 times:

(N = number of BCD digits)

AND	MQ, MSK, R1, SLCMQ	Extract one digit; Circular left shift MQ.
ADD	ACC, R1, R1, SLCMQ	Add extracted digit to accumulator, and store result in R1; Circular left shift MQ.
BCDBIN	R1, R1, ACC	Perform BCDBIN instruction, and store result in accumulator $[4 \times (\text{ACC} + 4 \times \text{digit})]$; Circular left shift MQ.
BCDBIN	ACC, R1, ACC	Perform BCDBIN instruction, and store result in accumulator $[10 \times (\text{ACC} + 10 \times \text{digit})]$; Circular left shift MQ.
(END REPEAT)		
AND	MQ MSK, R1	Fetch last digit.
ADD	ACC, R1, ACC	Add in last digit and store result in accumulator.

FUNCTION

$S' + C_n$ for selected bytes of S .

DESCRIPTION

Bytes with $\overline{SIO0}$ programmed low compute $S' + C_n$. Bytes with $\overline{SIO0}$ programmed high pass S unaltered. Multiple bytes can be selected only if they are adjacent to one another. At least one byte must be nonselected.

Available R Bus Source Operands

3

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
No	No	No	No

SN74ACT8832

Available S Bus Source Operands

RF (B5-B0)	DB-Port	MQ Register
Yes	Yes	Yes

Available Destination Operands Shift Operations

RF (C5-C0)	RF (B5-B0)	Y-Port	ALU	MQ
Yes	No	Yes	None	None

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inactive
$\overline{SIO0}$	Yes	Byte select
$\overline{SIO1}$	Yes	Byte select
$\overline{SIO2}$	Yes	Byte select
$\overline{SIO3}$	Yes	Byte select
Cn	Yes	Propagates through nonselected bytes; increments selected byte(s) if programmed high.

Status Signals

ZERO	= 1 if result (selected bytes) = 0
N	= 0
OVR	= 1 if signed arithmetic overflow (selected bytes)
C	= 1 if carry-out (most significant selected byte) = 1

EXAMPLE (assumes a 32-bit configuration)

Invert bytes 0 and 1 of register 3 and add them to the carry (bytes 2 and 3 are not changed). Store the result in register 3.

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel EB1- EA EBO	Dest Addr C5-C0	Destination Selects								CF2- SI03- IESIO3- SI00 IESIO0		
					WE3-	SELRF1-	WE0	SELRFO	OEA	OEB	OEY3-	OEY0	OES		
1100 1000	XX XXXX	00 0001	X 00	00 0011	0	0000	10	X	X	XXXX	0	1	110	1100	0000

Assume register file 3 holds A3018181 (Hex):

Source

1010 0011 0000 0001 1000 0001 1000 0001

 Sn \leftarrow RF(3)n

ALU

0101 1100 1111 1110 0111 1110 0111 1111

 Fn \leftarrow S'n + Cn

Destination

1010 0011 0000 0001 0111 1110 0111 1111

 RF(3)n \leftarrow Fn or Sn[†]

[†]F = ALU result

n = nth byte

Register file 3 gets F if byte selected, S if byte not selected.

FUNCTION

Increments selected bytes of S if the carry is set.

DESCRIPTION

Bytes with \overline{SIO}' inputs programmed low compute $S + C_n$. Bytes with \overline{SIO} inputs programmed high, pass S unaltered. Multiple bytes can be selected only if they are adjacent to one another. At least one byte must be nonselected.

Available R Bus Source Operands

3

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
No	No	No	No

SN74ACT8832

Available S Bus Source Operands

RF (B5-B0)	DB-Port	MQ Register
Yes	Yes	Yes

Available Destination Operands Shift Operations

RF (C5-C0)	RF (B5-B0)	Y-Port	ALU	MQ
Yes	No	Yes	None	None

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inactive
$\overline{SIO0}$	Yes	Byte select
$\overline{SIO1}$	Yes	Byte select
$\overline{SIO2}$	Yes	Byte select
$\overline{SIO3}$	Yes	Byte select
Cn	Yes	Propagates through nonselected bytes; increments selected byte(s) if programmed high.

Status Signals

ZERO	= 1 if result (selected bytes) = 0
N	= 0
OVR	= 1 if signed arithmetic overflow (selected bytes)
C	= 1 if carry-out (most significant selected byte) = 1

EXAMPLE (assumes a 32-bit configuration)

Add bytes 1 and 2 of register 7 to the carry (bytes 0 and 3 are not changed). Store the result in register 2.

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel EB1- EA EBO	Dest Addr C5-C0	Destination Selects								Cn	CF2- SIO3- IESIO3- SIOO	CF0	SIO3- IESIO3- SIOO
1011 1000	XX XXXX	00 0111	X 00	00 0010	0	0000	10	X	X	XXXX	0	1	110	1100	0000	

Assume register file 7 holds 408FBEBE (Hex):

Source 0100 0000 1000 1111 1011 1110 1011 1110 Sn \leftarrow RF(7)n

ALU 0100 0000 1000 1111 1011 1111 1011 1110 Fn \leftarrow Sn + Cn

Destination 0100 0000 1000 1111 1011 1111 1011 1110 RF(2)n \leftarrow Fn or Sn[†]

[†]F = ALU result

n = nth byte

Register file 11 gets F if byte selected, S if byte not selected.

FUNCTION

Converts a binary number to excess-3 representation.

DESCRIPTION

This instruction converts an N-digit binary number to a N/4 digit excess-3 number representation in $2N + 3$ clocks. The data on the R and S buses are added to the carry-in, which contains the most significant bit of the MQ register. The contents of the MQ register are rotated one bit to the left. The most significant bit is shifted out and passed to the least significant bit position. Depending on the configuration selected, this shift may be within the same byte or from the most significant byte to the least significant byte.

Recommended R Bus Source Operands

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
Yes	No	No	No

Recommended S Bus Source Operands

RF (B5-B0)	DB-Port	MQ Register
Yes	Yes	No

Recommended Destination Operands**Shift Operations**

RF (C5-C0)	RF (B5-B0)	Y-Port
Yes	No	Yes

ALU	MQ
None	Left

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inactive
SIO0	No	Inactive
SIO1	No	Inactive
SIO2	No	Inactive
SIO3	No	Inactive
Cn	No	Holds MSB of MQ register.

Status Signals

ZERO	= 1 if result = 0
N	= 1 if MSB = 1
OVR	= 1 if signed arithmetic overflow
C	= 1 if carry-out = 1

ALGORITHM

The following code converts an N-digit binary number to a N/4 digit excess-3 number in $2N+3$ clocks. It employs the standard conversion formula for a binary number:

$$\begin{aligned} a_n 2^n + a_{n-1} 2^{n-1} + a_{n-2} 2^{n-2} + \dots + a_0 = \\ \{(2a_n + a_{n-1}) \times 2 + a_{n-1}\} \times 2 + \dots + a_0 \} \times 2 + a^0. \end{aligned}$$

The conversion begins with the most significant bit. Addition during the BINEX3 instruction is performed in radix 10 (excess-3).

LOADMQ NUM	Load MQ with binary number.
SUB ACC, ACC, ACC	Clear accumulator;
SET1 ACC, 33 (Hex)	Store 33 (Hex) in all bytes of accumulator.

Repeat N times:

(N = number of bits in binary number)

BINEX3 ACC, ACC, ACC	Double accumulator and add in most significant bit of MQ register. Circular left shift MQ.
EX3C ACC	Perform excess-3 correction.

(END REPEAT)

FUNCTION

Evaluates R OR S of selected bytes.

DESCRIPTION

Bytes with \overline{SIO} inputs programmed low evaluate R OR S. Bytes with \overline{SIO} inputs programmed high, pass S unaltered. Multiple bytes can be selected only if they are adjacent to one another. At least one byte must be nonselected.

Available R Bus Source Operands

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
Yes	No	Yes	No

Available S Bus Source Operands

RF (B5-B0)	DB-Port	MQ Register
Yes	Yes	Yes

Available Destination Operands Shift Operations

RF (C5-C0)	RF (B5-B0)	Y-Port	ALU	MQ
Yes	No	Yes	None	None

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inactive
$\overline{SIO0}$	Yes	Byte select
$\overline{SIO1}$	Yes	Byte select
$\overline{SIO2}$	Yes	Byte select
$\overline{SIO3}$	Yes	Byte select
Cn	No	Inactive

BOR

Byte OR R and S (Byte Inclusive OR R and S)

F	8
---	---

Status Signals

ZERO = 1 if result (selected bytes) = 0
 N = 0
 OVR = 0
 C = 0

EXAMPLE (assumes a 32-bit configuration)

Logically OR bytes 1 and 2 of register 12 with bytes 1 and 2 on the DB bus. Concatenate with DB bytes 0 and 3, storing the result in register 12.

3

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel EB1- EA EBO	Dest Addr C5-C0	Destination Selects								Cn	CF2- CFO	SIO3- SIO0	IESIO3- IESIO0		
					WE3-	SELRF1-	OEY3-	SELMO	WEO	SELRF0	OEA	OEB	OEOY	OES				
1111 1000	00 1100	XX XXXX	0 10	00 1100	0	0000	10	X	X	XXXX	0	X	110	1001	0000			

Assume register file 12 holds 578FBEBE (Hex) and the DB bus holds 1C90BEBE (Hex):

Source 0101 0111 1000 1111 1011 1110 1011 1110 Rn \leftarrow RF(12)n

Source 0001 1100 1001 0000 1011 1110 1011 1100 Sn \leftarrow DBn

Destination 0001 1100 1001 1111 1011 1110 1011 1110 RF(12)n \leftarrow Fn or Sn[†]

[†]F = ALU result

n = nth package

Register file 12 gets F if byte selected, S if byte not selected.

SN74ACT8832

FUNCTION

Subtracts R from S in selected bytes.

DESCRIPTION

Bytes with \overline{SIO} inputs programmed low compute $R' + S + C_n$. Bytes with \overline{SIO} inputs programmed high, pass S unaltered. Multiple bytes can be selected only if they are adjacent to one another. At least one byte must be nonselected.

Available R Bus Source Operands

3

SN74ACT8832

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
Yes	No	Yes	No

Available S Bus Source Operands

RF (B5-B0)	DB-Port	MQ Register
Yes	Yes	Yes

Available Destination Operands Shift Operations

RF (C5-C0)	RF (B5-B0)	Y-Port	ALU	MQ
Yes	No	Yes	None	None

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inactive
$\overline{SIO0}$	Yes	Byte select
$\overline{SIO1}$	Yes	Byte select
$\overline{SIO2}$	Yes	Byte select
$\overline{SIO3}$	Yes	Byte select
Cn	Yes	Propagates through nonselected bytes; should be set high for two's complement subtraction.

Status Signals

ZERO	= 1 if result (selected bytes) = 0
N	= 0
OVR	= 1 if signed arithmetic overflow (selected bytes)
C	= 1 if carry-out (most significant selected byte) = 1

EXAMPLE (assumes a 32-bit configuration)

Subtract bytes 1 and 2 of register 1 with carry from bytes 1 and 2 of register 3. Concatenate with bytes 0 and 3 of register 3, storing the result in register 11.

Instr Code 17-10	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel EB1- EA EBO	Dest Addr C5-C0	Destination Selects						CF2- CFO SIO3- ESI03-	SIO0- ESI00	
1010 1000	00 0001	00 0011	0 00	00 1011	0	0000	10	X	X	XXXX	0	1	110 1001 0000

Assume register file 1 holds 091B5858 (Hex) and register file 3 holds 703A9898 (Hex):

Source 0000 1001 0001 1011 0101 1000 0101 1000 Rn \leftarrow RF(1)n

Source 0111 0000 0011 1010 1001 1000 1001 1000 Sn \leftarrow RF(3)n

ALU 0110 0111 0001 1111 0100 0000 0100 0000 Fn \leftarrow R'n + Sn + Cn

Destination 0111 0000 0001 1111 0100 0000 1001 1000 RF(11)n \leftarrow Fn or Sn[†]

[†]F = ALU result

n = nth package

Register file 11 gets F if byte selected, S if byte not selected.

FUNCTION

Subtracts S from R in selected bytes.

DESCRIPTION

Bytes with \overline{SIO} inputs programmed low compute $R + S' + C_n$. Bytes with \overline{SIO} inputs programmed high, pass S unaltered. Multiple bytes can be selected only if they are adjacent to one another. At least one byte must be nonselected.

Available R Bus Source Operands

3

SN74ACT832

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
Yes	No	Yes	No

Available S Bus Source Operands

RF (B5-B0)	DB-Port	MQ Register
Yes	Yes	Yes

Available Destination Operands Shift Operations

RF (C5-C0)	RF (B5-B0)	Y-Port	ALU	MQ
Yes	No	Yes	None	None

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inactive
$\overline{SIO0}$	Yes	Byte select
$\overline{SIO1}$	Yes	Byte select
$\overline{SIO2}$	Yes	Byte select
$\overline{SIO3}$	Yes	Byte select
Cn	Yes	Propagates through nonselected bytes; should be set high for two's complement subtraction.

Status Signals

ZERO	= 1 if result (selected bytes) = 0
N	= 0
OVR	= 1 if signed arithmetic overflow (selected bytes)
C	= 1 if carry-out (most significant selected byte) = 1

EXAMPLE (assumes a 32-bit configuration)

Subtract bytes 1 and 2 of register 3 with carry from bytes 1 and 2 of register 1. Concatenate with bytes 0 and 3 of register 3, storing the result in register 11.

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel EB1- EA EBO	Dest Addr C5-C0	Destination Selects								Cn	CF2- SIO3- IESIO3- SIO0 IESIO0	
					WE3- SELMQ	SELRF1- WE0	SELRF0 OEAE	OEAO	OEBO	OEYO	OES				
1001 1000	00 0001	00 0011	0 00	00 1011	0	0000	10	X	X	XXXX	0	1	110	1001	0000

Assume register file 1 holds 5288B8B8 (Hex) and register file 3 holds 143A9898 (Hex):

Source 0101 0010 1000 1000 1011 1000 1011 1000 Rn \leftarrow RF(1)n

Source 0001 0100 0011 1010 1001 1000 1001 1000 Sn \leftarrow RF(3)n

ALU 0011 1110 0100 1110 0010 0000 0010 0000 Fn \leftarrow Rn + S'n + Cn

Destination 0101 0010 0100 1110 0010 0000 1011 1000 RF(11)n \leftarrow Fn or Sn[†]

[†]F = ALU result

n = nth byte

Register file 11 gets F if byte selected, S if byte not selected.

3

SN74ACT8832

FUNCTION

Evaluates R exclusive OR S in selected bytes.

DESCRIPTION

Bytes with \overline{SIO} inputs programmed low evaluate R exclusive OR S. Bytes with \overline{SIO} inputs programmed high, pass S unaltered. Multiple bytes can be selected only if they are adjacent to one another. At least one byte must be nonselected.

Available R Bus Source Operands

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
Yes	No	Yes	No

3

SN74ACT8832

Available S Bus Source Operands

RF (B5-B0)	DB-Port	MQ Register
Yes	Yes	Yes

Available Destination Operands Shift Operations

RF (C5-C0)	RF (B5-B0)	Y-Port	ALU	MQ
Yes	No	Yes	None	None

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inactive
$\overline{SIO0}$	Yes	Byte select
$\overline{SIO1}$	Yes	Byte select
$\overline{SIO2}$	Yes	Byte select
$\overline{SIO3}$	Yes	Byte select
Cn	No	Inactive

Status Signals

ZERO = 1 if result (selected bytes) = 0

N = 0

OVR = 0

C = 0

EXAMPLE (assumes a 32-bit configuration)

Exclusive OR bytes 1 and 2 of register 6 with bytes 1 and 2 on the DB bus; concatenate the result with DB bytes 0 and 3, storing the result in register 10.

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel EB1- EA EBO	Dest Addr C5-C0	Destination Selects								CF2- Cn	SIO3- SIO0	IESI03- IESI00
					WE3-	SELRF1-	SELMQ	WEO	SELRF0	OEA	OEB	OEY3-	OES		
1101 1000	00 0110	XX XXXX	0 10	00 1010	0	0000	10	X	X	XXXX	0	1	110	1001	0000

Assume register file 6 holds 938FBEBE (Hex) and the DB bus holds 4190BEBE (Hex):

Source 1001 0011 1000 1111 1011 1110 1011 1110 Rn \leftarrow RF(6)n

Source 0100 0001 1001 0000 1011 1110 1011 1110 Sn \leftarrow DBn

Destination 0100 0001 0001 1111 0000 0000 1011 1110 RF(10)n \leftarrow Fn or Sn[†]

[†]F = ALU result

n = nth package

Register file 10 gets F if byte selected, S if byte not selected.

1	F
---	---

†

CLEAR**CLR**

FUNCTION

Forces ALU output to zero and clears the BCD flip-flops.

DESCRIPTION

ALU output is forced to zero and the BCD flip-flops are cleared.

†This instruction may also be coded with the following opcodes:
[2] [F], [3] [F], [4] [F], [6] [F], [B] [F], [C] [F], [E] [F]

Available R Bus Source Operands

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
No	No	No	No

3**SN74ACT8832**

Available S Bus Source Operands

RF (B5-B0)	DB-Port	MQ Register
No	No	No

Available Destination Operands Shift Operations

RF (C5-C0)	RF (B5-B0)	Y-Port
Yes	No	Yes

ALU	MQ
None	None

Status Signals

ZERO = 1
N = 0
OVR = 0
Cn = 0

FUNCTION

Evaluates R exclusive OR S for use with cyclic redundancy check codes.

DESCRIPTION

Data on the R bus is exclusive ORed with data on the S bus. If MQ0 XNORed with S0 is zero (MQ0 is the LSB of the MQ register and S0 is the LSB of S-bus data), the result is sent to the ALU shifter. Otherwise, data on the S bus is sent to the ALU shifter.

A right shift is performed; the MSB is filled with R0 (MQ0 XOR S0), where R0 is the LSB of R-bus data. A circular right shift is performed on MQ data.

Recommended R Bus Source Operands

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
Yes	No	No	No

Recommended S Bus Source Operands

RF (B5-B0)	DB-Port	MQ Register
Yes	Yes	No

Recommended Destination Operands Shift Operations

RF (C5-C0)	RF (B5-B0)	Y-Port
Yes	No	No

ALU	MQ
Right	Right

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inactive
SIO0	No	Inactive
SIO1	No	Inactive
SIO2	No	Inactive
SIO3	No	Inactive
Cn	No	Inactive

Status Signals

ZERO	= 1 if result = 0
N	= 0
OVR	= 0
C _n	= 0

CYCLIC REDUNDANCY CHARACTER CHECK**DESCRIPTION**

3

SN74ACT8832

Serial binary data transmitted over a channel is susceptible to error bursts. These bursts may be detected and corrected by standard encoding methods such as cyclic redundancy check codes, fire codes, or computer generated codes. These codes all divide the message vector by a generator polynomial to produce a remainder that contains parity information about the message vector.

If a message vector of m bits, $a(x)$, is divided by a generator polynomial, $g(x)$, of order $k-1$, a k bit remainder, $r(x)$, is formed. The code vector, $c(x)$, consisting of $m(x)$ and $r(x)$ of length $n = m+k$ is transmitted down the channel. The receiver divides the received vector by $g(x)$.

After m divide iterations, $r(x)$ will be regenerated only if there is no error in the message bits. After k more iterations, the result will be zero if and only if no error has occurred in either the message or the remainder.

ALGORITHM

An algorithm for a cyclic redundancy character check, using the 'ACT8832 as a receiver, is given below:

LOADMQ VEC(X)

Load MQ with first 32 message bits of received vector $c'(x)$.

LOAD POLY

Load register with polynomial $g(x)$.

CLEAR SUM

Clear register acting as accumulator.

REPEAT (n/32) TIMES:

SUM = SUM CRC POLY

Perform CRC instruction where
 R Bus = POLY
 S Bus = SUM
 Store result in SUM.

LOADMQ VEC(X)

Load MQ with next 32 message bits of received vector $c'(x)$.

(END REPEAT)

SUM now contains the remainder [r'(x)] of c'(x). A syndrome generation routine may be called next, if required.

Note that the most significant bit of

$$g(x) = (g_{k-1})(x^{k-1}) + (g_{k-2})(x^{k-2}) + \dots (g_0)(x^0)$$

is implied and that POLY(0) is set to zero if the length of g(x) requires fewer bits than are in the machine word width.

FUNCTION

Corrects the remainder of nonrestoring division routine if correction is required.

DESCRIPTION

DIVRF tests the result of the final step in nonrestoring division iteration: SDIVIT (for signed division) or UDIVIT (for unsigned division). An error in the remainder results when it is nonzero and the signs of the remainder and the dividend are different.

The R bus must be loaded with the divisor and the S bus with the most significant half of the previous result. The least significant half is in the MQ register. The Y bus result must be stored in the register file for use during the subsequent SDIVQF instruction.

DIVRF tests to determine whether a fix is required and evaluates:

$$Y \leftarrow S + R' + 1 \text{ if a fix is necessary}$$

$$Y \leftarrow S + R + 0 \text{ if a fix is unnecessary}$$

Overflow is reported to OVR at the end of the division routine (after SDIVQF).

Recommended R Bus Source Operands

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
Yes	No	No	No

Recommended S Bus Source Operands

RF (B5-B0)	DB-Port	MQ Register
Yes	Yes	No

Recommended Destination Operands Shift Operations

RF (C5-C0)	RF (B5-B0)	Y-Port
Yes	No	No

ALU	MQ
None	None

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inactive
SIO0	No	Inactive
SIO1	No	Inactive
SIO2	No	Inactive
SIO3	No	Inactive
Cn	Yes	Should be programmed high

Status Signals

ZERO = 1 if remainder = 0
N = 0
OVR = 0
Cn = 1 if carry-out = 1

3

SN74ACT8832

FUNCTION

Tests the two most significant bits of a double precision number. If they are the same, shifts the number to the left.

DESCRIPTION

This instruction is used to normalize a two's complement, double precision number by shifting the number one bit to the left and filling a zero into the LSB unless S100 is low. The S bus holds the most significant half; the MQ register holds the least significant half.

- 3** Normalization is complete when overflow occurs. The shift is inhibited whenever normalization is attempted on a number already normalized.

SN74ACT8832

Available R Bus Source Operands

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
No	No	No	No

Recommended S Bus Source Operands (MSH)

RF (B5-B0)	DB-Port	MQ Register
Yes	No	No

Recommended Destination Operands

RF (C5-C0)	RF (B5-B0)	Y-Port
Yes	No	No

Shift Operations (conditional)

ALU	MQ
Left	Left

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inactive
SIO0	Yes	When low, selects a one end-fill bit in LSB
SIO1	No	Passes internally generated end-fill bits
SIO2	No	
SIO3	No	
Cn	No	

Status Signals

ZERO = 1 if result = 0
 N = 1 if MSB = 1
 OVR = 1 if MSB XOR 2nd MSB = 1
 Cn = 0

EXAMPLE (assumes a 32-bit configuration)

Normalize a double-precision number.

(This example assumes that the MSH of the number to be normalized is in register 3 and the LSH is in the MQ register. The zero on the OVR pin at the end of the instruction cycle indicates that normalization is not complete and the instruction should be repeated).

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel EB1- EA EBO	Dest Addr C5-C0	Destination Selects								Cn	CF2- CFO	
					WE3- SELMQ	SELRF1- WE0	SELRF0 OEA	OEB	OEV3- OEYO	OEV0 OES					
0011 0000	XX XXXX	00 0011	X 00	00 0011	0	0000	10	X	X	XXXX	0	X	110		

Assume register file 3 holds FA75D84E (Hex) and MQ register holds 37F6D843 (Hex):

Source 1111 1010 0111 0101 1101 1000 0100 1110 ALU shifter \leftarrow RF(3)

Source 0011 0111 1111 0110 1101 1000 0100 0011 MQ shifter \leftarrow MQ register

Destination 1111 0100 1110 1011 1011 0000 1001 1101 8RF(3) \leftarrow Result (MSH)

Destination 0110 1111 1110 1101 1011 0000 1000 0110 MQ register \leftarrow Result (LSH)

0 OVR \leftarrow 0[†]

[†]Normalization not complete at the end of this instruction cycle.

FUNCTION

Output contents of the divide/BCD flip-flops.

DESCRIPTION

The contents of the divide/BCD flip-flops are passed through the MQ register to the Y output multiplexer.

Available R Bus Source Operands

3

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
No	No	No	No

SN74ACT8832

Available S Bus Source Operands

RF (B5-B0)	DB-Port	MQ Register
No	No	No

Available Destination Operands Shift Operations

RF (C5-C0)	RF (B5-B0)	Y-Port	ALU	MQ
No	No	Yes	None	None

Status Signals

ZERO = 0
N = 0
OVR = 0
Cn = 0

EXAMPLES (assumes a 32-bit configuration)

Dump divide/BCD flip-flops to Y output.

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel EB1- EA EBO	Dest Addr C5-C0	Destination Selects								Cn	CF2- CF0
					WE3- SELMQ	WE0 SELRF1-	SELRF0	OEA OEB	OEB OEA	OY3- OEY0	OES OEA			
0101 1111	XX XXXX	XX XXXX	X XX	XX XXXX	1	XXXX	XX	X	X	0000	X	X		110

Assume divide/BCD flip-flops contain 2A055470 (Hex):

Source 0010 1010 0000 0101 0101 0100 0111 0000 MQ register \leftarrow Divide/BCD flip-flops

3

Destination 0010 1010 0000 0101 0101 0100 0111 0000 Y output \leftarrow MQ register

SN74ACT8832

FUNCTION

Corrects the result of excess-3 addition or subtraction in selected bytes.

DESCRIPTION

This instruction corrects excess-3 additions or subtractions in the byte mode. For correct excess-3 arithmetic, this instruction must follow each add or subtract. The operand must be on the S bus.

Data on the S bus is added to a constant on the R bus determined by the state of the BCD flip flops and previous overflow condition reported on the SSF pin. Bytes with SIO inputs programmed low evaluate the correct excess-3 representation. Bytes with SIO inputs programmed high or floating, pass S unaltered.

SN74ACT8832

Available R Bus Source Operands

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
No	No	No	No

Available S Bus Source Operands

RF (B5-B0)	DB-Port	MQ Register
Yes	No	No

Available Destination Operands Shift Operations

RF (C5-C0)	RF (B5-B0)	Y-Port	ALU	MQ
Yes	No	No	No	No

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inactive
SIO0	Yes	Byte select
SIO1	Yes	Byte select
SIO2	Yes	Byte select
SIO3	Yes	Byte select
Cn	No	Inactive

Status Signals

ZERO	= 0
N	= 0
OVR	= 1 if arithmetic signed overflow
Cn	= 1 if carry-out = 1

EXAMPLE (assumes a 32-bit configuration)

Add two BCD numbers and store the sum in register 3. Assume data comes in on DB bus.

1. Clear accumulator (SUB ACC, ACC)
2. Store 33 (Hex) in all bytes of register (SET1 R2, H/33/)
3. Add 33 (Hex) to selected bytes of first BCD number (BADD DB, R2, R1)
4. Add 33 (Hex) to selected bytes of second BCD number (BADD DB, R2, R3)
5. Add selected bytes of registers 1 and 3 (BADD, R1, R3, R3)
6. Correct the result (EX3BC, R3, R3)

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel EB1- EA EBO	Dest Addr C5-C0	Destination Selects										Cn	CF2- CF0	SIO3- SIO0	IESI03- IESI00				
					WE3-		SELRF1-			OEY3-												
					SELMQ	WE0	SELRF0	OEA	OEB	OEY0	OES											
1111 0010	00 0010	XX XXXX	0 XX	00 0010	0	0000	10	X	X	XXXX	0	1	110	XXXX	XXXX							
0000 1000	00 0010	XX XXXX	0 XX	00 0010	0	0000	10	X	X	XXXX	0		X	110	XXXX	XXXX						
1000 1000	00 0010	XX XXXX	0 10	00 0001	0	0000	10	X	X	XXXX	0		0	110	1100	0000						
1000 1000	00 0010	XX XXXX	0 10	00 0011	0	0000	10	X	X	XXXX	0		0	110	1100	0000						
1000 1000	00 0001	00 0011	0 00	00 0011	0	0000	10	X	X	XXXX	0		0	110	1100	0000						
1000 1111	XX XXXX	00 0011	X 00	00 0011	0	0000	10	X	X	XXXX	0		0	110	1100	0000						

Assume DB bus holds 51336912 at third instruction and 34867162 at fourth instruction.

- 1

0000 0000 0000 0000 0000 0000 0000 0000

 RF(2) ← 0
- 2

0000 0000 0000 0000 0011 0011 0011 0011

 RF(2) ← 00003333 (Hex)
- 3

0101 0001 0011 0011 1001 1100 0100 0101

 RF(1) ← RF(2) + DB
- 4

0011 0100 1000 0110 1010 0100 1001 0101

 RF(3) ← RF(2) + DB
- 5

0011 0100 1000 0110 0100 0000 1101 1010

 RF(3)n ← RF(1)n + RF(3)n
- 6

0011 0100 1000 0110 0100 0000 0111 0100

 RF(3)n ← Corrected RF(3)n result

FUNCTION

Corrects the result of excess-3 addition or subtraction.

DESCRIPTION

This instruction corrects excess-3 additions or subtractions in the word mode. For correct excess-3 arithmetic, this instruction must follow each add or subtract. The operand must be on the S bus.

Data on the S bus is added to a constant on the R bus determined by the state of the BCD flip-flops and previous overflow condition reported on the SSF pin.

3

SN74ACT8832

Available R Bus Source Operands

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
No	No	No	No

Available S Bus Source Operands

RF (B5-B0)	DB-Port	MQ Register
Yes	No	No

Available Destination Operands Shift Operations

RF (C5-C0)	RF (B5-B0)	Y-Port	ALU	MQ
Yes	No	Yes	No	No

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inactive
<u>SIO0</u>	No	Inactive
<u>SIO1</u>	No	Inactive
<u>SIO2</u>	No	Inactive
<u>SIO3</u>	No	Inactive
Cn	No	Inactive

Status Signals

ZERO = 0
N = 1 if MSB = 1
OVR = 1 if arithmetic signed overflow
Cn = 1 if carry-out = 1

EXAMPLE (assumes a 32-bit configuration)

Add two BCD numbers and store the sum in register 3. Assume data comes in on DA bus.

1. Clear accumulator (SUB ACC, ACC)
2. Store 33 (Hex) in all bytes of register (SET1 R2, H/33/)
3. Add 33 (Hex) to all bytes of first BCD number (ADD DB, R2, R1)
4. Add 33 (Hex) to all bytes of second BCD number (ADD DB, R2, R3)
5. Add the excess-3 data (ADD, R1, R3, R3)
6. Correct the excess-3 result (EX3C, R3, R3)
7. Subtract the excess-3 bias to go to BCD result.

3

SN74ACT8832

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel EB1- EB0	Dest C5-C0	Destination Selects								Cn CF0 CF2-	
					WE3-		SELRF1-		OEY3-					
					SELMQ	WE0	SELRF0	OE0	OEA	OEB	OEY0	OES		
1111 0010	00 0010	XX XXXX	0 XX	00 0010	0	0000	10	X	X	XXXX	0	1	110	
0000 1000	00 0010	XX XXXX	0 XX	00 0010	0	0000	10	X	X	XXXX	0	X	110	
1111 0001	00 0010	XX XXXX	0 10	00 0001	0	0000	10	X	X	XXXX	0	0	110	
1111 0001	00 0010	XX XXXX	0 10	00 0011	0	0000	10	X	X	XXXX	0	0	110	
1111 0001	00 0001	00 0011	0 00	00 0011	0	0000	10	X	X	XXXX	0	0	110	
1001 1111	XX XXXX	00 0011	X 00	00 0011	0	0000	10	X	X	XXXX	0	0	110	
1111 0010	00 0010	00 0011	0 00	00 0011	0	0000	10	X	X	XXXX	0	0	110	

Assume DB bus holds 51336912 at third instruction and 34867162 at fourth instruction.

Results of Instruction Cycles:

3**SN74ACT832**

- 1

0000 0000 0000 0000 0000 0000 0000 0000

 RF(2) \leftarrow 0
- 2

0011 0011 0011 0011 0011 0011 0011 0011

 RF(2) \leftarrow 33333333 (Hex)
- 3

1000 0100 0110 0110 1001 1100 0100 0101

 RF(1) \leftarrow RF(2) + DB
- 4

0110 0111 1011 1001 1010 0100 1001 0101

 RF(3) \leftarrow RF(2) + DB
- 5

1110 1100 0010 0000 0100 0000 1101 1010

 RF(3) \leftarrow RF(1) + RF(3)
- 6

1011 1001 0101 0011 0111 0011 1010 0111

 RF(3) \leftarrow Corrected RF(3) result
- 7

1000 0110 0010 0000 0100 0000 0111 0100

 RF(3) \leftarrow RF(3) - RF(2)

FUNCTION

Evaluates $R' + C_n$.

DESCRIPTION

Data on the R bus is inverted and added with carry. The result appears at the ALU and MQ shifters.

*The result of this instruction can be shifted in the same microcycle by specifying a shift instruction in the upper nibble (I7-I4) of the instruction field. The result may also be passed without shift. Possible instructions are listed in Table 15.

Available R Bus Source Operands

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
Yes	No	Yes	No

Available S Bus Source Operands

RF (B5-B0)	DB-Port	MQ Register
No	No	No

Available Destination Operands

RF (C5-C0)	RF (B5-B0)	Y-Port	ALU Shifter	MQ Shifter
Yes	No	Yes	Yes	Yes

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Affect shift instructions programmed in bits I7-I4 of instruction field.
SIO0	No	
SIO1	No	
SIO2	No	
SIO3	No	
Cn	Yes	Increments if programmed high.

Status Signals[†]

ZERO = 1 if result = 0

N = 1 if MSB = 1

OVR = 1 if signed arithmetic overflow

C = 1 if carry-out = 1

[†]C is ALU carry out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

EXAMPLE (assumes a 32-bit configuration)

3

Convert the data on the DA bus to two's complement and store the result in register 4.

SN74ACT8832

Instr Code 17-10	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel EB1- EA EBO	Dest Addr C5-C0	Destination Selects						Cn	CF2- CF0	
					WE3- SELMQ	WE0- SELRF1-	WE0- SELRF0	OE _A	OE _B	OEY3- OEY0			
1111 0111	XX XXXX	XX XXXX	1 XX	00 0100	0	0000	10	X	X	XXXX	0	1	110

Assume register file 1 holds 3791FEF6 (Hex):

Source 0011 0111 1001 0001 1111 1110 1111 0110 R ← DADestination 1100 1000 0110 1110 0000 0001 0000 1010 RF(4) ← R' + Cn

FUNCTION

Evaluates S' + Cn.

DESCRIPTION

Data on the S bus is inverted and added to the carry. The result appears at the ALU and MQ shifters.

*The result of this instruction can be shifted in the same microcycle by specifying a shift instruction in the upper nibble (I7-I4) of the instruction field. The result may also be passed without shift. Possible instructions are listed in Table 15.

Available R Bus Source Operands

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
No	No	No	No

Available S Bus Source Operands

RF (B5-B0)	DB-Port	MQ Register
Yes	Yes	Yes

Available Destination Operands

RF (C5-C0)	RF (B5-B0)	Y-Port	ALU Shifter	MQ Shifter
Yes	No	Yes	Yes	Yes

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Affect shift instructions programmed in bits I7-I4 of instruction field.
SIO0	No	
SIO1	No	
SIO2	No	
SIO3	No	
Cn	Yes	Increments if programmed high.

Status Signals[†]

ZERO = 1 if result = 0

N = 1 if MSB = 1

OVR = 1 if signed arithmetic overflow

C = 1 if carry-out = 1

[†]C is ALU carry-out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

EXAMPLE (assumes a 32-bit configuration)

3

Convert the data on the MQ register to one's complement and store the result in register 4.

SN74ACT8832

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel EB1- EA EBO	Dest Addr C5-C0	Destination Selects								Cn CF2- CF0
					WE3- SELMOQ	WE0 WEO	SELRF1- SELRFO	OE3- OEA	OE1- OEB	OEY3- OEYO	OEY1- OES		
1111 0101	XX XXXX	XX XXXX	X 11	00 0100	0	0000	10	X	X	XXXX	0	0	110

Assume MQ register file 1 holds 3791FEF6 (Hex):

Source 0011 0111 1001 0001 1111 1110 1111 0110 S ← MQ register

Destination 1100 1000 0110 1110 0000 0001 0000 1001 RF(4) ← S' + Cn

FUNCTION

Increments R if the carry is set.

DESCRIPTION

Data on the R bus is added to the carry. The sum appears at the ALU and MQ shifters.

*The result of this instruction can be shifted in the same microcycle by specifying a shift instruction in the upper nibble (I7-I4) of the instruction field. The result may also be passed without shift. Possible instructions are listed in Table 15.

Available R Bus Source Operands

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
Yes	No	Yes	No

Available S Bus Source Operands (MSH)

RF (B5-B0)	DB-Port	MQ Register
No	No	No

Available Destination Operands

RF (C5-C0)	RF (B5-B0)	Y-Port	ALU Shifter	MQ Shifter
Yes	No	Yes	Yes	Yes

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Affect shift instructions programmed in bits I7-I4 of instruction field.
SIO0	No	
SIO1	No	
SIO2	No	
SIO3	No	
Cn	Yes	Increments R if programmed high.

Status Signals[†]

ZERO = 1 if result = 0

N = 1 if MSB = 1

OVR = 1 if signed arithmetic overflow

Cn = 1 if carry-out = 1

[†]C is ALU carry-out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

EXAMPLE (assumes a 32-bit configuration)

Increment the data on the DA bus and store the result in register 4.

3

SN74ACT8832

Instr Code 17-10	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel EB1- EA EBO	Dest Addr C5-C0	Destination Selects										
					WE3-	SELRF1-		OEY3-						Cn	CF2- CF0
1111 0110	XX XXXX	XX XXXX	1 XX	00 0100	0	0000	10	X	X	XXXX	0	1	110		

Assume register file 1 holds 3791FEF6 (Hex).

Source 0001 0111 1001 0001 1111 1110 1111 0110 R ← DADestination 0001 0111 1001 0001 1111 1110 1111 0111 RF(4) ← R + Cn

FUNCTION

Increments S if the carry is set.

DESCRIPTION

Data on the S bus is added to the carry. The sum appears at the ALU and MQ shifters.

*The result of this instruction can be shifted in the same microcycle by specifying a shift instruction in the upper nibble (I7-I4) of the instruction field. The result may also be passed without shift. Possible instructions are listed in Table 15.

Available R Bus Source Operands

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
No	No	No	No

3

SN74ACT8832

Available S Bus Source Operands

RF (B5-B0)	DB-Port	MQ Register
Yes	Yes	Yes

Available Destination Operands

RF (C5-C0)	RF (B5-B0)	Y-Port	ALU Shifter	MQ Shifter
Yes	No	Yes	Yes	Yes

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Affect shift instructions programmed in bits I7-I4 of instruction field.
SIO0	No	
SIO1	No	
SIO2	No	
SIO3	No	
Cn	Yes	Increments S if programmed high.

Status Signals[†]

ZERO	= 1 if result = 0
N	= 1 if MSB = 1
OVR	= 1 if signed arithmetic overflow
C	= 1 if carry-out = 1

[†]C is ALU carry-out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

EXAMPLE (assumes a 32-bit configuration)**3**

Increment the data in the MQ register and store the result in register 4.

SN74ACT8832

Instr Code 17-10	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel EB1- EA EBO	Dest Addr C5-C0	Destination Selects								Cn	CF2- CFO
					WE3-	SELRF1-	WE0	SELRF0	OEA	OEB	OEOY	OES		
1111 0100	XX XXXX	XX XXXX	X 11	00 0100	0	0000	10	X	X	XXXX	0	1	110	

Assume MQ register holds 54FF00FF (Hex):

Source

0101 0100 1111 1111 0000 0000 1111 1111

 S ← MQ registerDestination

0101 0100 1111 1111 0000 0001 0000 0000

 RF(4) ← S + Cn

FUNCTION

Load divide/BCD flip-flops from external data input.

DESCRIPTION

Uses an internal bypass path to load data from the S MUX directly into the divide/BCD flip-flops.

Available R Bus Source Operands

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
No	No	No	No

3

SN74ACT8832

Available S Bus Source Operands

RF (B5-B0)	DB-Port	MQ Register
Yes	Yes	Yes

Available Destination Operands

RF (C5-C0)	RF (B5-B0)	Y-Port	ALU Shifter	MQ Shifter
No	No	No	No	No

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inactive
<u>SIO0</u>	No	Inactive
<u>SIO1</u>	No	Inactive
<u>SIO2</u>	No	Inactive
<u>SIO3</u>	No	Inactive
Cn	No	Inactive

0 F**Load Divide/BCD Flip-Flops****LOADFF****Status Signals**

ZERO = 0
N = 0
OVR = 0
C = 0

EXAMPLE (assumes a 32-bit configuration)

Load the divide/BCD flip-flops with data from the DB input bus.

3**SN74ACT8832**

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel EB1- EA EBO	Dest Addr C5-C0	Destination Selects								CF2- CFO Cn
					WE3- SELMQ	WE0- SELMO	SELRF1- SELRF0	OEA	OEB	OEY3- OEYO	OES		
0000 1111	XX XXXX	XX XXXX	X 10	XX XXXX	X	XXXX	XX	X	X	XXXX	X	X	110

Assume DB input holds 2A08C618 (Hex):

Source

0010 1010 0000 1000 1100 0110 0001 1000

 S ← DB busDestination

0010 1010 0000 1000 1100 0110 0001 1000

 Divide/BCD flip-flops ← S

FUNCTION

Passes the result of the ALU instruction specified in the lower nibble of the instruction field to Y and the MQ register.

DESCRIPTION

The result of the arithmetic or logical operation specified in the lower nibble of the instruction field (I3-I0) is passed unshifted to Y and the MQ register.

*A list of ALU operations that can be used with this instruction is given in Table 15.

Shift Operations

ALU Shifter	MQ Shifter
None	None

3

SN74ACT8832

Available Destination Operands

RF (C5-C0)	RF (B5-B0)	Y-Port
Yes	No	Yes

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Outputs MQ0 (LSB)
SIO0'	No	Inactive
SIO1	No	Inactive
SIO2	No	Inactive
SIO3	No	Inactive
Cn	No	Inactive

Status Signals[†]

ZERO	= 1 if result = 0
N	= 1 if MSB of result = 1
	= 0 if MSB of result = 0
OVR	= 1 if signed arithmetic overflow
C	= 1 if carry-out = 1

[†]C is ALU carry-out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

E ***Pass ($Y \leftarrow F$) and Load MQ with F****LOADMQ****EXAMPLE (assumes a 32-bit configuration)**

Load the MQ register with data from register 1, and pass the data to the Y port.

(In this example, data is passed to the ALU by and INCR instruction without carry-in.)

Instr Code 17-10	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel EB1- EA EBO	Dest Addr C5-C0	Destination Selects						Cn	CF2- CF0	
					WE3-	SELRF1-	OEY3-			OEY0	OES		
SELMQ	WE0	SELRF0	OE _A	OE _B	OEY0	OES	Cn	CF0					
1111 0110	00 0001	XX XXXX	0 XX	XX XXXX	0	XXXX	XX	X	X	XXXX	0	0	110

Assume register file 1 holds 2A08C618 (Hex):

3Source 0010 1010 0000 1000 1100 0110 0001 1000 $R \leftarrow RF(1)$ Destination 0010 1010 0000 1000 1100 0110 0001 1000 MQ register $\leftarrow R + Cn$ **SN74ACT8832**

FUNCTION

Passes the result of the ALU instruction specified in the upper nibble of the instruction field to Y MUX. Performs a circular left shift on MQ.

DESCRIPTION

The result of the arithmetic or logical operation specified in the lower nibble of the instruction field (I3-I0) is passed unshifted to Y MUX.

The contents of the MQ register are rotated one bit to the left. The MSB is rotated out and passed to the LSB of the same word, which may be 1, 2, or 4 bytes long.

The shift may be made conditional on SSF. If SSF is high or floating, the shift result will be sent to the MQ register. If SSF is low, the MQ register will not be altered.

*A list of ALU operations that can be used with this instruction is given in Table 15.

Shift Operations

ALU Shifter	MQ Shifter
None	Circular Left

Available Destination Operands (ALU Shifter)

RF (C5-C0)	RF (B5-B0)	Y-Port
Yes	No	Yes

Control/Data Signals

Signal	User Programmable	Use
SSF	Yes	Passes shift result if high or floating; retains MQ without shift if low.
SIO0	No	Inactive
SIO1	No	Inactive
SIO2	No	Inactive
SIO3	No	Inactive
Cn	No	Affects arithmetic operation programmed in bits I3-I0 of instruction field.

Status Signals[†]

ZERO	= 1 if result = 0
N	= 1 if MSB of result = 1
	= 0 if MSB of result = 0
OVR	= 1 if signed arithmetic overflow
C	= 1 if carry-out = 1

[†]C is ALU carry-out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

3

EXAMPLE (assumes a 32-bit configuration)

Add data in register 1 to data on the DB bus with carry-in and store the unshifted result in register 1. Circular shift the contents of the MQ register one bit to the left.

SN74ACT8832

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Opnd Sel EB1- EA E80	Dest Addr C5-C0	Destination Selects								Cn	CF2- CFO		
					WE3-	SELRF1-	OEY3-		OEY0		OES					
SELMQ	WE0	SELRFO	OEA	OEB	OEO	OES										
1101 0001	00 0001	XX XXXX	0 10	00 0001	0	0000	10	X	X	XXXX	0	1	110			

Assume register file 1 holds 2508C618 (Hex), DB bus holds 11007530 (Hex), and MQ register holds 4DA99AOE (Hex).

Source 0010 0101 0000 1000 1100 0110 0001 1000 R \leftarrow RF(1)

Source 0001 0001 0000 0000 0111 0101 0011 0000 S \leftarrow DB bus

Destination 0011 0110 0000 1001 0011 1011 0100 1001 RF(1) \leftarrow R + S + Cn

Source 0100 1101 1010 1001 1001 1010 0000 1110 MQ shifter \leftarrow MQ register

Destination 1001 1011 0101 0011 0011 0100 0001 1100 MQ register \leftarrow MQ shifter

FUNCTION

Passes the result of the ALU instruction specified in the upper nibble of the instruction field to Y MUX. Performs a left shift on MQ.

DESCRIPTION

The result of the arithmetic or logical operation specified in the lower nibble of the instruction field (I3-I0) is passed unshifted to Y MUX.

The contents of the MQ register are shifted one bit to the left. A zero is filled into the least significant bit of each word unless the SIO input for that word is programmed low; this will force the least significant bit to one. The MSB is dropped from each word, which may be 1, 2, or 4 bytes long, depending on the configuration selected.

3

The shift may be made conditional on SSF. If SSF is high or floating, the shift result will be sent to the MQ register. If SSF is low, the MQ register will not be altered.

*A list of ALU operations that can be used with this instruction is given in Table 15.

SN74ACT8832

Shift Operations

ALU Shifter	MQ Shifter
None	Logical Left

Available Destination Operands (ALU Shifter)

RF (C5-C0)	RF (B5-B0)	Y-Port
Yes	No	Yes

Control/Data Signals

Signal	User Programmable	Use
SSF	Yes	Passes shift result if high or floating; retains MQ without shift if low.
<u>SIO0</u>	Yes	Fills a zero in LSB of MQ shifter if high or floating; sets LSB to one if low.
<u>SIO1</u>	No	Inactive in 32-bit configuration; used in configurations to select end-fill in LSBs.
<u>SIO2</u>	No	
<u>SIO3</u>	No	
Cn	No	Affects arithmetic operation programmed in bits I3-I0 of instruction field.

Status Signals[†]

ZERO	= 1 if result = 0
N	= 1 if MSB of result = 1
	= 0 if MSB of result = 0
OVR	= 1 if signed arithmetic overflow
C	= 1 if carry-out = 1

[†]C is ALU carry-out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

3**SN74ACT8832****EXAMPLE (assumes a 32-bit configuration)**

Add data in register 7 to data on the DB bus with carry-in and store the unshifted result in register 7. Shift the contents of the MQ register one bit to the left, filling a zero into the least significant bit.

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel EB1- EA EBO	Dest Addr C5-C0	Destination Selects								CF2- Cn	SIO3- SIO0	IESIO3- IESIO0
					WE3- SELMQ	SELRF1- WE0	SELRF0- OEA	OEA	OEB	OEB	OEY3- OEYO	OES			
1100 0001	00 0111	XX XXXX	0 10	00 0111	0	0000	10	X	X	XXXX	0	1	110	1111	0000

Assume register file 7 holds 7308C618 (Hex), DB bus holds 54007530 (Hex), and MQ register holds 61A99A0E (Hex).

Source 0111 0011 0000 1000 1100 0110 0001 1000 R \leftarrow RF(7)

Source 0101 0100 0000 0000 0111 0101 0011 0000 S \leftarrow DB bus

Destination 1100 0111 0000 1001 0011 1011 0100 1001 RF(7) \leftarrow R + S + Cn

Source 0110 0001 1010 1001 1001 1010 0000 1100 MQ shifter \leftarrow MQ register

Destination 1100 0011 0101 0011 0011 0100 0001 1000 MQ register \leftarrow MQ shifter

FUNCTION

Passes the result of the ALU instruction specified in the upper nibble of the instruction field to Y MUX. Performs an arithmetic right shift on MQ.

DESCRIPTION

The result of the arithmetic or logical operation specified in the lower nibble of the instruction field (I3-I0) is passed unshifted to Y MUX.

The contents of the MQ register are rotated one bit to the right. The sign bit of the most significant byte is retained. Bit 0 of the least significant byte is dropped.

The shift may be made conditional on SSF. If SSF is high or floating, the shift result will be sent to the MQ register. If SSF is low, the MQ register will not be altered.

*A list of ALU operations that can be used with this instruction is given in Table 15.

Shift Operations

ALU Shifter	MQ Shifter
None	Arithmetic Right

Available Destination Operands (ALU Shifter)

RF (C5-C0)	RF (B5-B0)	Y-Port
Yes	No	Yes

Control/Data Signals

Signal	User Programmable	Use
SSF	Yes	Passes shift result if high or floating; retains MQ without shift if low.
SIO0	No	Outputs LSB of MQ shifter (inverted).
SIO1	No	Inactive in 32-bit configurations; used in other configurations to output LSBs from MQ shifter (inverted).
SIO2	No	
SIO3	No	
Cn	No	Affects arithmetic operation programmed in bits I3-I0 of instruction field.

Status Signals[†]

ZERO	= 1 if result = 0
N	= 1 if MSB of result = 1
	= 0 if MSB of result = 0
OVR	= 1 if signed arithmetic overflow
C	= 1 if carry-out = 1

[†]C is ALU carry-out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

3**SN74ACT8832****EXAMPLE (assumes a 32-bit configuration)**

Add data in register 1 to data in register 10 with carry-in and store the unshifted result in register 1. Shift the contents of the MQ register one bit to the right, retaining the sign bit.

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Opnd Sel EB1- EA EBO	Dest Addr C5-C0	Destination Selects								Cn	CF2- CFO
					WE3- SELMQ	WE0	SELRF1- SELRF0	OE3- OEA	OE0- OEB	OEY3- OEYO	OEY0- OES			
1010 0001	00 0001	00 1010	0 00	00 0001	0	0000	10	X	X	XXXX	0	1	110	

Assume register file 1 holds 5608C618 (Hex), register file 10 holds 14007530 (Hex), and MQ register holds 98A99AOE (Hex).

Source 0101 0110 0000 1000 1100 0110 0001 1000 R \leftarrow RF(1)

Source 0001 0100 0000 0000 0111 0101 0011 0000 S \leftarrow RF(10)

Destination 0110 1010 0000 1001 0011 1011 0100 1001 RF(1) \leftarrow R + S + Cn

Source 1001 1000 1010 1001 1001 1010 0000 1110 MQ shifter \leftarrow MQ register

Destination 1100 1100 0101 0100 1100 1101 0000 0111 MQ register \leftarrow MQ shifter

FUNCTION

Passes the result of the ALU instruction specified in the upper nibble of the instruction field to Y MUX. Performs a right shift on MQ.

DESCRIPTION

The result of the arithmetic or logical operation specified in the lower nibble of the instruction field (I3-I0) is passed unshifted to Y MUX.

The contents of the MQ register are shifted one bit to the right. A zero is placed in the sign bit of the most significant byte unless the \overline{SIO} input for that byte is set to zero; this will force the sign bit to 1. Bit 0 of the least significant byte is dropped.

The shift may be made conditional on SSF. If SSF is high or floating, the shift result will be sent to the MQ register. If SSF is low, the MQ register will not be altered.

*A list of ALU operations that can be used with this instruction is given in Table 15.

Shift Operations

ALU Shifter	MQ Shifter
None	Logical Right

Available Destination Operands (ALU Shifter)

RF (C5-C0)	RF (B5-B0)	Y-Port
Yes	No	Yes

Control/Data Signals

Signal	User Programmable	Use
SSF	Yes	Passes shift result if high or floating; retains MQ without shift if low.
$\overline{SIO0}$	Yes	Fills a zero in LSB of MQ shifter if high or floating; sets LSB to one if low.
$\overline{SIO1}$	No	Inactive in 32-bit configuration; used in other configurations to select end-fill in LSBs.
$\overline{SIO2}$	No	
$\overline{SIO3}$	No	
Cn	No	Affects arithmetic operation programmed in bits I3-I0 of instruction field.

Status Signals[†]

ZERO	= 1 if result = 0
N	= 1 if MSB of result = 1
	= 0 if MSB of result = 0
OVR	= 1 if signed arithmetic overflow
C	= 1 if carry-out = 1

[†]C is ALU carry-out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

3

SN74ACT8832

EXAMPLE (assumes a 32-bit configuration)

Add data in register 1 to data on the DB bus with carry-in and store the unshifted result in register 1. Shift the contents of the MQ register one bit to the left.

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel EB1- EA EBO	Dest Addr C5-C0	Destination Selects						Cn	CF2- CFO	
					WE3- SELMOQ	WE0- SELRF0	SELRF1- OEA	OEB	OEY3- OEYO	OES			
1011 0001	00 0001	XX XXXX	0 10	00 0001	0	0000	10	X	X	XXXX	0	1	110

Assume register file 1 holds 5608C618 (Hex), DB bus holds 14007530 (Hex), and MQ register holds 98A99AOE (Hex).

Source 0101 0110 0000 1000 1100 0110 0001 1000 R \leftarrow RF(1)

Source 0001 0100 0000 0000 0111 0101 0011 0000 S \leftarrow DB bus

Destination 0110 1010 0000 1001 0011 1011 0100 1001 RF(1) \leftarrow R + S + Cn

Source 1001 1000 1010 1001 1001 1010 0000 1110 MQ shifter \leftarrow MQ register

Destination 0100 1100 0101 0100 1100 1101 0000 0111 MQ register \leftarrow MQ shifter

FUNCTION

Evaluates the logical expression R NAND S.

DESCRIPTION

Data on the R bus is NANDed with data on the S bus. The result appears at the ALU and MQ shifters.

*The result of this instruction can be shifted in the same microcycle by specifying a shift instruction in the upper nibble (I7-I4) of the instruction field. The result may also be passed without shift. Possible instructions are listed in Table 15.

Available R Bus Source Operands

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
Yes	No	Yes	No

Available S Bus Source Operands

RF (B5-B0)	DB-Port	MQ Register
Yes	Yes	Yes

Available Destination Operands

RF (C5-C0)	RF (B5-B0)	Y-Port	ALU Shifter	MQ Shifter
Yes	No	Yes	Yes	Yes

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Affect shift instructions programmed in bits I7-I4 of instruction field.
SIO0	No	
SIO1	No	
SIO2	No	
SIO3	No	
Cn		Inactive

Status Signals[†]

ZERO	= 1 if result = 0
N	= 1 if MSB = 1
OVR	= 0
C	= 0

[†]C is ALU carry out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

EXAMPLE (assumes a 32-bit configuration)

3

Logically NAND the contents of register 3 and register 5, and store the result in register 5.

SN74ACT8832

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel EB1- EA EBO	Dest Addr C5-C0	Destination Selects								Cn	CF2- CF0	
					WE3- SELMQ	WE0 SELRF1-	SELRF0	OEA	OEB	OEY3- OEYO	OEY0	OES			
1111 1100	00 0011	00 0101	0 00	00 0101	0	0000	10	X	X	XXXX	0	X	110		

Assume register file 1 holds 60F6D840 (Hex) and register file 5 holds 13F6D377 (Hex).

Source 0110 0000 1111 0110 1101 1000 0100 0000 R ← RF(3)

Source 0001 0011 1111 0110 1101 0011 0111 0111 S ← RF(5)

Destination 1111 1111 0000 1001 0010 1111 1011 1111 RF(5) ← R NAND S

NOP**No Operation**

F	F
---	---

FUNCTION

Forces ALU output to zero.

DESCRIPTION

This instruction forces the ALU output to zero. The BCD flip-flops retain their old value. Note that the clear instruction (CLR) forces the ALU output to zero and clears the BCD flip-flops.

Available R Bus Source Operands

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
No	No	No	No

3

SN74ACT8832

Available S Bus Source Operands

RF (B5-B0)	DB-Port	MQ Register
No	No	No

Available Destination Operands Shift Operations

RF (C5-C0)	RF (B5-B0)	Y-Port	ALU	MQ
Yes	No	Yes	None	None

Status Signals

ZERO = 1
N = 0
OVR = 0
C = 0

F F**No Operation****NOP****EXAMPLE** (assumes a 32-bit configuration)

Clear register 12.

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel EB1- EA EA EBO	Dest Addr C5-C0	Destination Selects								Cn	CF2- CF0
					SELMQ	WE3- WE0	SELRF1- SELRFO	OE _A	OE _B	OEY3- OEY0	OE _S			
1111 1111	XX XXXX	XX XXXX	X XX	00 1100	0	0000	10	X	X	XXXX	0	X	110	

Destination 0000 0000 0000 0000 0000 0000 0000 0000 RF(12) ← 0**3**

SN74ACT8832

FUNCTION

Evaluates the logical expression R NOR S.

DESCRIPTION

Data on the R bus is NORed with data on the S bus. The result appears at the ALU and MQ shifters.

*The result of this instruction can be shifted in the same microcycle by specifying a shift instruction in the upper nibble (I7-I4) of the instruction field. The result may also be passed without shift. Possible instructions are listed in Table 15.

Available R Bus Source Operands

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
Yes	No	Yes	No

Available S Bus Source Operands

RF (B5-B0)	DB-Port	MQ Register
Yes	Yes	Yes

Available Destination Operands

RF (C5-C0)	RF (B5-B0)	Y-Port	ALU Shifter	MQ Shifter
Yes	No	Yes	Yes	Yes

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Affect shift instructions programmed in bits I7-I4 of instruction field.
SIO0	No	
SIO1	No	
SIO2	No	
SIO3	No	
Cn	No	Inactive

*

D**Logical NOR (R NOR S)****NOR****Status Signals[†]**

ZERO = 1 if result = 0

N = 1 if MSB = 1

OVR = 0

C = 0

[†]C is ALU carry out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

EXAMPLE (assumes a 32-bit configuration)**3**

Logically NOR the contents of register 3 and register 5, and store the result in register 5.

SN74ACT8832

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel EB1- EA EBO	Dest Addr C5-C0	Destination Selects						Cn	CF2- CF0	
					WE3- SELMQ	WE0- WEO	SELRF1- SELRFO	OE3- OEA	OE2- OEB	OEY3- OEYO	OEY2- OES		
1111 1011	00 0011	00 0101	0 00	00 0101	0	0000	10	X	X	XXXX	0	X	110

Assume register file 3 holds 60F6D840 (Hex) and register file 5 holds 13F6D377 (Hex).

Source 0110 0000 1111 0110 1101 1000 0100 0000 R ← RF(3)

Source 0001 0011 1111 0110 1101 0011 0111 0111 S ← RF(5)

Destination 1000 1100 0000 1001 0010 0100 1000 1000 RF(5) ← R NOR S

FUNCTION

Evaluates the logical expression R OR S.

DESCRIPTION

Data on the R bus is ORed with data on the S bus. The result appears at the ALU and MQ shifters.

*The result of this instruction can be shifted in the same microcycle by specifying a shift instruction in the upper nibble (I7-I4) of the instruction field. The result may also be passed without shift. Possible instructions are listed in Table 15.

Available R Bus Source Operands

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
Yes	No	Yes	No

Available S Bus Source Operands

RF (B5-B0)	DB-Port	MQ Register
Yes	Yes	Yes

Available Destination Operands

RF (C5-C0)	RF (B5-B0)	Y-Port	ALU Shifter	MQ Shifter
Yes	No	Yes	Yes	Yes

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Affect shift instructions programmed in bits I7-I4 of instruction field.
SIO0	No	
SIO1	No	
SIO2	No	
SIO3	No	
Cn	No	Inactive

*

B**Logical OR (R OR S)****OR****Status Signals[†]**

ZERO = 1 if result = 0

N = 1 if MSB = 1

OVR = 0

C = 0

[†]C is ALU carry out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

EXAMPLE (assumes a 32-bit configuration)**3**

Logically OR the contents of register 5 and register 3, and store the result in register 3.

SN74ACT8832

Instr Code 17-10	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel EB1- EA EBO	Dest Addr C5-C0	Destination Selects								Cn CF2- CF0
					WE3- SELMQ	WE0- WEO	SELRF1- SELRF0	OEA	OEB	OEY3- OEYO	OES		
1111 1011	00 0101	00 0011	0 00	00 0011	0	0000	10	X	X	XXXX	0	X	110

Assume register file 5 holds 60F6D840 (Hex) and register file 3 holds 13F6D377 (Hex).

Source 0110 0000 1111 0110 1101 1000 0100 0000 R ← RF(5)

Source 0001 0011 1111 0110 1101 0011 0111 0111 S ← RF(3)

Destination 0111 0011 1111 0110 1101 1011 0111 0111 RF(3) ← R OR S

FUNCTION

Passes the result of the ALU instruction specified in the lower nibble of the instruction field to Y MUX.

DESCRIPTION

The result of the arithmetic or logical operation specified in the lower nibble of the instruction field (I3-I0) is passed unshifted to Y MUX.

*A list of ALU operations that can be used with this instruction is given in Table 15.

Available Destination Operands

RF (C5-C0)	RF (B5-B0)	Y-Port	ALU Shifter	MQ Shifter
Yes	No	Yes	None	None

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inactive
SIO0	No	Inactive
SIO1	No	Inactive
SIO2	No	Inactive
SIO3	No	Inactive
Cn	No	Affects arithmetic operation specified in bits I3-I0 of instruction field.

Status Signals[†]

ZERO = 1 if result = 0
N = 1 if MSB of result = 1
= 0 if MSB of result = 0
OVR = 1 if signed arithmetic overflow
C = 1 if carry-out condition

[†]C is ALU carry out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

F	*
---	---

Pass (Y \leftarrow F)

PASS

EXAMPLE (assumes a 32-bit configuration)

Add data in register 1 to data on the DB bus with carry-in and store the unshifted result in register 10.

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel EB1- EA EBO	Dest Addr C5-C0	Destination Selects								Cn	CF2- CF0
					WE3- SELMQ	SELRF1- WEO	SELRF0 OEA	OEB	OEY3- OEO	OEY0 OES				
1111 0001	00 0001	XX XXXX	0 10	00 1010	0	0000	10	X	X	XXXX	0	1	110	

Assume register file 3 holds 9308C618 (Hex) and DB bus holds 24007530 (Hex).

3

Source

1001 0011 0000 1000 1100 0110 0001 1000

 R \leftarrow RF(1)Source

0010 0100 0000 0000 0111 0101 0011 0000

 S \leftarrow DB busDestination

1011 0111 0000 1001 0011 1011 0100 1001

 RF(10) \leftarrow R + S + Cn

SN74ACT8832

FUNCTION

Performs one of N-2 iterations of nonrestoring signed division by a test subtraction of the N-bit divisor from the 2N-bit dividend. An algorithm using this instruction is given in the "Other Arithmetic Instructions" section.

DESCRIPTION

SDIVI performs a test subtraction of the divisor from the dividend to generate a quotient bit. The test subtraction passes if the remainder is positive and fails if negative. If it fails, the remainder will be corrected during the next instruction.

SDIVI checks the pass/fail result of the test subtraction from the previous instruction, and evaluates

$$\begin{aligned} F &\leftarrow R + S && \text{if the test fails} \\ F &\leftarrow R' + S + C_n && \text{if the test passes} \end{aligned}$$

A double precision left shift is performed; bit 7 of the most significant byte of the MQ shifter is transferred to bit 0 of the least significant byte of the ALU shifter. Bit 7 of the most significant byte of the ALU shifter is lost. The unfixed quotient bit is circulated into the least significant bit of the MQ shifter.

The R bus must be loaded with the divisor, the S bus with the most significant half of the result of the previous instruction (SDIVI during iteration or SDIVIS at the beginning of iteration). The least significant half of the previous result is in the MQ register. Carry-in should be programmed high. Overflow occurring during SDIVI is reported to OVR at the end of the signed divide routine (after SDIVQF).

Available R Bus Source Operands

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
Yes	No	Yes	No

Recommended S Bus Source Operands

RF (B5-B0)	DB-Port	MQ Register
Yes	Yes	No

Recommended Destination Operands

RF (C5-C0)	RF (B5-B0)	Y-Port
Yes	No	Yes

Shift Operations

ALU	MQ
Left	Left

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inactive
$\overline{SIO0}$	No	Pass internally generated end-fill bits.
$\overline{SIO1}$	No	
$\overline{SIO2}$	No	
$\overline{SIO3}$	No	
Cn	Yes	Should be programmed high

3**Status Signals**

ZERO = 1 if intermediate result = 0

N = 0

OVR = 0

C = 1 if carry-out

SN74ACT8832

FUNCTION

Initializes 'ACT8832 for nonrestoring signed division by shifting the dividend left and internally preserving the sign bit. An algorithm using this instruction is given in the "Other Arithmetic Instructions section.

DESCRIPTION

This instruction prepares for signed divide iteration operations by shifting the dividend and storing the sign for future use.

The preceding instruction should load the MQ register with the least significant half of the dividend. During SDIVIN, the S bus should be loaded with the most significant half of the dividend, and the R bus with the divisor. Y-output should be written back to the register file for use in the next instruction.

A double precision logical left shift is performed; bit 7 of the most significant byte of the MQ shifter is transferred to bit 0 of the least significant byte of the ALU shifter. Bit 7 of the most significant byte of the ALU shifter is lost. The unfixed quotient sign bit is shifted into the least significant bit of the MQ shifter.

Available R Bus Source Operands

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
Yes	No	Yes	No

Recommended S Bus Source Operands

RF (B5-B0)	DB-Port	MQ Register
Yes	Yes	No

Recommended Destination Operands

RF (C5-C0)	RF (B5-B0)	Y-Port
Yes	No	Yes

Shift Operations

ALU	MQ
Left	Left

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inactive
SIO0	No	Pass internally generated end-fill bits.
SIO1	No	
SIO2	No	
SIO3	No	
Cn	No	Inactive

3**Status Signals**

ZERO = 1 if divisor = 0

N = 0

OVR = 0

Cn = 0

SN74ACT8832

FUNCTION

Computes the first quotient bit of nonrestoring signed division. An algorithm using this instruction is given in the "Other Arithmetic Instructions" section..

DESCRIPTION

SDIVIS computes the first quotient bit during nonrestoring signed division by subtracting the divisor from the dividend, which was left-shifted during the prior SDIVIN instruction. The resulting remainder due to subtraction may be negative. If so, the subsequent SDIVI instruction will restore the remainder during the next subtraction.

The R bus must be loaded with the divisor and the S bus with the most significant half of the remainder. The result on the Y bus should be loaded back into the register file for use in the next instruction. The least significant half of the remainder is in the MQ register. Carry-in should be programmed high.

A double precision left shift is performed; bit 7 of the most significant byte of the MQ shifter is transferred to bit 0 of the least significant byte of the ALU shifter. Bit 7 of the most significant byte of the ALU shifter is lost. The unfixed quotient bit is circulated into the least significant bit of the MQ shifter.

Overflow occurring during SDIVIS is reported to OVR at the end of the signed division routine (after SDIVQF).

3

SN74ACT8832

Available R Bus Source Operands

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
Yes	No	Yes	No

Recommended S Bus Source Operands

RF (B5-B0)	DB-Port	MQ Register
Yes	Yes	No

Recommended Destination Operands Shift Operations

RF (C5-C0)	RF (B5-B0)	Y-Port
Yes	No	Yes

ALU	MQ
Left	Left

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inactive
SIO0	No	Pass internally generated end-fill bits.
SIO1	No	
SIO2	No	
SIO3	No	
Cn	Yes	Should be programmed high.

3**Status Signals**

ZERO = 1 if intermediate result = 0

N = 0

OVR = 0

C = 1 if carry-out

SN74ACT8832

FUNCTION

Solves the final quotient bit during nonrestoring signed division. An algorithm using this instruction is given in the "Other Arithmetic Instructions" section.

DESCRIPTION

SDIVIT performs the final subtraction of the divisor from the remainder during nonrestoring signed division. SDIVIT is preceded by N-2 iterations of SDIVI, where N is the number of bits in the dividend.

The R bus must be loaded with the divisor, and the S bus must be loaded with the most significant half of the result of the last SDIVI instruction. The least significant half lies in the MQ register. The Y bus result must be loaded back into the register file for use in the subsequent DIVRF instruction. Carry-in should be programmed high.

SDIVIT checks the pass/fail result of the previous instruction's test subtraction and evaluates;

$$\begin{array}{ll} Y \leftarrow R + S & \text{if the test fails} \\ Y \leftarrow R' + S + C_n & \text{if the test passes} \end{array}$$

The contents of the MQ register are shifted one bit to the left; the unfixed quotient bit is circulated into the least significant bit.

Overflow during this instruction is reported to OVR at the end of the signed division routine (after SDIVQF).

Available R Bus Source Operands

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
Yes	No	Yes	No

Recommended S Bus Source Operands

RF (B5-B0)	DB-Port	MQ Register
Yes	Yes	No

Recommended Destination Operands

RF (C5-C0)	RF (B5-B0)	Y-Port
Yes	No	Yes

Shift Operations

ALU	MQ
Left	Left

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inactive
$\overline{SIO0}$	No	Pass internally generated end-fill bits.
SIO1	No	
SIO2	No	
$\overline{SIO3}$	No	
Cn	Yes	Should be programmed high

3**Status Signals**

ZERO = 1 if intermediate result = 0

N = 0

OVR = 0

C = 1 if carry-out

SN74ACT8832

FUNCTION

Tests for overflow during nonrestoring signed division. An algorithm using this instruction is given in the "Other Arithmetic Instructions section.

DESCRIPTION

This instruction performs an initial test subtraction of the divisor from the dividend. If overflow is detected, it is preserved internally and reported at the end of the divide routine (after SDIVQF). If overflow status is ignored, the SDIVO instruction may be omitted.

The divisor must be loaded onto the R bus; the most significant half of the previous SDIVIN result must be loaded onto the S bus. The least significant half is in the MQ register.

The result on the Y bus should not be stored back into the register file; WE' should be programmed high.

Carry-in should also be programmed high.

Available R Bus Source Operands

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
Yes	No	Yes	No

Recommended S Bus Source Operands

RF (B5-B0)	DB-Port	MQ Register
Yes	Yes	No

Recommended Destination Operands

RF (C5-C0)	RF (B5-B0)	Y-Port
Yes	No	Yes

Shift Operations

ALU	MQ
None	None

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inactive
$\overline{SIO0}$	No	Inactive
$\overline{SIO1}$	No	Inactive
$\overline{SIO2}$	No	Inactive
$\overline{SIO3}$	No	Inactive
Cn	Yes	Should be programmed high

3**Status Signals**

ZERO = 1 if divisor = 0

N = 0

OVR = 0

C = 1 if carry-out

SN74ACT8832

FUNCTION

Tests the quotient result after nonrestoring signed division and corrects it if necessary. An algorithm using this instruction is given in the "Other Arithmetic Instructions" section.

DESCRIPTION

SDIVQF is the final instruction required to compute the quotient of a 2N-bit dividend by an N-bit divisor. It corrects the quotient if the signs of the divisor and dividend are different and the remainder is nonzero.

The fix is implemented by incrementing S:

$$\begin{aligned} Y &\leftarrow S + 1 && \text{if a fix is required} \\ Y &\leftarrow S + 0 && \text{if no fix is required} \end{aligned}$$

The R bus must be loaded with the divisor, and the S bus with the most significant half of the result of the preceding DIVRF instruction. The least significant half is in the MQ register.

3

SN74ACT8832

Available R Bus Source Operands

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
Yes	No	Yes	No

Recommended S Bus Source Operands

RF (B5-B0)	DB-Port	MQ Register
Yes	Yes	No

Recommended Destination Operands Shift Operations

RF (C5-C0)	RF (B5-B0)	Y-Port
Yes	No	Yes

ALU	MQ
None	None

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inactive
SIO0	No	Inactive
SIO1	No	Inactive
SIO2	No	Inactive
SIO3	No	Inactive
Cn	Yes	Should be programmed high

3**Status Signals**

ZERO	= 1 if quotient = 0
N	= 1 if sign of quotient + 1
	= 0 if sign of quotient + 0
OVR	= 1 if divide overflow
C	= 1 if carry-out

SN74ACT8832

SEL**Select S/R**

1 0

FUNCTION

Selects S if SSF is high; otherwise selects R.

DESCRIPTION

Data on the S bus is passed to Y if SSF is programmed high or floating; data on the R bus is passed without carry to Y if SSF is programmed low.

Available R Bus Source Operands

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
Yes	No	Yes	No

**Available S Bus Source
Operands (MSH)**

RF (B5-B0)	DB-Port	MQ Register
Yes	Yes	Yes

Available Destination Operands

RF (C5-C0)	RF (B5-B0)	Y-Port
Yes	No	Yes

Shift Operations

ALU	MQ
None	None

Control/Data Signals

Signal	User Programmable	Use
SSF	Yes	Selects S if high, R if low.
SIO0	No	Inactive
SIO1	No	Inactive
SIO2	No	Inactive
SIO3	No	Inactive
Cn	No	Inactive

SN74ACT8832

1 0

Select S/R

SEL

Status Signals

ZERO = 1 if result = 0

N = 1 if MSB = 1

OVR = 0

C = 0

EXAMPLE (assumes a 32-bit configuration)

Compare the two's complement numbers in registers 1 and 3 and store the larger in register 5.

3

1. Subtract (SUBS) data in register 3 from data in register 1 and pass the result to the Y bus.
2. Perform Select S/R instruction and pass result to register 5.

SN74ACT8832

[This example assumes the SSF is set by the negative status (N) from the previous instruction].

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel EB1- EA EBO	Dest Addr C5-C0	Destination Selects						Cn CF2- CF0	
					WE3- WEO	SELRF1- SELRF0	OEA	OEB	OEY3- OEY0	OES		
1111 0011	00 0001	00 0011	0 00	XX XXXX	0	XXXX	XX	X	0000	0	1	110
0001 0000	00 0001	00 0011	0 00	00 0101	0	0000	10	X	XXXX	0	0	110

Assume register file 1 holds 008497D0 (Hex) and register file 3 holds 01C35250 (Hex).

Instruction Cycle 1

Source 0000 0000 1000 0100 1001 0111 1101 0000 R ← RF(1)

Source 0000 0001 1100 0011 0101 0010 0101 0000 S ← RF(3)

Destination 1111 1110 1100 0001 0100 0101 1000 0000 Y bus ← R + S' + Cn

1 N ← 1

Instruction Cycle 2

Source 0000 0000 1000 0100 1001 0111 1101 0000 R ← RF(1)

1 SSF ← 1

Source 0000 0001 1100 0011 0101 0010 0101 0000 S ← RF(3)

Destination 0000 0001 1100 0011 0101 0010 0101 0000 RF(5) ← S

FUNCTION

Resets bits in selected bytes of S-bus data using mask in C3-C0::A3-A0.

DESCRIPTION

The register addressed by B5-B0 is both the source and destination for this instruction. The source word is passed on the S bus to the ALU, where it is compared to an 8-bit mask, consisting of a concatenation of the C3-C0 and A3-A0 address ports (C3-C0::A3-A0). The mask is input via the R bus. All bits in the source word that are in the same bit position as ones in the mask are reset. Bytes with their SIO inputs programmed low perform the Reset Bit instruction. Bytes with their SIO inputs programmed high or floating pass S unaltered.

3

SN74ACT8832

Available R Bus Source Operands

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
No	No	No	Yes

**Available S Bus Source
Operands (MSH)**

RF (B5-B0)	DB-Port	MQ Register
Yes	Yes	Yes

Available Destination Operands

RF (C5-C0)	RF (B5-B0)	Y-Port
No	Yes	Yes

Shift Operations

ALU	MQ
None	None

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inactive
<u>SIO0</u>	No	Byte-select
<u>SIO1</u>	No	Byte-select
<u>SIO2</u>	No	Byte-select
<u>SIO3</u>	No	Byte-select
Cn	No	Inactive

Status Signals

ZERO = 1 if result (selected bytes) = 0

N = 0

OVR = 0

C = 0

EXAMPLE (assumes a 32-bit configuration)

Set bits 3-0 of bytes 1 and 2 of register file 8 to zero and store the result back in register 8.

3

SN74ACT8832

Instr Code I7-I0	Mask (LSH) A3-A0	Oprd Addr B5-B0	Oprd Sel EB1- EA EBO	Mask (MSH) C3-C0	Destination Selects								CF2- Cn	SIO3- CFO	IESIO3- SIO0	IESIO0
					WE3-	SELRF1-	SELMO	WEO	SELRFO	OEA	OEB	OEY0	OES			
0001 1000	1111	00 1000	X 00	0000	0	0000	10	X	X	XXXX	0	X	110	1001	0000	

Assume register file 8 holds A083BEBE (Hex).

Source 0000 1111 0000 1111 0000 1111 0000 1111 Rn \leftarrow C3-C0::A3-A0

Source 1010 0000 1000 0011 1011 1110 1011 1110 Sn \leftarrow RF(3)n

ALU 1010 0000 1000 0000 1011 0000 1011 1110 Fn \leftarrow Sn AND Rn

Destination 1010 0000 1000 0000 1011 0000 1011 1110 RF(8)n \leftarrow Fn or Sn[†]

[†]F = ALU result

n = nth byte

Register file 8 gets F if byte selected, S if byte not selected.

FUNCTION

Sets bits in selected bytes of S-bus data using mask in C3-C0::A3-A0.

DESCRIPTION

The register addressed by B5-B0 is both the source and destination for this instruction. The source word is passed on the S bus to the ALU, where it is compared to an 8-bit mask, consisting of a concatenation of the C3-C0 and A3-A0 address ports (C3-C0::A3-A0). The mask is input via the R bus. All bits in the source word that are in the same bit position as ones in the mask are forced to a logical one. Bytes with their \overline{SIO} inputs programmed low perform the Set Bit instruction. Bytes with their \overline{SIO} inputs programmed high or floating pass S unaltered.

3

Available R Bus Source Operands

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
No	No	No	Yes

**Available S Bus Source
Operands (MSH)**

RF (B5-B0)	DB-Port	MQ Register
Yes	Yes	Yes

Available Destination Operands

RF (C5-C0)	RF (B5-B0)	Y-Port
No	Yes	Yes

Shift Operations

ALU	MQ
None	None

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inactive
$\overline{SIO0}$	Yes	Byte-select
$\overline{SIO1}$	No	Byte-select
$\overline{SIO2}$	No	Byte-select
$\overline{SIO3}$	No	Byte-select
Cn	No	Inactive

SN74ACT8832

Status Signals

ZERO = 1 if result (selected bytes) = 0

N = 0

OVR = 0

C = 0

EXAMPLE (assumes a 32-bit configuration)

Set bits 3-0 of byte 1 of register file 1 to zero and store the result back in register 1.

Instr Code I7-I0	Mask (LSH) A3-A0	Oprd Addr B5-B0	Oprd Sel EB1- EA EBO	Mask (MSH) C3-C0	Destination Selects								CF2- Cn	SIO3- SIO0 IESIO3- IESIO0	
0000 1000	1111	00 0001	X 00	0000	0	0000	10	X	X	XXXX	0	X	110	1101	0000

Assume register file 8 holds A083BEBE (Hex).

Source 0000 1111 0000 1111 0000 1111 0000 1111 Rn \leftarrow C3-C0::A3-A0

Source 1010 0000 1000 0011 1011 1110 1011 1110 Sn \leftarrow RF(1)n

ALU 1010 0000 1000 0011 1011 1111 1011 1110 Fn \leftarrow Sn OR Rn

Destination 1010 0000 1000 0011 1011 1111 1011 1110 RF(1)n \leftarrow Fn or Sn[†]

[†]F = ALU result

n = nth byte

Register file 1 gets F if byte selected, S if byte not selected.

FUNCTION

Performs arithmetic left shift on result of ALU operation specified in lower nibble of instruction field.

DESCRIPTION

The result of the ALU operation specified in instruction bits I3-I0 is shifted one bit to the left. A zero is filled into bit 0 of the least significant byte of each word unless the \overline{SIO} input is programmed low; this will force bit 0 to one. Bit 7 is dropped from the most significant byte in each word, which may be 1, 2, or 4 bytes long, depending on the configuration selected.

The shift may be made conditional on SSF. If SSF is high or floating, the shift result will be sent to the MQ register. If SSF is low, the MQ register will not be altered.

* A list of ALU operations that can be used with this instruction is given in Table 15.

Shift Operations

ALU Shifter	MQ Shifter
Arithmetic Left	None

Available Destination Operands (ALU Shifter)

RF (C5-C0)	RF (B5-B0)	Y-Port
Yes	No	Yes

Control/Data Signals

Signal	User Programmable	Use
SSF	Yes	Passes shift result if high; passes ALU result if low.
$\overline{SIO0}$	Yes	Fills a zero in LSB of each word if high; fills a one in LSB if low.
$\overline{SIO1}$	Yes	
$\overline{SIO2}$	Yes	
$\overline{SIO3}$	Yes	
Cn	No	Affects arithmetic operation programmed in bits I3-I0 of instruction field.

Status Signals[†]

ZERO	= 1 if result = 0
N	= 1 if MSB of result = 1
	= 0 if MSB of result = 0
OVR	= 1 if signed arithmetic overflow or if MSB XOR MSB-1 = 1 before shift
C	= 1 if carry-out condition

[†]C is ALU carry-out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

3

EXAMPLE (assumes a 32-bit configuration)

Perform the computation $A = 2(A + B)$, where A and B are single-precision, two's complement numbers. Let A be stored in register 1 and B be input via the DB bus.

SN74ACT8832

Instr Code 17-10	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel EB1- EA EBO	Dest Addr C5-C0	Destination Selects								Cn	CF2- CFO	SIO3- SIO0	IESI03- IESI00	SSF
					WE3- SELMQ	WE0- WE0	SELRF1- SELRF0	OEA	OEB	OEY3- OEY0	OES						
0100 0001	00 0001	XX XXXX	0 10	00 0001	0	0000	10	X	X	XXXX	0	0	110	1110	0000	1	

Assume register file 1 holds 1308C618 (Hex), DB bus holds 44007530 (Hex).

Source	0001 0011 0000 1000 1100 0110 0001 1000	R \leftarrow RF(1)
Source	0100 0100 0000 0000 0111 0101 0011 0000	S \leftarrow DB bus
Intermediate Result	0101 0111 0000 1001 0011 1011 0100 1000	ALU Shifter \leftarrow R + S + Cn
Destination	1010 1110 0001 0010 0111 0110 1001 0001	RF(1) \leftarrow ALU shift result

FUNCTION

Performs arithmetic left shift on MQ register (LSH) and result of ALU operation (MSH) specified in lower nibble of instruction field.

DESCRIPTION

The result of the ALU operation specified in instruction bits I3-I0 is used as the upper half of a double-precision word, the contents of the MQ register as the lower half.

The contents of the MQ register are shifted one bit to the left. A zero is filled into bit 0 of the least significant byte of each word unless the SIO input for the word is set to zero; this will force bit 0 to one. Bit 7 of the most significant byte in the MQ shifter is passed to bit 0 of the least significant byte of the ALU shifter. Bit 7 of the most significant byte in the ALU shifter is dropped.

The shift may be made conditional on SSF. If SSF is high or floating, the shift result will be sent to the Y MUX and MQ register. If SSF is low, the ALU output and MQ register will not be altered.

*A list of ALU operations that can be used with this instruction is given in Table 15.

Shift Operations

ALU Shifter	MQ Shifter
Arithmetic Left	Arithmetic Left

Available Destination Operands (ALU Shifter)

RF (C5-C0)	RF (B5-B0)	Y-Port
Yes	No	Yes

Control/Data Signals

Signal	User Programmable	Use
SSF	Yes	Passes shift result if high; passes ALU result if low.
SIO0	Yes	Fills a zero in LSB of each word if high; fills a one in LSB if low.
SIO1	Yes	
SIO2	Yes	
SIO3	Yes	
Cn	No	Affects arithmetic operation specified in bits I3-I0 of instruction field.

Status Signals[†]

ZERO = 1 if result = 0

N = 1 if MSB of result = 1

= 0 if MSB of result = 0

OVR = 1 if signed arithmetic overflow or if MSB XOR MSB-1 = 1 before shift

C = 1 if carry-out condition

[†]C is ALU carry-out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

3

EXAMPLE (assumes a 32-bit configuration)

SN74ACT8832

Perform the computation A = 2(A + B), where A and B are two's complement numbers.

Let A be a double precision number residing in register 1 (MSH) and the MQ register (LSH). Let B be a single precision number which is input through the DB bus.

Instr Code 17-10	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel EB1- EA EBO	Dest Addr C5-C0	Destination Selects								Cn	CF2- CFO SIO3- SIO0	IESI03- IESI00 SSF		
					WE3-	SELRF1-	OEY3-	WE0-	SELRF0	OEA	OEB	OEO	OES				
0101 0001	00 0001	XX XXXX	0 10	00 0001	0	0000	10	X	X	XXXX	0	0	0	110	1110	0000	1

Assume register file 1 holds 2408C618 (Hex), DB bus holds 26007530 (Hex), and MQ register holds 50A99AOE (Hex).

MSH

Source 0010 0100 0000 1000 1100 0110 0001 1000 R ← RF(1)

Source 0010 0110 0000 0000 0111 0101 0011 0000 S ← DB bus

Intermediate Result 0100 1010 0000 1001 0011 1011 0100 1000 ALU Shifter ← R + S + Cn

Destination 1001 0100 0001 0010 0111 0110 1001 0000 RF(1) ← ALU shift register

LSH

Source 0101 0000 1010 1001 1001 1010 0000 1110 MQ shifter ← MQ register

Destination 1010 0001 0101 0011 0011 0100 0001 1101 MQ register ← MQ shift result

FUNCTION

Performs circular left shift on result of ALU operation specified in lower nibble of instruction field.

DESCRIPTION

The result of the ALU operation specified in instruction bits I3-I0 is rotated one bit to the left. Bit 7 of the most significant byte in each word is passed to bit 0 of the least significant byte in the word, which may be 1, 2, or 4 bytes long.

The shift may be made conditional on SSF. If SSF is high or floating, the shift result will be sent to Y MUX. If SSF is low, F is passed unaltered.

*A list of ALU operations that can be used with this instruction is given in Table 15.

3

SN74ACT8832

Shift Operations

ALU Shifter	MQ Shifter
Circular Left	None

Available Destination Operands (ALU Shifter)

RF (C5-C0)	RF (B5-B0)	Y-Port
Yes	No	Yes

Control/Data Signals

Signal	User Programmable	Use
SSF	Yes	Passes shift result if high; passes ALU result if low.
SIO0	No	Bit 7 of ALU result
SIO1	No	Bit 15 of ALU result
SIO2	No	Bit 23 of ALU result
SIO3	No	Bit 31 of ALU result
Cn	No	Affects arithmetic operation specified in bits I3-I0 of instruction field.

Status Signals[†]

ZERO	= 1 if result = 0
N	= 1 if MSB of result = 1
	= 0 if MSB of result = 0
OVR	= 1 if signed arithmetic overflow
C	= 1 if carry-out condition

[†]C is ALU carry-out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

3

EXAMPLE (assumes a 32-bit configuration)

Perform a circular left shift of register 6 and store the result in register 1.

SN74ACT8832

Instr Code 17-10	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel EB1- EA EBO	Dest Addr C5-C0	Destination Selects								Cn	CF2- CFO	SSF
					WE3-	SELMQ	WE0	SELRF1-	OE4	OEY3-	OEY0	OE3			
0110 0110	00 0110	XX XXXX	0 00	00 0001	0	0000	10	X	X	XXXX	0	0	0	110	1

Assume register file 6 holds 3788C618 (Hex).

Source

0011 0111 1000 1000 1100 0110 0001 1000

 R ← RF(6)

Intermediate Result

0011 0111 1000 1000 1100 0110 0001 1000

 ALU Shifter ← R + Cn

Destination

0110 1111 0001 0001 1000 1100 0011 0000

 RF(1) ← ALU shifter result

FUNCTION

Performs circular left shift on MQ register (LSH) and result of ALU operation specified in lower nibble of instruction field (MSH).

DESCRIPTION

The result of the ALU operation specified in instruction bits I3-I0 is used as the upper half of a double-precision word, the contents of the MQ register as the lower half.

The contents of the MQ and ALU registers are rotated one bit to the left. Bit 7 of the most significant byte in the MQ shifter is passed to bit 0 of the least significant byte of the ALU shifter. Bit 7 of the most significant byte is passed to bit 0 of the least significant byte in the MQ shifter.

The shift may be made conditional on SSF. If SSF is high or floating, the shift result will be sent to Y MUX. If SSF is low, F is passed unaltered and the MQ register is not changed.

*A list of ALU operations that can be used with this instruction is given in Table 15.

Shift Operations

ALU Shifter	MQ Shifter
Circular Left	Circular Left

Available Destination Operands (ALU Shifter)

RF (C5-C0)	RF (B5-B0)	Y-Port
Yes	No	Yes

Control/Data Signals

Signal	User Programmable	Use
SSF	Yes	Passes shift result if high; passes ALU result if low.
SIO0	No	Bit 7 of ALU result
SIO1	No	Bit 15 of ALU result
SIO2	No	Bit 23 of ALU result
SIO3	No	Bit 31 of ALU result
Cn	No	Affects arithmetic operation specified in bits I3-I0 of instruction field.

Status Signals[†]

ZERO	= 1 if result = 0
N	= 1 if MSB of result = 1
	= 0 if MSB of result = 0
OVR	= 1 if signed arithmetic overflow
C	= 1 if carry-out condition

[†]C is ALU carry-out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

3**EXAMPLE (assumes a 32-bit configuration)**

Perform a circular left double precision shift of data in register 6 (MSH) and MQ (LSH), and store the result back in register 6 and the MQ register.

SN74ACT8832

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel EB1- EA EBO	Dest Addr C5-C0	Destination Selects								CF2- CFO	SSF	
					WE3-	SELRF1-	OEY3-	Cn							
SELMO	WE0	SELRF0	DEA	DEB	OEY0	DES									
0111 0110	00 0110	XX XXXX	0 00	00 0110	0	0000	10	X	X	XXXX	0	0	110	1	

Assume register file 6 holds 3708C618 (Hex) and MQ register holds 50A99AOE (Hex).

MSH

Source 0011 0111 0000 1000 1100 0110 0001 1000 R ← RF(6)

Intermediate Result 0011 0111 0000 1000 1100 0110 0001 1000 ALU Shifter ← R + Cn

Destination 0110 1111 0001 0001 1000 1100 0011 0000 RF(6) ← ALU shifter result

LSH

Source 0101 0000 1010 1001 1001 1010 0000 1110 MQ register ← MQ register

Destination 1010 0001 0101 0011 0011 0100 0001 1100 MQ register ← MQ shift result

FUNCTION

Converts data on the S bus from sign magnitude to two's complement or vice versa.

DESCRIPTION

The S bus provides the source word for this instruction. The number is converted by inverting S and adding the result to the carry-in, which should be programmed high for proper conversion; the sign bit of the result is then inverted. An error condition will occur if the source word is a negative zero (negative sign and zero magnitude). In this case, SMT generates a positive zero, and the OVR pin is set high to reflect an illegal conversion.

The sign bit of the selected operand in the most significant byte is tested; if it is high, the converted number is passed to the destination. Otherwise the operand is passed unaltered.

3

SN74ACT8832

Available R Bus Source Operands

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
No	No	No	No

Available S Bus Source Operands

RF (B5-B0)	DB-Port	MQ Register
Yes	Yes	Yes

Available Destination Operands Shift Operations

RF (C5-C0)	RF (B5-B0)	Y-Port
Yes	No	Yes

ALU	MQ
None	None

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inactive
<u>SIO0</u>	No	Inactive
<u>SIO1</u>	No	Inactive
<u>SIO2</u>	No	Inactive
<u>SIO3</u>	No	Inactive
Cn	Yes	Should be programmed high for proper conversion

3

SN74ACT8832

Status Signals

ZERO = 1 if result = 0

N = 1 if MSB = 1

OVR = 1 if input of most significant byte is 80 (Hex) and results in all other bytes are 00 (Hex).

C = 1 if S = 0

EXAMPLES (assumes a 32-bit configuration)

Convert the two's complement number in register 1 to sign magnitude representation and store the result in register 4.

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel EB1- EA EBO	Dest Addr C5-C0	Destination Selects								CF2- CF0
					<u>WE3-</u> SELMQ	<u>WEO</u>	<u>SELRF1-</u> SELRFO	<u>OEA</u>	<u>OEB</u>	<u>OEY3-</u> OEA	<u>OEO</u>	<u>OES</u>	
0101 1000	XX XXXX	00 0001	X 00	00 0100	0	0000	10	X	X	XXXX	0	1	110

Example 1: Assume register file 1 holds C3F6D840 (Hex).

Source	1100 0011 1111 0110 1101 1000 0100 0000	S \leftarrow RF(1)
--------	---	----------------------

Destination	1011 1100 0000 1001 0010 0111 1100 0000	RF(4) \leftarrow S' + Cn
-------------	---	----------------------------

Example 2: Assume register file 1 holds 550927C0 (Hex).

Source	0101 0101 0000 1001 0010 0111 1100 0000	S \leftarrow RF(1)
--------	---	----------------------

Destination	0101 0101 0000 1001 0010 0111 1100 0000	RF(4) \leftarrow S
-------------	---	----------------------

FUNCTION

Computes one of N-1 signed or N mixed multiplication iterations for computing an N-bit by N-bit product. Algorithms for signed and mixed multiplication using this instruction are given in the "Other Arithmetic Instructions" section.

DESCRIPTION

SMULI checks to determine whether the multiplicand should be added with the present partial product. The instruction evaluates:

$$\begin{aligned} F &\leftarrow R + S + C_n && \text{if the addition is required} \\ F &\leftarrow S && \text{if no addition is required} \end{aligned}$$

A double precision right shift is performed. Bit 0 of the least significant byte of the ALU shifter is passed to bit 7 of the most significant byte of the MQ shifter; carry-out is passed to the most significant bit of the ALU shifter.

The S bus should be loaded with the contents of an accumulator and the R bus with the multiplicand. The Y bus result should be written back to the accumulator after each iteration of UMULI. The accumulator should be cleared and the MQ register loaded with the multiplier before the first iteration.

3

SN74ACT8832

Available R Bus Source Operands

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
Yes	No	Yes	No

Recommended S Bus Source Operands

RF (B5-B0)	DB-Port	MQ Register
Yes	Yes	No

Recommended Destination Operands Shift Operations

RF (C5-C0)	RF (B5-B0)	Y-Port
Yes	No	No

ALU	MQ
Right	Right

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inactive
SIO0	No	Passes LSB from ALU shifter to MSB of MQ shifter.
SIO1	No	
SIO2	No	
SIO3	No	
Cn	Yes	Should be programmed low

3**Status Signals**

ZERO = 1 if result = 0

N = 1 if MSB = 1

OVR = 0

C = 1 if carry-out

SN74ACT8832

FUNCTION

Performs the final iteration for computing an N-bit by N-bit signed product. An algorithm for signed multiplication using this instruction is given in the “other Arithmetic Instructions” section.

DESCRIPTION

SMULI checks the present multiplier bit (the least significant bit of the MQ register) to determine whether the multiplicand should be added with the present partial product. The instruction evaluates:

$$\begin{aligned} F &\leftarrow R' + S + C_n && \text{if the addition is required} \\ F &\leftarrow S && \text{if no addition is required} \end{aligned}$$

with the correct sign in the product.

A double precision right shift is performed. Bit 0 of the least significant byte of the ALU shifter is passed to bit 7 of the most significant byte of the MQ shifter.

The S bus should be loaded with the contents of an register file holding the previous iteration result; the R bus must be loaded with the multiplicand. After executing SMULT, the Y bus contains the most significant half of the product, and MQ contains the least significant half.

3

SN74ACT8832

Available R Bus Source Operands

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
Yes	No	Yes	No

Recommended S Bus Source Operands

RF (B5-B0)	DB-Port	MQ Register
Yes	Yes	No

Available Destination Operands Shift Operations

RF (C5-C0)	RF (B5-B0)	Y-Port	ALU	MQ
Yes	No	No	Right	Right

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inactive
<u>SIO0</u>	No	Passes LSB from ALU shifter to MSB of MQ shifter.
<u>SIO1</u>	No	
<u>SIO2</u>	No	
<u>SIO3</u>	No	
Cn	Yes	Should be programmed low

3**Status Signals**

ZERO = 1 if result = 0

N = 1 if MSB = 1

OVR = 0

C = 1 if carry-out

SN74ACT8832

FUNCTION

Tests the two most significant bits of the MQ register. If they are the same, shifts the number to the left.

DESCRIPTION

This instruction is used to normalize a two's complement number in the MQ register by shifting the number one bit position to the left and filling a zero into the LSB (unless the \overline{SIO} input for that word is low). Data on the S bus is added to the carry, permitting the number of shifts performed to be counted and stored in one of the register files.

The shift and the S bus increment are inhibited whenever normalization is attempted on a number already normalized. Normalization is complete when overflow occurs.

3

SN74ACT8832

Available R Bus Source Operands

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
No	No	No	No

Available S Bus Source Operands (Count)

RF (B5-B0)	DB-Port	MQ Register
Yes	No	No

Available Destination Operands (Count) Shift Operations (Conditional)

RF (C5-C0)	RF (B5-B0)	Y-Port
Yes	No	Yes

ALU	MQ
No	Left

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inactive
$\overline{SIO0}$	No	Passes internally generated end-fill bit.
$\overline{SIO1}$	No	
$\overline{SIO2}$	No	
$\overline{SIO3}$	No	
Cn	Yes	Increments S bus (shift count) if set to one.

3

Status Signals

ZERO = 1 if result = 0

N = 1 if MSB of MQ register = 1

OVR = 1 if MSB of MQ register XOR 2nd MSB = 1

C = 1 if carry-out = 1

SN74ACT8832

EXAMPLE (assumes a 32-bit configuration)

Normalize the number in the MQ register, storing the number of shifts in register 3.

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel EB1- EA EBO	Dest Addr C5-C0	Destination Selects						Cn	CF2- CF0	
					WE3-	SELRF1-	SELRF0	OE3-	OEY3-	OEY0	OE		
0010 0000	XX XXXX	00 0011	X 00	00 0011	0	0000	10	X	X	XXXX	0	1	110

Assume register file 3 holds 00000003 (Hex) and MQ register holds 3699D84E (Hex).

OperandSource 0011 0110 1001 1001 1101 1000 0100 1110 MQ shifter \leftarrow MQ registerDestination 0110 1101 0011 0011 1011 0000 1001 1100 MQ register \leftarrow MQ shifter**Count**Source 0000 0000 0000 0000 0000 0000 0000 0011 S \leftarrow RF(3)Destination 0000 0000 0000 0000 0000 0000 0000 0100 RF(3) \leftarrow S + Cn

FUNCTION

Performs arithmetic right shift on result of ALU operation specified in lower nibble of instruction field.

DESCRIPTION

The result of the ALU operation specified in instruction bits I3-I0 is shifted one bit to the right. The sign bit of the most significant byte is retained unless it is inverted as a result of overflow. Bit 0 of the least significant byte is dropped.

The shift may be made conditional on SSF. If SSF is high or floating, the shift result will be sent to the Y MUX. If SSF is low, the ALU result will be passed unshifted to the Y MUX.

*A list of ALU operations that can be used with this instruction is given in Table 15.

Shift Operations

ALU Shifter	MQ Shifter
Arithmetic Right	None

Available Destination Operands (ALU Shifter)

RF (C5-C0)	RF (B5-B0)	Y-Port
Yes	No	Yes

Control/Data Signals

Signal	User Programmable	Use
SSF	Yes	Passes shifted output if high; passes ALU result if low.
SIO0	No	LSB is shifted out from each word, which may be 1, 2, or 4 bytes long depending on selected configuration
SIO1	No	
SIO2	No	
SIO3	No	
Cn	No	Affects arithmetic operation specified in bits I3-I0 of instruction field.

Status Signals[†]

ZERO	= 1 if result = 0
N	= 1 if MSB of result = 1
	= 0 if MSB of result = 0
OVR	= 0
C	= 1 if carry-out condition

[†]C is ALU carry-out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

3

EXAMPLE (assumes a 32-bit configuration)

Perform the computation A = (A + B)/2, where A and B are single-precision numbers. Let A reside in register 1 and B be input via the DB bus.

SN74ACT8832

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel EB1- EA EBO	Dest Addr C5-C0	Destination Selects								Cn	CF2- CFO	SSF	
					WE3-	SELRF1-	SELMO	WE0	SELRFO	OEA	OEB	OEY3-				
0000 0001	00 0001	XX XXXX	0 10	00 0001	0	0000	10	X	X	XXXX	0	0	110	1		

Assume register file 1 holds 6A08C618 (Hex) and DB bus holds 51007530 (Hex).

Source 0110 1010 0000 1000 1100 0110 0001 1000 R ← RF(1)

Source 0101 0001 0000 0000 0111 0101 0011 0000 S ← DB bus

Intermediate[‡]
Result 1011 1011 0000 1001 0011 1011 0100 1000 ALU Shifter ← R + S + Cn

Destination 0101 1101 1000 0100 1001 1101 1010 0100 RF(1) ← ALU shift result

[‡]After the intermediate operation (ADD), overflow has occurred and OVR status signal is set high. When the arithmetic right shift is executed, the sign bit is corrected (see Table 16 for shift definition notes).

FUNCTION

Performs arithmetic right shift on MQ register (LSH) and result of ALU operation (MSH) specified in lower nibble of instruction field.

DESCRIPTION

The result of the ALU operation specified in instruction bits I3-I0 is used as the upper half of a double precision word, the contents of the MQ register as the lower half.

The contents of the ALU are shifted one bit to the right. The sign bit of the most significant byte is retained unless the sign bit is inverted as a result of overflow. Bit 0 of the least significant byte in the ALU shifter is passed to bit 7 of the most significant byte of the MQ register. Bit 0 of the MQ register's least significant byte is dropped.

The shift may be made conditional on SSF. If SSF is high or floating, the shift result will be sent to the Y MUX. If SSF is low, the ALU result will be passed unshifted to the Y MUX.

*A list of ALU operations that can be used with this instruction is given in Table 15.

Shift Operations

ALU Shifter	MQ Shifter
Arithmetic Right	Arithmetic Right

Available Destination Operands (ALU Shifter)

RF (C5-C0)	RF (B5-B0)	Y-Port
Yes	No	Yes

Control/Data Signals

Signal	User Programmable	Use
SSF	Yes	Passes shifted output if high; passes ALU result if low.
SIO0	No	LSB of ALU shifter is passed to MSB of MQ shifter, and LSB of MQ shifter is dropped.
SIO1	No	
SIO2	No	
SIO3	No	
Cn	No	Affects arithmetic operation specified in bits I3-I0 of instruction field.

Status Signals[†]

ZERO	= 1 if result = 0
N	= 1 if MSB of result = 1
	= 0 if MSB of result = 0
OVR	= 0
C	= 1 if carry-out condition

[†]C is ALU carry-out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

3

EXAMPLE (assumes a 32-bit configuration)

Perform the computation $A = (A + B)/2$, where A and B are two's complement numbers. Let A be a double precision number residing in register 1 (MSH) and MQ (LSH). Let B be a single precision number which is input through the DB bus.

SN74ACT8832

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel EB1- EA EB0	Dest Addr C5-C0	Destination Selects								Cn	CF2- CFO	SSF	
					WE3-	SELRF1-	OEY3-		OEY0		OES					
SELMO	WE0	SELRF0	DEA	OEB	OEY0	OES	Cn	CF2- CFO	SSF							
0001 0001	00 0001	XX XXXX	0 10	00 0001	0	0000	10	X	X	XXXX	0	0	110	1		

Assume register file 1 holds 4A08C618 (Hex), and DB bus holds 51007530 (Hex), and MQ register holds 17299A0F (Hex).

MSH

Source 0100 1010 0000 1000 1100 0110 0001 1000 $R \leftarrow RF(1)$

Source 0101 0001 0000 0000 0111 0101 0011 0000 $S \leftarrow DB\ bus$

Intermediate[‡]
Result 1001 1011 0000 1001 0011 1011 0100 1000 $ALU\ Shifter \leftarrow R + S + Cn$

Destination 0100 1101 1000 0100 1001 1101 1010 0100 $RF(1) \leftarrow ALU\ shift\ result$

LSH

Source 0001 0111 0010 1001 1001 1010 0000 1111 $MQ\ shifter \leftarrow MQ\ register$

Destination 0000 1011 1001 0100 1100 1101 0000 0111 $MQ\ register \leftarrow MQ\ shift\ result$

[‡]After the intermediate operation (ADD), overflow has occurred and OVR status signal is set high. When the arithmetic right shift is executed, the sign bit is corrected (see Table 16 for shift definition notes).

FUNCTION

Performs circular right shift on result of ALU operation specified in lower nibble of instruction field.

DESCRIPTION

The result of the ALU operation specified in instruction bits I3-I0 is shifted one bit to the right. Bit 0 of the least significant byte is passed to bit 7 of the most significant byte in the same word, which may be 1, 2, or 4 bytes long depending on the selected configuration.

The shift may be made conditional on SSF. If SSF is high or floating, the shift result will be sent to the Y MUX. If SSF is low, the ALU result will be passed unshifted to the Y MUX.

*A list of ALU operations that can be used with this instruction is given in Table 15.

Shift Operations

ALU Shifter	MQ Shifter
Circular Right	None

Available Destination Operands (ALU Shifter)

RF (C5-C0)	RF (B5-B0)	Y-Port
Yes	No	Yes

Control/Data Signals

Signal	User Programmable	Use
SSF	Yes	Passes shift result if high; passes ALU result if low.
SIO0	No	Rotates LSB to MSB of the same word, which may be 1, 2, or 4 bytes long depending on configuration
SIO1	No	
SIO2	No	
SIO3	No	
Cn	No	Affects arithmetic operation specified in bits I3-I0 of instruction field.

Status Signals[†]

ZERO	= 1 if result = 0
N	= 1 if MSB of result = 1
	= 0 if MSB of result = 0
OVR	= 1 if signed arithmetic overflow
C	= 1 if carry-out condition

[†]C is ALU carry-out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

3

EXAMPLE (assumes a 32-bit configuration)

Perform a circular right shift of register 6 and store the result in register 1.

SN74ACT8832

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel EB1- EA EBO	Dest Addr C5-C0	Destination Selects						CF2- CFO	SSF	
					WE3-	SELRF1-		OEY3-					
					SELMQ	WE0	SELRF0	OE _A	OE _B	OEY0	OE _S		
1000 0110	00 0110	XX XXXX	0 XX	00 0001	0	0000	10	X	X	XXXX	0	0	110 1

Assume register file 6 holds 3788C618 (Hex).

Source 0011 0111 1000 1000 1100 0110 0001 1000 R ← RF(6)

Intermediate
Result 0011 0111 1000 1000 1100 0110 0001 1000 ALU Shifter ← R + Cn

Destination 0001 1011 1100 0100 0110 0011 0000 1100 RF(1) ← ALU shift result

FUNCTION

Performs circular right shift on MQ register (LSH) and result of ALU operation (MSH) specified in lower nibble of instruction field.

DESCRIPTION

The result of the ALU operation specified in instruction bits I3-I0 is used as the upper half of a double precision word, the contents of the MQ register as the lower half.

The contents of the ALU and MQ shifters are rotated one bit to the right. Bit 0 of the least significant byte in the ALU shifter is passed to bit 7 of the most significant byte of the MQ shifter. Bit 0 of the least significant byte is passed to bit 7 of the most significant byte of the ALU shifter.

The shift may be made conditional on SSF. If SSF is high or floating, the shift result will be sent to the Y MXU and MQ register. If SSF is low, the Y MUX and MQ register will not be altered.

*A list of ALU operations that can be used with this instruction is given in Table 15.

Shift Operations

ALU Shifter	MQ Shifter
Circular Right	Circular Right

Available Destination Operands (ALU Shifter)

RF (C5-C0)	RF (B5-B0)	Y-Port
Yes	No	Yes

Control/Data Signals

Signal	User Programmable	Use
SSF	Yes	Passes shift result if high; passes ALU result and retains MQ register if low.
SIO0	No	Rotates LSB of ALU shifter to MSB of MQ shifter,
SIO1	No	and LSB of MQ shifter to MSB of ALU shifter
SIO2	No	
SIO3	No	
Cn	No	Affects arithmetic operation specified in bits I3-I0 of instruction field.

Status Signals[†]

ZERO	= 1 if result = 0
N	= 1 if MSB of result = 1
	= 0 if MSB of result = 0
OVR	= 1 if signed arithmetic overflow
C	= 1 if carry-out condition

[†]C is ALU carry-out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

3

EXAMPLE (assumes a 32-bit configuration)

Perform a circular right double precision shift of the data in register 6 (MSH) and MQ (LSH), and store the result back in register 6 and the MQ register.

SN74ACT8832

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel EB1- EA EBO	Dest Addr C5-C0	Destination Selects								Cn	CF2- CFO	
					WE3- SELMO	WE0- SELMQ	SELRF1- SELRF0	OEA	OEB	OEY3- OEOY0	OEY0	OES			
1001 0110	00 0110	XX XXXX	0 XX	00 0110	0	0000	10	X	X	XXXX	0	0	110		

Assume register file 6 holds 3788C618 (Hex) and MQ register holds 50A99A0F (Hex).

MSH

Source 0011 0111 0000 1000 1100 0110 0001 1000 R \leftarrow RF(6)

Intermediate Result 0011 0111 0000 1000 1100 0110 0001 1000 ALU shifter \leftarrow R + Cn

Destination 1001 1011 1000 0100 0110 0011 0000 1100 RF(6) \leftarrow ALU shift result

LSH

Source 0101 0000 1010 1001 1001 1010 0000 1111 MQ shifter \leftarrow MQ register

Destination 0010 1000 0101 0100 1100 1101 0000 0111 MQ register \leftarrow MQ shift result

FUNCTION

Performs logical right shift on result of ALU operation specified in lower nibble of instruction field.

DESCRIPTION

The result of the ALU operation specified in instruction bits I3-I0 is shifted one bit to the right. A zero is placed in the bit 7 of the most significant byte of each word unless the \overline{SIO} input for the word is programmed low; this will force the sign bit to one. The LSB is dropped from the word, which may be 1, 2, or 4 bytes long depending on selected configuration.

The shift may be made conditional on SSF. If SSF is high or floating, the shift result will be sent to the Y MUX. If SSF is low, the ALU result will be passed unshifted to the Y MUX.

*A list of ALU operations that can be used with this instruction is given in Table 15.

Shift Operations

ALU Shifter	MQ Shifter
Logical Right	None

Available Destination Operands (ALU Shifter)

RF (C5-C0)	RF (B5-B0)	Y-Port
Yes	No	Yes

Control/Data Signals‡

Signal	User Programmable	Use
SSF		Passes shift result if high or floating; passes ALU result if low.
$\overline{SIO0}$	Yes	Fills a zero in MSB of the word if high or floating; fills a one in MSB if low.
$\overline{SIO1}$	Yes	
$\overline{SIO2}$	Yes	
$\overline{SIO3}$	Yes	
Cn		Inactive

‡Cn is ALU carry-out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

EXAMPLE (assumes a 32-bit configuration)

Perform a logical right single precision shift on data on the DA bus, and store the result in register 1.

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel EB1- EA E80	Dest Addr C5-C0	Destination Selects								Cn	CF2- CFO	SIO3- SIO0	IESIO3- IESIO0	SSF
					WE3-	SELRF1-	OEY3-	Cn	CF2-	SIO3- SIO0	IESIO3- IESIO0	SSF					
0010 0110	XX XXXX	XX XXXX	1 XX	00 0001	0	0000	10	X	X	XXXX	0	0	110	XXX1	0000	1	

Assume DA bus holds 2DA8C615.

3

SN74ACT8832

Source 0010 1101 1010 1000 1100 0110 0001 0101 R ← DA bus

Intermediate Result 0010 1101 1010 1000 1100 0110 0001 0101 ALU Shifter ← R + Cn

Destination 0001 0110 1101 0100 0110 0011 0000 1010 RF(1) ← ALU shift result

FUNCTION

Performs logical right shift on MQ register (LSH) and result of ALU operation (MSH) specified in lower nibble of instruction field.

DESCRIPTION

The result of the ALU operation specified in instruction bits I3-I0 is used as the upper half of a double precision word, the contents of the MQ register as the lower half.

The ALU result is shifted one bit to the right. A zero is placed in the sign bit of the most significant byte unless the \overline{SIO} input for that word is programmed low; this will force the sign bit to one. Bit 0 of the least significant byte is passed to bit 7 of the most significant byte of the MQ shifter. Bit 0 of the least significant byte of the MQ shifter is dropped.

The shift may be made conditional on SSF. If SSF is high or floating, the shift result will be sent to the Y MUX and MQ register. If SSF is low, the ALU result and MQ register will not be altered.

*A list of ALU operations that can be used with this instruction is given in Table 15.

Shift Operations

ALU Shifter	MQ Shifter
Logical Right	Logical Right

Available Destination Operands (ALU Shifter)

RF (C5-C0)	RF (B5-B0)	Y-Port
Yes	No	Yes

Control/Data Signals

Signal	User Programmable	Use
SSF	Yes	Passes shift result if high; passes ALU result and retains MQ
$\overline{SIO0}$	Yes	Fills a zero in MSB if high or floating;
$\overline{SIO1}$	Yes	fills a one MSB if low.
$\overline{SIO2}$	Yes	
$\overline{SIO3}$	Yes	
Cn	No	Affects arithmetic operation specified in bits I3-I0 of instruction field.

Status Signals[†]

ZERO	= 1 if result = 0
N	= 1 if MSB of result = 1
	= 0 if MSB of result = 0
OVR	= 1 if signed arithmetic overflow
C	= 1 if carry-out condition

[†]C is ALU carry-out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

3

EXAMPLE (assumes a 32-bit configuration)

Perform a logical right double precision shift of the data in register 1 (MSH) and MQ (LSH), filling a one into the most significant bit, and store the result back in register 1 and the MQ register.

SN74ACT8832

Instr Code 17-10	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel EB1- EA EBO	Dest Addr C5-C0	Destination Selects								CF2- Cn	SIO3- SIO0	IESI03- IESI00
					WE3-	SELRF1-	OEY3-	CF0-							
0011 0110	XX XXXX	00 0001	X 00	00 0001	0	0000	10	X	X	XXXX	0	0	110	1110	0000

Assume register file 1 holds 2DA8C615 (Hex) and MQ register holds 50A99A0E (Hex).

MSH

Source 0010 1101 1010 1000 1100 0110 0001 0101 R \leftarrow RF(1)

Intermediate Result 0010 1101 1010 1000 1100 0110 0001 0101 ALU Shifter \leftarrow S + Cn

Destination 1001 0110 1101 0100 0110 0011 0000 1010 RF(1) \leftarrow ALU shift result

LSH

Source 0101 0000 1010 1001 1001 1010 0000 1110 MQ shifter \leftarrow MQ register

Destination 1010 1000 0101 0100 1100 1101 0000 0111 MQ register \leftarrow MQ shift result

FUNCTION

Subtracts four-bit immediate data on A3-A0 with carry from S-bus data.

DESCRIPTION

Immediate data in the range 0 to 15, supplied by the user at A3-A0, is inverted and added with carry to S.

Available R Bus Source Operands (Constant)

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
No	Yes	No	No

3

SN74ACT8832

Available S Bus Source Operands

RF (B5-B0)	DB-Port	MQ Register
Yes	Yes	Yes

Available Destination Operands Shift Operations

RF (C5-C0)	RF (B5-B0)	Y-Port	ALU	MQ
Yes	No	Yes	None	None

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inactive
SIO0	No	Inactive
SIO1	No	Inactive
SIO2	No	Inactive
SIO3	No	Inactive
Cn	Yes	Two's complement subtraction if programmed high.

Status Signals

ZERO	= 1 if result = 0
N	= 1 if MSB = 1
OVR	= 1 if arithmetic signed overflow
C	= 1 if carry-out

EXAMPLE (assumes a 32-bit configuration)

Subtract the value 12 from data on the DB bus, and store the result into register file 1.

3

SN74ACT18832

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel EB1- <u>EA</u> EBO	Dest Addr C5-C0	Destination Selects								Cn	CF2- CFO
					WE3- SELMQ	WE0- WEO	SELRF1- SELRF0	OE _A	OE _B	OEY3- OEYO	OEY0	OES		
0111 1000	00 1100	XX XXXX	X 10	00 0001	0	0000	10	X	X	XXXX	0	1	110	

Assume bits A3-A0 hold C (Hex) and DB bus holds 24000100 (Hex).

Source 0000 0000 0000 0000 0000 0000 0000 1100 R ← A3-A0

Source 0010 0100 0000 0000 0000 0001 0000 0000 S ← DB bus

Destination 0010 0100 0000 0000 0000 0000 1111 0100 RF(1) ← R' + S + Cn

FUNCTION

Subtracts data on the R bus from S with carry.

DESCRIPTION

Data on the R bus is subtracted with carry from data on the S bus. The result appears at the ALU and MQ shifters.

- * The result of this instruction can be shifted in the same microcycle by specifying a shift instruction in the upper nibble (I7-I4) of the instruction field. The result may also be passed without shift. Possible instructions are listed in Table 15.

Available R Bus Source Operands

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
Yes	No	Yes	No

Available S Bus Source Operands

RF (B5-B0)	DB-Port	MQ Register
Yes	Yes	Yes

Available Destination Operands

RF (C5-C0)	RF (B5-B0)	Y-Port	ALU Shifter	MQ Shifter
Yes	No	Yes	Yes	Yes

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Affect shift instructions programmed in bits I7-I4 of instruction field.
SIO0	No	
SIO1	No	
SIO2	No	
SIO3	No	
Cn	Yes	Two's complement subtraction if programmed high.

Status Signals[†]

ZERO	= 1 if result = 0
N	= 1 if MSB = 1
OVR	= 1 if signed arithmetic overflow
C	= 1 if carry-out

[†]C is ALU carry-out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

EXAMPLE (assumes a 32-bit configuration)

3

Subtract data in register 1 from data on the DB bus, and store the result in the MQ register.

SN74ACT8832

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel EB1- EA EBO	Dest Addr C5-C0	Destination Selects								Cn	CF2- CF0
					WE3- SELMQ	WE0- SELRF1-	SELRF0	OEA	OEB	OEY3- OEY0	OES			
1110 0010	00 0001	XX XXXX	0 10	XX XXXX	1	XXXX	XX	X	X	XXXX	0	1	110	

Assume register file 1 holds 150084D0 (Hex) and DB bus holds 4900C350 (Hex).

Source 0001 0101 0000 0000 1000 0100 1101 0000 R ← RF(1)

Source 0100 1001 0000 0000 1100 0011 0101 0000 S ← DB bus

Destination 0011 0100 0000 0000 0011 1110 1000 0000 MQ register ← R' + S + Cn

FUNCTION

Subtracts data on the S bus from R with carry.

DESCRIPTION

Data on the S bus is subtracted with carry from data on the R bus. The result appears at the ALU and MQ shifters.

*The result of this instruction can be shifted in the same microcycle by specifying a shift instruction in the upper nibble (I7-I4) of the instruction field. The result may also be passed without shift. Possible instructions are listed in Table 15.

Available R Bus Source Operands

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
Yes	No	Yes	No

Available S Bus Source Operands

RF (B5-B0)	DB-Port	MQ Register
Yes	Yes	Yes

Available Destination Operands

RF (C5-C0)	RF (B5-B0)	Y-Port	ALU Shifter	MQ Shifter
Yes	No	Yes	Yes	Yes

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Affect shift instructions programmed in bits I7-I4 of instruction field.
SIO0	No	
SIO1	No	
SIO2	No	
SIO3	No	
Cn	Yes	Two's complement subtraction if programmed high.

Status Signals[†]

ZERO = 1 if result = 0
 N = 1 if MSB = 1
 OVR = 1 if signed arithmetic overflow
 C = 1 if carry-out

[†]C is ALU carry-out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

EXAMPLE (assumes a 32-bit configuration)

3

Subtract data on the DB bus from data in register 1, and store the result in the MQ register.

SN74ACT18832

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel EB1- EA EBO	Dest Addr C5-C0	Destination Selects								Cn	CF2- CFO
					WE3- SELMO	WE0- WEO	SELRF1- SELRF0	OEA	OEB	OEY3- OEYO	OES			
1110 0011	00 0001	XX XXXX	0 10	XX XXXX	1	XXXX	XX	X	X	XXXX	0	1	110	

Assume register file 1 holds 150084D0 (Hex) and DB bus holds 4900C350 (Hex).

Source 0001 0101 0000 0000 1000 0100 1101 0000 R ← RF(1)

Source 0100 1001 0000 0000 1100 0011 0101 0000 S ← DB bus

Destination 1100 1011 1111 1111 1100 0001 1000 0000 MQ register ← R + S' + Cn

FUNCTION

Tests bits in selected bytes of S-bus data for zeros using mask in C3-C0::A3-A0.

DESCRIPTION

The S bus is the source word for this instruction. The source word is passed to the ALU, where it is compared to an 8-bit mask, consisting of a concatenation of the C3-C0 and A3-A0 address ports (C3-C0::A3-A0). The mask is input via the R bus. The test will pass if the selected byte has zeros at all bit locations specified by the ones of the mask. Bytes are selected by programming the \overline{SIO} inputs low. Test results are indicated on the ZERO output, which goes to one if the test passes. Register write is internally disabled during this instruction.

3

Available R Bus Source Operands

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
No	No	No	Yes

Available S Bus Source Operands

RF (B5-B0)	DB-Port	MQ Register
Yes	Yes	Yes

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inactive
$\overline{SIO0}$	Yes	Byte Select
$\overline{SIO1}$	Yes	Byte Select
$\overline{SIO2}$	Yes	Byte Select
$\overline{SIO3}$	Yes	Byte Select
Cn	No	Inactive

SN74ACT8832

Status Signals

ZERO	= 1 if result (selected bytes) = Pass
N	= 0
OVR	= 0
C	= 0

EXAMPLE (assumes a 32-bit configuration)

Test bits 7, 6 and 5 of bytes 0 and 2 of data in register 3 for zeroes.

Instr Code 17-10	Mask (LSH) A3-A0	Oprd Addr B5-B0	Oprd Sel EB1- EA EBO	Mask (MSH) C3-C0	Destination Selects								Cn	CF2- SIO3- SIO0	IESIO3- IESIO0
0011 1000	0000	00 0011	X 00	1110	X	XXXX	XX	X	X	XXXX	0	X	110	1010	0000

Assume register file 3 holds 881CD003 (Hex).

Source R \leftarrow Mask (C3-C0::A3-A0)

Source SN \leftarrow RF(3) n^{\dagger}

t_n = nth byte

Output ZERO \leftarrow 1

FUNCTION

Tests bits in selected bytes of S-bus data for ones using mask in C3-C0::A3-A0.

DESCRIPTION

The S bus is the source word for this instruction. The source word is passed to the ALU, where it is compared to an 8-bit mask, consisting of a concatenation of the C3-C0 and A3-A0 address ports (C3-C0::A3-A0). The mask is input via the R bus. The test will pass if the selected byte has ones at all bit locations specified by the ones of the mask. Bytes are selected by programming the \overline{SIO} inputs low. Test results are indicated on the ZERO output, which goes to one if the test passes. Register write is internally disabled for this instruction.

Available R Bus Source Operands

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
No	No	No	Yes

Available S Bus Source Operands

RF (B5-B0)	DB-Port	MQ Register
Yes	Yes	Yes

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inactive
$\overline{SIO0}$	Yes	Byte Select
$\overline{SIO1}$	Yes	Byte Select
$\overline{SIO2}$	Yes	Byte Select
$\overline{SIO3}$	Yes	Byte Select
Cn	No	Inactive

Status Signals

ZERO	= 1 if result (selected bytes) = Pass
N	= 0
OVR	= 0
C	= 0

EXAMPLE (assumes a 32-bit configuration)

Test bits 7, 6 and 5 of bytes 1 and 2 of data in register 3 for ones.

3

SN74ACT8832

Instr Code 17-10	Mask (LSH) A3-A0	Oprd Addr B5-B0	Oprd Sel EB1- EA EBO	Mask (MSH) C3-C0	Destination Selects								CF2-	SIO3- SIO0	IESI03- IESI00
0010 1000	0000	00 0011	X 00	1110	X	XXXX	XX	X	X	XXXX	0	X	110	1001	0000

Assume register file 3 holds 881CF003 (Hex).

Mask 1110 0000 1110 0000 1110 0000 1110 0000 $R_n \leftarrow \text{Mask } (C3-C0::A3-A0)$

Source 1000 1000 0001 1100 1101 0000 0000 0011 $S_n \leftarrow RF(3)_n^{\dagger}$

Output 0 $ZERO \leftarrow 0$

$\dagger n$ = nth byte

FUNCTION

Performs one of N-2 iterations of nonrestoring unsigned division by a test subtraction of the N-bit divisor from the 2N-bit dividend. An algorithm using this instruction can be found in the "Other Arithmetic Instructions" section.

DESCRIPTION

UDIVI performs a test subtraction of the divisor from the dividend to generate a quotient bit. The test subtraction may pass or fail and is corrected in the subsequent instruction if it fails. Similarly a failed test from the previous instruction is corrected during evaluation of the current UDIVI instruction (see the "Other Arithmetic Instructions" section for more details).

The R bus must be loaded with the divisor, the S bus with the most significant half of the result of the previous instruction (UDIVI during iteration or UDIVIS at the beginning of iteration). The least significant half of the previous result is in the MQ register.

UDIVI checks the result of the previous pass/fail test and then evaluates:

$$\begin{array}{ll} F \leftarrow R + S & \text{if the test is failed} \\ F \leftarrow R' + S + C_n & \text{if the test is passed} \end{array}$$

A double precision left shift is performed; bit 7 of the most significant byte of the MQ shifter is transferred to bit 0 of the least significant byte of the ALU shifter. Bit 7 of the most significant byte of the ALU shifter is lost. The unfixed quotient bit is circulated into the least significant bit of the MQ shifter.

Available R Bus Source Operands

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
Yes	No	Yes	No

Recommended S Bus Source Operands

RF (B5-B0)	DB-Port	MQ Register
Yes	Yes	No

Recommended Destination Operands Shift Operations

RF (C5-C0)	RF (B5-B0)	Y-Port
Yes	No	Yes

ALU	MQ
Left	Left

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inactive
$\overline{SIO0}$	No	Passes internally generated end-fill bit.
SIO1	No	
SIO2	No	
SIO3	No	
Cn	Yes	Should be programmed high.

Status Signals

ZERO = 1 if result = 0
N = 0
OVR = 0
C = 1 if carry-out

FUNCTION

Computes the first quotient bit of nonrestoring unsigned division. An algorithm using this instruction is given in the "Other Arithmetic Instructions" section.

DESCRIPTION

UDIVIS computes the first quotient bit during nonrestoring unsigned division by subtracting the divisor from the dividend. The resulting remainder due to subtraction may be negative; the subsequent UDIVI instruction may have to restore the remainder during the next operation.

The R bus must be loaded with the divisor and the S bus with the most significant half of the remainder. The result on the Y bus should be loaded back into the register file for use in the next instruction. The least significant half of the remainder is in the MQ register.

UDIVIS computes:

$$F \leftarrow R' + S + C_n$$

A double precision left shift is performed; bit 7 of the most significant byte of the MQ shifter is transferred to bit 0 of the least significant byte of the ALU shifter. Bit 7 of the most significant byte of the ALU shifter is lost. The unfixed quotient bit is circulated into the least significant bit of the MQ shifter.

Available R Bus Source Operands

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
Yes	No	Yes	No

Recommended S Bus Source Operands

RF (B5-B0)	DB-Port	MQ Register
Yes	Yes	No

Recommended Destination Operands Shift Operations

RF (C5-C0)	RF (B5-B0)	Y-Port
Yes	No	Yes

ALU	MQ
Left	Left

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inactive
SIO0	No	Passes internally generated end-fill bit.
SIO1	No	
SIO2	No	
SIO3	No	
Cn	Yes	Should be programmed high.

3**Status Signals**

ZERO = 1 if intermediate result = 0
N = 0
OVR = 1 if divide overflow
C = 1 if carry-out

SN74ACT8832

FUNCTION

Solves the final quotient bit during nonrestoring unsigned division. An algorithm using this instruction is given in the "Other Arithmetic Instructions" section.

DESCRIPTION

UDIVIT performs the final subtraction of the divisor from the remainder during nonrestoring signed division. UDIVIT is preceded by N-1 iterations of UDIVI, where N is the number of bits in the dividend.

The R bus must be loaded with the divisor, the S bus must be loaded with the most significant half of the result of the last UDIVI instruction. The least significant half lies in the MQ register. The Y bus result must be loaded back into the register file for use in the subsequent DIVRF instruction.

UDIVIT checks the results of the previous pass/fail test and evaluates:

$$\begin{array}{ll} Y \leftarrow R + S & \text{if the test is failed} \\ Y \leftarrow R' + S + C_n & \text{if the test is passed} \end{array}$$

The contents of the MQ register are shifted one bit to the left; the unfixed quotient bit is circulated into the least significant bit.

3

SN74ACT8832

Available R Bus Source Operands

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
Yes	No	Yes	No

Recommended S Bus Source Operands

RF (B5-B0)	DB-Port	MQ Register
Yes	Yes	No

Recommended Destination Operands Shift Operations

RF (C5-C0)	RF (B5-B0)	Y-Port
Yes	No	Yes

ALU	MQ
None	Left

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inactive
SIO0	No	Passes internally generated end-fill bit.
SIO1	No	
SIO2	No	
SIO3	No	
Cn	Yes	Should be programmed high.

3**Status Signals**

ZERO = 1 if intermediate result = 0

N = 0

OVR = 0

C = 1 if carry-out

SN74ACT8832

FUNCTION

Performs one of N unsigned multiplication iterations for computing an N-bit by N-bit product. An algorithm for unsigned multiplication using this instruction is given in the “Other Arithmetic Instructions” section.

DESCRIPTION

UMULI checks to determine whether the multiplicand should be added with the present partial product. The instruction evaluates:

$$\begin{array}{ll} F \leftarrow R + S + C_n & \text{if the addition is required} \\ F \leftarrow S & \text{if no addition is required} \end{array}$$

A double precision right shift is performed. Bit 0 of the least significant byte of the ALU shifter is passed to bit 7 of the most significant byte of the MQ shifter; carry-out is passed to the most significant bit of the ALU shifter.

The S bus should be loaded with the contents of an accumulator and the R bus with the multiplicand. The Y bus result should be written back to the accumulator after each iteration of UMULI. The accumulator should be cleared and the MQ register loaded with the multiplier before the first iteration.

3

SN74ACT8832

R Bus Source Operands

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
Yes	No	Yes	No

Recommended S Bus Source Operands

RF (B5-B0)	DB-Port	MQ Register
Yes	Yes	No

Recommended Destination Operands Shift Operations

RF (C5-C0)	RF (B5-B0)	Y-Port
Yes	No	Yes

ALU	MQ
Right	Right

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Holds LSB of MQ.
$\overline{SIO0}$	No	Passes internal input (shifted bit).
$\overline{SIO1}$	No	
$\overline{SIO2}$	No	
$\overline{SIO3}$	No	
Cn	Yes	Should be programmed low.

3**Status Signals[†]**

ZERO = 1 if result = 0
N = 1 if MSB = 1
OVR = 0
C = 1 if carry-out

[†]Valid only on final execution of multiply iteration

SN74ACT8832

FUNCTION

Evaluates the logical expression R XOR S.

DESCRIPTION

Data on the R bus is exclusive ORed with data on the S bus. The result appears at the ALU and MQ shifters.

*The result of this instruction can be shifted in the same microcycle by specifying a shift instruction in the upper nibble (I7-I4) of the instruction field. The result may also be passed without shift. Possible instructions are listed in Table 15.

Available R Bus Source Operands

RF (A5-A0)	A3-A0 Immed	DA-Port	C3-C0 :: A3-A0 Mask
Yes	No	Yes	No

Available S Bus Source Operands

RF (B5-B0)	DB-Port	MQ Register
Yes	Yes	Yes

Available Destination Operands

RF (C5-C0)	RF (B5-B0)	Y-Port	ALU Shifter	MQ Shifter
Yes	No	Yes	Yes	Yes

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Affect shift instructions programmed in bits I7-I4 of instruction field.
SIO0	No	
SIO1	No	
SIO2	No	
SIO3	No	
Cn	No	Inactive

Status Signals[†]

ZERO	= 1 if result = 0
N	= 1 if MSB = 1
OVR	= 0
C	= 0

[†]C is ALU carry-out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

EXAMPLE (assumes a 32-bit configuration)

3

Exclusive OR the contents of register 3 and register 5, and store the result in register 5.

SN74ACT8832

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel EB1 EA EBO	Dest Addr C5-C0	Destination Selects								CF2- CFO Cn
					WE3 SELMQ	WE0 WE0	SELRF1- SELRF0	OEA OEA	OEB OEB	OEV3 OEY0	OES OEY0		
1111 1001	00 0011	00 0101	0 00	00 0101	0	0000	10	X	X	XXXX	0	X	110

Assume register file 3 holds 33F6D840 (Hex) and register file 5 holds 90F6D842 (Hex)..

Source 0011 0011 1111 0110 1101 1000 0100 0000 R ← RF(3)

Source 1001 0000 1111 0110 1101 1000 0100 0010 S ← RF(5)

Destination 1010 0011 0000 0000 0000 0000 0000 0010 RF(5) ← R XOR S

Overview**1****SN74ACT8818 16-Bit Microsequencer****2****SN74ACT8832 32-Bit Registered ALU****3****SN74ACT8836 32- × 32-Bit Parallel Multiplier****4****SN74ACT8837 64-Bit Floating Point Processor****5****SN74ACT8841 Digital Crossbar Switch****6****SN74ACT8847 64-Bit Floating Point/Integer Processor****7****Support****8****Mechanical Data****9**

4

SN74ACT8836

SN74ACT8836 32-Bit by 32-Bit Multiplier/Accumulator

The SN74ACT8836 is a 32-bit integer multiplier/accumulator (MAC) that accepts two 32-bit inputs and computes a 64-bit product. An on-board adder is provided to add or subtract the product or the complement of the product from the accumulator.

To speed-up calculations, many modern systems off-load frequently-performed multiply/accumulate operations to a dedicated single-cycle MAC. In such an arrangement, the 'ACT8836 MAC can accelerate 32-bit microprocessors, building block processors, or custom CPUs. The 'ACT8836 is well-suited for digital signal processing applications, including fast fourier transforms, digital filtering, power series expansion, and correlation.

4

SN74ACT8836

SN74ACT8836

32-BIT BY 32-BIT MULTIPLIER/ACCUMULATOR

D3046, JANUARY 1988

- Performs Full 32-Bit by 32-Bit Multiply/Accumulate in Flow-Through Mode in 60 ns (Max)
- Can be Pipelined for 36 ns (Max) Operation
- Performs 64-Bit by 64-Bit Multiplication in Five Cycles
- Supports Division Using Newton-Raphson Approximation
- Signed, Unsigned, or Mixed-Mode Multiply Operations
- EPIC™ (Enhanced-Performance Implanted CMOS) 1- μ m Process
- Multiplier, Multiplicand, and Product Can be Complemented
- Accumulator Bypass Option
- TTL I/O Voltage Compatibility
- Three Independent 32-Bit Buses for Multiplicand, Multiplier, and Product
- Parity Generation/Checking
- Master/Slave Fault Detection
- Single 5-V Power Supply
- Integer or Fractional Rounding

description

The 'ACT8836 is a 32-bit by 32-bit parallel multiplier/accumulator suitable for low-power, high-speed operations in applications such as digital signal processing, array processing, and numeric data processing. High speed is achieved through the use of a Booth and Wallace Tree architecture.

Data is input to the chip through two registered 32-bit DA and DB input ports and output through a registered 32-bit Y output port. These registers have independent clock enable signals and can be made transparent for flowthrough operations.

The device can perform two's complement, unsigned, and mixed-data arithmetic. It can also operate as a 64-bit by 64-bit multiplier. Five clock cycles are required to perform a 64-bit by 64-bit multiplication and multiplex the 128-bit result. Division is supported using Newton-Raphson approximation.

A multiply/accumulate mode is provided to add or subtract the accumulator from the product or the complement of the product. The accumulator is 67 bits wide to accommodate possible overflow. A warning flag (ETPERR) indicates whether overflow has occurred.

A rounding feature in the 'ACT8836 allows the result to be truncated or rounded to the nearest 32-bits. To ensure data integrity, byte parity checking is provided at the input ports, and a parity generator and master/slave error detection comparator are provided at the output port.

The SN74ACT8836 is characterized for operation from 0°C to 70°C.

EPIC is a trademark of Texas Instruments Incorporated

ADVANCE INFORMATION documents contain information on new product in the sampling or preproduction phase of development. Characteristic data and other specifications are subject to change without notice.



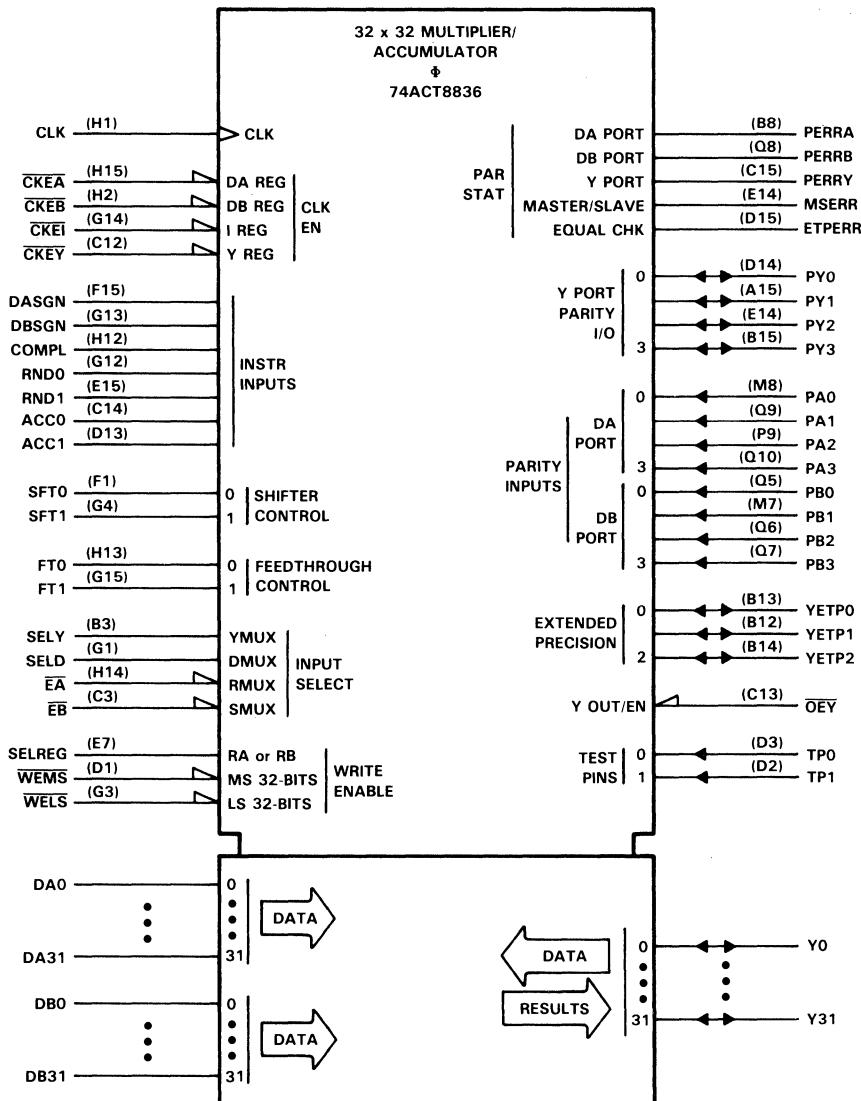
POST OFFICE BOX 655012 • DALLAS, TEXAS 75265

Copyright © 1988, Texas Instruments Incorporated

SN74ACT8836

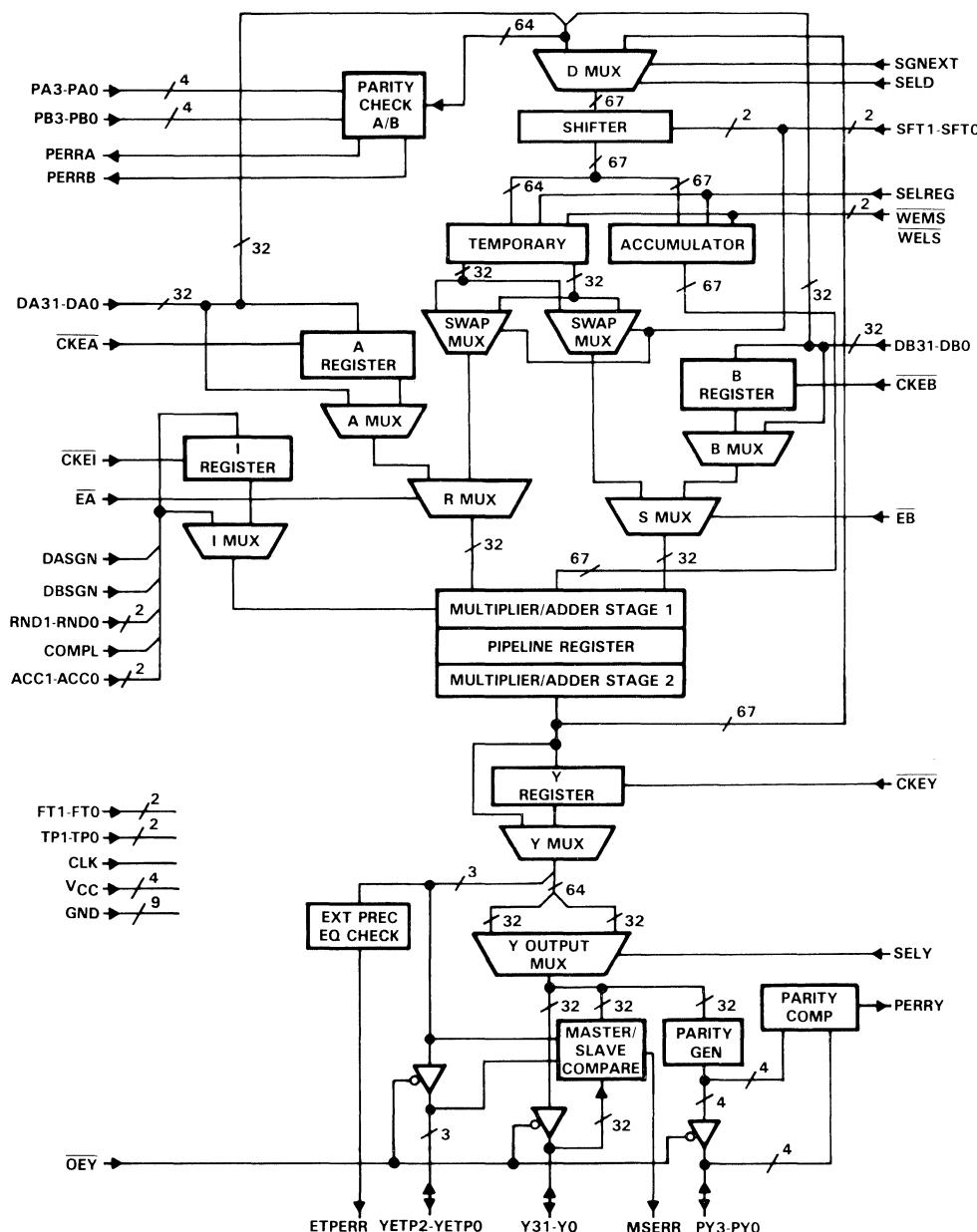
32-BIT BY 32-BIT MULTIPLIER/ACCUMULATOR

logic symbol



SN74ACT8836
32-BIT BY 32-BIT MULTIPLIER/ACCUMULATOR

functional block diagram (positive logic)



SN74ACT8836

32-BIT BY 32-BIT MULTIPLIER/ACCUMULATOR

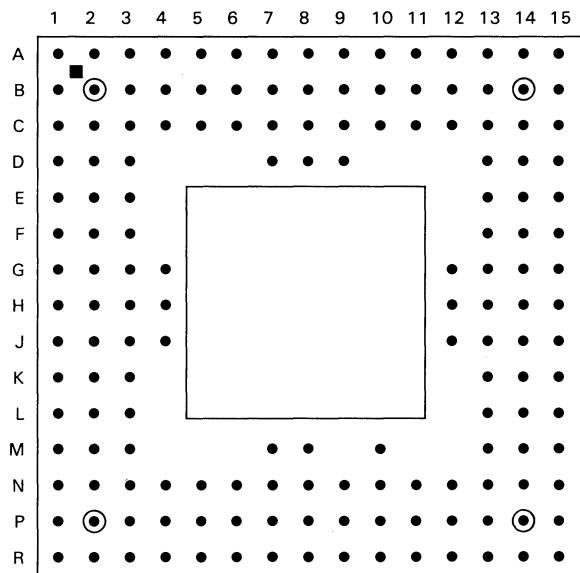
4

SN74ACT8836

ADVANCE INFORMATION

GB PIN-GRID-ARRAY PACKAGE

(TOP VIEW)



GB PACKAGE PIN ASSIGNMENTS

PIN NO.	PIN NAME	PIN NO.	PIN NAME	PIN NO.	PIN NAME	PIN NO.	PIN NAME
A1	Y8	B12	YETP1	D14	PYO	H12	COMPL
A2	Y10	B13	YETP0	D15	ETPERR	H13	FTO
A3	Y11	B14	YETP2	E1	SELREG	H14	EA
A4	Y13	B15	PY3	E2	Y3	H15	CKEA
A5	Y14	C1	Y0	E3	GND	J1	DB2
A6	Y16	C2	Y4	E13	GND	J2	DB3
A7	Y18	C3	EB	E14	PY2	J3	DB5
A8	Y19	C4	Y5	E15	RND1	J4	DB7
A9	Y21	C5	VCC	F1	SFTO	J12	DA26
A10	Y23	C6	GND	F2	Y1	J13	DA24
A11	Y25	C7	Y15	F3	GND	J14	DA30
A12	Y27	C8	GND	F13	GND	J15	DA31
A13	Y28	C9	Y22	F14	MSERR	K1	DB4
A14	Y30	C10	GND	F15	DASGN	K2	DB9
A15	PY1	C11	VCC	G1	SELD	K3	DB11
B1	Y2	C12	CKEY	G2	SGNEXT	K13	DA22
B2	Y6	C13	OEY	G3	WELS	K14	DA28
B3	SELY	C14	ACCO	G4	SFT1	K15	DA29
B4	Y7	C15	PERRY	G12	RND0	L1	DB6
B5	Y9	D1	WE&MS	G13	DBSGN	L2	DB15
B6	Y12	D2	TP1	G14	CKE1	L3	DB13
B7	Y17	D3	TPO	G15	FT1	L13	DA18
B8	Y20	D7	GND	H1	CLK	L14	DA20
B9	Y26	D8	VCC	H2	CKEB	L15	DA27
B10	Y29	D9	Y24	H3	DB0	M1	DB8
B11	Y31	D13	ACC1	H4	DB1	M2	DB17

 **TEXAS
INSTRUMENTS**

POST OFFICE BOX 655012 • DALLAS, TEXAS 75265

SN74ACT8836
32-BIT BY 32-BIT MULTIPLIER/ACCUMULATOR

PIN NAME	PIN NO.	I/O	DESCRIPTION
ACCO	C14	I	Accumulate mode opcode (see Table 2)
ACC1	D13		
CLK	H1	I	System clock
CKEA	H15	I	Clock enable for A register, active low
CKEB	H2	I	Clock enable for B register, active low
CKEI	G14	I	Clock enable for I register, active low
CKEY	C12	I	Clock enable for Y register, active low
COMPL	H12	I	Product complement control; high complements multiplier result, low passes multiplier unaltered to accumulator.
DA0	N9		
DA1	R11		
DA2	P10		
DA3	R12		
DA4	N10		
DA5	R13		
DA6	M10		
DA7	R14		
DA8	P11		
DA9	R15		
DA10	N11		
DA11	P14		
DA12	P12		
DA13	N12		
DA14	P13		
DA15	N13	I	DA port input data bits 0 through 31
DA16	M13		
DA17	M14		
DA18	L13		
DA19	N14		
DA20	L14		
DA21	P15		
DA22	K13		
DA23	N15		
DA24	J13		
DA25	M15		
DA26	J12		
DA27	L15		
DA28	K14		
DA29	K15		
DA30	J14		
DA31	J15		
DASGN	F15	I	Sign magnitude control; high identifies DA input data as two's complement, low identifies DA input data as unsigned

SN74ACT8836
32-BIT BY 32-BIT MULTIPLIER/ACCUMULATOR

4

SN74ACT8836

ADVANCE INFORMATION

PIN NAME	PIN NO.	I/O	DESCRIPTION
DB0	H3		
DB1	H4		
DB2	J1		
DB3	J2		
DB4	K1		
DB5	J3		
DB6	L1		
DB7	J4		
DB8	M1		
DB9	K2		
DB10	N1		
DB11	K3		
DB12	P1		
DB13	L3		
DB14	R1		
DB15	L2		
DB16	P2	I	DB port input data bits 0 through 31
DB17	M2		
DB18	M3		
DB19	N2		
DB20	N3		
DB21	N4		
DB22	P4		
DB23	N5		
DB24	P3		
DB25	P5		
DB26	R2		
DB27	N6		
DB28	R3		
DB29	P6		
DB30	R4		
DB31	P7		
DBSGN	G13	I	Sign magnitude control; high identifies DB input data as two's complement, low identifies DB input data as unsigned.
EA	H14	I	Core multiplier operand select. A high on this signal selects DA register for input on the R bus; a low selects the swap MUX.
EB	C3	I	Core multiplier operand select. A high on this signal selects DB register for input on the S bus; a low selects the swap MUX.
ETPERR	D15	O	Equality check result. A low on this signal indicates that bits 67 through 64 of the core multiplier results are equal to bit 63.
FT0	H13	I	Feedthrough control signals for A, B, I, Pipeline and Y registers (see Table 4).
FT1	G15		

PIN NAME	PIN NO.	I/O	DESCRIPTION
GND	C6		
GND	C8		
GND	C10		
GND	D7		
GND	E3		Ground pins. All ground pins should be used and connected.
GND	E13		
GND	F3		
GND	F13		
GND	N8		
MSERR	F14	O	Master/slave error flag. This signal goes high when the contents of the Y output multiplexer and the value at the external port are not equal.
OEY	C13	I	Y, YETP2-YETPO, and PY3-PYO output enable, active low.
PA0	M8		
PA1	R9		
PA2	P9	I	Parity input data bus for DA input data
PA3	R10		
PB0	R5		
PB1	M7		
PB2	R6	I	Parity input data bus for DB input data
PB3	R7		
PY0	D14		
PY1	A15		
PY2	E14	I/O	Y output parity data bus. Outputs data from parity generator ($\overline{OEY} = L$) or inputs external parity data ($\overline{OEY} = H$).
PY3	B15		
PERRA	P8	O	DA port parity status pin. Goes high if even-parity test on any byte fails.
PERRB	R8	O	DB port parity status pin. Goes high if even-parity test on any byte fails.
PERRY	C15	O	Y port parity status pin. Goes high if even-parity test on any byte fails.
RNDO	G12	I	Multiplier/accumulator rounding control; high rounds integer result; low leaves result unaltered.
RND1	E15	I	Multiplier/accumulator rounding control; high rounds fractional result; low leaves result unaltered.
SELD	G1	I	D multiplexer select. High selects DA and DB ports; low selects multiplier core output.
SELREG	E1	I	Write enable for temporary register and accumulator. High enables the temporary register; low enables the accumulator.
SELY	B3	I	Y multiplexer select. High selects most significant 32 bits of Y register output; low selects least significant 32 bits.
SGNEXT	G2	I	Sign extend control for multiplexer. A low fills shift matrix bits 66-64 with zeros; a high fills DA31 in bits 66-64.
SFT0	F1	I	Shift multiplexer control (see Table 4).
SFT1	G4		
TPO	D3		
TP1	D2	I	Test pins (see Table 5)
VCC	C5		
VCC	C11		
VCC	D8		
VCC	N7		
WEMS	D1	I	Write enable for most significant 32 bits of temporary register and accumulator active low.
WELS	G3	I	Write enable for least significant 32 bits of temporary register and accumulator active low.

SN74ACT8836

32-BIT BY 32-BIT MULTIPLIER/ACCUMULATOR

4

SN74ACT8836

ADVANCE INFORMATION

PIN NAME	PIN NO.	I/O	DESCRIPTION
Y0	C1		
Y1	F2		
Y2	B1		
Y3	E2		
Y4	C2		
Y5	C4		
Y6	B2		
Y7	B4		
Y8	A1		
Y9	B5		
Y10	A2		
Y11	A3		
Y12	B6		
Y13	A4		
Y14	A5		
Y15	C7	I/O	Y port data bus. Outputs data from Y register ($\overline{OEY} = L$); inputs data to master/slave comparator ($\overline{OEY} = H$).
Y16	A6		
Y17	B7		
Y18	A7		
Y19	A8		
Y20	B8		
Y21	A9		
Y22	C9		
Y23	A10		
Y24	D9		
Y25	A11		
Y26	B9		
Y27	A12		
Y28	A13		
Y29	B10		
Y30	A14		
Y31	B11		
YETPO	B13	I/O	Data bus for extended precision product. Outputs three most significant bits of the 67-bit multiplier core result; inputs external data to master/slave comparator.
YETP1	B12		
YETP2	B14		

TABLE 1. INSTRUCTION INPUTS

Signal	High	Low
DASGN	Identifies DA Input data as two's complement	Identifies DA input data as unsigned
DBSGN	Identifies DB input data as two's complement	Identifies DB input data as unsigned
RND0	Rounds integer result	Leaves integer result unaltered
RND1	Rounds fractional result	Leaves fractional result unaltered
COMPL	Complements the product from the multiplier before passing it to the accumulator	Passes the product from the multiplier to the accumulator unaltered
ACCO ACC1	See Table 2	See Table 2

SN74ACT8836

32-BIT BY 32-BIT MULTIPLIER/ACCUMULATOR

TABLE 2. MULTIPLIER/ADDER CONTROL INPUTS

ACC1	ACCO	E _A	E _B	Operation
0	0	X	X	$\pm(R \times S) + 0$
0	1	X	X	$\pm(R \times S) + ACC$
1	0	X	X	$\pm(R \times S) - ACC$
1	1	0	0	$\pm 1 \times 1 + 0$
1	1	0	1	$\pm 1 \times DB + 0$
1	1	1	0	$\pm DA \times 1 + 0$
1	1	1	1	$\pm DA \times DB + 0$

ACC is the data stored in the accumulator

TABLE 3. SHIFTER CONTROL INPUTS

SFT1	SFT0	Shifter Operation
L	L	Pass data without shift
L	H	Shift one bit left; fill with zero
H	L	Swap upper and lower halves of temporary register
H	H	Shift 32 bits right; fill with sign bit

TABLE 4. FLOWTHROUGH CONTROL INPUTS

Control Inputs FT1 FTO	Registers Bypassed				
	Pipeline	Y	I	A	B
L L	Yes	Yes	Yes	Yes	Yes
L H	Yes	No	No	No	No
H L	Yes	Yes	No	No	No
H H	No	No	No	No	No

TABLE 5. TEST PIN CONTROL INPUTS

TP1	TP0	Operation
L	L	All outputs and I/Os forced low
L	H	All outputs and I/Os forced high
H	L	All outputs placed in a high impedance state
H	H	Normal operation (default state)

data flow

Two 32-bit input data ports, DA and DB, are provided for input of the multiplicand and multiplier to registers A and B and the multiplier/adder. Input data can be clocked to the A and B registers before being passed to the multiplier/adder if desired. Two multiplexers, R and S, in conjunction with a flowthrough decoder select the multiplier operands from DA and DB inputs, A and B registers, or the temporary register. Data is supplied to the temporary register from a shifter that operates on external DA/DB data or a previous multiplier/adder result. The 67-bit multiplier/adder result can be output through the Y port or passed through the shifter to the accumulator.

External DA and DB data is also available to the accumulator via the shifter. This 64-bit data can be extended with zeros or the sign bit. The 64 least significant bits from the shifter may also be latched in the 64-bit temporary register and input to the multiplier through the R and S multiplexers. A swap option allows the most significant and least significant 32-bit halves of temporary register data to be swapped before being made available to the R and S multiplexers. This allows either 32-bit half of the temporary register to be used as a multiplier.

SN74ACT8836

32-BIT BY 32-BIT MULTIPLIER/ACCUMULATOR

architectural elements

Included in the functional block diagram of the 'ACT8836 are the following blocks.

1. Two 32-bit registered input data ports DA and DB
2. A parity checker at the DA and DB inputs
3. An instruction decoder (I register)
4. A flowthrough decoder that permits selected registers to be bypassed to support up to three levels of pipelining
5. R and S multiplexers to select operands for the multiplier/ adder from DA and DB inputs, registers A and B, or temporary register
6. A D multiplexer that selects the operand for the shifter from the 67-bit sign-extended DA and DB inputs or the multiplier/adder output
7. A shifter block that operates on DA/DB input data or on multiplier/adder outputs for scaling or Newton-Raphson division
8. A Y output multiplexer that selects the most significant half or the least significant half of the multiplier/ adder result for output at the registered Y port
9. An extended precision error check that tests for overflow
10. A master/slave comparator and parity generator/comparator at the Y output port for master/slave and parity checking
11. Registers at the external data and instruction input ports and the shifter and multiplier/adder output port to support pipe-lining

4

input data parity checker

An even-parity check is performed on each byte of input data at the DA, DB and Y ports. If the parity test fails for any byte, a high appears at the parity error output pin (PERRA for DA data, PERRB for DB data, PERRY for Y data).

A and B registers

Register A can be loaded with data from the DA bus, which normally holds a 32-bit multiplicand. Register B is loaded from the DB bus which holds a 32-bit multiplier. Separate clock enables, CKEA and CKEB, allow the registers to be loaded separately. This is useful when performing double precision multiplication or using the temporary register as an input to the multiplier/adder. The registers can be made transparent using the FT inputs (see Table 4).

instruction register

Instruction inputs to the device are shown in Table 1. These signals control signed, unsigned, and mixed multiplication modes, fractional and integer rounding, accumulator operations and complementing of products. They can be latched into instruction register I when clock enable CKEI is low.

Sign control inputs DASGN and DBSGN identify DA and DB input data as signed (high) or unsigned (low).

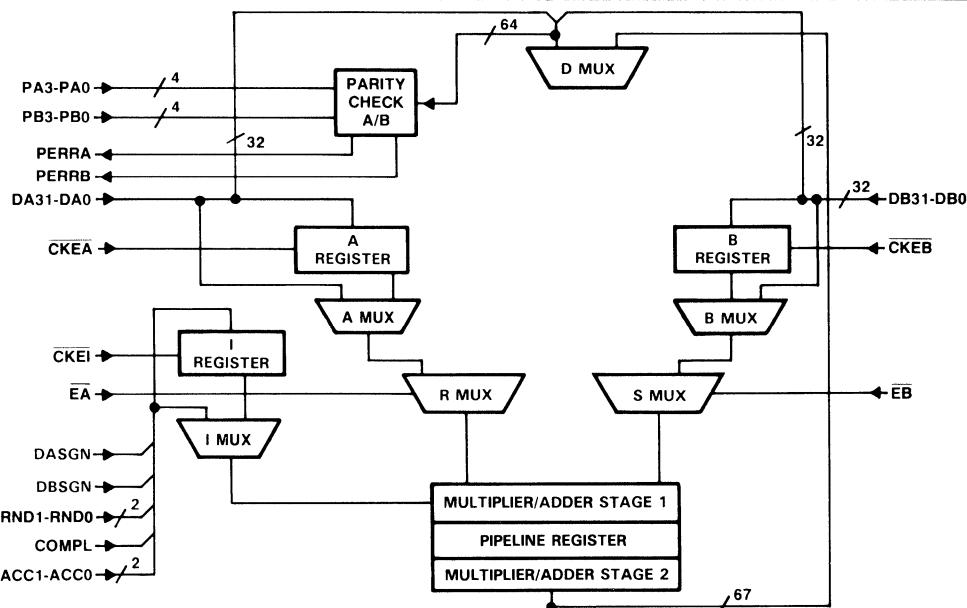
Rounding inputs RND0 and RND1 control rounding operations in the multiplier/adder. A low on these inputs passes the results unaltered. If a high appears on RND1, the result will be rounded by adding a one to bit 30. RND1 should be set high if the multiplier/adder result is to be shifted in order to maintain precision of the least significant bit following the shift operation. If a high appears on RND0, the result will be rounded by adding a one to bit 31. This code should be used when the adder result will not be shifted.

A complement control, COMPL, is used to complement the product from the multiplier before passing it to the accumulator. The complement will occur if COMPL is high; the product will be passed unaltered if COMPL is low.

ACC1-ACCO control the operation of the multiplier/adder. Possible operations are shown in Table 2.

SN74ACT8836

ADVANCE INFORMATION



INPUT REGISTERS AND PARITY CHECK

R, S, and swap multiplexers

The R and S multiplexers select the multiplier/adder operands from external data or from the temporary register.

When \overline{EA} is low, the R multiplexer selects data from the swap multiplexer. When \overline{EA} is high, the R multiplexer selects data from DA or the A register, depending on the state of the flowthrough control inputs (see Table 4). When \overline{EB} is low, the S multiplexer selects data from the swap multiplexer. When \overline{EB} is high, the S multiplexer switches data from DB or the B register, depending on the state of the flowthrough control inputs.

\overline{EA} and \overline{EB} are also used in conjunction with the multiplier/adder control inputs to force a numeric one on the R or S inputs (see Table 2).

The swap multiplexers are controlled by the shifter control inputs. When SFT1 is high and SFT0 is low, the most significant half of the temporary register is available to the S multiplexer, and the least significant half is available to the R multiplexer. When SFT1-SFT0 are set to other values, the most significant half of the temporary register is available to the R multiplexer, and the least significant half is available to the S multiplexer.

multiplier/adder

The multiplier performs 32-bit multiplication and generates a 67-bit product. The product can be latched in the pipeline to increase cycle speed. The product is complemented when COMPL is set high as shown in Table 1. The adder computes the sum or the difference of the accumulator and the product and gives a 67-bit sum. Bits 66-64 are used for overflow and sign extension.

SN74ACT8836

32-BIT BY 32-BIT MULTIPLIER/ACCUMULATOR

D multiplexer

The D multiplexer selects input data for the shifter. Two sources are available to the multiplexer: a 64-bit word formed by concatenating DA and DB bus data, and the 67-bit sum from the multiplier/adder. If SELD is high, external DA/DB data is selected; if SELD is low, the sum is selected.

If the 64-bit word is selected for input to the shifter, three bits are added to the word based on the state of the sign extend signal (SGNEXT). If SGNEXT is low, bits 66-64 are zero-filled; if SGNEXT is high, bits 66-64 are filled with the value on DA31.

temporary register and accumulator (Figure 1)

Output from the shifter will be stored in the temporary register if SELREG is high and in the accumulator register if SELREG is low. The 64-bit temporary register can be used to store temporary data, constants and scaled binary fractions.

Separate clock controls, \overline{WELS} and \overline{WEMS} , allow the most significant and least significant halves of the shifter output to be loaded separately. The 32 least significant bits of the selected register are loaded when \overline{WELS} is low; the most significant bits when \overline{WEMS} is low. When \overline{WELS} and \overline{WEMS} are both low, the entire word from the shifter is loaded into the selected register.

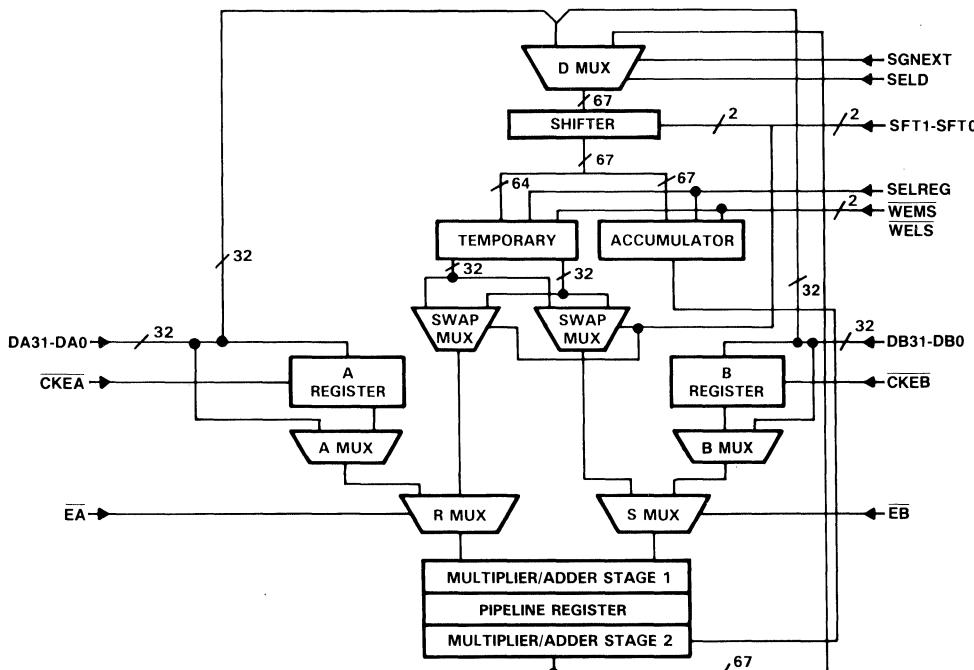


FIGURE 1. TEMPORARY REGISTER AND ACCUMULATOR

shifter

The shifter can be used to multiply by two for Newton-Raphson operations or perform a 32-bit shift for double precision multiplication. The shifter is controlled by two SFT inputs, as shown in Table 3.

Y register

Final or intermediate multiplier/adder results will be clocked into Y register when \overline{CKEY} is low.

Results can be passed directly to the Y output multiplexer using flowthrough decoder signals to bypass the register (see Table 4).

Y multiplexer and Y output multiplexer

The Y multiplexer allows the 64-bit result or the contents of the Y register to be switched to the Y bus, depending upon the state of the flowthrough control outputs. The upper 32 bits are selected for output when the Y output multiplexer control SELY is high; the lower 32 bits are selected for output when SELY is low. Note that the Y output multiplexer can be switched at twice the clock rate so that the 64-bit result can be output in one clock cycle.

flowthrough decoder

To enable the device to operate in pipelined or flowthrough modes, on-chip registers can be bypassed using flowthrough control signals FT1 and FT0. Up to three levels of pipeline can be supported, as shown in Table 4.

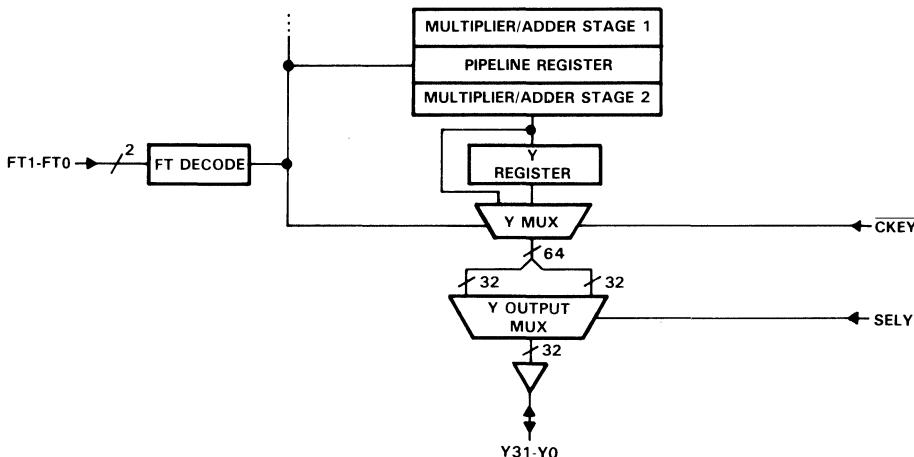


FIGURE 2. Y OUTPUT

SN74ACT8836

32-BIT BY 32-BIT MULTIPLIER/ACCUMULATOR

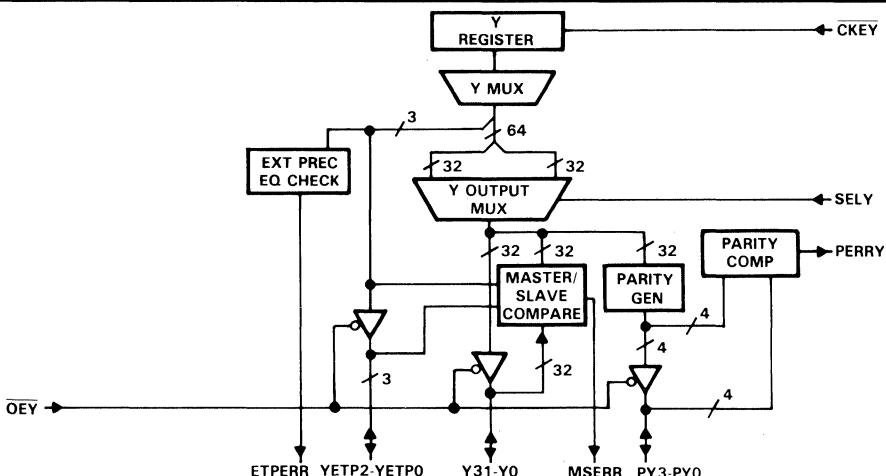


FIGURE 3. OUTPUT ERROR CONTROL

4

SN74ACT8836

ADVANCE INFORMATION

extended precision check

Three extended product outputs, YETP2-YETP0, are provided to recover three bits of precision during overflow. An extended precision check error signal (ETPERR) goes high whenever overflow occurs. If sign controls DASGN and DBSGN are both low, indicating an unsigned operation, the extended precision bits 66-64 are compared for equality. Under all other sign control conditions, bits 66-63 are compared for equality.

master slave comparator

A master/slave comparator is provided to compare data bytes from the Y output multiplexer with data bytes on the external Y port when OEV is high. A comparison of the three extended precision bits of the multiplier/adder result or Y register output with external data in the YETP1-YETP0 port is performed simultaneously. If the data is not equal, a high signal is generated on the master slave error output pin (MSERR). A similar comparison is performed for parity using the PY3-PY0 inputs. This feature is useful in fault-tolerant design where several devices vote to ensure hardware integrity.

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 5).

data formats

The 'ACT8836 performs single-precision and double-precision multiplication in two's complement, unsigned magnitude, and mixed formats for both integer and fractional numbers.

Input formats for the multiplicand (R) and multiplier (S) are given below, followed by output formats for the fully extended product. The fully extended product (PRDT) is 67 bits wide. It includes the extended product (XTP) bits YETP1-YETP0, the most significant product (MSP) bits Y63-Y32, and the least significant product (LSP) bits Y31-Y0.

This can be represented in notational form as follows:

$$\begin{aligned} \text{PRDT} &= \text{XTP} :: \text{MSP} :: \text{LSP} \\ \text{or} \\ \text{PRDT} &= \text{YETP}2 - \text{YETP}0 :: \text{Y}63 - \text{Y}0 \end{aligned}$$

Table 6 shows the output formats generated by two's complement, unsigned and mixed-mode multiplications.

TABLE 6. GENERATED OUTPUT FORMATS

	Two's Complement	Unsigned Magnitude
Two's Complement	Two's Complement	Two's Complement
Unsigned Magnitude	Two's Complement	Unsigned Magnitude

examples

Representative examples of single-precision multiplication, double-precision multiplication, and division using Newton-Raphson binary division algorithm are given below.

single-precision multiplication

Microcode for the multiplication of two signed numbers is shown in Figure 1. In this example, the result is rounded and the 32 most significant bits are output on the Y bus. A second instruction (SELY = 0) would be required to output the least significant half if rounding were not used.

Unsigned and mixed mode single-precision multiplication are executed using the same code. (The sign controls must be modified accordingly.) Following are the input and output formats for signed, unsigned, and mixed mode operations.

Two's Complement Integer Inputs									
Input Operand A						Input Operand B			
Input Operand A						Input Operand B			
31	30	29	2	1	0	31	30	29
-2 ³¹	2 ³⁰	2 ²⁹	2 ²	2 ¹	2 ⁰	-2 ³¹	2 ³⁰	2 ²⁹
(Sign)							2 ²	2 ¹	2 ⁰

Unsigned Integer Inputs									
Input Operand A						Input Operand B			
Input Operand A						Input Operand B			
31	30	29	2	1	0	31	30	29
2 ³¹	2 ³⁰	2 ²⁹	2 ²	2 ¹	2 ⁰	2 ³¹	2 ³⁰	2 ²⁹
							2 ²	2 ¹	2 ⁰

Two's Complement Fractional Inputs									
Input Operand A						Input Operand B			
Input Operand A						Input Operand B			
31	30	29	2	1	0	31	30	29
-2 ⁰	2 ⁻¹	2 ⁻²	2 ⁻²⁹	2 ⁻³⁰	2 ⁻³¹	-2 ⁰	2 ⁻¹	2 ⁻²
(Sign)							2 ⁻²⁹	2 ⁻³⁰	2 ⁻³¹

SN74ACT8836

32-BIT BY 32-BIT MULTIPLIER/ACCUMULATOR

4

SN74ACT8836

ADVANCE INFORMATION

Unsigned Fractional Inputs									
Input Operand A					Input Operand B				
31 30 29 2 1 0					31 30 29 2 1 0				
2-1 2-2 2-3 2-30 2-31 2-32					2-1 2-2 2-3 2-30 2-31 2-32				
Two's Complement Integer Outputs									
Extended Product (YETP2-YETP0)			Most Significant Product (Y63-Y32)				Least Significant Product (Y31-Y0)		
66 65 64 $\underbrace{-266 \quad 265 \quad 264}_{\text{(Sign)}}$			63 62 61 30 31 32 263 262 261 234 233 232				31 30 29 2 1 0 231 230 229 2 ² 2 ¹ 2 ⁰		
Unsigned Integer Outputs									
Extended Product (YETP2-YETP0)			Most Significant Product (Y63-Y32)				Least Significant Product (Y31-Y0)		
66 65 64 266 265 264			63 62 61 30 31 32 263 262 261 234 233 232				31 30 29 2 1 0 231 230 229 2 ² 2 ¹ 2 ⁰		
Two's Complement Fractional Outputs									
Extended Product (YETP2-YETP0)			Most Significant Product (Y63-Y32)				Least Significant Product (Y31-Y0)		
66 65 64 $\underbrace{-24 \quad 2^3 \quad 2^2}_{\text{(Sign)}}$			63 62 61 30 31 32 2 ¹ 2 ⁰ 2 ⁻¹ 2-28 2-29 2-30				31 30 29 2 1 0 2-31 2-32 2-33 2-60 2-61 2-62		
Unsigned Fractional Outputs									
Extended Product (YETP2-YETP0)			Most Significant Product (Y63-Y32)				Least Significant Product (Y31-Y0)		
66 65 64 2 ² 2 ¹ 2 ⁰			63 62 61 30 31 32 2 ⁻¹ 2 ⁻² 2 ⁻³ 2-30 2-31 2-32				31 30 29 2 1 0 2-33 2-34 2-35 2-62 2-63 2-64		

double-precision multiplication

To simplify discussion of double-precision multiplication, the following example implements an algorithm using one 'ACT8836 device. It should be noted that even higher speeds can be achieved through the use of two 'ACT8836s to implement a parallel multiplier.

The example is based on the following algorithm where A and B are 64-bit signed numbers.

Let

$A_m = a_8, a_{62}, a_{61}, \dots, a_{32}$
and

$A_l = a_{31}, a_{30}, a_{29}, \dots, a_0$ (a_0 = LSB)

Therefore:

$$A = (A_m \times 2^{32}) + A_l$$

Likewise:

$$B = (B_m \times 2^{32}) + B_l$$

Thus:

$$\begin{aligned} A \times B &= [(A_m \times 2^{32}) + A_l] \times [(B_m \times 2^{32}) + B_l] \\ &= (A_m \times B_m) 2^{64} + (A_m \times B_l + A_l \times B_m) 2^{32} + A_l \times B_l \end{aligned}$$

Therefore, four products and three summations with rank adjustments are required.

Basic implementation of this algorithm uses a single 'ACT8836. The result is a two's complement 128-bit product. Microcode signals to implement the algorithm are shown in Figure 4.

The first instruction cycle computes the first product, $A_l \times B_l$. The least significant half of the result is output through the Y port for storage in an external RAM or some other 32-bit register; this will be the least significant 32-bit portion of the final result.

The instruction also uses the shifter to shift the $A_l \times B_l$ product 32 bits to the right in order to adjust for ranking in the next multiplication-addition sequence. The least significant half of the shift result is stored in the lower 32-bit portion of the accumulator; the upper 32 bits contain the zero and fill.

The second instruction produces the second product, $A_l \times B_m$, adds it to the contents of the accumulator, and stores the result in the accumulator for use in the third instruction.

Instruction 3 computes $A_m \times B_l$, adds the result to the accumulator, and outputs the least significant 32 bits of the addition for use as bits 63-32 of the final product.

This instruction also shifts the result 32 bits to the right to provide the necessary rank adjustment and stores the shift result (the most significant half of the addition result) in the lower 32 bits of the accumulator. Bits ACC63-ACC32 are filled with zeros; the sign is extended into the three upper bits (ACC66-ACC64).

Instruction 4 computes the fourth product ($A_m \times B_m$), adds it to the accumulator, and outputs the least significant half at the Y port for use as bits 95-64 of the final product.

This example assumes that the chip is operating in feed-through mode. A fifth instruction is therefore required to perform the fourth iteration again so that bits 127-96 of the final product can be output.

SN74ACT8836 ADVANCE INFORMATION

Example 1. Single Precision Multiply, 32-Bit Result

Operand Select R bus S bus EA EB	Instruction Inputs					D-MUX Select	Sign Extend	Shift-MUX Control	Register Load Select	Register Write Enable	Feed-through Control	Clock Enables				Y-MUX Select	Y/PY Output Enable
	Sign	Rounding Control	Product Complement	Multiplier/ Adder Mode	ACC1 ACC0							CKEI CKEA	CKEB CKEY	SELY	OEY		
1 1	1 1	0 1	0	0 0	0	X	0 0	0	0 0	0 0	0 0	0 0	0 0	0	1	0	

Example 2. Double-Precision Multiply, 64-Bit Result

Instruction Number	Operand Select R bus S bus EA EB	Instruction Inputs					D-MUX Select	Sign Extend	Shift-MUX Control	Register Load Select	Register Write Enable	Feed-through Control	Clock Enables				Y-MUX Select	Y/PY Output Enable
		Sign	Rounding Control	Product Complement	Multiplier/ Adder Mode	ACC1 ACC0							CKEI CKEA	CKEB CKEY	SELY	OEY		
(1)	1 1	0 0	0 0	0	0 0	0	0	0	1 1	0	0 0	0 0	1 1	1 1	1	0	0	
(2)	1 1	0 1	0 0	0	0 1	0	0	0	0 0	0	0 0	0 0	1 1	1 1	1	X	0	
(3)	1 1	1 0	0 0	0	0 1	0	0	1	1 1	0	0 0	0 0	1 1	1 1	1	0	0	
(4)	1 1	1 1	0 0	0	0 1	X	0	0	0 0	X	X X	0 0	1 1	1 1	1	0	0	
(5)	1 1	1 1	0 0	0	0 1	X	0	0	0 0	X	X X	0 0	1 1	1 1	1	1	0	

Example 3. Newton-Raphson Division

Instruction Number	Operand Select R bus S bus EA EB	Instruction Inputs					D-MUX Select	Sign Extend	Shift-MUX Control	Register Load Select	Register Write Enable	Feed-through Control	Clock Enables				Y-MUX Select	Y/PY Output Enable
		Sign	Rounding Control	Product Complement	Multiplier/ Adder Mode	ACC1 ACC0							CKEI CKEA	CKEB CKEY	SELY	OEY		
Repeat N Times*																		
(1)	0 1	0 0	0 0	0	0 0	0	0	0	1 1	1	1 0	1 0	0 0	0 0	0	1	0	
(2)	0 0	0 0	0 1	0	0 0	0	0	0	0 1	1	0 1	1 0	0 0	0 0	0	1	0	
End Repeat																		
(3)	0 1	0 0	0 0	0	0 0	0	0	0	1 1	1	1 0	1 0	0 0	0 0	0	1	0	
(4)	0 0	0 0	0 0	0	0 0	0	0	0	0 1	1	0 1	1 0	0 0	0 0	0	1	0	

*N = $\frac{32}{2m+1}$ Where m = number of bits in the seed (assuming 32-bits of precision)

FIGURE 4. MICROCODED EXAMPLES

Newton-Raphson binary division algorithm

The following explanation illustrates how to implement the Newton-Raphson binary division algorithm using the 'ACT8836 multiplier/accumulator. The Newton-Raphson algorithm is an iterative procedure that generates the reciprocal of the divisor through a convergence method.

Consider the equation $Q = A/B$. This equation can be rewritten as $Q = A \times (1/B)$. Therefore, the quotient Q can be computed by simply multiplying the dividend A by the reciprocal of the divisor (B). Finding the divisor reciprocal $1/B$ is the objective of the Newton-Raphson algorithm.

To calculate $1/B$ the Newton-Raphson equation, $X_{i+1} = X_i(2-BX_i)$ is calculated in an iterative process. In the equation, B represents the divisor and X represents successively closer approximations to the reciprocal $1/B$. The following sequence of computation illustrates the iterative nature of the Newton-Raphson algorithm.

$$\begin{aligned}\text{Step 1} \quad X_1 &= X_0(2-BX_0) \\ \text{Step 2} \quad X_2 &= X_1(2-BX_1) \\ \text{Step 3} \quad X_3 &= X_2(2-BX_2) \\ \text{Step } n \quad X_n &= X_{n-1}(2-BX_{n-1})\end{aligned}$$

The successive approximation of X_i , for all i , approaches the reciprocal $1/B$ as the number of iterations increases; that is

$$\lim_{i \rightarrow n} X_i = 1/B$$

The iterative operation is executed until the desired tolerance or error is reached. The required accuracy for $1/B$ can be determined by subtracting each x_i from its corresponding $x_i + 1$. If the difference $|X_{i+1} - X_i|$ is less than or equal to a predetermined round off error, then the process is terminated. The desired tolerance can also be achieved by executing a fixed number of iterations based on the accuracy of the initial guess of $1/B$ stored in RAM of PROM.

The initial guess, X_0 , is called the seed approximation. The seed must be supplied to the Newton-Raphson process externally and must fall within the range of $0 < X_0 < 2/B$ if B is greater than 0 or $0 < X_0 < 2/B$ if B is less than 0.

To perform the Newton-Raphson binary division algorithm using the 'ACT8836, the divisor, B , must be a positive fraction. As a positive fraction, B is limited within the range of $1/2 \leq B < 1$.

Since X_i from Newton-Raphson must lie between $0 < X_i < 2/B$ and since the range of the positive fraction B is $1/2 \leq B < 1$, then the limits of X_i become $1 \leq X_i < 2$.

The range of $-BX_i$ will therefore be $-2 \leq -BX_i \leq -1/2$.

The limits of $-BX_i$ are shown in Table 7 as they would appear in the 'ACT8836 extended bit, binary fraction format.

TABLE 7. LIMITS OF $-BX_i$ IN 'ACT8836 EXTENDED BIT FORMAT

	Extended Bits 66 65 64			63	62	61	2	1	0
-2	1	1	1	0	0	0	0	0	0
$-\frac{1}{2}$	1	1	1	1	1	0	0	0	0

The diagram indicates that $-BX_i$ is always of the form:

$$1\ 1\ 1\ d_0 .\ d_1\ d_2 d_{n-2}\ d_{n-1}$$

SN74ACT8836

32-BIT BY 32-BIT MULTIPLIER/ACCUMULATOR

The next step in Newton-Raphson is to complete the $2 - BX_i$ equation. The fractional representation of 2 is:

$$0\ 0\ 1\ 0\ .\ 0\ 0\0\ 0$$

Completion of the $2 - BX_i$ equation is shown in Table 8.

TABLE 8. COMPLETION OF $2 - BX_i$ EQUATION

Extended Bits 66 65 64			63	62	61	1	0	
1	1	1	d_0	d_1	d_2	d_{n-2}	d_{n-1}	
+	0	0	1	0	0	0	0	
=	0	0	0	d_0	d_1	d_2	d_{n-2}	d_{n-1}

Since this step only affects the extended bits (66-64) on the 'ACT8836, this step can be skipped. The following algorithm can therefore be used to perform Newton-Raphson binary division with the 'ACT8836.

Assuming B is on the DB bus (or stored in the B register) and X_i is stored in the temporary register:

Step 1

$$\begin{aligned} \text{Accumulator} &\leftarrow -(DB \times \text{temporary register}) \\ &= 2 - BX_i \end{aligned}$$

Step 2

$$\begin{aligned} \text{Temporary Register} &\leftarrow \text{Left shift one bit of} \\ &(\text{accumulator times temporary register}) \\ &= X_i + 1 \\ &= X_i (2 - BX_i) \end{aligned}$$

Step 3

Repeat Steps 1 and 2 until $|X_i + 1 - X_i| \leq$ a predetermined round-off error

Two cycles are required for each iteration. The left shift that is performed in Step 2 is required to realign X_i after the signed fraction multiply. Microcode for this example is shown in Figure 4.

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 = 0 to V _{CC})	±50 mA
Continous 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

		MIN	NOM	MAX	UNIT
V _{CC}	Supply voltage	4.5	5	5.5	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	
d _{t/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			T _A = 0°C to 70°C		UNIT
			MIN	TYP	MAX	MIN	MAX	
V _{OH}	I _{OH} = -20 μA	4.5 V	4.4			4.4		V
		5.5 V	5.4			5.4		
	I _{OH} = -8 mA	4.5 V	3.8			3.7		V
		5.5 V	4.8			4.7		
V _{OL}	I _{OL} = 20 μA	4.5 V		0.1		0.1		V
		5.5 V		0.1		0.1		
	I _{OL} = 8 mA	4.5 V		0.32		0.4		V
		5.5 V		0.32		0.4		
I _I	V _I = V _{CC} or 0	5.5 V		0.1	±1.0			μA
I _{CC}	V _I = V _{CC} or 0, I _O	5.5 V		50		100		μA
C _I	V _I = V _{CC} or 0	5 V		5	10		10	pF
ΔI _{CC} [‡]	One input at 3.4 V, other inputs at 0 or V _{CC}	5.5 V		1		1		mA
I _{OZH}	V _I = V _{CC} or 0	5 V		0.5		5		μA
I _{OZL}	V _I = V _{CC} or 0	5 V	-0.5			-5		μA

[†] This is the increase in supply current for each input that is at one of the specified TTL voltage levels rather than 0 or V_{CC}.

SN74ACT8836

32-BIT BY 32-BIT MULTIPLIER/ACCUMULATOR

setup and hold times

PARAMETER	MIN	MAX	UNIT
t _{su1} Instruction before CLK↑	14		
t _{su2} Data before CLK↑	12		
t _{su3} CKEA before CLK↑	14		
t _{su4} CKEB before CLK↑	14		
t _{su5} CKEI before CLK↑	10		
t _{su6} CKEY before CLK↑	19		
t _{su7} SELREG before CLK↑	12		
t _{su8} WEMS before CLK↑	11		
t _{su9} WELS before CLK↑	11		
t _{h1} Instruction after CLK↑	0		
t _{h2} Data after CLK↑	0		
t _{h3} CKEA after CLK↑	0		
t _{h4} CKEB after CLK↑	0		
t _{h5} CKEI after CLK↑	0		
t _{h6} CKEY after CLK↑	0		
t _{h7} SELREG after CLK↑	0		
t _{h8} WEMS after CLK↑	0		
t _{h9} WELS after CLK↑	0		

4

SN74ACT8836

ADVANCE INFORMATION

SN74ACT8836
32-BIT BY 32-BIT MULTIPLIER/ACCUMULATOR

switching characteristics over recommended ranges of supply voltage and free-air temperature (see Figure 2) for load circuit and voltage waveforms)

PARAMETER	FROM (INPUT)	TO (OUTPUT)	FT MODE (FT1—FT0)	MIN	TYP	MAX	UNIT
t_{pd1}^{\dagger}	CLK	PIPE	11			36	ns
t_{pd2}^{\dagger}	PIPE	Y REG	11			36	
t_{pd3}^{\dagger}	PIPE	ACCUM	11			36	
t_{pd4}^{\dagger}	Y REG	Y	All modes			18	
t_{pd5}	SELY	Y	All modes			18	
t_{pd6}^{\dagger}	CLK	Y REG	01			54	
t_{pd7}^{\dagger}	CLK	ACCUM	10 or 01			67	
t_{pd8}	CLK	Y	10			67	
t_{pd9}	DATA	Y	00			60	
t_{pd10}^{\dagger}	DATA	ACCUM	00			56	
t_{pd11}	CLK	YETP	11 or 10			18	
t_{pd12}	CLK	ETPERR	11 or 10			18	
t_{pd13}	CLK	YETP	00			67	
t_{pd14}	CLK	ETPERR	01			67	
t_{pd15}	DATA	YETP	00			60	
t_{pd16}	DATA	ETPERR	00			60	
t_{pd17}	PA	PERRA	All modes			20	
t_{pd18}	DA	PERRA	All modes			20	
t_{pd19}	PB	PERRB	All modes			20	
t_{pd20}	DB	PERRB	All modes			20	
t_{pd21}	PY	PERRY	All modes			20	
t_{pd22}	Y	MSERR	All modes			22	
t_{pd23}	YETP	MSERR	All modes			22	
t_{en2}	OEY	YETP	All modes			20	
t_{en1}	OEY	Y	All modes			20	
t_{dis1}	OEY	YETP	All modes			15	
t_{dis2}	OEY	Y	All modes			15	

clock requirements

PARAMETER	SN74ACT8836		UNIT
	MIN	MAX	
t_{w1} CLK high		5	
t_{w2} CLK low		20	ns

[†]These parameters cannot be measured but can be inferred from device operation and other measurable parameters.

SN74ACT8836

32-BIT BY 32-BIT MULTIPLIER/ACCUMULATOR

PARAMETER MEASUREMENT INFORMATION

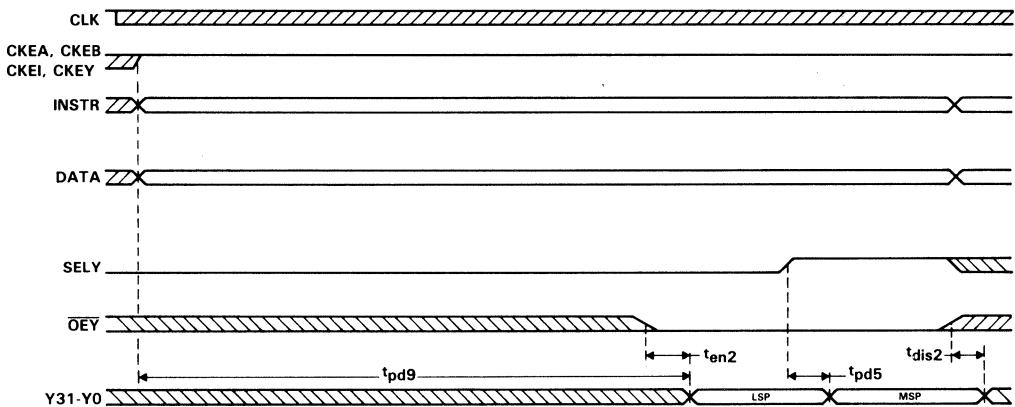


FIGURE 5. FULL FLOWTHROUGH MODE (FT = 00)

4

SN74ACT8836

ADVANCE INFORMATION

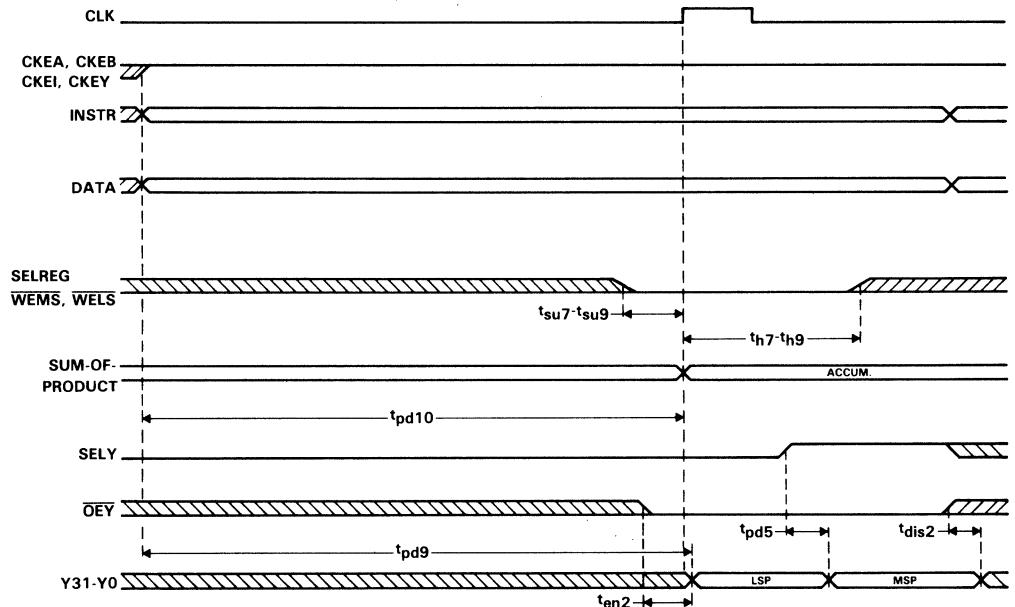


FIGURE 6. FULL FLOWTHROUGH MODE, ACCUMULATOR MODE (FT = 00)

PARAMETER MEASUREMENT INFORMATION

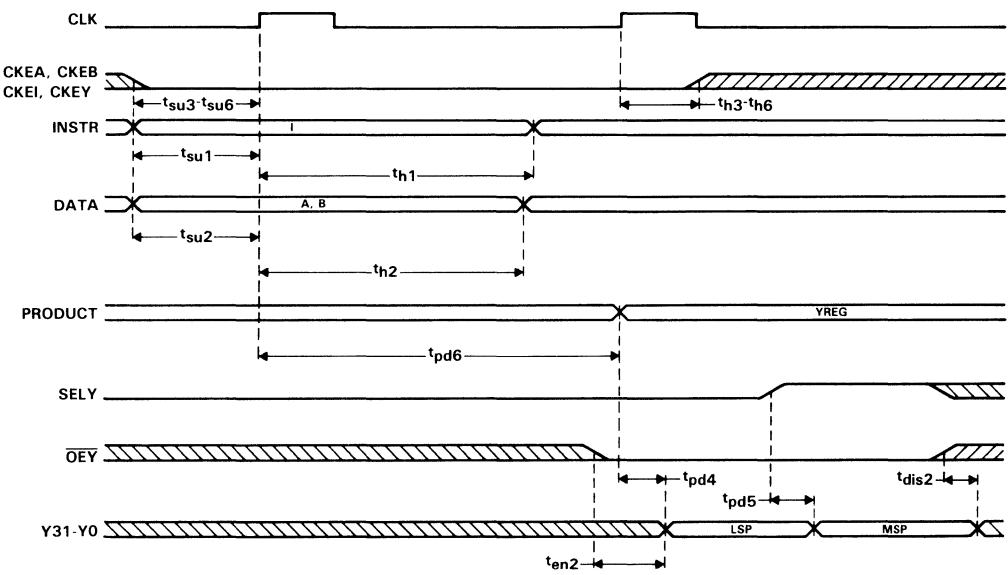


FIGURE 7. FLOWTHROUGH PIPE ONLY VOLTAGE WAVEFORMS (FT = 01)

SN74ACT8836
32-BIT BY 32-BIT MULTIPLIER/ACCUMULATOR

PARAMETER MEASUREMENT INFORMATION

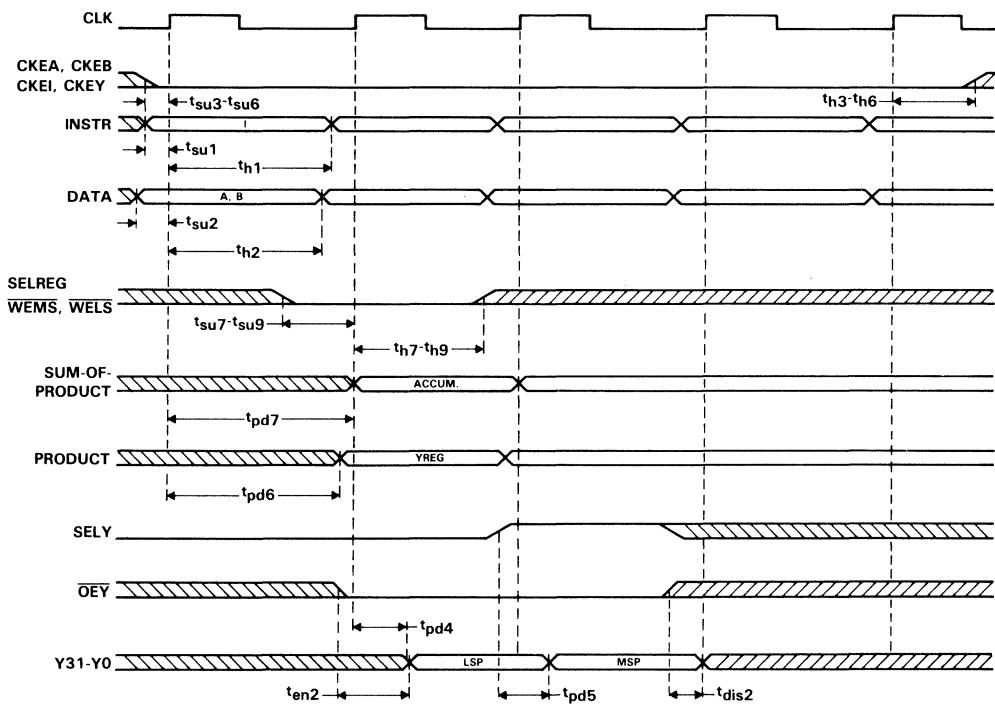


FIGURE 8. FLOWTHROUGH PIPE ONLY, ACCUMULATOR MODE (FT = 01)

PARAMETER MEASUREMENT INFORMATION

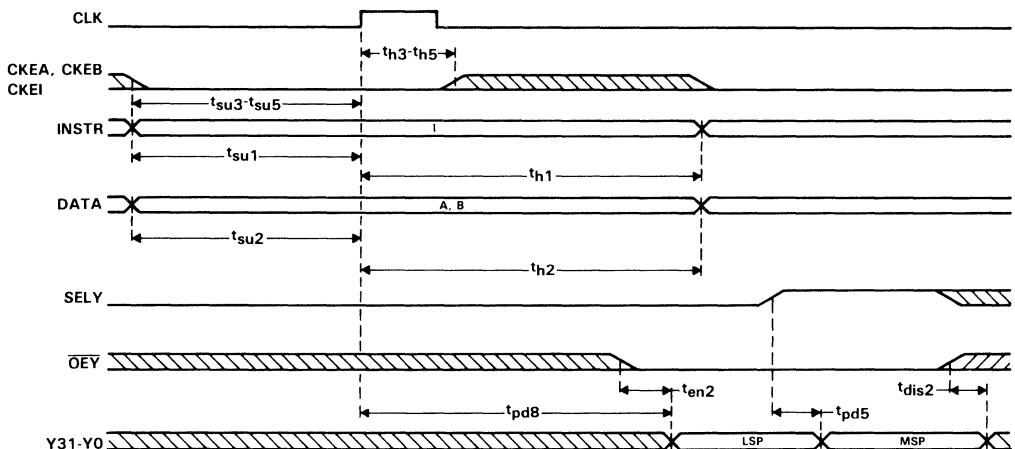


FIGURE 9. FLOWTHROUGH PIPE AND Y ONLY (FT = 10)

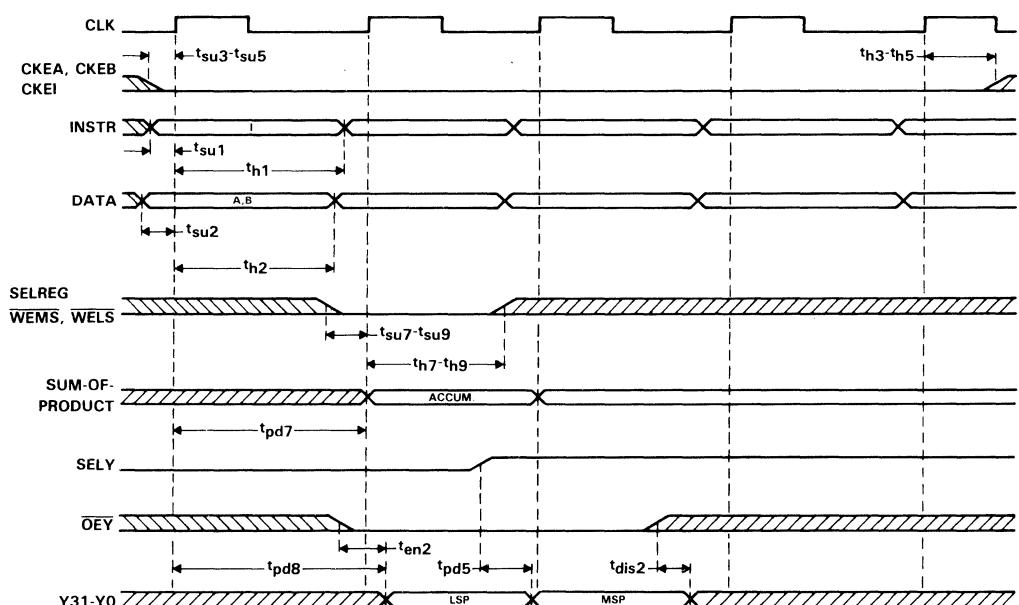


FIGURE 10. FLOWTHROUGH PIPE AND Y ONLY, ACCUMULATOR MODE (FT = 10)

SN74ACT8836
32-BIT BY 32-BIT MULTIPLIER/ACCUMULATOR

PARAMETER MEASUREMENT INFORMATION

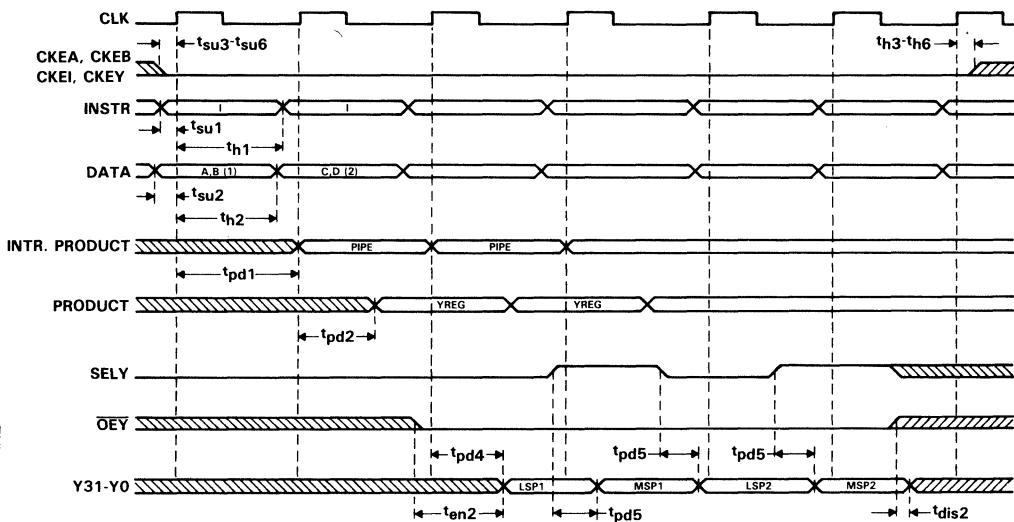


FIGURE 11. ALL REGISTERS ENABLED (FT = 11)

4

SN74ACT8836

ADVANCE INFORMATION

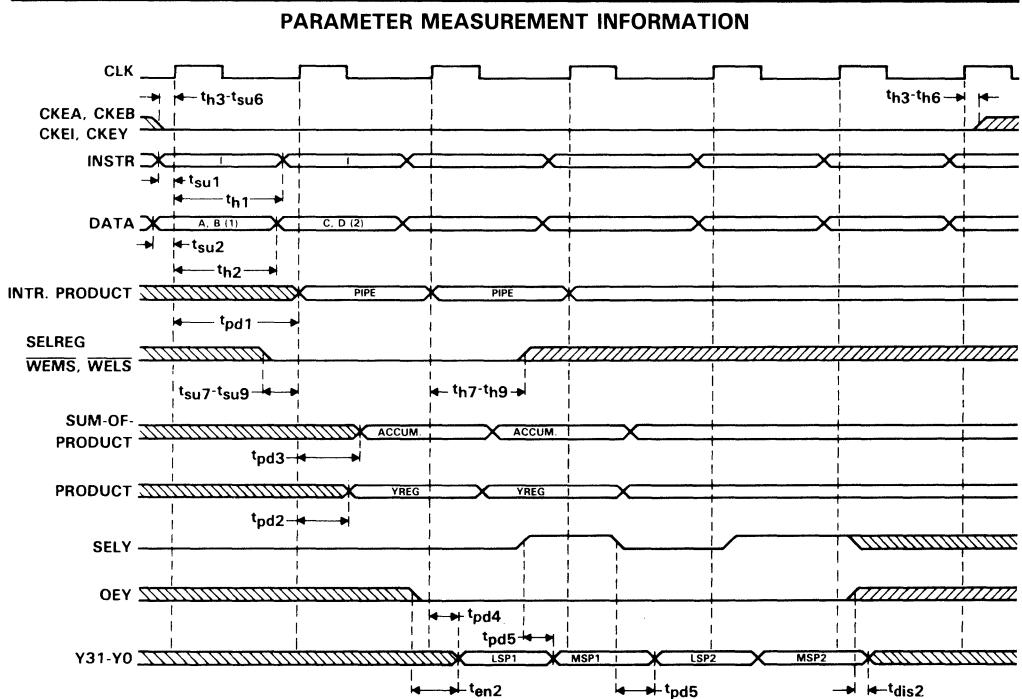
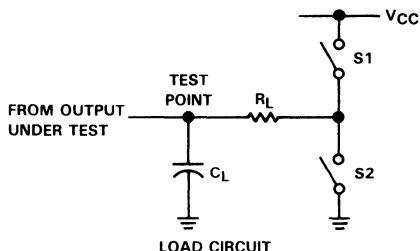


FIGURE 12. ALL REGISTERS ENABLED, ACCUMULATOR MODE (FT = 11)



PARAMETER	R_L	C_L^{\dagger}	$S_1 z$	$S_2 z$
t_{en}	t_{PZH}	1 k Ω	50 pF	OPEN
	t_{PLZ}			CLOSED
t_{dis}	t_{PHZ}	1 k Ω	50 pF	OPEN
	t_{PLZ}			CLOSED
t_{pd}	-	50 pF	OPEN	OPEN

[†] C_L includes probe and test fixture capacitance

All input pulses are supplied by generators having the following characteristics: PRR \leq 1 MHz, Z_{out} = 50 Ω , t_r = 50 Ω , t_f = 6 ns.

FIGURE 13. LOAD CIRCUIT

4

SN74ACT8836

Overview**1****SN74ACT8818 16-Bit Microsequencer****2****SN74ACT8832 32-Bit Registered ALU****3****SN74ACT8836 32- × 32-Bit Parallel Multiplier****4****SN74ACT8837 64-Bit Floating Point Processor****5****SN74ACT8841 Digital Crossbar Switch****6****SN74ACT8847 64-Bit Floating Point/Integer Processor****7****Support****8****Mechanical Data****9**

5

SN74ACT8837

SN74ACT8837

64-Bit Floating Point Unit

- Multiplier and ALU in One Chip
- 65-ns Pipelined Performance
- Low-Power EPIC™ CMOS
- Meets IEEE Standard for 32- and 64-Bit Multiply, Add, and Subtract
- Three-Port Architecture, 64-Bit Internal Bus
- Pipelined or Flowthrough Operation
- Floating Point-to-Integer and Integer-to-Floating Point Conversions
- Supports Division Using Newton-Raphson Algorithm
- Parity Generation/Checking

5

The SN74ACT8837 single-chip floating point processor performs high-speed 32- and 64-bit floating point operations. More than just a coprocessor, the 'ACT8837 integrates on one chip, two double-precision floating point functions, an ALU and multiplier.

The wide dynamic range and high precision of floating point format minimize the need for scaling and overflow detection. Computationally-intense applications, such as high-end graphics and digital signal processing, need double-precision floating point accuracy to maintain data integrity. Floating point processors in general-purpose computing must often support double-precision formats to match existing software.

By integrating its two functions on one chip, the 'ACT8837 reduces data routing problems and processing overhead. Its three data ports and 64-bit internal bus structure let the user load two operands and take a result in a single clock cycle.

SN74ACT8837

EPIC is a trademark of Texas Instruments Incorporated.

5

SN74ACT8837

Contents

	<i>Page</i>
Introduction	5-13
Understanding the 'ACT8837 Floating Point Unit	5-13
Microprogramming the 'ACT8837	5-13
Support Tools	5-14
Design Support	5-14
Systems Expertise	5-15
'ACT8837 Logic Symbol	5-16
'ACT8837 Pin Descriptions	5-17
'ACT8837 Specification Tables	5-24
SN74ACT8837 Floating Point Unit	5-27
Data Flow	5-27
Input Data Parity Check	5-27
Temporary Input Register	5-29
RA and RB Input Registers	5-29
Multiplier/ALU Multiplexers	5-30
Pipelined ALU	5-31
Pipelined Multiplier	5-31
Product, Sum, and C Registers	5-31
Parity Generators	5-31
Master/Slave Comparator	5-34
Status and Exception Generator/Register	5-34
Flowthrough Mode	5-37
Fast and IEEE Modes	5-37
Rounding Mode	5-38
Test Pins	5-38
Summary of Control Inputs	5-38

Contents (Continued)

	<i>Page</i>
Instruction Set	5-40
Loading External Data Operands	5-40
Configuration Controls (CONFIG1-CONFIG0)	5-40
CLKMODE Settings	5-40
Internal Register Operations	5-41
Data Register Controls (PIPES2-PIPES0)	5-41
C Register Controls (SRCC, CLKC)	5-42
Operand Selection (SELOP7-SELOP0)	5-42
Rounding Controls (RND1-RND0)	5-43
Status Exceptions	5-43
Handling of Denormalized Numbers (FAST)	5-45
Data Output Controls (SELMS/LS, OEY)	5-47
Status Output Controls (SELST1-SELST0, OES, OEC)	5-47
Stalling the Device (HALT)	5-47
Instruction Inputs (I9-I0)	5-48
Independent ALU Operations	5-48
Independent Multiplier Operations	5-48
Chained Multiplier/ALU Operations	5-51
Microprogramming the 'ACT8837	5-52
Single-Precision Operations	5-52
Single-Precision ALU Operations	5-52
Single-Precision Multiplier Operations	5-52
Sample Single-Precision Microinstructions	5-53
Double-Precision Operations	5-58
Double-Precision ALU Operations	5-58
Double-Precision ALU Operations with CLKMODE=0	5-60
Double-Precision ALU Operations with CLKMODE=1	5-66
Double-Precision Multiplier Operations	5-73
Double-Precision Multiplication with CLKMODE=0	5-73
Double-Precision Multiplication with CLKMODE=1	5-79
Chained Multiplier/ALU Operations	5-86
Fully Pipelined Double-Precision Operations	5-87
Mixed Operations and Operands	5-90
Matrix Operations	5-92
Representation of Variables	5-92
Sample Matrix Transformation	5-93
Microinstructions for Sample Matrix Manipulation	5-100

Contents (Concluded)

	<i>Page</i>
Sample Microprograms for Binary Division and Square Root	5-105
Binary Division Using the Newton-Raphson Algorithm	5-105
Single-Precision Newton-Raphson Binary Division	5-108
Double-Precision Newton-Raphson Binary Division	5-111
Binary Square Root Using the Newton-Raphson Algorithm	5-114
Single-Precision Square Root Using a Double-Precision Seed ROM	5-114
Double-Precision Square Root	5-117
Glossary	5-123
Implementing a Double-Precision Seed ROM	5-124

5

SN74ACT8837

List of Illustrations

<i>Figure</i>		<i>Page</i>
1	'ACT8837 Floating Point Unit	5-28
2	Single-Precision Operation, All Registers Disabled (PIPES = 111, CLKMODE = 0)	5-53
3	Single-Precision Operation, Input Registers Enabled (PIPES = 110, CLKMODE = 0)	5-54
4	Single-Precision Operation, Input and Output Registers Enabled (PIPES = 010, CLKMODE = 0)	5-55
5	Single-Precision Operation, All Registers Enabled (PIPES = 000, CLKMODE = 0)	5-57
6	Double-Precision ALU Operation, All Registers Disabled (PIPES = 111, CLKMODE = 0)	5-59
7	Double-Precision ALU Operation, Input Registers Enabled (PIPES = 110, CLKMODE = 0)	5-61
8	Double-Precision ALU Operation, Input and Output Registers Enabled (PIPES = 010, CLKMODE = 0)	5-63
9	Double-Precision ALU Operation, All Registers Enabled (PIPES = 000, CLKMODE = 0)	5-65
10	Double-Precision ALU Operation, All Registers Disabled (PIPES = 111, CLKMODE = 1)	5-67
11	Double-Precision ALU Operation, Input Registers Enabled (PIPES = 110, CLKMODE = 1)	5-68
12	Double-Precision ALU Operation, Input and Output Registers Enabled (PIPES = 010, CLKMODE = 1)	5-70
13	Double-Precision ALU Operation, All Registers Enabled (PIPES = 000, CLKMODE = 1)	5-72
14	Double-Precision Multiplier Operation, All Registers Disabled (PIPES = 111, CLKMODE = 0)	5-74
15	Double-Precision Multiplier Operation, Input Registers Enabled (PIPES = 110, CLKMODE = 0)	5-75
16	Double-Precision Multiplier Operation, Input and Output Registers Enabled (PIPES = 010, CLKMODE = 0)	5-76
17	Double-Precision Multiplier Operation, All Registers Enabled (PIPES = 000, CLKMODE = 0)	5-78
18	Double-Precision Multiplier Operation, All Registers Disabled (PIPES = 111, CLKMODE = 1)	5-80

List of Illustrations (Concluded)

<i>Figure</i>		<i>Page</i>
19	Double-Precision Multiplier Operation, Input Registers Enabled (PIPES = 110, CLKMODE = 1)	5-81
20	Double-Precision Multiplier Operation, Input and Output Registers Enabled (PIPES = 010, CLKMODE = 1)	5-83
21	Double-Precision Multiplier Operation, All Registers Enabled (PIPES = 000, CLKMODE = 1)	5-85
22	Mixed Operations and Operands (PIPES2-PIPES0 = 110, CLKMODE = 0)	5-91
23	Mixed Operations and Operands (PIPES2-PIPES0 = 000, CLKMODE = 1)	5-92
24	Sequence of Matrix Operations	5-96
25	Resultant Matrix Transformation	5-103
26	IEEE Double-Precision Seed ROM for Newton-Raphson Division and Square Root	5-125

List of Tables

<i>Table</i>	<i>Page</i>
1 'ACT8837 Pin Grid Allocations	5-17
2 'ACT8837 Pin Functional Description	5-18
3 Double-Precision Input Data Configuration Modes	5-29
4 Single-Precision Input Data Configuration Mode	5-30
5 Double-Precision Input Data Register Sources	5-30
6 Multiplier Input Selection	5-30
7 ALU Input Selection	5-30
8 Independent ALU Operations, Single Operand (I9 = 0, I6 = 0)	5-32
9 Independent ALU Operations, Two Operands (I9 = 0, I5 = 0)	5-33
10 Independent Multiplier Operations (I9 = 0, I6 = 1)	5-33
11 Independent Multiplier Operations Selected by I4-I2 (I9 = 0, I6 = 1)	5-34
12 Operations Selected by I8-I7 (I9 = 0, I6 = 1)	5-34
13 Chained Multiplier/ALU Operations (I9 = 1)	5-35
14 Comparison Status Outputs	5-36
15 Status Outputs	5-36
16 Status Output Selection (Chain Mode)	5-37
17 Pipeline Controls (PIPES2-PIPES0)	5-37
18 Rounding Modes	5-38
19 Test Pin Control Inputs	5-38
20 Control Inputs	5-39
21 IEEE Floating-Point Representations	5-44
22 Handling Wrapped Multiplier Outputs	5-46
23 Independent ALU Operations with One Operand	5-49
24 Independent ALU Operations with Two Operands	5-50
25 Independent Multiplier Operations	5-50
26 Chained Multiplier/ALU Operations	5-51
27 Single-Precision Sum of Products (PIPES2-PIPES0 = 010)	5-86
28 Sample Microinstructions for Single-Precision Sum of Products	5-87

List of Tables (Concluded)

<i>Table</i>		<i>Page</i>
29	Pseudocode for Fully Pipelined Double-Precision Sum of Products (CLKM = 0, CONFIG = 10, PIPES = 000, CLKC \longleftrightarrow SYSCLK)	5-88
30	Pseudocode for Fully Pipelined Double-Precision Product of Sums (CLKM = 0, CONFIG = 10, PIPES = 000, CLKC \longleftrightarrow SYSCLK)	5-89
31	Microinstructions for Sample Matrix Manipulation	5-101
32	Single-Precision Matrix Multiplication (PIPES2-PIPES0 = 010)	5-102
33	Fully Pipelined Sum of Products (PIPES2-PIPES0 = 000)	5-104
34	Sample Data Values and Representations	5-106
35	Binary Division Using the Newton-Raphson Algorithm . .	5-107
36	Single-Precision Newton-Raphson Binary Division	5-109
37	Double-Precision Newton-Raphson Binary Division	5-111
38	Single-Precision Binary Square Root	5-115
39	Double-Precision Binary Square Root	5-118

Introduction

Each of these floating point units (FPU), the SN74ACT8837 combines a multiplier and an arithmetic-logic unit in a single microprogrammable VLSI device. The 'ACT8837 is implemented in Texas Instruments one-micron CMOS technology to offer high speed and low power consumption in an FPU with exceptional flexibility and functional integration. The FPU can be microprogrammed to operate in multiple modes to support a variety of floating point applications.

The 'ACT8837 is fully compatible with the IEEE standard for binary floating point arithmetic, STD 754-1985. This FPU performs both single- and double-precision operations, including division and square-root using the Newton-Raphson algorithm.

Understanding the 'ACT8837 Floating Point Unit

To support floating point processing in IEEE format, the 'ACT8837 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.

The 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 a later section. The 'ACT8837 multiplier operates only on normalized floating-point numbers or wrapped numbers.

Microprogramming the 'ACT8837

The 'ACT8837 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.

The microprogram which controls operation of the FPU is stored in the microprogram memory (or control store). Execution of the microprogram is controlled by a microsequencer such as the TI SN74ACT8818 16-bit microsequencer. A discussion of microprogrammed architecture and the operation of the 'ACT8818 is presented in this Data Manual.

Support Tools

Texas Instruments has developed a functional evaluation model of the 'ACT8837 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.

Texas Instruments has also designed a family of low-cost real-time evaluation modules (EVM) to aid with initial hardware and microcode design. Each EVM is a small self-contained system which provides a convenient means to test and debug simple microcode, allowing software and hardware evaluation of components and their operation.

At present, the 74AS-EVM-8 Bit-Slice Evaluation Module has been completed, and a 16-bit EVM is in an advanced stage of development. EVMs and support tools for devices in the VLSI family are planned for future development.

Design Support

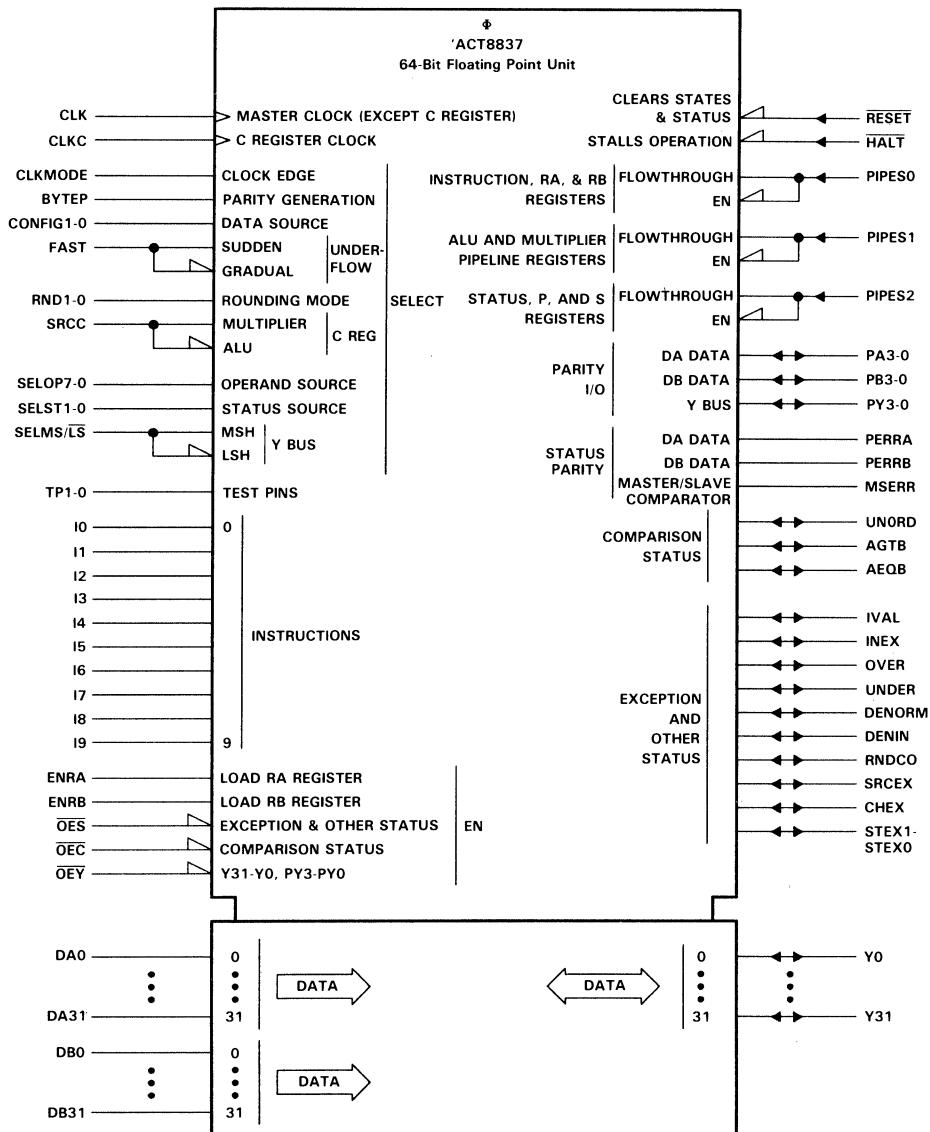
TI's '8837 64-bit floating point unit is supported by a variety of tools developed to aid in design evaluation and verification. These tools will streamline all stages of the design process, from assessing the operation and performance of the '8837 to evaluating a total system application. The tools include a functional model, behavioral model, and microcode development software and hardware. Section 8 of this manual provides specific information on the design tools supporting TI's SN74ACT8800 Family.

Systems Expertise

Texas Instruments VLSI Logic applications group is available to help designers analyze TI's high-performance VLSI products, such as the '8837 64-bit floating point unit. The group works directly with designers to provide ready answers to device-related questions and also prepares a variety of applications documentation.

The group may be reached in Dallas, at (214) 997-3970.

'ACT8837 Logic Symbol



'ACT8837 Pin Descriptions

Pin descriptions and grid allocations for the 'ACT8837 are given on the following pages.

208 PIN . . . GB PACKAGE

(TOP VIEW)

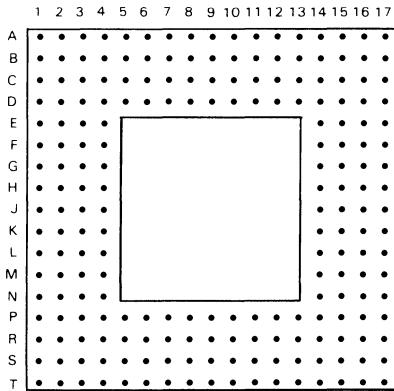


Table 1. 'ACT8837 Pin Grid Allocations

PIN NO.	PIN NAME	PIN NO.	PIN NAME	PIN NO.	PIN NAME	PIN NO.	PIN NAME
A1 NC	C2 Y0	E3 FAST	J15 NC	P1 NC	S1 NC		
A2 NC	C3 Y3	E4 GND	J16 SRCC	P2 PIPES0	S2 PB0		
A3 Y5	C4 Y6	E14 GND	J17 BYTEP	P3 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 DAO	S11 DB30		
A12 Y29	C13 UNDER	F15 PERRA	L1 SELOP6	P12 DA4	S12 DA2		
A13 PY0	C14 INEX	F16 O'EY	L2 SELOP7	P13 DAB	S13 DA6		
A14 PY3	C15 DENIN	F17 O'ES	L3 CLK	P14 DA12	S14 DA10		
A15 IVAL	C16 SRCEX	G1 I7	L4 VCC	P15 DA19	S15 DA14		
A16 NC	C17 CHEX	G2 I6	L14 GND	P16 DA22	S16 DA15		
A17 NC	D1 I1	G3 I4	L15 DA30	P17 DA23	S17 DA17		
B1 NC	D2 RND1	G4 VCC	L16 DA31	R1 PIPES1	T1 NC		
B2 Y2	D3 Y1	G14 VCC	L17 PA0	R2 HALT	T2 PB3		
B3 Y4	D4 GND	G15 OEC	M1 ENRB	R3 PB2	T3 DB3		
B4 Y7	D5 VCC	G16 SELMS/LS	M2 ENRA	R4 DB2	T4 DB7		
B5 Y10	D6 GND	G17 TP1	M3 CLKC	R5 DB6	T5 DB8		
B6 Y13	D7 GND	H1 I9	M4 GND	R6 DB10	T6 DB13		
B7 Y16	D8 VCC	H2 NC	M14 VCC	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 VCC	H14 GND	M17 DA29	R10 DB27	T10 DB25		
B11 Y28	D12 GND	H15 TPO	N1 CONFIG0	R11 DB31	T11 DB29		
B12 Y31	D13 GND	H16 SELST1	N2 CONFIG1	R12 DA3	T12 DA1		
B13 PY2	D14 VCC	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 SELPO0	N15 DA24	R16 DA20	T16 NC		
B17 NC	E1 I2	J4 VCC	N16 DA25	R17 DA21	T17 NC		
C1 PERRB	E2 IO	J14 VCC	N17 DA26				

Table 2. 'ACT8837 Pin Functional Description

PIN NAME	NO.	I/O	DESCRIPTION
AEQB	E16	I/O	Comparison status 1 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.
AGTB	E15	I/O	Comparison status pin. When high, indicates that A operand is greater than B operand.
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).
CHEX	C17	I/O	Status pin indicating an exception during a chained function. If I6 is low, indicates the multiplier is the source of the exception. If I6 is high, indicates the ALU is the source of the exception.
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).
CONFIG0	N1	I	Select data sources for RA and RB registers from DA bus, DB bus and temporary register.
DA1	T12	I	DA 32-bit input data bus. Data can be latched in a 64-bit temporary register or loaded directly into an input register.
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		

Table 2. 'ACT8837 Pin Functional Description (Continued)

PIN NAME	NO.	I/O	DESCRIPTION
DA24	N15		
DA25	N16		
DA26	N17		
DA27	M15		
DA28	M16		
DA29	M17		
DA30	L15		
DA31	L16		
DB0	S3		
DB1	P5		
DB2	R4		
DB3	T3		
DB4	S4		
DB5	P6		
DB6	R5		
DB7	T4		
DB8	T5		
DB9	P7		
DB10	R6		
DB11	S5		
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		
DENIN	C15	I/O	Status pin indicating a denormal input to the multiplier. When DENIN goes high, the STEX pins indicate which port had the denormal input.

Table 2. 'ACT8837 Pin Functional Description (Continued)

PIN NAME	NO.	I/O	DESCRIPTION
DENORM	B16	I/O	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.
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).
FAST	E3	I	When low, selects gradual underflow (IEEE mode). When high, selects sudden underflow, forcing all denormalized inputs and outputs to zero.
GND	D4		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		
HALT	R2	I	Stalls operation without altering contents of instruction or data registers. Active low.
I0	E2		Instruction inputs
I1	D1		
I2	E1		
I3	F2		
I4	G3	I	
I5	F1		
I6	G2		
I7	G1		
I8	H3		
I9	H1		
INEX	C14	I/O	Status pin indicating an inexact output

Table 2. 'ACT8837 Pin Functional Description (Continued)

PIN NAME	PIN NO.	I/O	DESCRIPTION
IVAL	A15	I/O	Status pin indicating that an invalid operation or a nonnumber (NaN) has been input to the multiplier or ALU.
MSERR	E17	O	Master/Slave error output pin
NC	A1		
	A2		
	A16		
	A17		
	B1		
	B17		
	H2		No internal connection. Pins should be left floating.
	J15		
	P1		
	S1		
OVER	T1		
	T16		
PA0	L17		
	PA1		
PA2	K15		
	K16		
PA3	K17		
	PA4		Parity inputs for DA data
PB0	S2		
	PB1		
PB2	P4		
	R3		
PB3	T2		
	T4		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.
PIPES0	P2	I	When low, enables instruction register, RA and RB input registers. When high, puts instruction register, RA and RB registers in flowthrough mode.
PIPES1	R1	I	When low, enables pipeline registers in ALU and multiplier. When high, puts pipeline registers in flowthrough mode.

Table 2. 'ACT8837 Pin Functional Description (Continued)

PIN NAME	NO.	I/O	DESCRIPTION
PIPS2	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.
PY0 PY1 PY2 PY3	A13 C12 B13 A14	I/O	Y port parity data
<u>RESET</u>	P3	I	Clears internal states and status with no effect to data registers. Active low.
RND0 RND1	F3 D2	I	Rounding mode control pins. Select four IEEE rounding modes (see Table 18).
RNDC0	B15	I	When high, indicates the mantissa of a wrapped number has been increased in magnitude by rounding.
SELMS/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.
SELOP0 SELOP1 SELOP2 SELOP3 SELOP4 SELOP5 SELOP6 SELOP7	J3 J2 J1 K1 K2 K3 L1 L2	I	Select operand sources for multiplier and ALU (See Tables 6 and 7)
SELST0 SELST1	H17 H16	I	Select status source during chained operation (see Table 16)
SRCC	J16	I	When low, selects ALU as data source for C register. When high, selects multiplier as data source for C register.
SRCEX	C16	I/O	Status pin indicating source of status, either ALU (SRCEX = L) or multiplier (SRCEX = H)
STEX0 STEX1	D16 D15	I/O	Status pins indicating that a nonnumber (NaN) or denormal number has been input on A port (STEX1) or B port (STEX0).
TP0 TP1	H15 G17	I	Test pins (see Table 19)
UNDER	C13	I/O	Status pin indicating that a result is inexact and less than minimum allowable value for format (exponent underflow).
UNORD	D17	I/O	Comparison status pin indicating that the two inputs are unordered because at least one of them is a nonnumber (NaN).

Table 2. 'ACT8837 Pin Functional Description (Concluded)

PIN NAME	NO.	I/O	DESCRIPTION
VCC	D5		
VCC	D8		
VCC	D11		
VCC	D14		
VCC	G4		
VCC	G14		5-V power supply
VCC	J4		
VCC	J14		
VCC	L4		
VCC	M14		
Y0	C2		
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	I/O	32-bit Y output data bus
Y16	B7		
Y17	A7		
Y18	C8		
Y19	B8		
Y20	A8		
Y21	A9		
Y22	B9		
Y23	C9		
Y24	A10		
Y25	B10		
Y26	C10		
Y27	A11		
Y28	B11		
Y29	A12		
Y30	C11		
Y31	B12		

'ACT8837 Specification Tables

**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 = 0 to 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

5

SN74ACT8837

PARAMETER	SN74ACT8837			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			SN74ACT8837			UNIT
			MIN	TYP	MAX	MIN	TYP	MAX	
V _{OH}	I _{OH} = -20 μA	4.5 V							V
		5.5 V							
	I _{OH} = -8 mA	4.5 V			3.76				
		5.5 V			4.76				
V _{OL}	I _{OL} = 20 μA	4.5 V							V
		5.5 V							
	I _{OL} = 8 mA	4.5 V						0.45	
		5.5 V						0.45	
I _I	V _I = V _{CC} or 0	5.5 V						±1	μA
I _{CC}	V _I = V _{CC} or 0, I _O	5.5 V						200	μA
C _i	V _I = V _{CC} or 0	5 V							pF

switching characteristics (see Note)

PARAMETER		SN74ACT8837-65		UNIT
		MIN	MAX	
t _{pd1}	Propagation delay from DA/DB/I inputs to Y output		125	ns
t _{pd2}	Propagation delay from input register to output buffer		118	ns
t _{pd3}	Propagation delay from pipeline register to output buffer		70	ns
t _{pd4}	Propagation delay from output register to output buffer		30	ns
t _{pd5}	Propagation delay from SELMS/L _S to Y output		32	ns
t _{d1}	Propagation delay from input register to output register		95	ns
t _{d2}	Delay time, input register to pipeline register or pipeline register to output register	65		ns

5

SN74ACT8837

Note: Switching data must be used with timing diagrams for different operating modes.

setup and hold times

PARAMETER		SN74ACT8837-65		UNIT
		MIN	MAX	
t_{su1}	Setup time, Instruction before CLK1	18		ns
t_{su2}	Setup time, data operand before CLK1	18		ns
t_{su3}	Setup time, data operand before second CLK1 for double-precision operation (input register not enabled)	65		ns
t_{h1}	Hold time, Instruction input after CLK1	0		ns

clock requirements

PARAMETER		SN74ACT8837-65		UNIT
		MIN	MAX	
t_w	Pulse duration	CLK high	15	ns
		CLK low	15	
Clock period				ns

SN74ACT8837 FLOATING POINT UNIT

The SN74ACT8837 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 and multiplication operations.

The 'ACT8837 input buses can be configured to operate as two 32-bit data buses or 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 register to be clocked on the rising edge or the falling edge of the clock to support double precision operations (except multiplication) at the same rate as single precision operations. A feedback register with a separate clock is provided for temporary storage of a multiplier result, ALU result or constant.

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.

Floating point division using a Newton-Raphson algorithm can be performed in a sum-of-products operating mode, one of two modes in which the multiplier and ALU operate in parallel. Absolute value conversions, floating point to integer and integer to floating point conversions, and a compare instruction are also available.

Data Flow

Data enters the 'ACT8837 through two 32-bit input data buses, DA and DB. The buses can be configured to operate as a single 64-bit data bus for double precision operations (see Table 7). Data can be latched in a 64-bit temporary register or loaded directly into the RA and RB registers for input to the multiplier and ALU.

Four multiplexers select the multiplier and ALU operands from the input register, 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 for output. The 64-bit C register is provided for temporary storage of a result from the ALU or multiplier.

Input Data Parity Check

When BYTEP is high, internal odd parity is generated for each byte of input data at the DA and DB ports and compared to the PA and PB parity inputs. If an odd number of bits is set high in a data byte, the parity bit for that byte is also set high. Parity bits are input on PA for DA data and PB for DB data. PAO and PBO are the parity bits for the least significant bytes of DA and DB, respectively. If the parity comparison fails for any byte, a high appears on the parity error output pin (PERRA for DA data and PERRB for DB data).

SN74ACT883

5

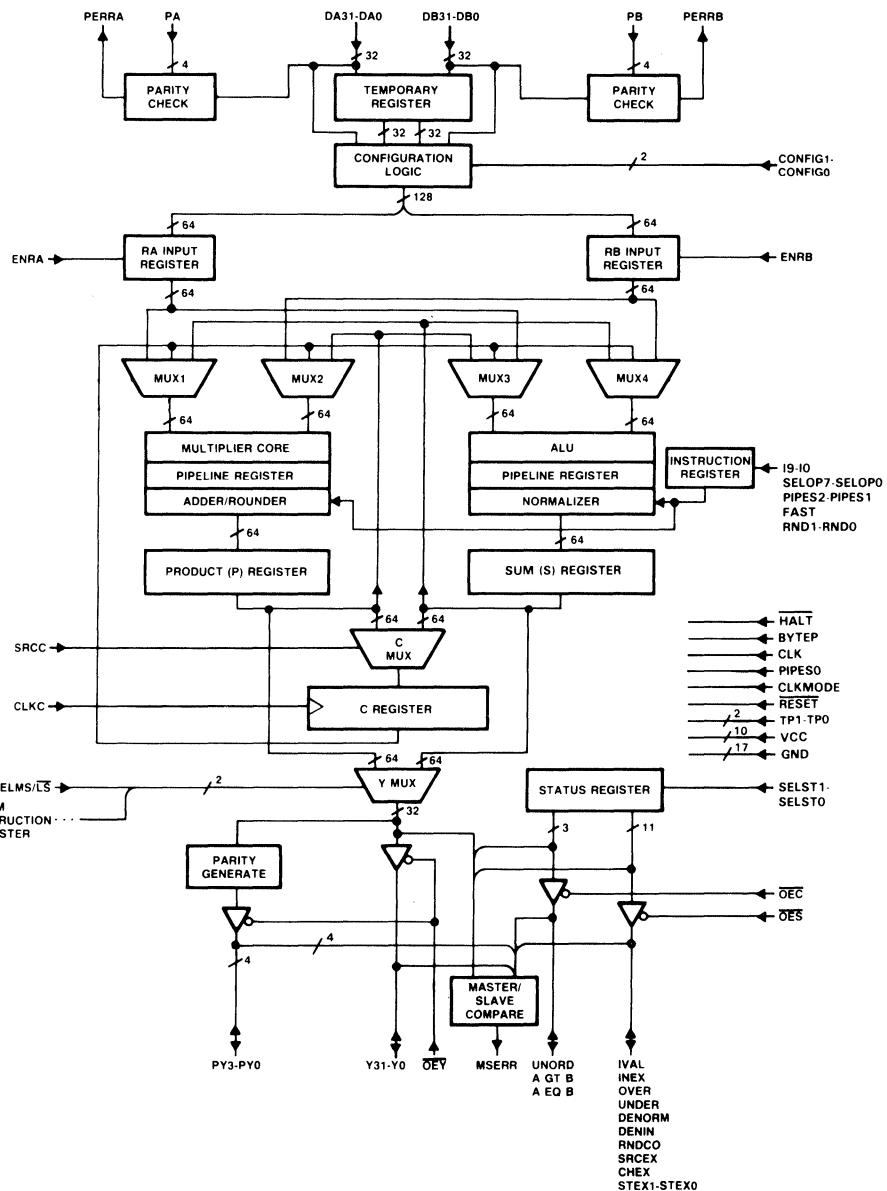


Figure 1. 'ACT8837 Floating Point Unit

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.

Temporary Input Register

A temporary input register is provided to enable double precision numbers on a single 32-bit input bus to be loaded 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; when CLKMODE is high, data is loaded on the falling edge.

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, according to configuration mode controls CONFIG1-CONFIG0 (see Tables 3 and 5). The registers are loaded on the rising edge of clock CLK. For single-precision operations, CONFIG1-CONFIG0 should ordinarily be set to 0 1 (see Table 4).

Table 3. Double-Precision Input Data Configuration Modes

CONFIG1	CONFIG0	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	
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, Table 17), 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.

Table 4. Single-Precision Input Data Configuration Mode

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

Table 5. Double-Precision Input Data Register Sources

CONFIG1	CONFIG0	RA SOURCE		RB SOURCE	
		MSH	LSH	MSH	LSH
0	0	DA	DB	TEMP REG (MSH)	TEMP REG (LSH)
0	1	DA	TEMP REG (MSH)	DB	TEMP REG (LSH)
1	0	TEMP REG (MSH)	DA	TEMP REG (LSH)	DB
1	1	TEMP REG (MSH)	TEMP REG (LSH)	DA	DB

5

Multiplier/ALU Multiplexers

Four multiplexers select the multiplier and ALU operands from the RA and RB registers, the previous multiplier or ALU result, or the C register. The multiplexers are controlled by input signals SELOP7-SELOPO as shown in Tables 6 and 7.

Table 6. 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

Table 7. 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

Pipelined ALU

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

The ALU may be operated independently or in parallel with the multiplier. Possible ALU functions during independent operation are given in Tables 8 and 9. Parallel ALU/multiplier functions are listed in Table 11.

Pipelined Multiplier

The pipelined multiplier performs a basic multiply function, $A * B$. The operands can be single-precision or double-precision numbers and can be converted to absolute values before multiplication takes place. Multiplier operations are summarized in Table 10.

An exception circuit is provided to detect denormalized inputs; these are indicated by a high on the DENIN signal.

The multiplier and ALU can be operated simultaneously by setting the I9 instruction input high. Possible operations in this chained mode are listed in Table 13.

Product, Sum, and C Registers

The results of the ALU and multiplier operations may optionally be latched into two output registers on the rising edge of the system clock (CLK). The P (product) register holds the result of the multiplier operation; the S (sum) register holds the ALU result.

An additional 64-bit register is provided for temporary storage of the result of an ALU or multiplier operation before feedback to the multiplier or ALU. The data source for this C register is selected by SRCC; a high on this pin selects the multiplier result; a low selects the ALU. A separate clock, CLKC, has been provided for this register.

Parity Generators

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 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.

Table 8. Independent ALU Operations, Single Operand (I9 = 0, I6 = 0)

CHAINED OPERATION I9	PRECISION RA I8	PRECISION RB I7	OUTPUT SOURCE I6	OPERAND TYPE I5	ABSOLUTE VALUE A I4	ALU OPERATION	
						I3-I0	RESULT
0 = Not Chained	0 = A(SP) 1 = A(DP)	0 = B(SP) 1 = B(DP)	0 = ALU result	1 = Single Operand	0 = A 1 = A	0000	Pass A operand
						0001	Negate A operand
						0010	Integer to floating point conversion [†]
						0011	Floating point to integer conversion
						0100	Undefined
						0101	Undefined
						0110	Floating point to floating point conversion [‡]
						0111	Undefined
						1000	Wrap (denormal) input operand
						1001	Undefined
						1010	Undefined
						1011	Undefined
						1100	Unwrap exact number
						1101	Unwrap inexact number
						1110	Unwrap rounded input
						1111	Undefined

[†]The precision of the integer to floating point conversion is set by I8.

[‡]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.

Table 9. Independent ALU Operations, Two Operands (I9 = 0, I5 = 0)

CHAINED OPERATION 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 = 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 10. Independent Multiplier Operations (I9 = 0, I6 = 1)

CHAINED OPERATION I9	PRECISION RA I8	PRECISION RB I7	OUTPUT SOURCE I6	I5	ABSOLUTE VALUE A I4 [†]	ABSOLUTE VALUE B I3 [†]	NEGATE RESULT I2 [†]	WRAP A I1	WRAP B I0
0 = Not chained	0 = A(SP) 1 = A(DP)	0 = B(SP) 1 = B(DP)	1 = Multi- plier result	0	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

[†]See Table 15.

Table 11. Independent Multiplier Operations Selected by I4-I2 (I9 = 0, I6 = 1)

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)

Table 12. Operations Selected by I8-I7 (I9 = 0, I6 = 1)

PRECISION SELECT RA I8	PRECISION RA INPUT	PRECISION SELECT RB I7	PRECISION RB INPUT	PRECISION OF RESULT
0	Single	0	Single	Single
0	Single Converted to Double	1	Double	Double
1	Double	0	Single Converted to Double	Double
1	Double	1	Double	Double

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{OYE} , \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).

Status and Exception Generator/Register

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

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 13. Chained Multiplier/ALU Operations (I9 = 1)

CHAINED OPERATION I9	PRECISION RA I8	PRECISION RB I7	OUTPUT SOURCE I6	ADD ZERO I5	MULTIPLY BY ONE I4	NEGATE ALU RESULT I3	NEGATE MULTI- PLIER RESULT I2	ALU OPERATIONS	
								I1-I0	RESULT
1 = Chained	0 = A(SP) 1 = A(DP)	0 = B(SP) 1 = B(DP)	0 = ALU result 1 = Multi- plier result	0 = Normal operation 1 = Forces B2 input of ALU to zero	0 = Normal operation 1 = Forces B1 input of multi- plier to one	0 = Normal operation 1 = Negate ALU result	0 = Normal operation 1 = Negate multiplier result	00 01 10 11	A + B A - B 2 - A B - A

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.)
AGTB	The A operand is greater than the B operand. (Only during a compare operation in the ALU)
UNORD	The two inputs of a comparison operation are unordered, i.e., one or both of the inputs is a NaN.

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 a 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.
INEX	The result of an operation is not exact.
IVAL	A NaN has been input to the multiplier or the ALU, or an invalid operation ($0 * \infty$ or $\pm \infty \mp \infty$) has been requested. When IVAL goes high, the STEX pins indicate which port had a NaN.
OVER	The result is greater than the largest allowable value for the specified format.
RNDCO	The mantissa of a wrapped number has been increased in magnitude by rounding and the unwrap round instruction can be used to unwrap properly 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.)
STEXO	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, status 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, CEX is set high. Status of the nonselected output source can be forced using the SELST pins, as shown in Table 16.

Table 16. Status Output Selection (Chain Mode)

SELST1- SELST0	STATUS SELECTED
00	Invalid
01	Selects multiplier status
10	Selects ALU status
11	Normal operation (selection based on result source specified by I6 input)

Flowthrough Mode

To enable the device to operate in pipelined or flowthrough modes, registers can be bypassed using pipeline control signals PIPES2-PIPES0 (see Table 17).

Table 17. Pipeline Controls (PIPES2-PIPES0)

PIPES2- PIPES0	REGISTER OPERATION SELECTED
X X 0	Enables input registers (RA, RB)
X X 1	Disables input registers (RA, RB)
X 0 X	Enables pipeline registers
X 1 X	Disables pipeline registers
0 X X	Enables output registers (P, S, Status)
1 X X	Disables output registers (P, S, Status)

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. If the multiplier result underflows, a wrapped number will be output.

Rounding Mode

The 'ACT8837 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)

Test Pins

Two pins, TP1-TPO, 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 19).

Table 19. Test Pin Control Inputs

TP1- TPO	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

Summary of Control Inputs

Control input signals for the 'ACT8837 are summarized in Table 20.

Table 20. Control Inputs

SIGNAL	HIGH	LOW
BYTEP	Selects byte parity generation and test	Selects single bit parity generation and test
CLK	Clocks all registers except C	No effect
CLKC	Clocks C register	No effect
CLKMODE	Enables temporary input register load on falling clock edge	Enables temporary input register load on rising clock edge
CONFIG1- CONFIG0	See Table 3 (RA and RB register data source selects)	See Table 3 (RA and RB register data source selects)
ENRA	If register is not in flow through, enables clocking RA register	If register is not in flow through, holds contents of RA register
ENRB	If register is not in flow through, enables clocking of RB register	If register is not in flow through, holds contents of RB register
FAST	Places device in FAST mode	Places device in IEEE mode
HALT	No effect	Stalls device operation but does not affect registers, internal states, or status
OEC	Disables compare pins	Enables compare pins
OES	Disables status outputs	Enables status outputs
OEY	Disables Y bus	Enables Y bus
PIPES2- PIPES0	See Table 17 (pipeline mode control)	See Table 17 (pipeline mode control)
RESET	No effect	Clears internal states and status but does not affect data registers
RND1- RND0	See Table 18 (rounding mode control)	See Table 18 (rounding mode control)
SELOPT7- SELOPT0	See Tables 6 and 7 (multiplier/ALU operand selection)	See Tables 6 and 7 (multiplier/ALU operand selection)
SELMS/LS	Selects MSH of 64-bit result for output on the Y bus	Selects LSH of 64-bit result for output on the Y bus (no effect during single precision operation)
SELST1- SELST0	See Table 15 (status output selection)	See Table 15 (status output selection)
SRCC	Selects multiplier result for input to C register	Selects ALU result for input to C register
TP1-TPO	See Table 19 (test pin control inputs)	See Table 19 (test pin control inputs)

5

SN74ACT8837

INSTRUCTION SET

Configuration and operation of the 'ACT8837 can be selected to perform single- or double-precision floating-point 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. The ALU and the multiplier of the 'ACT8837 can operate either independently or simultaneously, depending on the setting of instruction inputs I9-I0 and related controls.

Controls for data flow and status results are discussed separately, prior to the discussions of ALU and multiplier operations. Then, in Tables 22 through 25, the instruction inputs to the ALU and the multiplier are summarized according to operating mode, whether independent or chained (ALU and multiplier in simultaneous operation).

Loading External Data Operands

Patterns of data input to the 'ACT8837 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 and register destinations for data inputs.

Configuration Controls (CONFIG1-CONFIG0)

5

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 only used during input of double-precision operands.

When single-precision or integer operands are loaded, the ordinary setting of CONFIG1-CONFIG0 is LH, as shown in Table 4. This setting loads each 32-bit operand in the most significant half (MSH) of its respective register. The 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 CONFIG1-CONFIG0 set to HH but two clock edges are required to load both the A and B operands on the DA bus.

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 Tables 3 and 5, four configuration modes for selecting input sources are available for loading data operands into the RA and RB registers.

CLKMODE 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 = L) or the falling edge of the clock (CLKMODE = H). 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.

SN74ACT8837

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 ALU 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, and the ALU 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 'ACT8837 is operated in clock mode 1.

Internal Register Operations

Six data registers in the 'ACT8837 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.

The RA and RB registers receive data inputs from the temporary register and the DA and DB buses. Data operands are then multiplexed into the multiplier, ALU, or both. To support simultaneous pipelined operations, the data paths through the multiplier and the ALU are both provided with pipeline registers and output registers. The control settings for the pipeline and output registers (PIPES2-PIPES1) are registered with the instruction inputs I9-I0.

A seventh register, the constant (C) register is available for storing a 64-bit constant or an intermediate result from the multiplier or the ALU. The C register has a separate clock input (CLKC) and input source select (SRCC). 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.

Status results are also registered whenever the output registers are enabled. Duration and availability of status results are affected by the same timing constraints that apply to data results on the Y output bus.

Data Register Controls (PIPES2-PIPES0)

Table 17 shows the settings of the registers controlled by PIPES2-PIPES0. Operating modes range from fully pipelined (PIPES2-PIPES0 = LLL) to flowthrough (PIPES2-PIPES0 = HHH).

In flowthrough mode all three levels of registers are disabled, 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 requires two cycles to execute, depending on the settings of PIPES2-PIPES0. The output may be valid for one or two cycles, depending on the precision of the next operation.

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 open clock cycle must intervene between the dissimilar operations.

C Register Controls (SRCC, CLKC)

5

The C register loads from the P or the S register output, depending on the setting of SRCC, the load source select. SRCC = H selects the multiplier as input source. Otherwise the ALU is selected when SRCC = L. In either case the C register only loads the selected input on a rising edge of the CLKC signal.

The C register does not load directly from an external data bus. One method for loading a constant without wasting a cycle is to input the value as an A operand during an operation which uses only the ALU or multiplier and requires no external data inputs. Since the B operand can be forced to zero in the ALU or to one in the multiplier, the A operand can be passed to the C register either by adding zero or multiplying by one, then selecting the input source with SRCC and causing the CLKC signal to go high. Otherwise, the C register can be loaded through the ALU with the Pass A Operand instruction, which requires a separate cycle.

Operand Selection (SELOP7-SELOP0)

As shown in Tables 6 and 7, data operands can be selected as five possible sources, including external inputs from the RA and RB registers, feedback from the P and S 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.

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

Rounding Controls (RND1-RNDO)

Because floating point operations may involve both inherent and procedural errors, it is important to select appropriate modes for handling rounding errors. To support the IEEE standard for binary floating-point arithmetic, the 'ACT8837 provides four rounding modes selected by RND1-RNDO.

Table 18 shows the four selectable rounding modes. The usual default rounding mode is round to nearest (RND1-RNDO = LL). In round-to-nearest mode, the 'ACT8837 supports the IEEE standard by rounding to even (LSB = 0) when two nearest representable values are equally near. Directed rounding toward zero, infinity, or minus infinity are also available.

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.

Status Exceptions

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.

To simplify the discussion of exception handling, it is useful to summarize the data formats for representing IEEE floating-point numbers which can be input to or output from the FPU (see Table 21). Since procedures for handling exceptions vary according to the requirements of specific applications, this discussion focuses on the conditions which cause particular status exceptions to be signalled by the FPU.

Table 21. IEEE Floating-Point Representations

TYPE OF OPERAND	EXPONENT (e)		FRACTION (f) (BINARY)	HIDDEN BIT	VALUE OF NUMBER REPRESENTED	
	SP (HEX)	DP (HEX)			SP (DECIMAL)†	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	$(-1)^s (\text{infinity})$	$(-1)^s (\text{infinity})$
NAN (Not a Number)	FF	7FF	Nonzero	N/A	None	None

† s = sign bit

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 eight 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. Further details of IEEE formats and exceptions are provided in the IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Std 754-1985.

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, DENIN, and STEX1-STEX0. If the IVAL flag is set, either an invalid operation 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. 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 = H) or a NaN (IVAL = H) is input to the multiplier or the ALU.

Nan inputs are all treated as IEEE signaling NaNs, causing the IVAL flag to be set. When output from the FPU, the fraction field from a NaN is set high (all 1's), regardless of the original fraction field of the input NaN.

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 signaled. The chained-mode exception signal CHEX indicates that an exception has been generated by the source not selected for output by I6. The exception type signaled 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. 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. 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 = H) may also cause INEX to be set, in which case UNDER is also signalled.

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 .IXXX. . . , 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 = L 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 22 describes the handling of wrapped multiplier results and the status flags that are set when wrapped numbers are output from the multiplier.

Table 22. Handling Wrapped Multiplier Outputs

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

Data Output Controls (SELMS/LS, OEY)

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/LS signal in the middle of the clock period.

When a single-precision result is output, the SELMS/LS signal has no effect. The SELMS/LS signal is set low only to read out the LSH of a double-precision result. Whenever this signal is selecting a valid result for output on the Y bus, the OEY enable must be pulled low at the beginning of that clock cycle.

Status Output Controls (SELST1-SELST0, OES, OEC)

5

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 = L). Otherwise, the status register is transparent. In either case, status outputs can be read by pulling the output enables low (OES, OEC, or both).

Stalling the Device (HALT)

SN74ACT8837

Operation of the 'ACT8837 can be stalled nondestructively by means of the HALT signal. Pulling the HALT input low causes the device to stall on the next low level of the clock. Register contents are unaltered when the device is stalled, and normal operation resumes at the next low clock period after the HALT signal is set high. Using HALT in microprograms can save power, especially using high clock frequencies and pipelined stages.

For some operations, such as a double-precision multiply with CLKMODE = 1, setting the 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 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 HALT signal is set high again and normal operation resumes.

Instruction Inputs (I9-I0)

Three modes of operation can be selected with inputs I9-I0, including independent ALU operation, independent multiplier operation, or simultaneous (chained) operation of ALU and multiplier. Each operating mode is treated separately in the following sections.

Independent ALU Operations

The ALU executes single- and double-precision operations which can be divided according to the number of operands involved, one or two. The ALU accepts integer, normalized, and denormalized numbers as operands. Table 22 shows independent ALU operations with one operand, along with the inputs I9-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. Wrapping and unwrapping of operands is also done in this mode.

Table 24 presents 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.

Independent Multiplier Operations

In this mode the multiplier operates on the RA and RB inputs which can be either single precision, double precision, or mixed. Operands may be normalized or wrapped numbers, as indicated by the settings for instruction inputs I1-I0. As shown in Table 25, the multiplier can be set to operate on the absolute value of either or both 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.

Table 23. Independent ALU Operations with One Operand

ALU OPERATION ON A OPERAND	INSTRUCTION INPUTS I9-I0	NOTES
Pass A operand	0x 001x 0000	
Negate A operand	0x 001x 0001	
Convert from integer to floating point [†]	0x 0010 0010	
Convert from floating point to integer	0x 001x 0011	
Undefined	0x 001x 0100	x = Don't care
Undefined	0x 001x 0101	I8 selects precision of A operand: 0 = A (SP) 1 = A (DP)
Convert from floating point to floating point (adjusts precision of input: SP→DP, DP→SP)	0x 001x 0110	I4 selects absolute value of A operand: 0 = A 1 = A
Undefined	0x 001x 0111	During integer to floating point conversion, A is not allowed as a result.
Wrap denormal operand	0x 001x 1000	
Undefined	0x 001x 1001	
Undefined	0x 001x 1010	
Undefined	0x 001x 1011	
Unwrap exact number	0x 001x 1100	
Unwrap inexact number	0x 001x 1101	
Unwrap rounded input	0x 001x 1110	
Undefined	0x 001x 1111	

[†] During this operation, I8 selects precision of the result.

Table 24. Independent ALU Operations with Two Operands

ALU OPERATIONS AND OPERANDS	INSTRUCTION INPUTS I9-I0	NOTES
Add A + B	0x x000 0x00	
Add A + B	0x x001 0x00	
Add A + B	0x x000 1x00	
Add A + B	0x x001 1x00	
Subtract A - B	0x x000 0x01	x = Don't Care I8 selects precision of A operand: 0 = A (SP) 1 = A (DP)
Subtract A - B	0x x001 0x01	
Subtract A - B	0x x000 1x01	
Subtract A - B	0x x001 1x01	I7 selects precision of B operand: 0 = B (SP) 1 = B (DP)
Compare A, B	0x x000 0x10	
Compare A , B	0x x001 0x10	
Compare A, B	0x x000 1x10	I2 selects either Y or its absolute value: 0 = Y
Compare A , B	0x x001 1x10	1 = Y
Subtract B - A	0x x000 0x11	
Subtract B - A	0x x001 0x11	
Subtract B - A	0x x000 1x11	
Subtract B - A	0x x001 1x11	

5

Table 25. Independent Multiplier Operations

MULTIPLIER OPERATION AND OPERANDS	INSTRUCTION INPUTS I9-I0	NOTES
Multiply A * B	0x x100 00xx	x = Don't Care
Multiply -(A * B)	0x x100 01xx	I8 selects A operand precision (0 = SP, 1 = DP)
Multiply A * B	0x x100 10xx	I7 selects B operand precision (0 = SP, 1 = DP)
Multiply -(A * B)	0x x100 11xx	
Multiply A * B	0x x101 00xx	I1 selects A operand format (0 = Normal, 1 = Wrapped)
Multiply -(A * B)	0x x101 01xx	
Multiply A * B	0x x101 10xx	I0 selects B operand format (0 = Normal, 1 = Wrapped)
Multiply -(A * B)	0x x101 11xx	

SN74ACT8837

Chained Multiplier/ALU Operations

In chained mode, the 'ACT8837 performs simultaneous operations in the multiplier and the ALU. Operations include addition, subtraction, and multiplication, except multiplication of wrapped operands. Several optional operations also increase the flexibility of the device.

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 (see Table 26).

Table 26. Chained Multiplier/ALU Operations

CHAINED OPERATIONS		OUTPUT SOURCE	INSTRUCTION INPUTS I9-I0	NOTES
MULTIPLIER	ALU			
A * B	A + B	ALU	1x x000 xx00	
A * B	A + B	Multiplier	1x x100 xx00	
A * B	A - B	ALU	1x x000 xx01	
A * B	A - B	Multiplier	1x x100 xx01	
A * B	2 - A	ALU	1x x000 xx10	
A * B	2 - A	Multiplier	1x x100 xx10	x = Don't Care
A * B	B - A	ALU	1x x000 xx11	I8 selects precision of RA inputs:
A * B	B - A	Multiplier	1x x100 xx11	0 = RA (SP)
A * B	A + 0	ALU	1x x010 xx00	1 = RA (DP)
A * B	A + 0	Multiplier	1x x110 xx00	I7 selects precision of RB inputs:
A * B	0 - A	ALU	1x x010 xx11	0 = RB (SP)
A * B	0 - A	Multiplier	1x x110 xx11	1 = RB (DP)
A * 1	A + B	ALU	1x x001 xx00	I3 negates ALU result:
A * 1	A + B	Multiplier	1x x101 xx00	0 = Normal
A * 1	A - B	ALU	1x x001 xx01	1 = Negated
A * 1	A - B	Multiplier	1x x101 xx01	I2 negates multiplier result:
A * 1	2 - A	ALU	1x x001 xx10	0 = Normal
A * 1	2 - A	Multiplier	1x x101 xx10	1 = Negated
A * 1	B - A	ALU	1x x001 xx11	
A * 1	B - A	Multiplier	1x x101 xx11	
A * 1	A + 0	ALU	1x x011 xx00	
A * 1	A + 0	Multiplier	1x x111 xx00	
A * 1	0 - A	ALU	1x x011 xx11	
A * 1	0 - A	Multiplier	1x x111 xx11	

MICROPROGRAMMING THE 'ACT8837

Because the 'ACT8837 is microprogrammable, it can be configured to operate on either 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 examples present not only control settings but the timings of the specific operations required to execute the sample instructions.

Timing of the sample operations varies with the precision of the data operands and the settings of CLKMODE and PIPES. Microinstructions and timing waveforms are given for all combinations of data precision, clock mode, and register settings. Following the presentation of ALU and multiplier operations is a brief sum-of-products operation using instructions for chained operating mode.

Single-Precision Operations

Two single-precision operands can be loaded on the 32-bit input buses without use of the temporary register so CLKMODE has no effect on single-precision operation. Both the ALU and the multiplier execute all single-precision instructions in one clock cycle, assuming that the device is not operating in flowthrough mode (all registers disabled). Settings of the register controls PIPES2-PIPES0 determine minimum cycle time and the rate of data throughput, as evident from the examples below.

5 Single-Precision ALU Operations

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. 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.

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.

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.

Single-Precision 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. If both operands are single precision, a single-precision result is output. Operations on mixed-precision data inputs produce double-precision results.

Sample Single-Precision Microinstructions

The following four single-precision microinstruction coding examples show the four register settings, ranging from flowthrough to fully pipelined. Timing diagrams accompany the sample microinstructions.

In the first example PIPES2-PIPES0 are all set high so the internal registers are all disabled. This microinstruction sets up a wrapped result from the multiplier to be unwrapped by the ALU as an exact denormalized number. In flowthrough mode the ‘unwrap exact’ operation is performed without a clock as soon as the instruction is input. Single-precision timing in flowthrough mode is shown in Figure 2.

CLKMODE = 0 PIPES = 111 Operation: Unwrap A operand exact

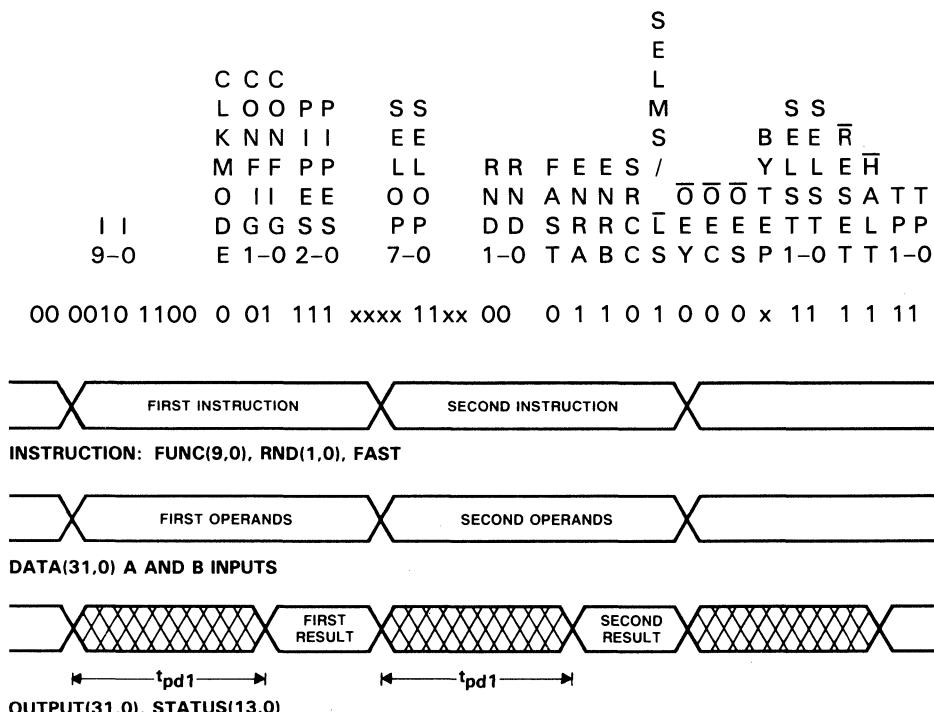


Figure 2. Single-Precision Operation, All Registers Disabled
(PIPES = 111, CLKMODE = 0)

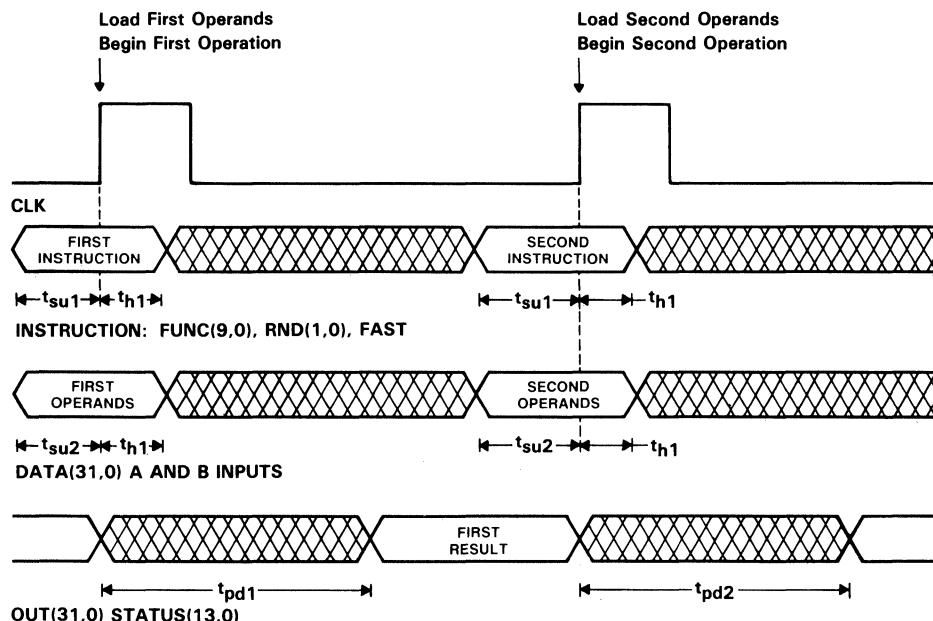
The second example shows a microinstruction causing the ALU to compare absolute values of A and B. Only the input registers are enabled (PIPES2-PIPES0 = 110) so the result is output in one clock cycle.

CLKMODE = 0 PIPES = 110 Operation: Compare $|A|, |B|$

C CCC	S
L O O P P	E
K N N I I	L
M F F P P	M
O I I E E	S S
I I D G G S S	B E E R
9-0 E 1-0 2-0	Y L L E H
7-0	N N A N N R
	D D S R R C L E E E E T T E L P P
	1-0 T A B C S Y C S P 1-0 T T 1-0

00 0001 1010 0 01 110 xxxx 1111 00 0 1 1 0 1 0 0 0 x 11 1 1 1 1

SN74ACT8837



**Figure 3. Single-Precision Operation, Input Registers Enabled
(PIPES = 110, CLKMODE = 0)**

Input and output registers are enabled in the third example, which shows the subtraction $B - A$. Two clock cycles are required to load the operands, execute the subtraction, and output the result (see Figure 4).

CLKMODE = 0 PIPES = 010 Operation: Subtract $B - A$

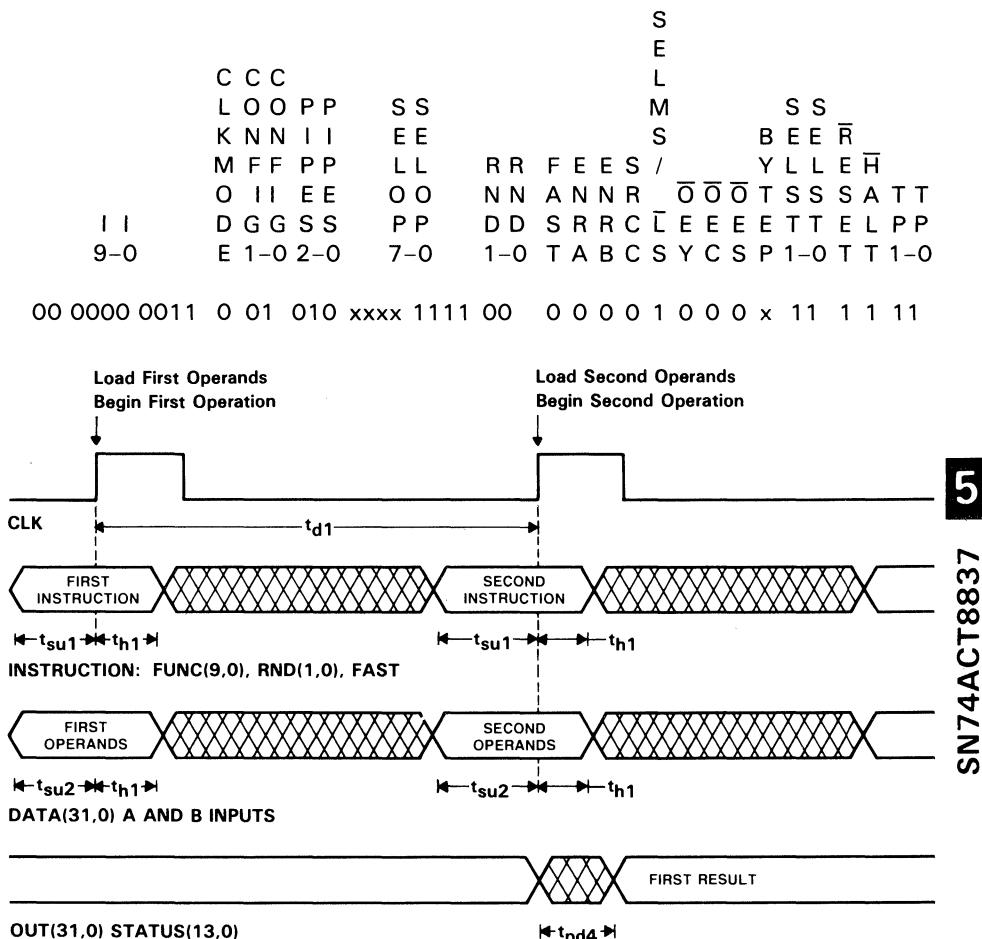


Figure 4. Single-Precision Operation, Input and Output Registers Enabled
(PIPES = 010, CLKMODE = 0)

The fourth example shows a multiplication A * B with all registers enabled. Three clock cycles are required to generate and output the product. Once the internal registers are all loaded with data or results, a result is available from the output register on every rising edge of the clock. The floating point unit produces its highest throughput when operated fully pipelined with single-precision operands.

CLKMODE = 0 PIPES = 000 Operation: Multiply A * B

C C C	S S	L	S S
L O O P P	E E	M	B E E R
K N N I I		S	
M F F P P	L L	R R F E E S /	Y L L E H
O I I E E	O O	N N A N N R	O O O T S S S A T T
I I	D G G S S	P P	D D S R R C L E E E T T E L P P
9-0	E 1-0 2-0	7-0	1-0 T A B C S Y C S P 1-0 T T 1-0

00 0100 0000 0 01 000 1111 xxxx 00 0 1 1 1 1 0 0 0 x 11 1 1 11

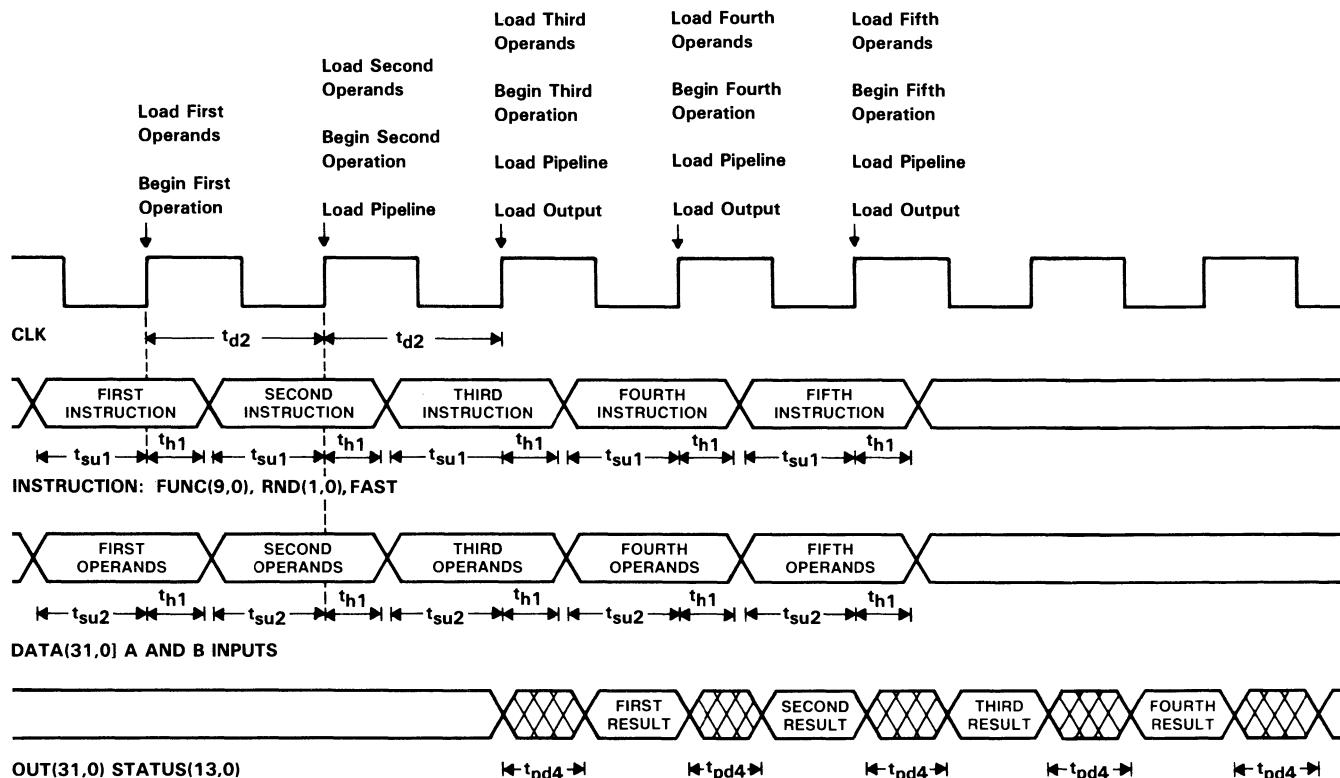


Figure 5. Single-Precision Operation, All Registers Enabled
(PIPES = 000, CLKMODE = 0)



Double-Precision Operations

Double-precision operations may be executed separately in the ALU or the multiplier, or simultaneously in both. Rates of execution and data throughput are affected by the settings of the register controls (PIPES2-PIPES0) and the clock mode (CLKMODE).

The temporary register can be loaded on either the rising edge (CLKMODE = L) or the falling edge of the clock (CLKMODE = H). Double-precision operands are always loaded by using the 64-bit temporary register to store half of the operands prior to inputting the other half of the operands on the DA and DB buses.

Input configuration is selected by CONFIG1-CONFIG0, allowing several options for the sequence in which data operands are set up in the temporary register and the RA and RB registers. Operands are then sent to either the ALU or multiplier, or both, depending on the settings for SELOP 7-0.

The ALU executes all double-precision operations in a single clock cycle. The multiplier requires two clock cycles to execute a double-precision operation. When the device operates in chained mode (simultaneous ALU and multiplier operations), the chained double-precision operation is executed in two clock cycles. The settings of PIPES2-PIPES0 determine whether the result is output without a clock (flowthrough) or after up to five clocks for a double-precision multiplication (all registers enabled and CLKMODE = L).

5

Double-Precision ALU Operations

Eight examples are provided to illustrate microinstructions and timing for double-precision ALU operations. The settings of CLKMODE and PIPES2-PIPES0 determine how the temporary register loads and which registers are enabled. Four examples are provided in each clock mode.

Double-Precision ALU Operations with CLKMODE = 0

The first example shows that, even in flowthrough mode, a clock signal is needed to load the temporary register with half the data operands (see Figure 6). The selected

operation is executed without a clock after the remaining half of the data operands are input on the RA and RB buses:

CLKMODE = 0 PIPES = 111 Operation: Add A + |B|

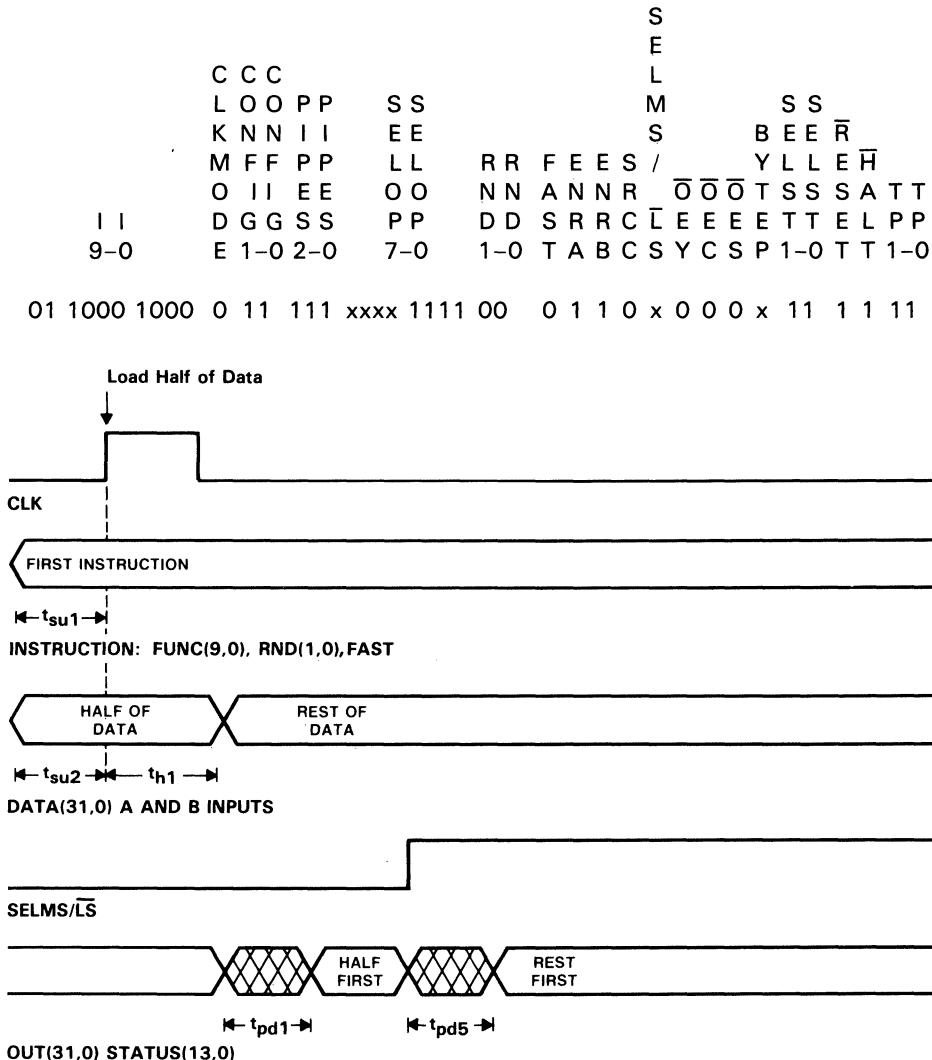


Figure 6. Double-Precision ALU Operation, All Registers Disabled
(PIPES = 111, CLKMODE = 0)

In the second example the input register is enabled ($\text{PIPES}2\text{-}\text{PIPES}0 = 110$). Operands A and B for the instruction, $|B| - |A|$, are loaded using $\text{CONFIG} = 00$ so that B is loaded first into the temporary register with MSH through the DA port and LSH through the DB port. On the second clock rising edge, the A operand is loaded in the same order directly to RA register while B is loaded from the temporary register to the RB register (see Figure 7).

$\text{CLKMODE} = 0 \quad \text{PIPES} = 110 \quad \text{Operation: } |B| - |A|$

			S
			E
C C C			L
L O O P P	S S		M
K N N I I	E E		S
M F F P P	LL	R R F E E S /	B E E \bar{R}
O I I E E	O O	N N A N N R	$\bar{O} \bar{O} \bar{O}$ T S S S A T T
I I	D G G S S	P P	D D S R R C L E E E E T T E L P P
9-0	E 1-0 2-0	7-0	1-0 T A B C S Y C S P 1-0 T T 1-0

01 1001 1011 0 00 110 xxxx 1111 00 0 1 1 0 x 0 0 0 x 11 1 1 11

5

SN74ACT8837

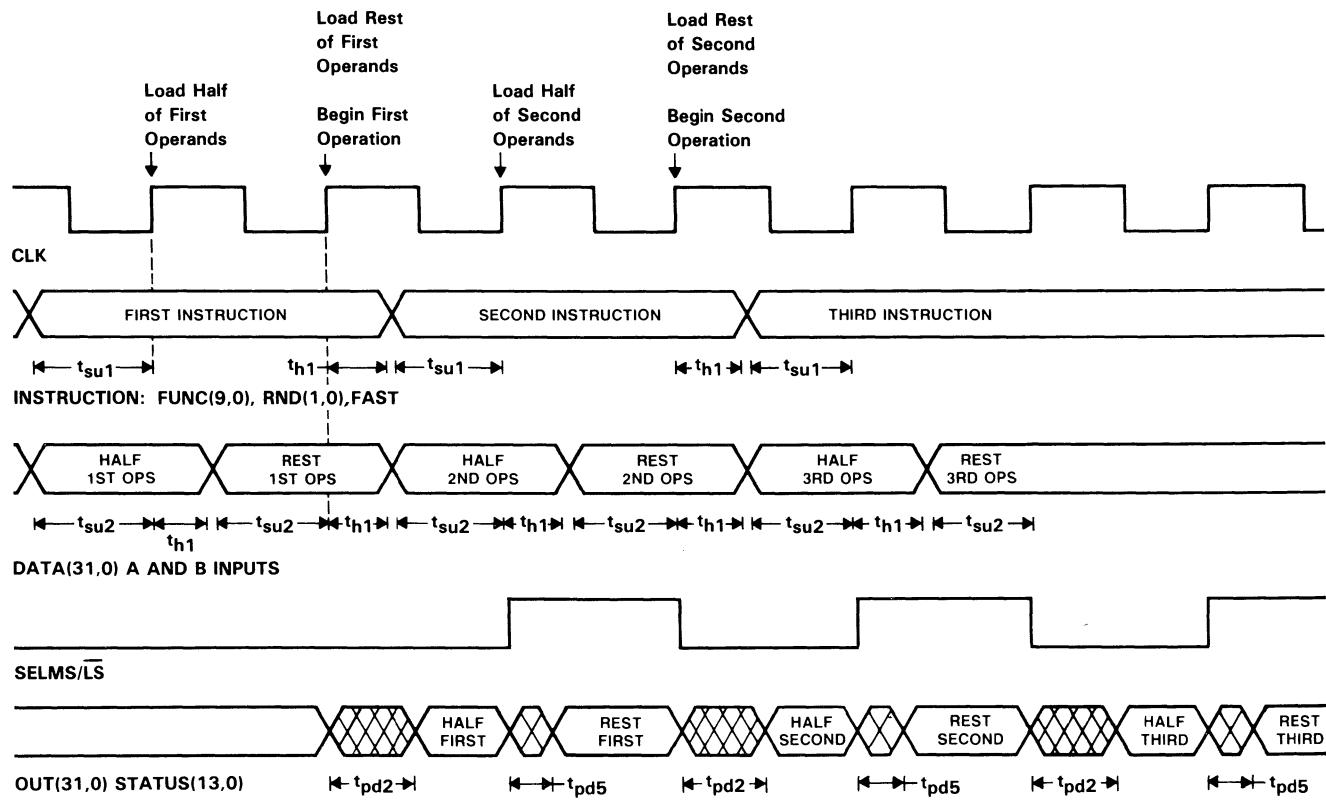


Figure 7. Double-Precision ALU Operation, Input Registers Enabled
 (PIPES = 110, CLKMODE = 0)

SN74ACT8837



Both the input and output registers are enabled (PIPES2-PIPES0 = 010) in the third example. The instruction sets up the ALU to wrap a denormalized number on the DA input bus. The wrapped output can be fed back from the S register to the multiplier input multiplexer by a later microinstruction. Timing for this operation is shown in Figure 8.

CLKMODE = 0 PIPES = 010 Operation: Wrap Denormal Input

C CCC	S S	S
L O O P P	S S	E
K N N I I	E E	L
M F F P P	L L	M
O I I E E	O O	S S
I I	D G G S S	B E E R
9-0	E 1-0 2-0	Y L L E H
	7-0	O O O T S S S A T T
	1-0	D D S R R C L E E E E T T E L P P
		T A B C S Y C S P 1-0 T T 1-0

01 1010 1000 0 01 010 xxxx 11xx 00 0 1 1 0 x 0 0 0 x 11 1 1 11

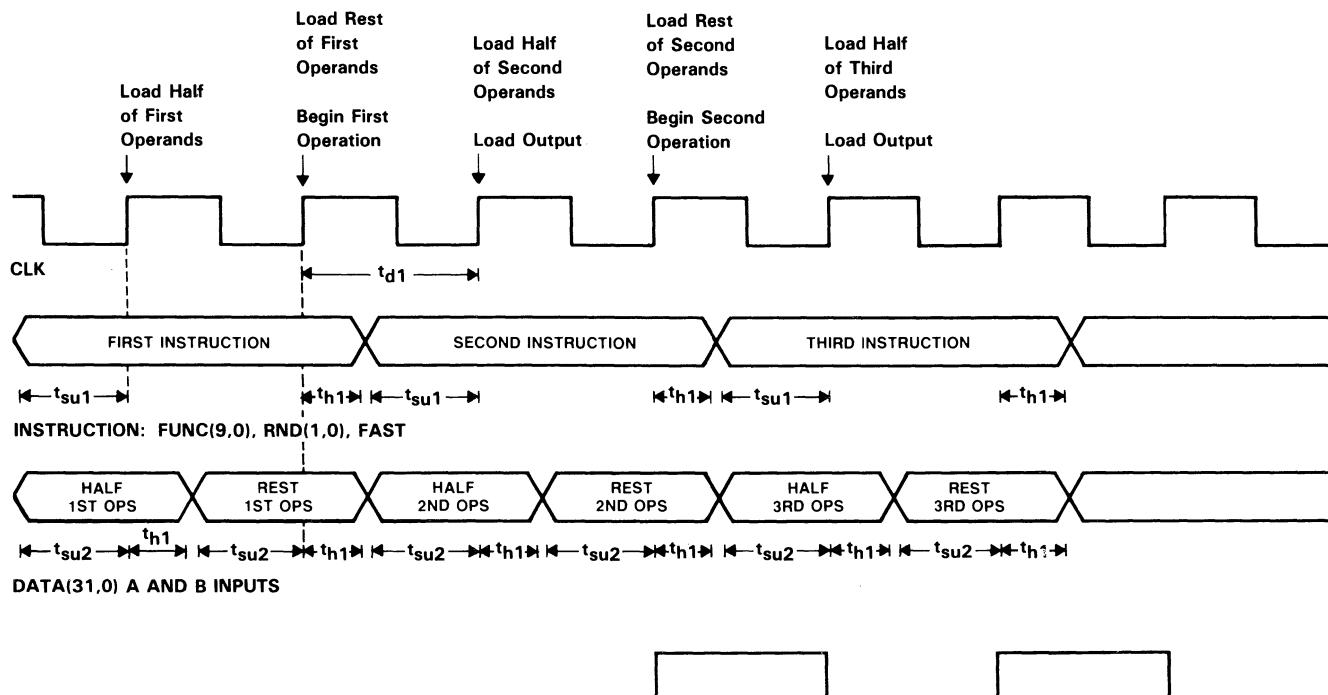


Figure 8. Double-Precision ALU Operation, Input and Output Registers Enabled
 (PIPES = 010, CLKMODE = 0)

SN74ACT8837



In the fourth example with CLKMODE = L, all three levels of internal registers are enabled. The instruction converts a double-precision integer operand to a double-precision floating-point operand. Figure 9 shows the timing for this operating mode.

CLKMODE = 0 PIPES = 000 Operation: Convert Integer to Floating Point

C C C		S	
L O O P P	S S	E	
K N N I I	E E	L	
M F F P P	L L	M	S S
O I I E E	O O	R R	B E E R
I I	D G G S S	F E E S /	Y L L E H
9-0	E 1-0 2-0	N N A N N R	O O O T S S S A T T
		D D S R R C L E E E E T T E L P P	
		1-0 T A B C S Y C S P 1-0 T T 1-0	

01 1010 0010 0 11 000 xxxx 1100 00 0 1 1 0 x 0 0 0 x 11 1 1 1 1

5

SN74ACT8837

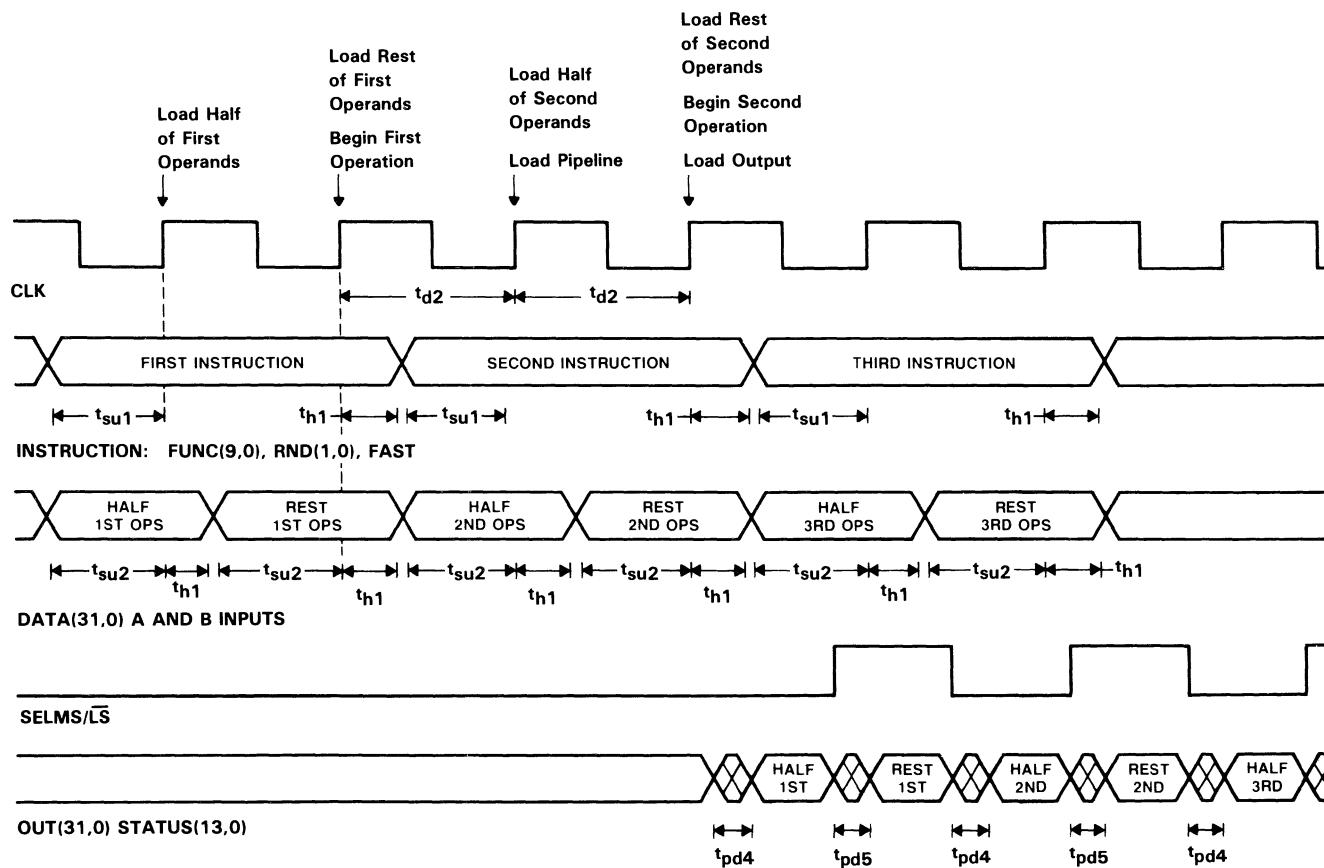


Figure 9. Double-Precision ALU Operation, All Registers Enabled
(PIPES = 000, CLKMODE = 0)

SN74ACT8837



Double-Precision ALU Operations with CLKMODE = 1

The next four examples are similar to the first four except that CLKMODE = H so that the temporary register loads on the falling edge of the clock. When the ALU is operating independently, setting CLKMODE high enables loading of both double-precision operands on successive falling and rising clock edges.

In this clock mode a double-precision ALU operation requires one clock cycle to load data inputs and execute, and both halves of the 64-bit result must be read out on the 32-bit Y bus within one clock cycle. The settings of PIPES2-PIPES0 determine the number of clock cycles which elapse between data input and result output.

In the first example all registers are disabled (PIPES2-PIPES0 = 111), and the addition is performed in flowthrough mode. As shown in Figure 10, a falling clock edge is needed to load half of the operands into the temporary register prior to loading the RA and RB registers on the next rising clock.

CLKMODE = 1 PIPES = 111 Operation: Add A + |B|

S
E
L
M S S
S B E E R
Y L L E H
T S S S A T T
D E E E E T T E L P P
A B C S Y C S P 1-0 T T 1-0

C C C L O O P P S S K N N I I E E M F F P P L L O I I E E O O I I D G G S S 9-0 E 1-0 2-0	R R F E E S / N N A N N R O O O T S S S A T T D D S R R R C L 1-0 T A B C S Y C S P	0 1 1 0 x 0 0 0 x x x 1 1 1 1
01 1000 1000 1 11 111 xxxx 1111 00 0 1 1 0 x 0 0 0 x xx 1 1 11		

5

SN74ACT8837

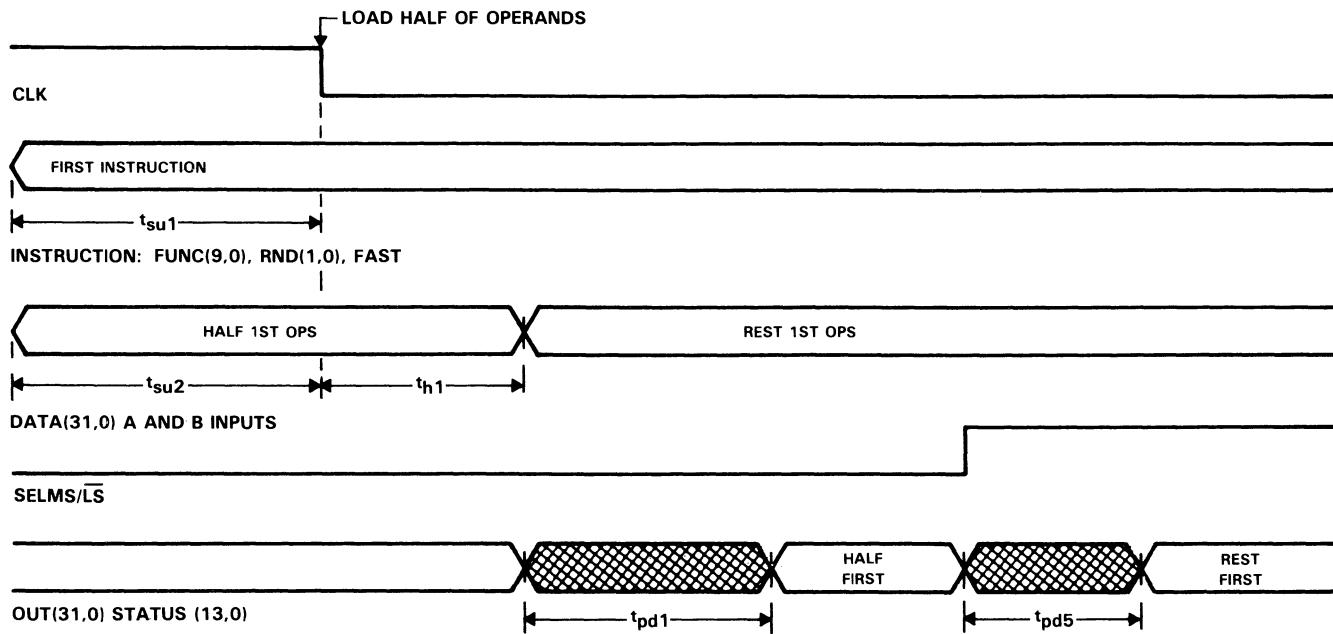


Figure 10. Double-Precision ALU Operation, All Registers Disabled
(PIPES = 111, CLKMODE = 1)



The second example executes subtraction of absolute values for both operands. Only the RA and RB registers are enabled (PIPS2-PIPES0 = 110). Timing is shown in Figure 11.

CLKMODE = 1 PIPES = 110 Operation: Subtract $|B| - |A|$

C C C	S
L O O P P	E
K N N I I	L
M F F P P	M
O I I E E	S S
I I	M
9-0	S
D G G S S	B E E R
E 1-0 2-0	Y L L E H
7-0	N N A N N R
D D S R R C	O O O T S S S A T T
T A B C S Y C S P	L E E E E T T E L P P
1-0	1-0
T T 1-0	T T 1-0

01 1001 1011 1 11 110 xxxx 1111 00 0 1 1 0 x 0 0 0 x xx 1 1 11

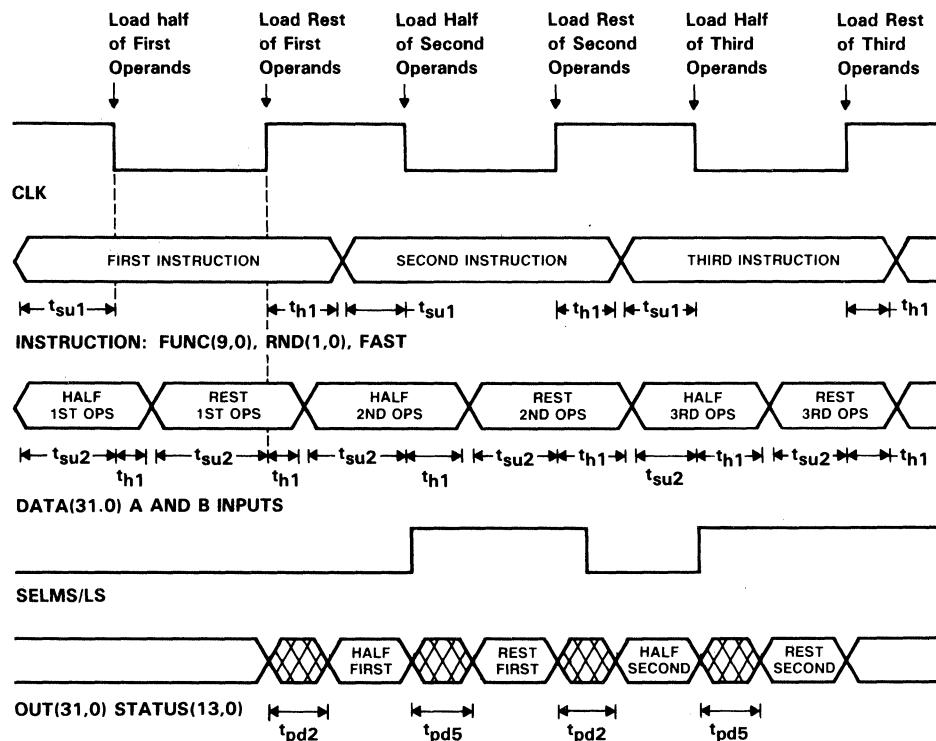


Figure 11. Double-Precision ALU Operation, Input Registers Enabled
(PIPS = 110, CLKMODE = 1)

The third example shows a single denormalized operand being wrapped so that it can be input to the multiplier. Both input and output registers are enabled (PIPES2-PIPES0 = 010). Timing is shown in Figure 12.

CLKMODE = 1 PIPES = 010 Operation: Wrap Denormal Input

C C C	S
L O O P P	E
K N N I I	L
M F F P P	M
O I I E E	S S
I I	B E E \bar{R}
9-0	Y L L E H
E 1-0 2-0	N N A N N R
7-0	D D S R R C
	L E E E E T T E L P P
	1-0 T A B C S Y C S P
	1-0 T T 1-0

01 1010 1000 1 11 010 xxxx 11xx 00 0 1 0 0 x 0 0 0 x xx 1 1 11

SN74ACT8837

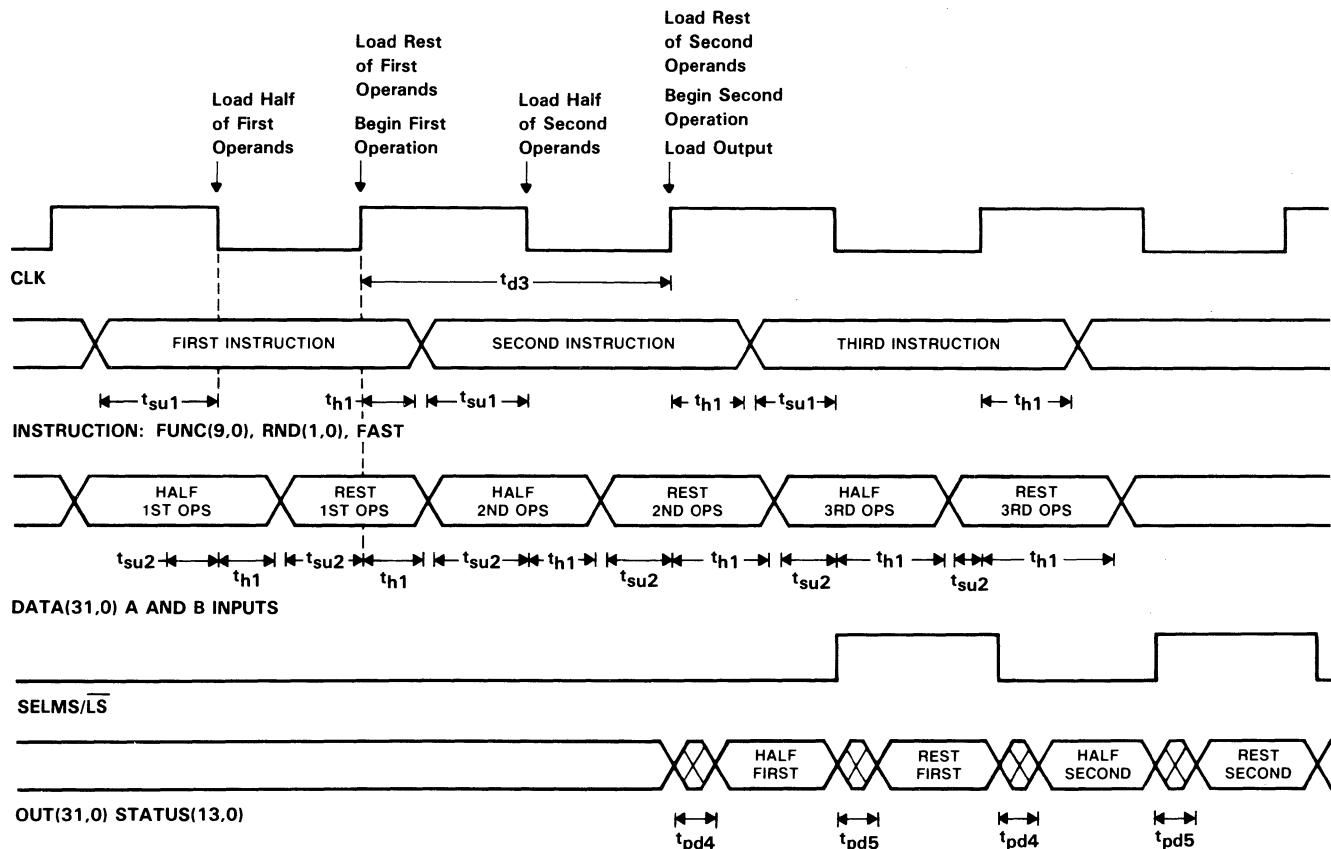


Figure 12. Double-Precision ALU Operation, Input and Output Registers Enabled
(PIPS = 010, CLKMODE = 1)

The fourth example shows a conversion from integer to floating point format. All three levels of data registers are enabled (PIPES2-PIPES0) so that the FPU is fully pipelined in this mode (see Figure 13).

CLKMODE = 1 PIPES = 000 Operation: Convert Integer to Floating Point

C C C		S	
L O O P P P	S S	E	
K N N I I	E E	L	
M F F P P	L L	M	S S
O I I E E	O O	S	B E E R
I I	D G G S S	R R F E E S /	Y L L E H
9-0	E 1-0 2-0	N N A N N R	— — — T S S S A T T
	7-0	D D S R R C L	— — — E E E E T T E L P P
		1-0 T A B C S Y C S P	1-0 T T 1-0

01 1010 0010 1 11 000 xxxx 1100 00 0 1 1 x x 0 0 0 x xx 1 1 11

SN74ACT8837

15

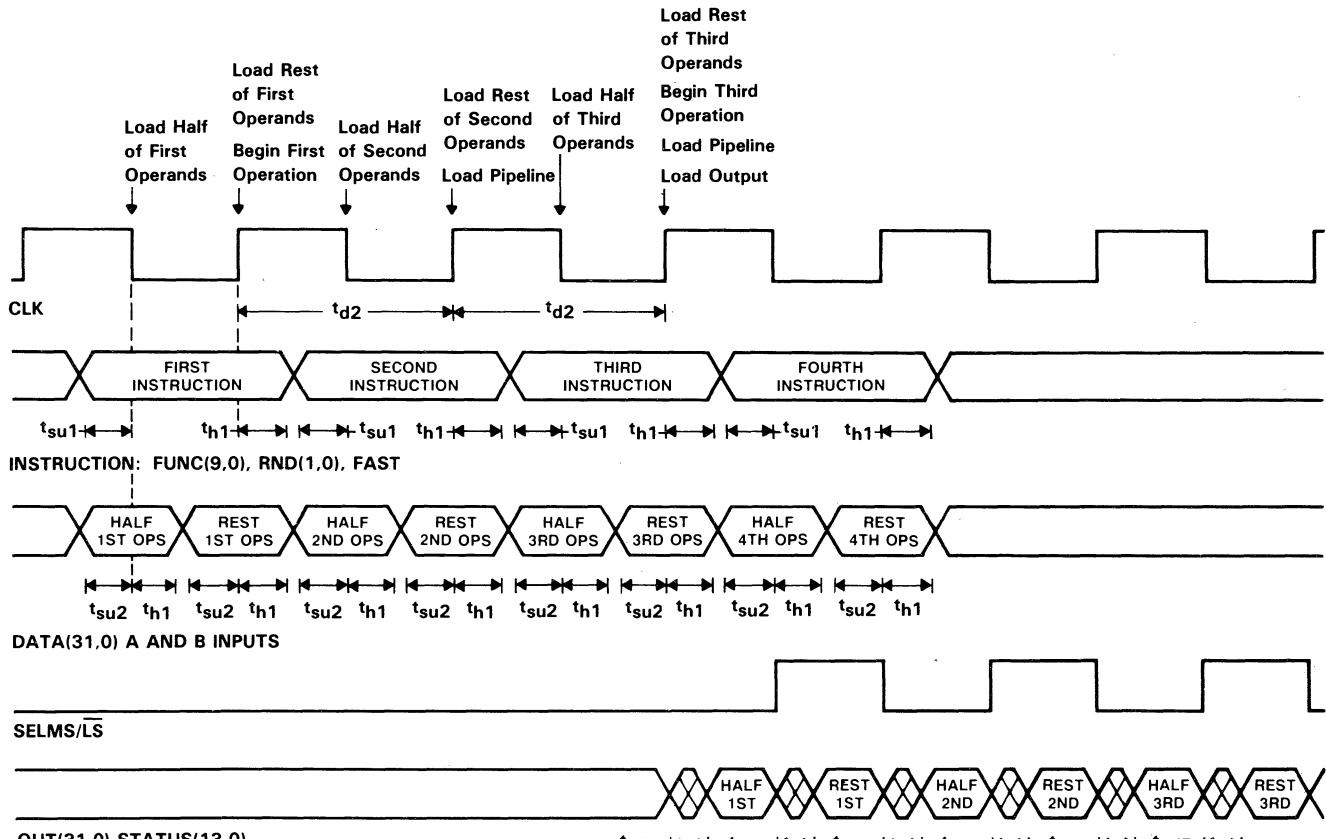


Figure 13. Double-Precision ALU Operation, All Registers Enabled
(PIPES = 000, CLKMODE = 1)

Double-Precision Multiplier Operations

Independent multiplier operations may also be performed in either clock mode and with various registers enabled. As before, examples for the two clock modes are treated separately. A double-precision multiply operation requires two clock cycles to execute (except in flowthrough mode) and from one to three other clock cycles to load the temporary register and to output the results, depending on the setting of PIPES2-PIPES0.

Even in flowthrough mode (PIPES2-PIPES0 = 111) two clock edges are required, the first to load half of the operands in the temporary register and the second to load the intermediate product in the multiplier pipeline register. Depending on the setting of CLKMODE, loading the temporary register may be done on either a rising or a falling edge.

Double-Precision Multiplication with CLKMODE = 0

In this first example, the A operand is multiplied by the absolute value of B operand. Timing for the operation is shown in Figure 14:

CLKMODE = 0 PIPES = 111 Operation: Multiply A * |B|

C C C L O O P P K N N I I M F F P P O I I E E I I 9-0	S S E E R R F E E S / N N A N N R D D S R R C L T A B C S Y C S P 1-0 2-0 7-0	S E L M S B E E R Y L L E H O O O T S S S A T T D D E E E T T E L P P 1-0 T T T 1-0	S S B E E R Y L L E H O O O T S S S A T T D D E E E T T E L P P 1-0 T T T 1-0	S E L M S B E E R Y L L E H O O O T S S S A T T D D E E E T T E L P P 1-0 T T T 1-0
---	--	--	--	--

01 1100 1000 0 11 111 1111 xxxx 00 0 x x x x 0 0 0 x xx 1 1 11

5

SN74ACT8837

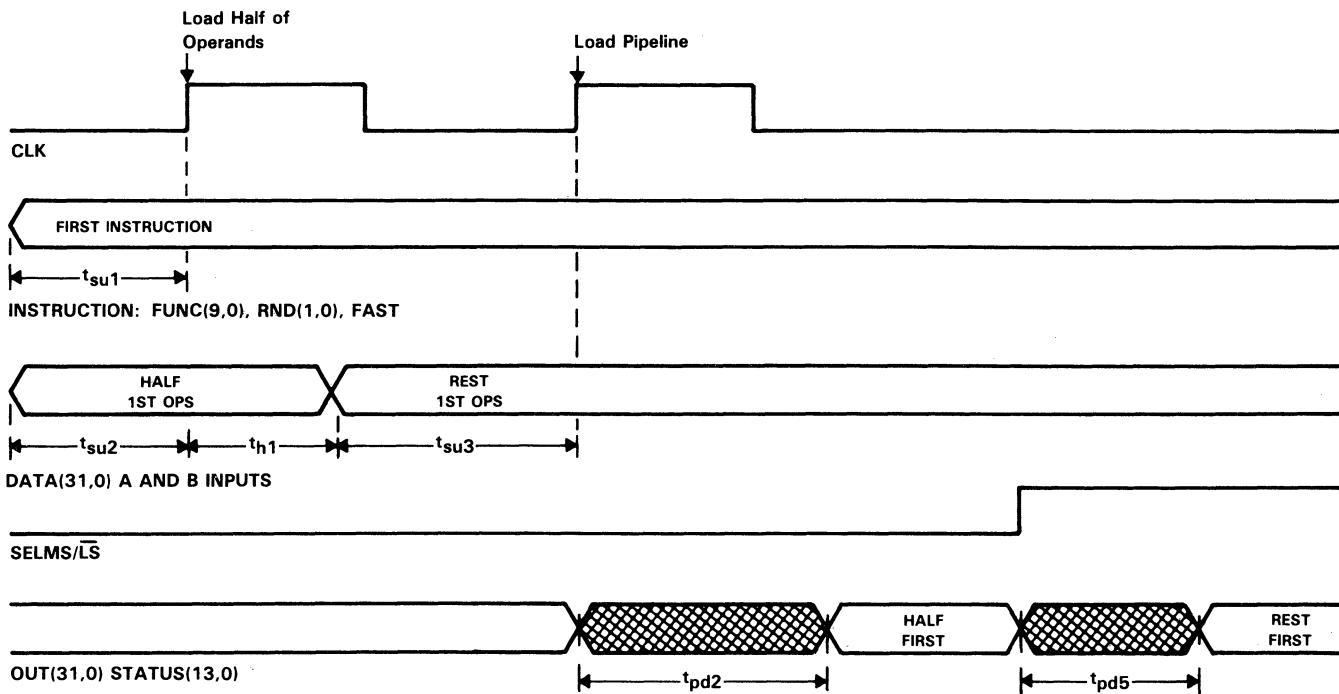


Figure 14. Double-Precision Multiplier Operation, All Registers Disabled
(PIPES = 111, CLKMODE = 0)

The second example assumes that the RA and RB input registers are enabled. With CLKMODE = 0 one clock cycle is required to input both the double-precision operands. The multiplier is set up to calculate the negative product of $|A|$ and B operands:

CLKMODE = 0 PIPES = 110 Operation: Multiply $-(|A| * B)$

$\begin{array}{c} \text{C C C} \\ \text{L O O P P} \\ \text{K N N I I} \\ \text{M F F P P} \\ \text{O I I E E} \\ \text{I I} \\ \text{9-0} \end{array}$	$\begin{array}{c} \text{S S} \\ \text{E E} \\ \text{L L} \\ \text{R R F E E S /} \\ \text{O O N N A N N R} \\ \text{D G G S S P P} \\ \text{E 1-0 2-0 7-0} \end{array}$	$\begin{array}{c} \text{S} \\ \text{E} \\ \text{L} \\ \text{M} \\ \text{S} \\ \text{B E E \bar{R}} \\ \text{Y L L E \bar{H}} \\ \text{\overline{O O O} T S S S A T T} \\ \text{D D S R R C \bar{L} E E E E T T E L P P} \\ \text{T A B C S Y C S P 1-0 T T 1-0} \end{array}$
---	---	--

01 1101 0100 0 11 110 1111 xxxx 00 0 1 1 x x 0 0 0 x xx 1 1 11

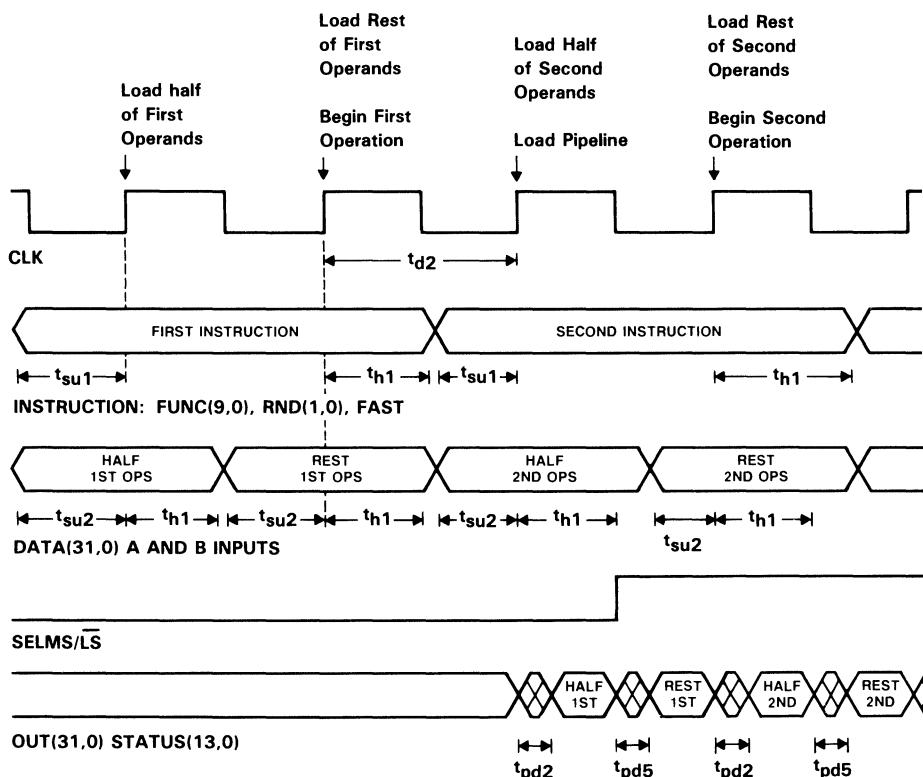


Figure 15. Double-Precision Multiplier Operation, Input Registers Enabled
(PIPES = 110, CLKMODE = 0)

Enabling both input and output registers in the third example adds an additional delay of one clock cycle, as can be seen from Figure 16. The sample instruction sets up calculation of the product of |A| and |B|:

CLKMODE = 0 PIPES = 010 Operation: Multiply |A| * |B|

C CCC L O O P P S S K N N I I E E M F F P P L L R R F E E S / Y L L E H O I I E E O O N N A N N R <u>O</u> <u>O</u> <u>O</u> T S S S A T T I I D G G S S P P D D S R R C L E E E E T T E L P P 9-0 E 1-0 2-0 7-0 1-0 T A B C S Y C S P 1-0 T T 1-0	S E L M S S S B E E <u>R</u> R R F E E S / Y L L E H N N A N N R <u>O</u> <u>O</u> <u>O</u> T S S S A T T D D S R R C L E E E E T T E L P P T A B C S Y C S P 1-0 T T 1-0
--	--

01 1101 1000 0 10 010 1111 xxxx 00 0 1 1 x x 0 0 0 x xx 1 1 11

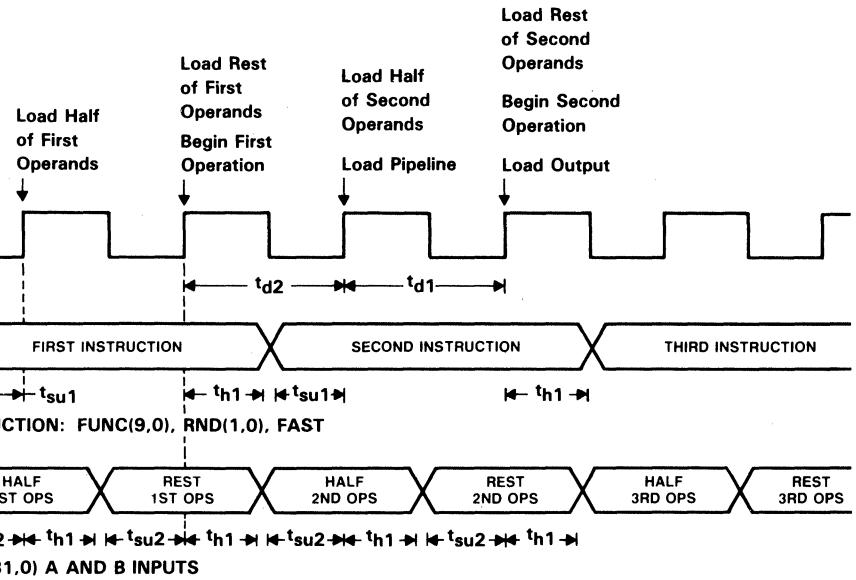


Figure 16. Double-Precision Multiplier Operation, Input and Output Registers Enabled (PIPES = 010, CLKMODE = 0)

With all registers enabled, the fourth example shows a microinstruction to calculate the negated product of operands A and B:

CLKMODE = 0 PIPES = 000 Operation: Multiply –(A * B)

C C C	S
L O O P P	E
K N N I I	L
M F F P P	M
O I I E E	S S
I I	B E E \bar{R}
9-0	Y L L E H
E 1-0 2-0	0 0 0 T S S S A T T
7-0	D D S R R C L E E E E T T E L P P
1-0	T A B C S Y C S P 1-0 T T 1-0

01 1100 0100 0 01 000 1111 xxxx 00 0 1 1 x x 0 0 0 x xx 1 1 11

SN74ACT837

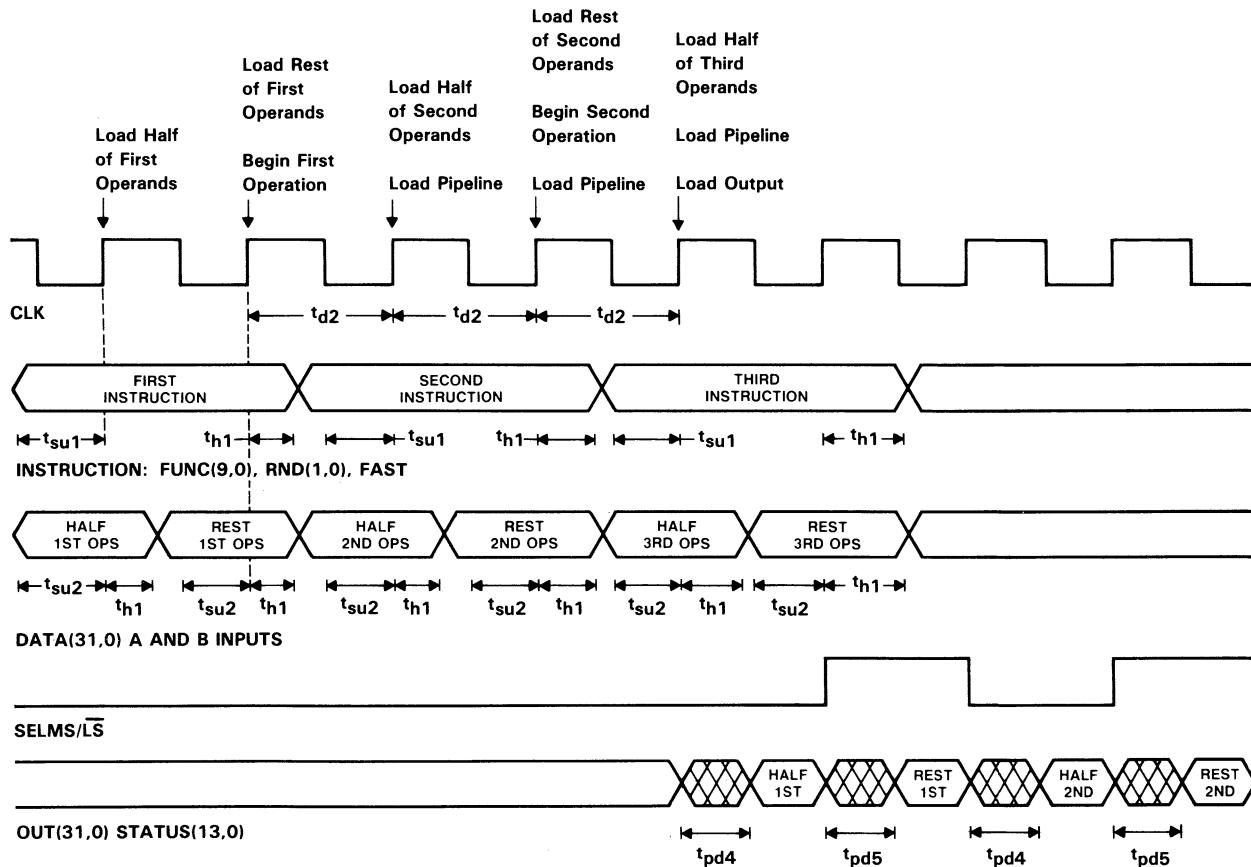


Figure 17. Double-Precision Multiplier Operation, All Registers Enabled
(PIPES = 000, CLKMODE = 0)

Double-Precision Multiplication with CLKMODE = 1

Setting the CLKMODE control high causes the temporary register to load on the falling edge of the clock. This permits loading both double-precision operands within the same clock cycle. The time available to output the result is also affected by the settings of CLKMODE and PIPES2-PIPES0, as shown in the individual timing waveforms.

The first multiplication example with CLKMODE set high shows a multiplication in flowthrough mode (PIPES2-PIPES0 = 111). Figure 18 shows the timing for this operating mode:

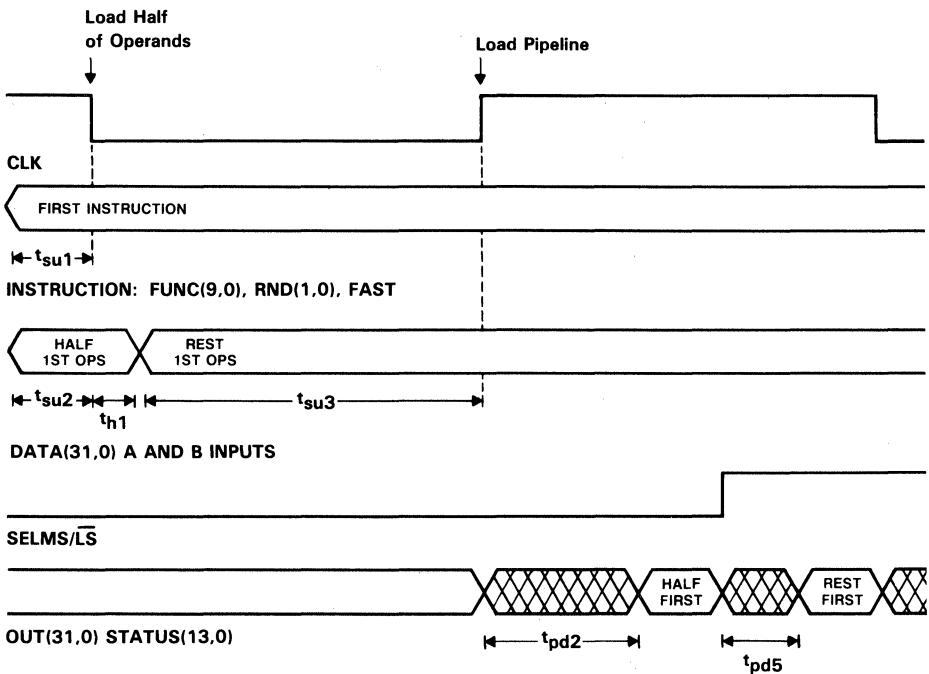
CLKMODE = 1 PIPES = 111 Operation: Multiply A * |B|

C C C L O O P P K N N I I M F F P P I I 9-0	S S E E L L R R F F E E S / O O O N N A N N R D D S R R C T A B C S Y C S P 1-0 2-0 7-0 1-0	S E L M S B E E R Y L L E H T S S S A T T E E E E T T E L P P 1-0 T T 1-0
--	--	--

01 1100 1000 1 11 111 1111 xxxx 00 0 x x x x 0 0 0 x xx 1 1 11

5

SN74ACT8837



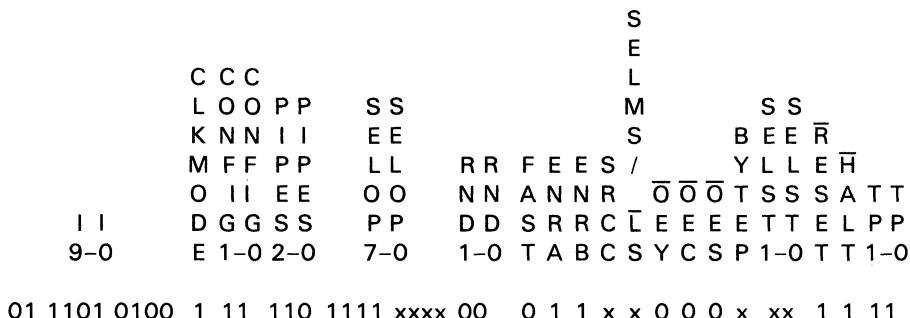
5

SN74ACT8837

Figure 18. Double-Precision Multiplier Operation, All Registers Disabled
(PIPES = 111, CLKMODE = 1)

In the second example, the input registers are enabled and the instruction is otherwise similar to the corresponding example for CLKMODE = 0. Timing is shown in Figure 19.

CLKMODE = 1 PIPES = 110 Operation: Multiply $-(|A| * B)$



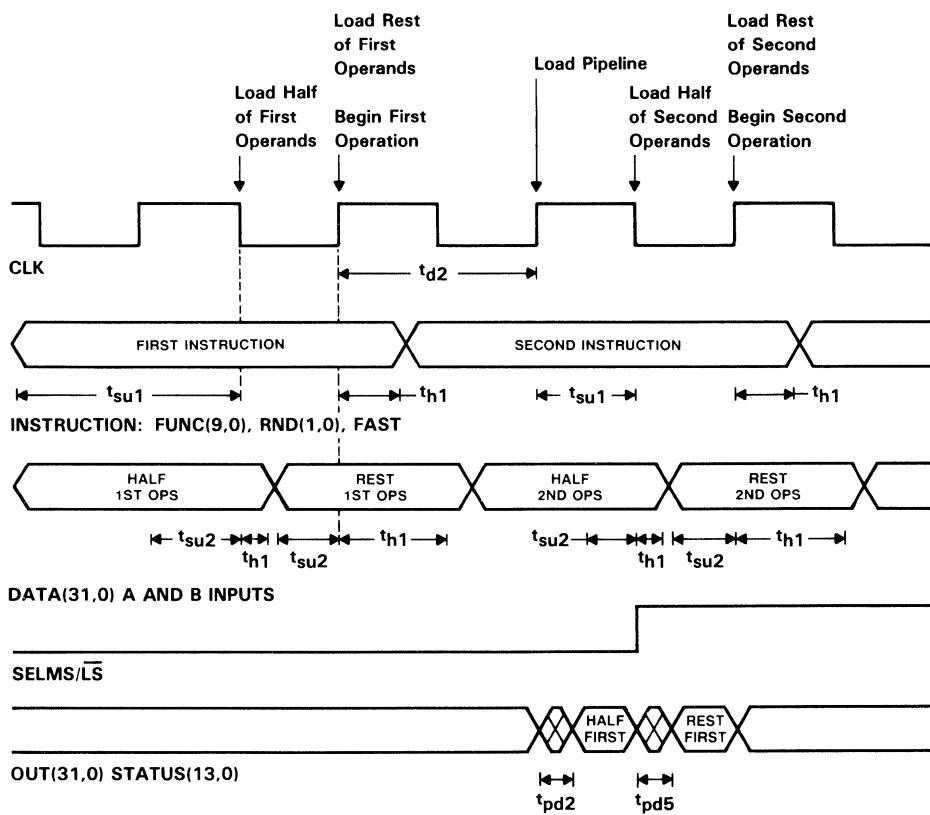


Figure 19. Double-Precision Multiplier Operation, Input Registers Enabled
(PIPES = 110, CLKMODE = 1)

With both input and output registers pipelined, the third example calculates the product of $|A|$ and $|B|$. Enabling the output register introduces a one-cycle delay in outputting the result (see Figure 20):

CLKMODE = 1 PIPES = 010 Operation: Multiply |A| * |B|

				S
				E
C C C			L	
L O O P P P	S S		M	S S
K N N I I I	E E		S	B E E R
M F F F P P P	L L	R R F E E S /		Y L L E H
O I I E E E	O O	N N A N N R	— — — O O O T S S S A T T	
I I	D G G S S S	P P	D D S R R C L E E E E T T E L P P	
9-0	E 1-0 2-0	7-0	1-0 T A B C S Y C S P 1-0 T T 1-0	

01 1101 1000 1 11 010 1111 xxxx 00 0 1 1 x x 0 0 0 x xx 1 1 11

5

SN74ACT8837

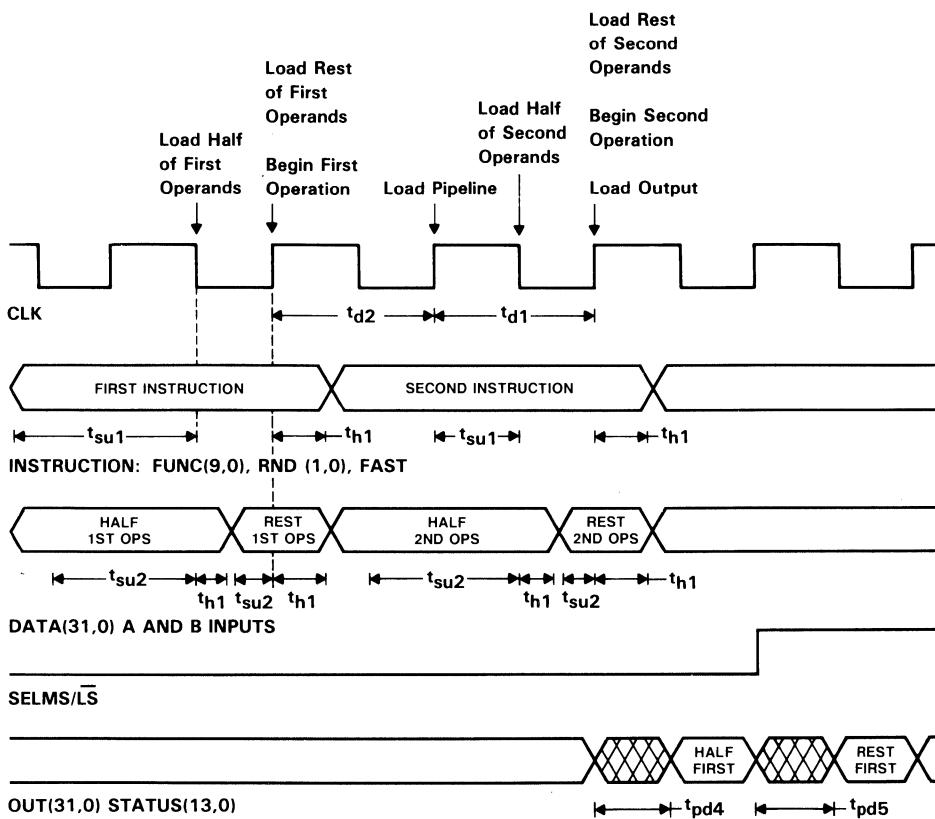


Figure 20. Double-Precision Multiplier Operation, Input and Output Registers Enabled
(PIPES = 010, CLKMODE = 1)

The fourth example shows the instruction and timing (Figure 21) to generate the negated product of the A and B operands. This operating mode with CLKMODE set high and all registers enabled permits use of the shortest clock period and produces the most data throughput, assuming that this is the primary operating mode in which the device is to function.

Additional considerations affecting timing and throughput are discussed in the sections on mixed operations and operands.

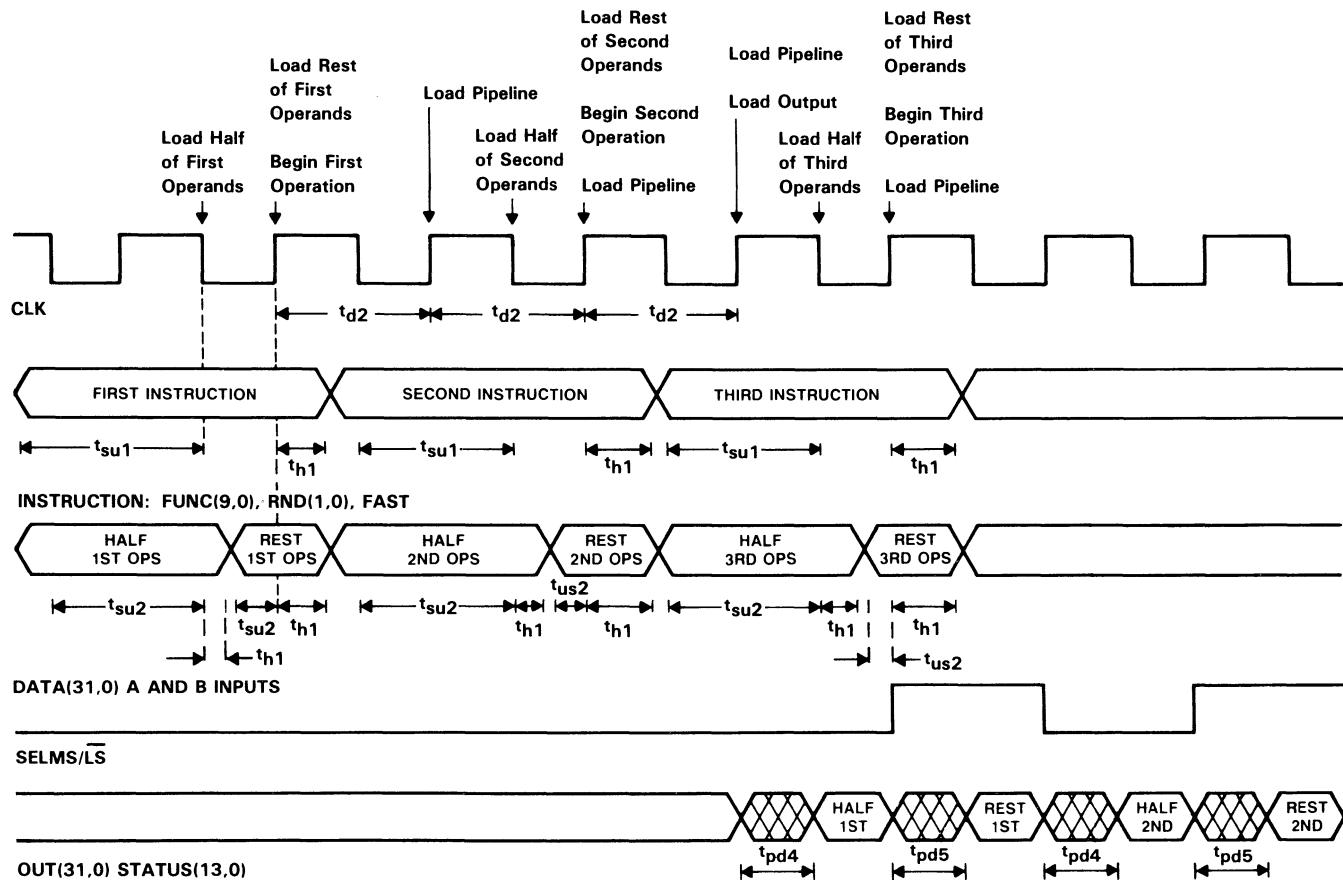
CLKMODE = 1 PIPES = 000 Operation: Multiply - (A * B)

			S
			E
C C C			L
L O O P P P	S S		M S S
K N N I I I	E E		S B E E R
M F F P P P	L L	R R F E E S /	Y L L E H
O I I E E E	O O	N N A N N R	O O O T S S S A T T
I I I	D G G S S S	P P P	D D S R R C L L E E E E T T E L L P P
9-0	E 1-0 2-0	7-0	1-0 T A B C S Y C S P 1-0 T T T 1-0

01 1100 0100 1 11 000 1111 xxxx 00 0 1 1 x x 0 0 0 x xx 1 1 11

5

SN74ACT8837



**Figure 21. Double-Precision Multiplier Operation, All Registers Enabled
(PIPS = 000, CLKMODE = 1)**

SN74ACT8837

Chained Multiplier/ALU Operations

Simultaneous multiplier and ALU functions can be selected in chained mode to support calculation of sums of products or products of sums. Operations selectable in chained mode (see Table 25) overlap partially with those selectable in independent multiplier or ALU operating mode. Format conversions, absolute values, and wrapping or unwrapping of denormal numbers are not available in chained mode.

To calculate sums of products, the FPU can operate on external data inputs in the multiplier while the ALU operates on feedback from the previous calculation. The operand selects SELOPS7-SELOPS0 can be set to select multiplier inputs from the RA and RB registers and ALU inputs from the P and S registers.

This mode of chained multiplier and ALU operation is used repeatedly in the division and square root calculations presented later. The sample microinstruction sequence shown in Tables 27 and 28 performs the operations for multiplying sets of data operands and accumulating the results, the basic operations involved in computing a sum of products.

Table 27 represents the operations, clock cycles, and register contents for a single-precision sum of four products. Registers used include the RA and RB input registers and the product (P) and sum (S) registers.

Table 27. Single-Precision Sum of Products (PIPES2-PIPES0 = 010)

CLOCK CYCLE	MULTIPLIER/ALU OPERATIONS	PSEUDOCODE
1	Load A, B A * B	A → RA, B → RB
2	Pass P(AB) to S Load C, D C * D	C → RA, D → RB A * B → P(AB)
3	S(AB) + P(CD) Load E, F E * F	P(AB) + 0 → S(AB) E → RA, F → RB C * D → P(CD)
4	S(AB + CD) + P(EF) Load G, H G * H	S(AB) + P(CD) → S(AB + CD) G → RA, H → RB E * F → P(EF)
5	S(AB + CD) + EF + P(GH)	S(AB + CD) + P(EF) → S(AB + CD + EF) G * H → P(GH)
6	New Instruction	S(AB + CD + EF) + P(GH) → S(AB + CD + EF + GH)

A microcode sequence to generate this sum of product is shown in Table 28. Only three instructions in chained mode are required, since the multiplier begins the calculation independently and the ALU completes it independently.

Table 28. Sample Microinstructions for Single-Precision Sum of Products

C	C	C	S	E	L	M	S	B	E	R
L	O	O	P	P	S	S				
K	N	N	I	I	E	E				
M	F	F	P	P	L	L	R	F	E	S /
O	I	I	E	E	O	O	N	N	A	N R
I	I	D	G	G	S	S	P	D	S	R R C L E E E E T T E L P P
9-0		E	1-0	2-0	7-0		1-0	T	A	B C S Y C S P 1-0 T T 1-0
xx	xxxx	xxxx	x	xx	xxx	xxxx	xxxx	xx	x	xx
00	0100	0000	0	01	010	1111	xxxx	00	0	1 1 x x x x x x x xx 1 1 11
10	0110	0000	0	01	010	1111	xxxx	00	0	1 1 x x x x x x x xx 1 1 11
10	0000	0000	0	01	010	1111	1010	00	0	1 1 x x x x x x x xx 1 1 11
10	0000	0000	0	01	010	xxxx	1010	00	0	1 1 x x x x x x x xx 1 1 11
00	0000	0000	0	01	010	xxxx	1010	00	0	x x x x x x x x x xx 1 1 11
xx	xxxx	xxxx	x	xx	xxx	xxxx	xxxx	xx	x	xx

Fully Pipelined Double-Precision Operations

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.

5

Pseudocode tables for these sequences are provided, (Table 29 and Table 30) 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 HL to load operands by MSH and LSH, all registers enabled (PIPES2 – PIPES0 = LLL), and the C register clock tied to the system clock.

SN74ACT8837

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 29. Pseudocode for Fully Pipelined Double-Precision Sum of Products (CLKM = 0, CONFIG = 10, PIPES = 000, CLKC \leftrightarrow SYSCLK)

**Table 30. Pseudocode for Fully Pipelined Double-Precision Product of Sums
(CLKM = 0, CONFIG = 10, PIPES = 000, CLKC↔SYSCLK)**

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(M)	B1(M)	A1,B1(M)	A1 + B1									
2	A1(L)	B1(L)	A1,B1(M)	A1 + B1	A1 + B1	A1	B1						
3	A2(M)	B2(M)	A2,B2(M)	A2 + B2	A1 + B1	A1	B1				A1 + B1		
4	A2(L)	B2(L)	A2,B2(M)	A2 + B2	A2 + B2	A2	B2				A1 + B1	S1	
5	A3(M)	B3(M)	A3,B3(M)	CR * SR A3 + B3	A2 + B2	A2	B2			S1	A2 + B2	S1	
6	A3(L)	B3(L)	A3,B3(M)	CR * SR A3 + B3	CR * SR A3 + B3	A3	B3			S1	A2 + B2	S2	
7	XXX	XXX	XXX	SP Add	CR * SR A3 + B3	A3	B3	S1 * S2		S1	A3 + B3	S2	
8	A4(M)	B4(M)	A4,B4(M)	PR * SR A4 + B4	CR * SR A3 + B3	ENRA = L A3	ENRB = L B3	S1 * S2		S1	A3 + B3	XXX	
9	A4(L)	B4(L)	A4,B4(M)	PR * SR A4 + B4	PR * SR A4 + B4	A4	B4	XXX	P1	S1	XXX	S3	
10	XXX	XXX	XXX	SP Add	PR * SR A4 + B4	A4	B4	P1 * S3	P1	S1	A4 + B4	S3	
11	A5(M)	B5(M)	A5,B5(M)	PR * SR A5 + B5	PR * SR A4 + B4	ENRA = L A4	ENRB = L B4	P1 * S3	XXX	S1	A4 + B4	XXX	
12	A5(L)	B5(L)	A5,B5(M)	PR * SR A5 + B5	PR * SR A5 + B5	A5	B5	XXX	P2	S1	XXX	S4	

NOTE: On CLK 7 and CLK10, put 0000000000 (Single-Precision Add) on the instruction bus.

SN74ACT8837



In the product of sums timing table, the two initial sums are generated in independent ALU mode. The remaining operations are shown as alternating chained operations followed by single-precision adds. The SP adds are necessary to provide an extra cycle during which the multiplier outputs the current intermediate product. The current sum and the latest intermediate product are then fed back to the multiplier inputs for the next chained operations. In this manner, a double-precision product of sums is generated in three system clocks, as opposed to two clocks for a double-precision sum of products.

Mixed Operations and Operands

Using mixed-precision data operands or performing sequences of mixed operations may require adjustments in timing, operand precision, and control settings. To simplify microcoding sequences involving mixed operations, mixed-precision operands, or both, it is useful to understand several specific requirements for mixed-mode or mixed-precision processing.

Calculations involving mixed-precision operands must be performed as double-precision operations (see Table 12). The instruction settings (I8-I7) should be set to indicate the precision of each operand from the RA and RB input registers. (Feedback operands from internal registers are also double-precision.) Mixed-precision operations should not be performed in chained mode.

5

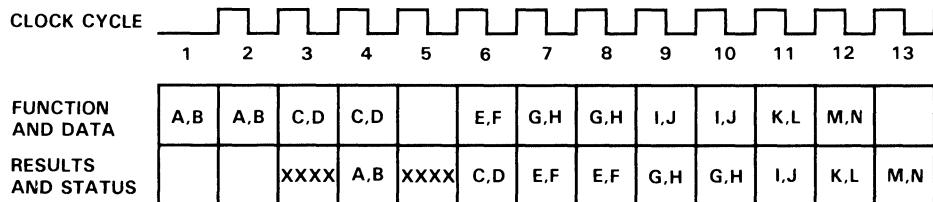
SN74ACT8837

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.

Most format conversions also involve double-precision timing. Conversions between single- and double-precision floating point format are treated as mixed-precision operations. During integer to floating point conversions, the integer input should be loaded into the upper half of the RA register.

In applications where mixed-precision operations is not required, it is possible to tie the I8-I7 instruction inputs together so that both controls always select the same precision.

Sequences of mixed operations may require changes in multiple control settings to deal with changes in timing of input, execution, and output of results. Figure 22 shows a simplified timing waveform for a series of mixed operations:



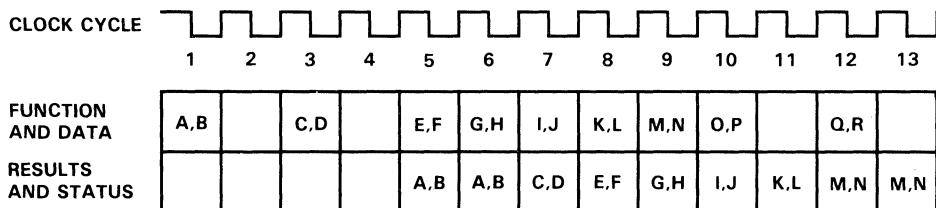
A,B,C,D — double precision multiply; E,F — single precision operation; G,H,I,J — double precision add; K,L — single precision operation. A double precision number is not required to be held on the outputs for two cycles unless it is followed by a like double precision function. If a double precision multiply is followed by single precision operation, there must be one open clock cycle.

**Figure 22. Mixed Operations and Operands
(PIPES2-PIPES0 = 110, CLKMODE = 0)**

In this sequence, the fifth cycle is left open because a single-precision multiply follows a double-precision multiply. If the SP multiply were input during the period following the fourth rising clock edge, the result of the preceding operation would be overwritten, since an SP multiply executes in one clock cycle. To avoid such a condition, the FPU will not load during the required open cycle.

Because the sequence of mixed operations places constraints on output timing, only one cycle is available to output the double-precision ($C * D$) result. By contrast, the SP multiply ($E * F$) is available for two cycles because the operation which follows it does not output a result in the period following the seventh rising clock edge. In general, the precision and timing of each operation affects the timing of adjacent operations.

Control settings for CLKMODE and registers must also be considered in relation to precision and speed of execution. In Figure 23, a similar sequence of mixed operations is set up for execution in fully pipelined mode:



A,B,C,D — double precision multiply; E,F — single precision operation; G,H — double precision add; I,J,K,L,M,N — single precision operation; O,P,Q,R — double precision multiply. In clock mode 1, a double precision result is two cycles long only when a double precision multiply is followed by a double precision multiply.

**Figure 23. Mixed Operations and Operands
(PIPES2-PIPES0 = 000, CLKMODE = 1)**

Although the data operands can be loaded in one clock cycle with CLKMODE set high, enabling two additional internal registers delays the $(A * B)$ result one cycle beyond the previous example. Again, an open cycle is required after the $(C * D)$ operation because the next operation is single precision. The result of the $(C * D)$ multiply is available for one cycle instead of two, also because the following operation is single precision. With this setting of CLKMODE and PIPES2-PIPES0, a double-precision result is only available for two clock cycles when one DP multiply follows another DP multiply.

5

SN74ACT8837

Matrix Operations

The 'ACT8837 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:

$$\frac{dx_1}{dt} = x_2, \dots, \frac{dx(n-1)}{dt} = x_n$$

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

$$\frac{d}{dt} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} b_{11} & \dots & b_{1n} \\ \vdots & & \vdots \\ \vdots & & \vdots \\ b_{n1} & \dots & b_{nn} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix}$$

$$\text{or, } \dot{X} = AX + BU$$

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

$$\dot{x}_1 = (a_{11} * x_1 + \dots + a_{1n} * x_n) + (b_{11} * u_1 + \dots + b_{1n} * u_n)$$

$$\text{where } \dot{x}_1 = dx_1/dt.$$

Sequences of multiplications and additions are required when such state space transformations are performed, and the 'ACT8837 has been designed to support such sum-of-products operations. An $n \times n$ matrix A multiplied by an $n \times n$ matrix X yields an $n \times n$ matrix C whose elements c_{ij} are given by this equation:

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

For the c_{ij} elements to be calculated by the 'ACT8837, the corresponding elements a_{ik} and x_{kj} must be stored outside the 'ACT8837 and fed to the 'ACT8837 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:

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

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

$$\begin{array}{c} \vdots \\ \vdots \\ 3 \times 3 \quad : \quad x \\ \vdots \\ 1 \\ \dots \\ 1 \times 3 \quad : \quad 1 \times 1 \end{array}$$

5 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.

SN74ACT837

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:

$$[x \ y \ z \ h] = [x \ y \ z \ 1] \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -a & -b & -c & 1 \end{pmatrix} \begin{array}{c} R \\ \downarrow \end{array} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ a & b & c & 1 \end{pmatrix} \begin{array}{c} \downarrow \\ \text{rotation} \\ \text{about} \\ \text{origin} \end{array} \quad (2) \quad \begin{array}{c} \downarrow \\ \text{translation} \\ \text{to origin} \end{array} \quad \begin{array}{c} \downarrow \\ \text{translation} \\ \text{back to initial} \\ \text{position} \end{array}$$

where R may be expressed as:

$$R = \begin{bmatrix} n_1^2 + (1-n)^2 \cos\phi & n_1n_2(1-\cos\phi) + n_3\sin\phi & n_1n_3(1-\cos\phi) - n_2\sin\phi & 0 \\ n_1n_2(1-\cos\phi) - n_3\sin\phi & n_2^2 + (1-n_2)^2 \cos\phi & n_2n_3(1-\cos\phi) + n_1\sin\phi & 0 \\ n_1n_3(1-\cos\phi) + n_2\sin\phi & n_2n_3(1-\cos\phi) - n_1\sin\phi & n_3^2 + (1-n_3)^2 \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

and $n_1 = q_1/(q_1^2 + q_2^2 + q_3^2)^{1/2}$ = direction cosine for x-axis of rotation

$n_2 = q_2/(q_1^2 + q_2^2 + q_3^2)^{1/2}$ = direction cosine for y-axis of rotation

$n_3 = q_3/(q_1^2 + q_2^2 + q_3^2)^{1/2}$ = direction cosine for z-axis of rotation

$\bar{n} = (n_1 \ n_2 \ n_3)$ = unit vector for \bar{Q}

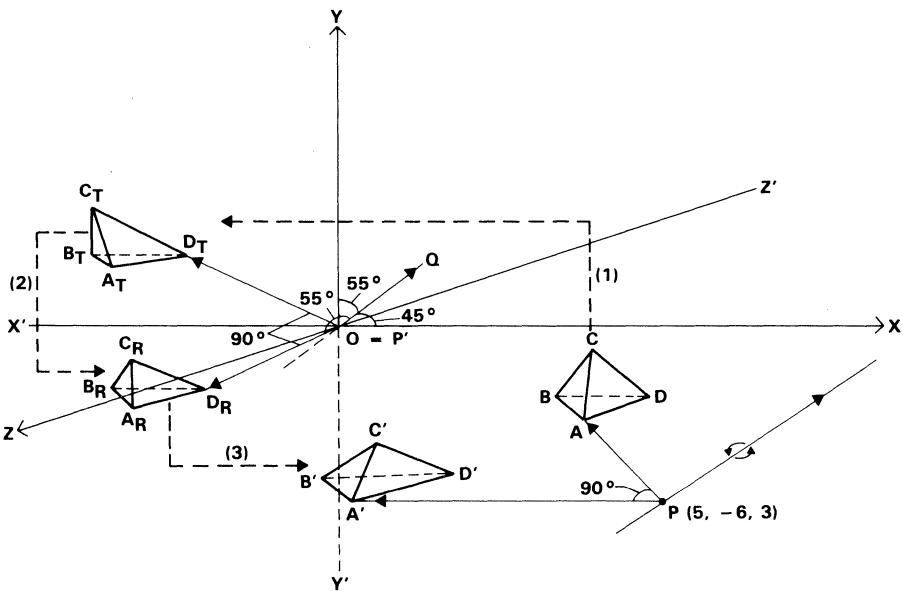
\bar{Q} = vector defining axis of rotation = $[q_1 \ q_2 \ q_3]$

ϕ = the rotation angle about \bar{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 [n_1, n_2, n_3], 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 [$n_1 = 0.866, n_2 = 0.5, n_3 = 0.707$]. The angle of rotation 0 is 90 degrees (see Figure 24). The rotation matrix [R] becomes

2	-3	3	1	→	A
1	-2	2	1	→	B
2	-1	2	1	→	C
2	-2	2	1	→	D

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



5

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

Figure 24. Sequence of Matrix Operations

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

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

$$\begin{array}{c|c|c|c}
\begin{bmatrix} 2 & -3 & 3 & 1 \\ 1 & -2 & 2 & 1 \\ 2 & -1 & 2 & 1 \\ 2 & -2 & 2 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -5 & 6 & -3 & 1 \end{bmatrix} & \begin{bmatrix} 0.750 & 1.140 & 0.112 & 0 \\ -0.274 & 0.250 & 1.22 & 0 \\ 1.112 & -0.513 & 0.500 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 5 & -6 & 3 & 1 \end{bmatrix} \\
\downarrow & \downarrow & \downarrow & \downarrow \\
\text{translation to origin} & \text{rotation about origin} & & \text{translation back to initial position}
\end{array} \quad (3)$$

The 'ACT8837 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 'ACT8837 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:

$$\begin{array}{c}
 \begin{array}{|c c c c|} \hline
 2 & -3 & 3 & 1 \\ \hline
 1 & -2 & 2 & 1 \\ \hline
 2 & -1 & 2 & 1 \\ \hline
 2 & -2 & 2 & 1 \\ \hline
 \end{array}
 \quad
 \begin{array}{|c c c c|} \hline
 1 & 0 & 0 & 0 \\ \hline
 0 & 1 & 0 & 0 \\ \hline
 0 & 0 & 1 & 0 \\ \hline
 -5 & 6 & -3 & 1 \\ \hline
 \end{array}
 \quad
 \begin{array}{|c c c c|} \hline
 (2-5) & (-3+6) & (3-3) & 1 \\ \hline
 (1-5) & (-2+6) & (2-3) & 1 \\ \hline
 (2-5) & (-1+6) & (2-3) & 1 \\ \hline
 (2-5) & (-2+6) & (2-3) & 1 \\ \hline
 \end{array}
 \\
 = \\
 \downarrow \qquad \qquad \qquad \downarrow \\
 \text{translation} \qquad \qquad \qquad \text{vertices of translated} \\
 \text{to origin} \qquad \qquad \qquad \text{tetrahedron}
 \end{array}$$

$$\begin{array}{c}
 \begin{array}{|c c c c|} \hline
 -3 & +3 & 0 & 1 \\ \hline
 -4 & +4 & -1 & 1 \\ \hline
 -3 & +5 & -1 & 1 \\ \hline
 -3 & +4 & -1 & 1 \\ \hline
 \end{array}
 \quad
 \begin{array}{l}
 \xrightarrow{} \text{AT} \\
 \xrightarrow{} \text{BT} \\
 \xrightarrow{} \text{CT} \\
 \xrightarrow{} \text{DT}
 \end{array}
 \\
 = \\
 \end{array}$$

The 'ACT8837 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:

$$\begin{array}{|c c|} \hline -3 & 3 & 0 & 1 \\ -4 & 4 & -1 & 1 \\ -3 & 5 & -1 & 1 \\ -3 & 4 & -1 & 1 \\ \hline \end{array} \quad \begin{array}{|c c c c|} \hline 0.750 & 1.140 & 0.112 & 0 \\ -0.274 & 0.250 & 1.22 & 0 \\ 1.112 & -0.513 & 0.500 & 0 \\ 0 & 0 & 0 & 1 \\ \hline \end{array} \quad = \quad \begin{array}{|c c c|} \hline -3.072 & -2.670 & 3.324 \\ -5.208 & -3.047 & 3.932 \\ -4.732 & -1.657 & 5.264 \\ -4.458 & -1.907 & 4.044 \\ \hline \end{array} \quad \begin{array}{c} \downarrow \\ \text{rotation about origin} \end{array} \quad \begin{array}{c} \downarrow \\ \text{rotated coordinates} \end{array}$$

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 produce the final coordinate matrix of the transformed object:

5

$$\begin{array}{c}
 \boxed{-3.072 \ -2.670 \ 3.324 \ 1} \\
 \boxed{-5.208 \ -3.047 \ 3.932 \ 1} \\
 \boxed{-4.732 \ -1.657 \ 5.264 \ 1} \\
 \boxed{-4.458 \ -1.907 \ 4.044 \ 1}
 \end{array}
 \quad
 \begin{array}{c}
 \boxed{1 \ 0 \ 0 \ 0} \\
 \boxed{0 \ 1 \ 0 \ 0} \\
 \boxed{0 \ 0 \ 1 \ 0} \\
 \boxed{5 \ -6 \ 3 \ 1}
 \end{array}
 \quad
 \begin{array}{c}
 = \\
 \downarrow \\
 \text{translate back}
 \end{array}
 \quad
 \begin{array}{c}
 \boxed{1.928 \ -8.670 \ 6.324 \ 1} \\
 \boxed{-0.208 \ -9.047 \ 6.932 \ 1} \\
 \boxed{0.268 \ -7.657 \ 8.264 \ 1} \\
 \boxed{0.542 \ -7.907 \ 7.044 \ 1}
 \end{array}
 \quad
 \begin{array}{c}
 \downarrow \\
 \text{final rotated coordinates}
 \end{array}$$

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:

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

5

SN74ACT8837

For example, the coordinates may be computed as follows:

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

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

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

$$\begin{aligned}h1 &= c_{14} = a_{11} * x_{14} + a_{12} * x_{24} + a_{13} * x_{34} + a_{14} * x_{44} \\&= 2 * 0 + (-3) * 0 + 3 * 0 + 1 * 1 \\&= 0 + 0 + 0 + 1 \\&= 1 \\A' &= [1.928 \quad -8.67 \quad 6.324 \quad 1]\end{aligned}$$

The other rotated vertices are computed in a similar manner:

$$\begin{aligned}B' &= [-5.208 \quad -3.047 \quad 3.932 \quad 1] \\C' &= [-4.732 \quad -1.657 \quad 5.264 \quad 1] \\D' &= [-4.458 \quad -1.907 \quad 4.044 \quad 1]\end{aligned}$$

Microinstructions for Sample Matrix Manipulation

The 'ACT8837 FPU can compute the coordinates for graphic objects over a broad dynamic range. Also, the homogeneous scalar factors h_1 , h_2 , h_3 and h_4 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.

A microcode sequence to generate this matrix multiplication is shown in Table 31. Table 32 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 31. Microinstructions for Sample Matrix Multiplication

						S					
C C C						E					
L O O P P	S S					L					
K N N I I	E E					M	S S				
M F F P P	L L	R R	F E E S /			S	B E E R				
O I I E E	O O	N N A N N R						Y L L E H			
I I D G G S S	P P	D D S R R C L E E E T T E L P P									
9-0 E 1-0 2-0	7-0	1-0 T A B C S Y C S P 1-0 T T 1-0									
00 0100 0000	0 01 0101111 xxxx	00	0 1 1 x x x x x x xx	1 1 11							
10 0110 0000	0 01 0101111 xxxx	00	0 1 1 x x x x x x xx	1 1 11							
10 0000 0000	0 01 0101111 1010	00	0 1 1 x x x x x x xx	1 1 11							
10 0000 0000	0 01 0101111 1010	00	0 1 1 x x x x x x xx	1 1 11							
10 0000 0000	0 01 0101111 1010	00	0 1 1 x x x x x x xx	1 1 11							
10 0110 0000	0 01 0101111 xxxx	00	0 1 1 x x x x x x xx	1 1 11							
10 0110 0000	0 01 0101111 xxxx	00	0 1 1 x x x x x x xx	1 1 11							
10 0000 0000	0 01 0101111 1010	00	0 1 1 x x x x x x xx	1 1 11							
10 0000 0000	0 01 0101111 1010	00	0 1 1 x x x x x x xx	1 1 11							
10 0000 0000	0 01 0101111 1010	00	0 1 1 x x x x x x xx	1 1 11							
10 0110 0000	0 01 0101111 xxxx	00	0 1 1 x x x x x x xx	1 1 11							

5

SN74ACT8837

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:

0
0
0
1

Table 32. Single-Precision Matrix Multiplication (PIPES2-PIPES0 = 010)

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

5

SN74ACT8837

The h-scalars h_1, h_2, h_3 , and h_4 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 25 summarizes the overall matrix transformation.

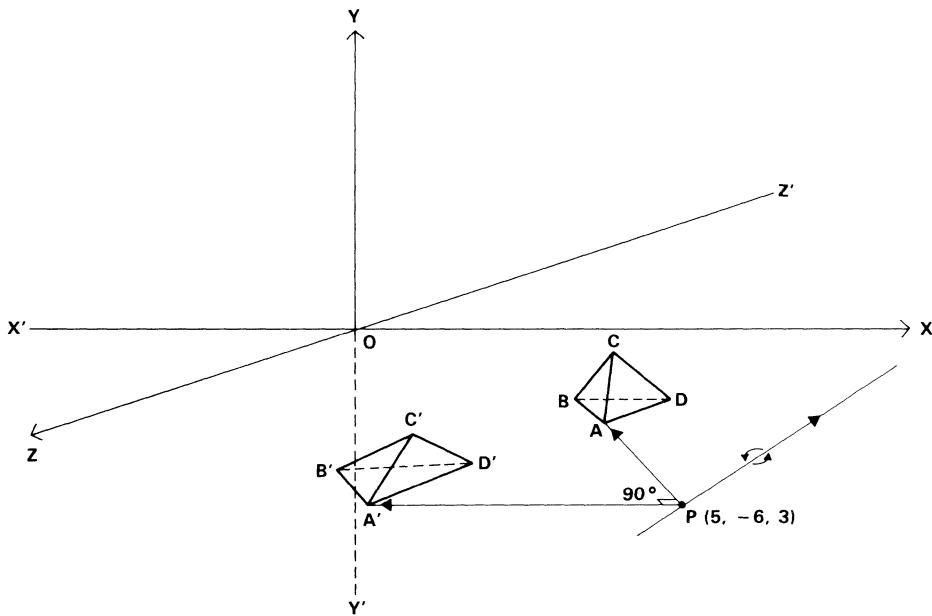


Figure 25. Resultant Matrix Transformation

5

SN74ACT8837

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 33 presents the sequence of calculations for the first two coordinates, x_a and y_a .

**Table 33. Fully Pipelined Sum of Products (PIPES2-PIPES0 = 000)
(Bus or Register Contents Following Each Rising Clock Edge)**

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					
4	Chn	x12	a11	Mul	x41	a14	p3	s1	p1			
5	Chn	x22	a12	Chn	x12	a11	p4	†	p2			
6	Chn	x32	a13	Chn	x22	a12	p5	s2	p3			
7	Chn	x42	a14	Chn	x32	a13	p6	s3	p4			
8	Chn	x13	a11	Chn	x42	a14	p7	s4	p5			
9	Chn	x23	a12	Chn	x13	a11	p8	xa	p6			
10	Chn	x33	a13	Chn	x23	a12	p9	s5	p7			
11	Chn	x43	a14	Chn	x33	a13	p10	s6	p8			
12	Chn	x14	a11	Chn	x43	a14	p11	s7	p9			
13	Chn	x24	a12	Chn	x14	a11	p12	ya	p10			
14	Chn	x34	a13	Chn	x24	a12	p13	s8	p11			
15	Chn	x44	a14	Chn	x34	a13	p14	s9	p12			
								p13	s7	p10		ya
									s8	p10		

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

Products in Table 33 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 33 are listed below:

$$\begin{array}{lll}
 s1 = p1 + 0 & s5 = p5 + p7 & s9 = p10 + p12 \\
 s2 = p1 + p3 & s6 = p6 + p8 & xa = p1 + p2 + p3 + p4 \\
 s3 = p2 + p4 & s7 = p9 + 0 & ya = p5 + p6 + p7 + p8 \\
 s4 = p5 + 0 & s8 = p9 + p11 &
 \end{array}$$

SAMPLE MICROPROGRAMS FOR BINARY DIVISION AND SQUARE ROOT

The SN74ACT8837 Floating Point Unit supports binary division and square root calculations using the Newton-Raphson algorithm. The 'ACT8837 performs these calculations by executing sequences of floating-point operations according to the control settings contained in specific microprogrammed routines. This implementation of the Newton-Raphson algorithm requires that a seed ROM provide values for the first approximations of the reciprocals of the divisors.

This application note presents several microprograms for floating-point division and square root using the Newton-Raphson algorithm. Each sample program is analyzed briefly to show details of the floating-point procedures being performed.

Binary Division Using the Newton-Raphson Algorithm

Binary division can be performed as an iterative procedure using the Newton-Raphson algorithm. For a dividend A, divisor B, and quotient Q, this procedure calculates a value for $1/B$ which is then used to evaluate the expression $Q = A * 1/B$. The calculation can be performed with either single- or double-precision operands, and examples of each precision are shown.

The basic algorithm calculates the value of a quotient Q by approximating the reciprocal of the divisor B to adequate precision and then multiplying the dividend A by the approximation of the reciprocal:

$$\begin{aligned} Q &= A/B = A * X_n, \text{ where } X_n = \text{the value of } X \text{ after the } n\text{th iteration} \\ n &= \text{the number of iterations to achieve the desired precision} \end{aligned}$$

Intermediate values of X are calculated using the following expression:

$$X_{i+1} = X_i * (2 - B * X_i), \text{ where } X_0 = \text{approximates } 1/B \text{ for the range } 0 < X_0 < 2/B$$

To illustrate a program using the Newton-Raphson algorithm, the sequence of calculations is presented in detail. For double-precision operations, three iterations are

needed to achieve adequate precision in the value of $1/B$. A value for the seed X_0 (approximately equal to $1/B$) is assumed to be given, and the following operations are performed to evaluate Q from double-precision inputs:

$$X_1 = X_0(2 - B * X_0)$$

$$X_2 = X_1(2 - B * X_1) = X_0(2 - B * X_0) * (2 - B * X_0(2 - B * X_0))$$

$$X_3 = X_2(2 - B * X_2)$$

$$X_3 = X_0(2 - B * X_0) * (2 - B * X_0(2 - B * X_0)) * (2 - B * X_0 * (2 - B * X_0) * (2 - B * X_0 * (2 - B * X_0)))$$

$$Q = A * 1/B = A * X_3$$

$$A/B = A * X_0(2 - B * X_0) * (2 - B * X_0(2 - B * X_0)) * (2 - B * X_0 * (2 - B * X_0) * (2 - B * X_0 * (2 - B * X_0)))$$

X1	X1	X1	X1
X2			X2
		X3	

Table 36 presents decimal and hexadecimal values for A , B , and X_0 , which are used in the sample calculation. The computed value of the quotient Q is also included, showing the representations of the results of this sample division.

Table 34. Sample Data Values and Representations

TERM	DECIMAL REPRESENTATION		IEEE HEXADECIMAL REPRESENTATION
	VALUE	MANTISSA • 2 EXPONENT	
A	22	1.375 * 2 4	40360000 00000000
B	7	1.75 * 2 2	401C0000 00000000
X ₀	1/7	1.140625 * 2 (-3)	3FC24000 00000000
Q	22/7	1.5714285714285713 * 2 1	40092492 49249249

In Table 35, the sequence and timing of this procedure is shown exactly as performed by the 'ACT8837. This example shows the steps in a double-precision division requiring three iterations to achieve the desired accuracy. In this table each operation is sequenced according to the clock cycles during which the instruction inputs for that operation are presented at the pins of the 'ACT8837. Operations are accompanied by a pseudocode summary of the operations performed by the 'ACT8837 and the clock cycle when an operand is available or a result is valid.

Each line of pseudocode indicates the operands being used, the operations being performed, the registers involved, and the clock cycles when the results appear. Each

register is represented by its usual abbreviation (RA, RB, P, S, or C) followed by the number of the clock cycle when an operand will be valid or available at the register. For example, "P.4" refers to the contents of the Product Register after the fourth clock cycle.

Table 35. Binary Division Using the Newton-Raphson Algorithm

CLOCK CYCLES	OPERATIONS	PSEUDOCODE
1, 2	$B * X_0$	$B \rightarrow RA.2, X_0 \rightarrow RB.2$ $RA.2 * RB.2 \rightarrow P.4$
3, 4	$2 - B * X_0$	$2 - P.4 \rightarrow S.6$
5, 6	$X_1 = X_0(2 - B * X_0)$	$RB.2 * S.6 \rightarrow P.8$
7, 8	$B * X_1$	$RA.2 * P.8 \rightarrow P.10$
9, 10	$2 - B * X_1$	$P.8 \rightarrow C.9, 2 - P.10 \rightarrow S.12$
11, 12	$X_2 = X_1(2 - B * X_1)$	$C.9 * S.12 \rightarrow P.14$
13, 14	$B * X_2$	$RA.2 * P.14 \rightarrow P.16$
15, 16	$2 - B * X_2$	$P.14 \rightarrow C.15, 2 - P.16 \rightarrow S.18$
17, 18	$X_3 = X_2(2 - B * X_2)$	$A \rightarrow RA.18, C.15 * S.18 \rightarrow P.20$
19, 20	$A * X_3$	$RA.18 * P.20 \rightarrow P.22$
21, 22	Output MSH	$P.22.MSH \rightarrow Y$

The sequence of operations can be microcoded for execution exactly as listed in the table above. Sample microprograms (with data and parity fields provided) are given below. To make the programs easier to follow, comment lines have been included to indicate clock timing, calculation performed by the instructions being loaded, and operations being represented, in the same pseudocode as in the preceding table. The fields in the microinstruction sequences presented below are arranged in the following order:

I	S								
N C C	E	L							
S L C	M								
L T K O P S		S	B S \bar{R} T						
I C R M N I E F E E S /			Y E E \bar{H} E D	D					
N C L U O F P L R A N N R			\bar{O} O O T L S A S A	D B	D				
E L K C D I E O N S R R C			L E E E E S E L T 3	A 3	B P P				
# K C T E G S P D T A B C S Y C S P T T T P			1-----0 1-----0 A B						

d h h hhh h h h hh h h h h h h h h h h h h h h h h hhhh hhhh hhhh h h

All fields in the sample microcode sequences (except for line numbers) are represented as hexadecimal numbers. Line numbers are the only decimal numbers in the samples.

Single-Precision Newton-Raphson Binary Division

Use of the Newton-Raphson algorithm is similar for both single- and double-precision operands. However, for implementations which handle both single- and double-precision division, it may be preferable to use a double-precision seed ROM, converting the double-precision seeds to single precision when necessary.

The following sample program involves conversion of a double-precision seed X_0 for use in single-precision division. Since B is given as a single-precision number, it must be converted to double precision in order to address a double-precision seed ROM. Then the seed X_0 , which is double precision, must be converted to single precision for the actual calculation.

Two iterations are used in the single-precision example. Thus, the formula $Q = A * 1/B$ may be rewritten with $n = 2$:

$$Q = A * 1/B = A * X_2$$

where $X_2 = X_1 * (2 - B * X_1)$ and $X_1 = X_0 * (2 - B * X_0)$

$$A * 1/B = A * X_0 * (2 - B * X_0) * [2 - B * X_0 * (2 - B * X_0)]$$

Table 36 presents a single-precision division using a double-precision seed ROM. This example divides 22/7.

Table 36. Single-Precision Newton-Raphson Binary Division

```

;
;
;Lines 1-2      Calculation: B s.p. → d.p.
;                  Operations: B → RA.1, (s.p. to d.p.)(RA.1) → S.2
;
01 0 0 026 1 1 3 FF 0 0 1 0 1 1 0 0 0 0 3 1 1 3 40E00000 00000000 0 0
02 1 0 026 1 1 3 FF 0 0 1 0 1 1 0 0 0 0 3 1 1 3 40E00000 00000000 0 0
;
;
;Lines 3-4      Calculation: Load X0
;                  Operations: X0 → RA.4
;
03 0 0 126 1 0 2 FF 0 0 1 0 1 1 0 0 0 0 3 1 1 3 3FC24000 00000000 0 0
04 1 0 126 1 0 2 FF 0 0 1 0 1 1 0 0 0 0 3 1 1 3 3FC24000 00000000 0 0
;
;
;Lines 5-6      Calculation: X0 d.p. → s.p.
;                  Operations: (d.p. to s.p.)(RA.4) → S.6
;
05 0 0 126 1 0 2 FF 0 0 1 0 1 1 0 0 0 0 3 1 1 3 3FC24000 00000000 0 0
06 1 0 126 1 0 2 FF 0 0 1 0 1 1 0 0 0 0 3 1 1 3 3FC24000 00000000 0 0
;
;
;Lines 7-8      Calculation: Load B, B * X0
;                  Operations: S.6 → C.7, B → RA.8 RA.8 * C.7 → P.10
;
07 0 1 040 1 0 2 DF 0 0 1 0 0 1 0 0 0 0 3 1 1 3 40E00000 00000000 0 0
08 1 0 040 1 0 2 DF 0 0 1 0 0 1 0 0 0 0 3 1 1 3 40E00000 00000000 0 0
;
;
;Lines 9-10     Calculation: 2 - (B * X0)
;                  Operations: 2 - P.10 → S.12
;
09 0 0 202 0 0 2 FB 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
10 1 0 202 0 0 2 FB 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
;
;
;Lines 11-12    Calculation: X1 = X0(2-B * X0)
;                  Operation: C.7 * S.12 → P.14
;
11 0 0 040 0 0 2 9F 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
12 1 0 040 0 0 2 9F 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0

```

Table 36. Single-Precision Newton-Raphson Binary Division (Concluded)

```

;
;
;Lines 13-14    Calculation: B * X1
;                  Operation: RA.8 * P.14 → P.16
;
13 0 0 040 0 0 2 EF 0 0 0 0 1 1 0 0 0 0 0 3 1 1 3 000000000 00000000 0 0
14 1 0 040 0 0 2 EF 0 0 0 0 1 1 0 0 0 0 0 3 1 1 3 000000000 00000000 0 0
;
;
;Lines 15-16    Calculation: 2 - (B * X1)
;                  Operations: P.14 → C.15, 2 - P.16 → S.18
;
15 0 0 202 0 0 2 FB 0 0 0 0 1 1 0 0 0 0 0 3 1 1 3 000000000 00000000 0 0
16 1 0 202 0 0 2 FB 0 0 0 0 1 1 0 0 0 0 0 3 1 1 3 000000000 00000000 0 0
;
;
;Lines 17-18    Calculation: X2 = X1(2 - B * X1)
;                  Operations: A → RA.18, C.15 * S.18 → P.20
;
17 0 0 040 0 0 2 9F 0 0 1 0 1 1 0 0 0 0 0 3 1 1 3 41B00000 00000000 0 0
18 1 0 040 0 0 2 9F 0 0 1 0 1 1 0 0 0 0 0 3 1 1 3 41B00000 00000000 0 0
;
;
;Lines 19-20    Calculation: A * X2
;                  Operations: RA.18 * P.20 → P.22
;
19 0 0 040 0 0 2 EF 0 0 0 0 1 1 0 0 0 0 0 3 1 1 3 000000000 00000000 0 0
20 1 0 040 0 0 2 EF 0 0 0 0 1 1 0 0 0 0 0 3 1 1 3 000000000 00000000 0 0
;
;
;Lines 21-22    Operation: P.22 → Y
;
;
21 0 0 020 0 0 2 EF 0 0 0 0 1 1 0 0 0 0 0 3 1 1 3 000000000 00000000 0 0
22 1 0 020 0 0 2 EF 0 0 0 0 1 1 0 0 0 0 0 3 1 1 3 000000000 00000000 0 0

```

Double-Precision Newton-Raphson Binary Division

If the value of B is given as a double-precision number and X0 is looked up in a double-precision seed ROM, no conversions are required prior to performing a double-precision division using the Newton-Raphson algorithm. Three iterations are used in the double-precision example ($n = 3$). The following formula represents the sequence of calculations to be performed:

$$\begin{aligned} A/B = & A * X_0 * (2 - B * X_0) * [2 - B * X_0 * (2 - B * X_0)] \\ & * (2 - B * X_0 * (2 - B * X_0) * [2 - B * X_0 * (2 - B * X_0)]) \end{aligned}$$

Table 37 shows a double-precision division using a double-precision seed ROM. The example divides 22/7.

Table 37. Double-Precision Newton-Raphson Binary Division

```

;
;
;
;Lines 1-4      Calculation: B * X0
;                  Operations: B → RA.4, X0 → RB.4, RA.4 * RB.4 → P.8
;
01 0 0 1C0 0 0 2 FF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 3FC24000 00000000 0 0
02 1 0 1C0 0 0 2 FF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 3FC24000 00000000 0 0
03 0 0 1C0 0 0 2 FF 0 0 1 1 1 1 0 0 0 0 3 1 1 3 401C0000 00000000 0 0
04 1 0 1C0 0 0 2 FF 0 0 1 1 1 1 0 0 0 0 3 1 1 3 401C0000 00000000 0 0
;
;
;Lines 5-8      Calculation: 2 - (B * X0)
;                  Operation: 2 - P.8 → S.12
;
05 0 0 382 0 0 2 FB 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
06 1 0 382 0 0 2 FB 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
07 0 0 382 0 0 2 FB 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
08 1 0 382 0 0 2 FB 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
;
;
;Lines 9-12     Calculation: X1 = X0(2-B * X0)
;                  Operation: RB.4 * S.12 → P.16
;
09 0 0 1C0 0 0 2 BF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
10 1 0 1C0 0 0 2 BF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
11 0 0 1C0 0 0 2 BF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
12 1 0 1C0 0 0 2 BF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0

```

Table 37. Double-Precision Newton-Raphson Binary Division (Continued)

```

; Lines 13-16      Calculation: B * X1
; Operations: RA.4 * P.16 → P.20
;
13 0 0 1C0 0 0 2 EF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
14 1 0 1C0 0 0 2 EF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
15 0 0 1C0 0 0 2 EF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
16 1 0 1C0 0 0 2 EF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
;
;
; Lines 17-20      Calculation: 2 - (B * X1)
; Operations: P.16 → C.18, 2 - P.20 → S.24
;
17 0 0 382 0 0 2 FB 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
18 1 1 382 0 0 2 FB 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
19 0 0 382 0 0 2 FB 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
20 1 0 382 0 0 2 FB 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
;
;
; Lines 21-24      Calculation: X2 = X1(2-B * X1)
; Operations: C.18 * S.24 → P.28
;
21 0 0 1C0 0 0 2 9F 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
22 1 0 1C0 0 0 2 9F 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
23 0 0 1C0 0 0 2 9F 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
24 1 0 1C0 0 0 2 9F 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
;
;
; Lines 25-28      Calculation: B * X2
; Operations: RA.4 * P.28 → P.32
;
25 0 0 1C0 0 0 2 EF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
26 1 0 1C0 0 0 2 EF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
27 0 0 1C0 0 0 2 EF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
28 1 0 1C0 0 0 2 EF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
;
;
; Lines 29-32      Calculation: 2 - (B * X2)
; Operations: P.28 → C.30, 2 - P.32 → S.36
;
29 0 0 382 0 0 2 FB 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
30 1 1 382 0 0 2 FB 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
31 0 0 382 0 0 2 FB 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
32 1 0 382 0 0 2 FB 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0

```

Table 37. Double-Precision Newton-Raphson Binary Division (Concluded)

```
;  
;  
;Lines 33-36      Calculation: X3 = X2(2-B * X2)  
;          Operations: A → RA.36, C.30 * S.36 → P.40  
;  
33 0 0 1C0 0 3 2 9F 0 0 1 1 1 1 0 0 0 0 3 1 1 3 40360000 00000000 0 0  
34 1 0 1C0 0 3 2 9F 0 0 1 1 1 1 0 0 0 0 3 1 1 3 40360000 00000000 0 0  
35 0 0 1C0 0 3 2 9F 0 0 1 1 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0  
36 1 0 1C0 0 3 2 9F 0 0 1 1 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0  
;  
;  
;Lines 37-40      Calculation: A * X3  
;          Operations: RA.36 * P.40 → P.44  
;  
37 0 0 1C0 0 0 2 EF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0  
38 1 0 1C0 0 0 2 EF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0  
39 0 0 1C0 0 0 2 EF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0  
40 1 0 1C0 0 0 2 EF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0  
;  
;  
;Lines 41-44      Operation: P.44.MSH → Y  
;  
;  
41 0 0 120 0 0 2 FF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0  
42 1 0 120 0 0 2 FF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0  
43 0 0 120 0 0 2 FF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0  
44 1 0 120 0 0 2 FF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0  
;  
;  
;Line 45          Operation: P.44.LSH → Y  
;  
;  
45 0 0 120 0 0 2 FF 0 0 0 0 1 0 0 0 0 0 3 1 1 3 00000000 00000000 0 0
```

SN74ACT8837

5

Binary Square Root Using the Newton-Raphson Algorithm

Square roots may be calculated iteratively using the Newton-Raphson algorithm. The procedure is similar to Newton-Raphson division and involves evaluating the following expression:

$$A = B * X^n$$

where X_n = the value of X after the n th iteration given

$$X_{i+1} = 0.5 * X_i * [3 - B * (X_i^2)]$$

X_0 = a guess at $1/\sqrt{B}$ where $0 < X_0 < \sqrt{3/B}$

and n = number of iterations to achieve the desired precision

Single-Precision Square Root Using a Double-Precision Seed ROM

When the value of B is given in single precision, it must be converted to a double-precision number before it can be used to address a double-precision seed ROM. Since the seed X_0 is stored as a double-precision number, it must first be converted to single precision before it is used in the calculation.

Two iterations ($n = 2$) are used in a single-precision calculation so the following expression for \sqrt{B} is to be evaluated:

$$A = B * X^2$$

$$\text{where } X_2 = 0.5 * X_1 * [3 - B * (X_1^2)]$$

$$\text{and } X_1 = 0.5 * X_0 * [3 - B * (X_0^2)]$$

$$A = B * 0.5 * 0.5 * X_0 * [3 - B * (X_0^2)] \\ * [3 - B * (0.5 * X_0 * [3 - B * (X_0^2)])^2]$$

Table 38. Single-Precision Binary Square Root

```

;
;
;Lines 1-2      Calculation: B s.p. → d.p.
;                  Operations: B → RA.1, (s.p. to d.p.)(RA.1) → S.2
;
01 0 0 026 1 1 3 FF 0 0 1 0 1 1 0 0 0 0 3 1 1 3 40000000 00000000 0 0
02 1 0 026 1 1 3 FF 0 0 1 0 1 1 0 0 0 0 3 1 1 3 40000000 00000000 0 0
;
;
;Lines 3-4      Calculation: Load X0
;                  Operation: X0 → RA.4
;
03 0 0 126 1 0 2 FF 0 0 1 0 1 1 0 0 0 0 3 1 1 3 3FE6A000 00000000 0 0
04 1 0 126 1 0 2 FF 0 0 1 0 1 1 0 0 0 0 3 1 1 3 3FE6A000 00000000 0 0
;
;
;Lines 5-6      Calculation: X0 d.p. → s.p.
;                  Operations: (d.p. to s.p.)(RA.4) → S.6
;
05 0 0 126 1 0 2 FF 0 0 1 0 1 1 0 0 0 0 3 1 1 3 3FE6A000 00000000 0 0
06 1 0 126 1 0 2 FF 0 0 1 0 1 1 0 0 0 0 3 1 1 3 3FE6A000 00000000 0 0
;
;
;Lines 7-8      Calculation: Load B, B * X0
;                  Operations: S.6 → C.7, B → RB.8, RB.8 * C.7 → P.10
;
07 0 1 040 1 0 2 7F 0 0 0 1 0 1 0 0 0 0 3 1 1 3 40000000 00000000 0 0
08 1 0 040 1 0 2 7F 0 0 0 1 0 1 0 0 0 0 3 1 1 3 40000000 00000000 0 0
;
;
;Lines 9-10     Calculation: B * X02
;                  Operations: P.10 * C.7 → P.12, 3 → RA.10 → S.12
;
09 0 0 260 0 0 2 6F 0 0 1 0 1 1 0 0 0 0 3 1 1 3 40400000 00000000 0 0
10 1 0 260 0 0 2 6F 0 0 1 0 1 1 0 0 0 0 3 1 1 3 40400000 00000000 0 0
;
;
;Lines 11-12    Calculation: 3 - (B * X02)
;                  Operation: S.12 - P.12 → S.14
;
11 0 0 003 0 0 2 FA 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
12 1 0 003 0 0 2 FA 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0

```

Table 38. Single-Precision Binary Square Root (Continued)

```

;
;
;Lines 13-14    Calculation: X0 * (3 - (B * X0 2))
;Operations: C.7 * S.14 → P.16, 1/2 → RA.14 → S.16
;
13 0 0 260 0 0 2 9F 0 0 1 0 1 1 0 0 0 0 3 1 1 3 3F000000 00000000 0 0
14 1 0 260 0 0 2 9F 0 0 1 0 1 1 0 0 0 0 3 1 1 3 3F000000 00000000 0 0
;
;
;Lines 15-16    Calculation: 1/2 * X0 * (3-(B * X0 2)) → X1
;Operations: S.16 * P.16 → P.18, 0 → RA.16,
;                         RA.16 + RB.8 S.18
;
15 0 0 240 0 0 2 AF 0 0 1 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
16 1 0 240 0 0 2 AF 0 0 1 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
;
;
;Lines 17-18    Calculation: B * X1
;Operations: S.18 * P.18 → P.20
;
5 17 0 0 040 0 0 2 AF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
18 1 0 040 0 0 2 AF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
;
;
;Lines 19-20    Calculation: B * X1 2
;Operations: P.18 → C.19, P.20 * C.19 → P.22,
;                         3 → RA.20 → S.22
;
19 0 1 260 0 0 2 6F 0 0 1 0 1 1 0 0 0 0 3 1 1 3 40400000 00000000 0 0
20 1 0 260 0 0 2 6F 0 0 1 0 1 1 0 0 0 0 3 1 1 3 40400000 00000000 0 0
;
;
;Lines 21-22    Calculation: 3 - (B * X1 2)
;Operations: S.22 - P.22 → S.24
;
21 0 0 003 0 0 2 FA 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
22 1 0 003 0 0 2 FA 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
;
;
;Lines 23-24    Calculation: X1 * (3 - (B * X1 2))
;Operations: C.19 * S.24 → P.26, 1/2 → RA.24 → S.26
;
23 0 0 260 0 0 2 9F 0 0 1 0 1 1 0 0 0 0 3 1 1 3 3F000000 00000000 0 0
24 1 0 260 0 0 2 9F 0 0 1 0 1 1 0 0 0 0 3 1 1 3 3F000000 00000000 0 0

```

Table 38. Single-Precision Binary Square Root (Concluded)

```
;  
;  
;Lines 25-26    Calculation: 1/2 * X1 * (3 - (B * X1_2)) → X2  
;          Operations: S.26 * P.26 → P.28, 0 → RA.26,  
;                      RA.26 + RB.8 S.28  
;  
;  
25 0 0 240 0 0 2 AF 0 0 1 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0  
26 1 0 240 0 0 2 AF 0 0 1 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0  
;  
;  
;Lines 27-28    Calculation: B * X2 → A  
;          Operations: S.28 * P.28 → P.30  
;  
27 0 0 040 0 0 2 AF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0  
28 1 0 040 0 0 2 AF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0  
;  
;  
;Lines 29-30    Calculation: NOP  
;          Operation: Y → Output  
;  
29 0 1 00A 0 0 2 FF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0  
30 1 0 00A 0 0 2 FF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
```

5

Double-Precision Square Root

The value of B is given as a double-precision number so X0 can be looked up from a double-precision seed ROM without conversion from one precision to the other. Three iterations ($n = 3$) are required in the double-precision calculation, and the following formula for $\text{sqrt}(B)$ is to be evaluated:

$$\begin{aligned} A = & B * 0.5 * 0.5 * 0.5 * X0 * [3 - B * (X0_2)] \\ & * [3 - B * (0.5 * X0 * [3 - B * (X0_2)])_2] \\ & * [3 - B * (0.5 * 0.5 * X0 * [3 - B * (X0_2)])_2] \\ & * [3 - B * (0.5 * X0 * [3 - B * (X0_2)])_2] \end{aligned}$$

SN74ACT8837

Table 39. Double-Precision Binary Square Root

```

;
;Lines 1-4      Calculations: Load B, Load X0, B * X0
;                  Operations: B → RB.4, X0 → RA.4, RA.4 * RB.4 → P.8
;                                         RA.4 → S.8 → C.10
;
;
01 0 0 3E0 0 0 2 FF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 40000000 00000000 0 0
02 1 0 3E0 0 0 2 FF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 40000000 00000000 0 0
03 0 0 3E0 0 0 2 FF 0 0 1 1 1 1 0 0 0 0 3 1 1 3 3FE6A000 00000000 0 0
04 1 0 3E0 0 0 2 FF 0 0 1 1 1 1 0 0 0 0 3 1 1 3 3FE6A000 00000000 0 0
;
;
;Lines 5-8      Calculations: B * X0 2
;                  Operations: P.8 * S.8 → P.12, 3 → RA.8 → S.12
;
;
05 0 0 3E0 0 0 2 AF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
06 1 0 3E0 0 0 2 AF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
07 0 0 3E0 0 0 2 AF 0 0 1 0 1 1 0 0 0 0 3 1 1 3 40080000 00000000 0 0
08 1 0 3E0 0 0 2 AF 0 0 1 0 1 1 0 0 0 0 3 1 1 3 40080000 00000000 0 0
;
;
5 ;Lines 9-12      Calculations: 3 - (B * X0 2)
;                  Operations: S.12 - P.12 → S.16
;
;
09 0 0 183 0 0 2 FA 0 0 0 0 0 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
10 1 1 183 0 0 2 FA 0 0 0 0 0 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
11 0 0 183 0 0 2 FA 0 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
12 1 0 183 0 0 2 FA 0 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
;
;
;Lines 13-16    Calculations: X0 * (3 - (B * X0 2))
;                  Operations: C.10 * S.16 → P.20, 1/2 → RA.16 → S.20
;
;
13 0 0 3E0 0 0 2 9F 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
14 1 0 3E0 0 0 2 9F 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
15 0 0 3E0 0 0 2 9F 0 0 1 0 1 1 0 0 0 0 3 1 1 3 3FE00000 00000000 0 0
16 1 0 3E0 0 0 2 9F 0 0 1 0 1 1 0 0 0 0 3 1 1 3 3FE00000 00000000 0 0

```

SN74ACT8837

Table 39. Double-Precision Binary Square Root (Continued)

```

;
;
;Lines 17-20    Calculations: 1/2 * X0 * (3-(B * X0 2)) → X1
;                  Operations: S.20 * P.20 → P.24 → C.25, 0 → RA.20,
;                               RA.20 + RB.4 → S.24
;
17 0 0 3C0 0 0 2 AF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
18 1 0 3C0 0 0 2 AF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
19 0 0 3C0 0 0 2 AF 0 0 1 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
20 1 0 3C0 0 0 2 AF 0 0 1 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
;
;
;Lines 21-24    Calculations: B * X1
;                  Operations: S.24 * P.24 → P.28
;
21 0 0 1C0 0 0 2 AF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
22 1 0 1C0 0 0 2 AF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
23 0 0 1C0 0 0 2 AF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
24 1 0 1C0 0 0 2 AF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
;
;
;Lines 25-28    Calculations: B * X1 2
;                  Operations: P.28 * C.25 → P.32, 3 → RA.28 → S.32
;
25 0 1 3E0 0 0 2 6F 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
26 1 0 3E0 0 0 2 6F 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
27 0 0 3E0 0 0 2 6F 0 0 1 0 1 1 0 0 0 0 3 1 1 3 40080000 00000000 0 0
28 1 0 3E0 0 0 2 6F 0 0 1 0 1 1 0 0 0 0 3 1 1 3 40080000 00000000 0 0
;
;
;Lines 29-32    Calculations: 3 - (B * X1 2)
;                  Operations: S.32 - P.32 → S.36
;
29 0 0 183 0 0 2 FA 0 0 0 0 0 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
30 1 0 183 0 0 2 FA 0 0 0 0 0 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
31 0 0 183 0 0 2 FA 0 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0
32 1 0 183 0 0 2 FA 0 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0

```

Table 39. Double-Precision Binary Square Root (Continued)

```

;
;
;Lines 33-36    Calculations: X1 * (3 - (B * X1_2))
;                  Operations: C.25 * S.36 → P.40, 1/2 → RA.36 S.40
;
33 0 0 3E0 0 0 2 9F 0 0 0 0 1 1 0 0 0 0 3 1 1 3 000000000 00000000 0 0
34 1 0 3E0 0 0 2 9F 0 0 0 0 1 1 0 0 0 0 3 1 1 3 000000000 00000000 0 0
35 0 0 3E0 0 0 2 9F 0 0 1 0 1 1 0 0 0 0 3 1 1 3 3FE00000 00000000 0 0
36 1 0 3E0 0 0 2 9F 0 0 1 0 1 1 0 0 0 0 3 1 1 3 3FE00000 00000000 0 0
;
;
;Lines 37-40    Calculations: 1/2 * X1 * (3 - (B * X1_2)) → X2
;                  Operations: S.40 * P.40 → P.44 → C.45, 0 → RA.40,
;                                         RA.40 + RB.4 S.44
;
37 0 0 3C0 0 0 2 AF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 000000000 00000000 0 0
38 1 0 3C0 0 0 2 AF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 000000000 00000000 0 0
39 0 0 3C0 0 0 2 AF 0 0 1 0 1 1 0 0 0 0 3 1 1 3 000000000 00000000 0 0
40 1 0 3C0 0 0 2 AF 0 0 1 0 1 1 0 0 0 0 3 1 1 3 000000000 00000000 0 0
;
;
;Lines 41-44    Calculations: B * X2
;                  Operations: S.44 * P.44 → P.48
;
41 0 0 1C0 0 0 2 AF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 000000000 00000000 0 0
42 1 0 1C0 0 0 2 AF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 000000000 00000000 0 0
43 0 0 1C0 0 0 2 AF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 000000000 00000000 0 0
44 1 0 1C0 0 0 2 AF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 000000000 00000000 0 0
;
;
;Lines 45-48    Calculations: B * X2_2
;                  Operations: P.48 * C.45 → P.52, 3 → RA.48 → S.52
;
45 0 1 3E0 0 0 2 6F 0 0 0 0 1 1 0 0 0 0 3 1 1 3 000000000 00000000 0 0
46 1 0 3E0 0 0 2 6F 0 0 0 0 1 1 0 0 0 0 3 1 1 3 000000000 00000000 0 0
47 0 0 3E0 0 0 2 6F 0 0 1 0 1 1 0 0 0 0 3 1 1 3 40080000 00000000 0 0
48 1 0 3E0 0 0 2 6F 0 0 1 0 1 1 0 0 0 0 3 1 1 3 40080000 00000000 0 0

```

5

SN74ACT8837

Table 39. Double-Precision Binary Square Root (Continued)

```
;  
;  
;Lines 49-52    Calculations: 3 - (B * X2 2)  
;                Operations: S.52 - P.52 → S.56  
;  
49 0 0 183 0 0 2 FA 0 0 0 0 0 1 0 0 0 0 3 1 1 3 000000000 00000000 0 0  
50 1 0 183 0 0 2 FA 0 0 0 0 0 1 0 0 0 0 3 1 1 3 000000000 00000000 0 0  
51 0 0 183 0 0 2 FA 0 0 0 0 1 1 0 0 0 0 3 1 1 3 000000000 00000000 0 0  
52 1 0 183 0 0 2 FA 0 0 0 0 1 1 0 0 0 0 3 1 1 3 000000000 00000000 0 0  
;  
;  
;Lines 53-56    Calculations: X2 * (3 - (B * X2 2))  
;                Operations: C.45 * S.56 → P.60, 1/2 → RA.56 → S.60  
;  
53 0 0 3E0 0 0 2 9F 0 0 0 0 1 1 0 0 0 0 3 1 1 3 000000000 00000000 0 0  
54 1 0 3E0 0 0 2 9F 0 0 0 0 1 1 0 0 0 0 3 1 1 3 000000000 00000000 0 0  
55 0 0 3E0 0 0 2 9F 0 0 1 0 1 1 0 0 0 0 3 1 1 3 3FE00000 00000000 0 0  
56 1 0 3E0 0 0 2 9F 0 0 1 0 1 1 0 0 0 0 3 1 1 3 3FE00000 00000000 0 0  
;  
;  
;Lines 57-60    Calculations: 1/2 * X2 * (3 - (B * X2 )) → X3  
;                Operations: S.60 * P.60 → P.64, 0 → RA.60,  
;                                RA.60 + RB.4 → S.64  
;  
57 0 0 3C0 0 0 2 AF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 000000000 00000000 0 0  
58 1 0 3C0 0 0 2 AF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 000000000 00000000 0 0  
59 0 0 3C0 0 0 2 AF 0 0 1 0 1 1 0 0 0 0 3 1 1 3 000000000 00000000 0 0  
60 1 0 3C0 0 0 2 AF 0 0 1 0 1 1 0 0 0 0 3 1 1 3 000000000 00000000 0 0  
;  
;  
;Lines 61-64    Calculations: B * X3 → A  
;                Operations: S.64 * P.64 → P.68 → Y.MSH  
;  
61 0 0 1C0 0 0 2 AF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 000000000 00000000 0 0  
62 1 0 1C0 0 0 2 AF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 000000000 00000000 0 0  
63 0 0 1C0 0 0 2 AF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 000000000 00000000 0 0  
64 1 0 1C0 0 0 2 AF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 000000000 00000000 0 0
```

Table 39. Double-Precision Binary Square Root (Concluded)

```
;  
;  
;Lines 65-68    Calculation:  NOP  
;                  Operation:  Y.MSH → Output  
;  
65 0 1 18A 0 0 2 FF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0  
66 1 0 18A 0 0 2 FF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0  
67 0 0 18A 0 0 2 FF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0  
68 1 0 18A 0 0 2 FF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 00000000 0 0  
;  
;  
;Line 69        Calculation:  NOP  
;                  Operation:  Y.LSH → Output  
;  
69 0 0 18A 0 0 2 FF 0 0 0 0 1 0 0 0 0 0 3 1 1 3 00000000 00000000 0 0
```

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 'ACT8837//ACT8847 produces a NaN whenever an invalid operation such as $0 * \infty$ 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 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.

Implementing a Double-Precision Seed ROM

The seed ROM assumed in the previous microcode examples is a double-precision seed ROM containing both division and square root seeds. Six chips are necessary to build this seed ROM: five 4×4096 registered PROMs and one latch (ordinarily implemented in a PAL). Figure 26 shows a sample implementation for a double-precision seed ROM.

Three of the PROMs are for generating the exponent part of the seed. All 11 exponent lines are necessary to accurately determine the exponent of the seed. There are 12 address lines in a 4×1024 PROM, so the last address line can be used for a microcode bit that tells whether a divide or square root seed is being read. Since there are only 11 bits in the exponent and three PROMs are used, there are 12 output bits but one bit is not used. The equations giving the contents of the PROMs is given in a later section.

The other two PROMs generate the mantissa part of the seed. One address line of the PROMs is used for the microcode bit telling whether a divide or square root seed is to be used. For a square root seed, the least significant bit of the exponent is needed in generating the mantissa seed. Therefore, another address line of the PROMs is used by the least significant exponent bit. This leaves 10 address lines to be used to look up the mantissa seed. Since there are eight output bits from the two PROMs, an eight-bit seed is generated.

5

The sign bit of B needs to be preserved for use when the seed is read. In the case of binary division, this requirement is obvious. In the square root calculation, the sign bit of B should always be zero. This condition should be tested by the microprogram.

Since every real square root has two answers, normally the positive answer is assumed. However, since the sign of B is meaningless to Newton-Raphson unless it is positive, the example microprograms assume that a negative B simply means that the negative of the square root of B is the desired answer instead of the positive root. This is accomplished by using the absolute value of B in all computations except for looking up the seed. If the seed is negative, then the answer generated will be the negative root.

PROM Contents

Because one address line of the PROMs selects divide or square root, the PROMs can be considered to be divided functionally into two halves: the divide half and the square root half. Each functional half is discussed separately in the sections below.

Divide PROMs

The exponent part of the seed is defined in the following manner. Assuming that $B = m * (2^e)$ and $X_0 = m' * (2^{e'})$, e' is computed as $e' = -e$. Using the definition of an IEEE number, the value of m can be represented as a number within the following interval: $1 \leq m < 2$.

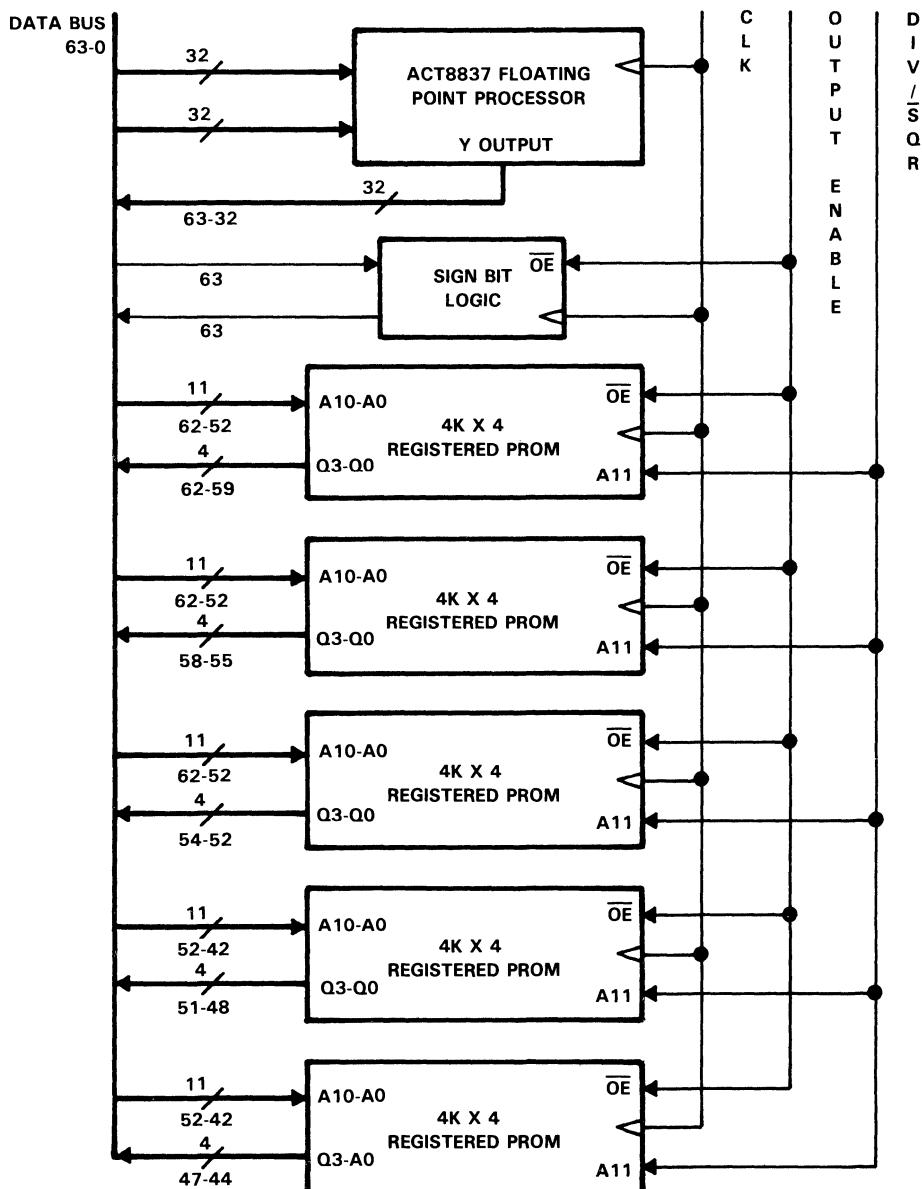


Figure 26. IEEE Double-Precision Seed ROM for Newton-Raphson Division and Square Root

This range of values of m can be subdivided into two cases:

$$m = 1, \text{ or } 1 < m < 2$$

Since m' is computed as $m' = 1 / m$, the range of m' will be

$$m' = 1, \text{ or } 1/2 < m' < 1$$

To be represented as a normalized IEEE number, m'' would be

$$m'' = m' * (2^1) = 2/m \quad (1)$$

This would make the range of m''

$$m'' = 2, \text{ or } 1 < m'' < 2$$

This is still not quite in the range of a valid IEEE number; however, $m'' = 2$ only when $m = 1$. Therefore, m'' can be forced to be just less than 2 in this case.

Since $X_0 = m' * (2^{e'})$, to use m'' in the PROMs, we must have an e'' in the exponent such that $X_0 = m'' * (2^{e''})$. This is true for $e'' = e' - 1$. Since, $X_0 = m'' * (2^{e''})$, the following substitution can be made:

$$\begin{aligned} X_0 &= (m' * (2^1)) * (2^{(e'-1)}) \\ &= m' * (2^1) * (2^{e'}) * (2^{(-1)}) \\ &= m' * (2^{e'}) * (2^{(1-1)}) \\ &= m' * (2^{e'}) * (2^0) \\ &= m' * (2^{e'}) \end{aligned}$$

Therefore, if e'' is used in the exponent PROMs and m'' is used in the mantissa PROMs, a normalized IEEE seed can be generated. The only exception to the formula is that for $m = 1$,

$$m'' = 2 / m - \text{delta}$$

$$\text{Where delta} = 2^{(-8)}$$

So $m'' = 2 / m$, and $e'' = (-e) - 1$.

Since IEEE exponents are represented in excess 1023 notation, a formula for X'' must be determined, given that X is the IEEE exponent. As an IEEE exponent, $X = e + 1023 \rightarrow e = X - 1023$ and $X'' = e'' + 1023$. So, for X'' in terms of X ,

$$\begin{aligned} X'' &= e'' + 1023 \\ &= (-e) - 1 + 1023 \\ &= (- (X - 1023)) + 1022 \\ &= 1023 - X + 1022 \\ &= 2045 - X \end{aligned}$$

So given the 11 bits of X as address of the seed exponent, the value stored at address X is

$$X'' = 2045 - X \quad (2)$$

Given that the mantissa seed ROM uses 10 bits of the mantissa to determine the seed, each seed X_m will be used for some range of mantissas, B_m to $(B_m + 2 * \text{delta})$. The formula for X_m is from formula (1).

$$\begin{array}{ll} 2/B_m & \rightarrow X_m \\ 2/(B_m + 2 * \text{delta}) & \rightarrow X_m \end{array}$$

$$\text{Where delta} = 2(-11)$$

This value is used since the actual X_m should be generated by the mantissa in the center of the given range:

$$X_m = 2/(B_m + \text{delta})$$

This would result in a more accurate seed on the average. Therefore, the formula used to generate the mantissa part of the seed is

$$X_m = 2/(B_m + (2(-11))) \quad (3)$$

Square Root PROMs

The seed for the square root, X_0 , is actually the reciprocal of the square root of the data, B :

$$X_0 = 1 / (B^{(1/2)})$$

Given $B = m * (2^e)$ and $X_0 = m' * (2^{e'})$, the expression for X_0 can be evaluated by substitution and reduction:

$$\begin{aligned} X_0 &= 1 / ((m * (2^e))^{(1/2)}) \\ &= 1 / (m^{(1/2)} * (2^{(e/2)})) \\ &= m^{(-1/2)} * (2^{(-e/2)}) \end{aligned}$$

Then m' and e' may be written as $m' = m^{(-1/2)}$ and $e' = -e/2$.

Next, it is necessary to verify that the above m' and e' form a valid normalized IEEE number. When e is an odd number, e' is not an integer and, therefore, it is not valid IEEE exponent. If the above expression is separated into two cases, e' can be represented in terms of a valid IEEE exponent, e'' :

$$\begin{array}{ll} e' = -e/2 & \text{for } e \text{ even} \\ e' = e'' + 1/2 & \text{for } e \text{ odd} \end{array}$$

Rewriting e'' in terms of e produces this expression:

$$e'' = e' - 1/2 = (-e/2) - 1/2 \quad \text{for } e \text{ odd}$$

Then a valid IEEE exponent, e'' , can be written for all e as

$$\begin{array}{ll} e'' = -e/2 & \text{for } e \text{ even} \\ e'' = (-e/2) - 1/2 & \text{for } e \text{ odd} \end{array}$$

This is equivalent to $e'' = \text{int}(-e/2)$ for all e. However, the 1/2 affects the mantissa:

$$\begin{aligned}X_0 &= m' * (2^{e'}) \\X_0 &= m' * (2(e'' + 1/2)) && \text{for odd } e \\X_0 &= m' * (2^{1/2}) * (2^{e''}) && \text{for odd } e\end{aligned}$$

Since $X_0 = m'' * (2^{e''})$ m'' can be rewritten as

$$\begin{aligned}m'' &= m' && \text{for even } e \\m'' &= m' * (2^{1/2}) && \text{for odd } e\end{aligned}$$

$$\begin{aligned}\text{In terms of } m, m'' &= m - 1/2 && \text{for even } e \\m'' &= (m - 1/2) * (2^{1/2}) && \text{for odd } e\end{aligned}$$

Simplifying m'' for odd e,

$$\begin{aligned}m'' &= (1/m^{1/2}) * (2^{1/2}) && \text{for odd } e \\m'' &= (2/m^{1/2}) && \text{for odd } e\end{aligned}$$

Just as the divide exponent needed to be converted to excess 1023 notation, so the same must be done for the square root:

$$\begin{aligned}X'' &= e'' + 1023 \\X &= e + 1023 \\X'' &= \text{int}(-e/2) + 1023 \\X'' &= \text{int}((1023-X)/2) + 1023\end{aligned}$$

5

The IEEE bits for the exponent seed, X'' , can be expressed in terms of the IEEE bits for the exponent of B, X:

$$X'' = \text{int}((1023-X)/2) + 1023$$

Because the formula for m'' depends on the least significant bit of e, that bit must be used as an address line to the mantissa.

Since $X = e + 1023$, an odd value of e will result in an even value of X, and an even value of e will result in an odd value of X. Therefore,

$$\begin{aligned}m'' &= m - 1/2 && \text{for odd } X \\m'' &= 2/m^{1/2} && \text{for even } X\end{aligned}$$

SN74ACT887

Overview**1****SN74ACT8818 16-Bit Microsequencer****2****SN74ACT8832 32-Bit Registered ALU****3****SN74ACT8836 32- × 32-Bit Parallel Multiplier****4****SN74ACT8837 64-Bit Floating Point Processor****5****SN74ACT8841 Digital Crossbar Switch****6****SN74ACT8847 64-Bit Floating Point/Integer Processor****7****Support****8****Mechanical Data****9**

6

SN74ACT8841

SN74ACT8841

Digital Crossbar Switch

The SN74ACT8841 is a single-chip digital crossbar switch that cost-effectively eliminates bottlenecks to speed data through complex bus architectures.

The 'ACT8841 has 16 four-bit bidirectional ports which can be connected in any conceivable combination. Total time for data transfer is 14-ns flowthrough.

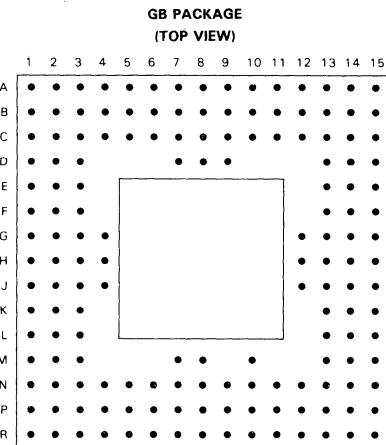
The 'ACT8841 is ideal for multiprocessor application, where memory bottlenecks tend to occur. For example, four 32-bit buses can be easily connected by two 'ACT8841 devices. System architectures based on the 16-port 'ACT8841 can include up to 16 switching nodes (i.e., processors, memories, or bus interfaces). Larger processor arrays can be built with multistage interconnect schemes.

6**SN74ACT8841**

SN74ACT8841 DIGITAL CROSSBAR SWITCH

JUNE 1988

- High-Speed Programmable Switch for Parallel Processing Applications
- Dynamically Reconfigurable for Fault-Tolerant Routing
- 64 Bidirectional Data I/Os in 16 Nibble (Four-Bit) Groups
- Data I/O Selection Programmable by Nibble
- Eight Banks of Control Flip-Flops for Storing Configuration Programs
- Two Selectable Hard-Wired Switching Configurations
- Selectable Stored-Data or Real-Time Inputs
- 156-Pin Grid-Array Package
- CMOS 1 μm EPIC™ Process
- Single 5-V Power Supply



description

The SN74ACT8841 is a flexible, high-speed digital crossbar switch. It is easily microprogrammable to support user-definable interconnection patterns. This crossbar switch is especially suited to multiprocessor interconnects that are dynamically reconfigurable or even reprogrammable after each system clock. The 'ACT8841 is built in Texas Instruments advanced 1 μm EPIC™ CMOS process to enhance performance and reduce power consumption. The switch requires only a 5-V power supply.

Because the 'ACT8841 is a 16-port device, system architectures based on the 'ACT8841 can include up to 16 switching nodes, which may be processors, data memories, or bus interfaces. Larger processor arrays can be built with multistage interconnection schemes. Most applications will use the crossbar switch as a broadband bus interface controller, for example, between closely coupled processors which must exchange data with very low propagation delays.

The 'ACT8841 has ten selectable control sources, including eight banks of programmable control flip-flops and two hard-wired control circuits. The device can switch from 1 to 16 nibbles (4 to 64 bits) of data in a single cycle.

The 64 I/O pins of the 'ACT8841 are arranged in 16 switchable nibbles (see Figure 1). A single input nibble can be broadcast to any combination of 15 output nibbles, or even to 16 nibbles (including itself) if operating off registered data. Multiple input nibbles can be switched to multiple outputs, depending on the programmed configurations available in the control flip-flops.

The digital crossbar switch is intended primarily for multiprocessor interconnection and parallel processing applications. The device can be used to select and transfer data from multiple sources to multiple destinations. Since it can be dynamically reprogrammed, it is suitable for use in reconfigurable networks for fault-tolerant routing.

PRODUCT PREVIEW

6

SN74ACT8841

EPIC is a trademark of Texas Instruments Incorporated

PRODUCT PREVIEW documents contain information on products in the formative or design phase of development. Characteristic data and other specifications are design goals. Texas Instruments reserves the right to change or discontinue these products without notice.

Copyright © 1988, Texas Instruments Incorporated

TEXAS
INSTRUMENTS
POST OFFICE BOX 655012 • DALLAS, TEXAS 75265

SN74ACT8841

DIGITAL CROSSBAR SWITCH

description (continued)

The 'ACT8841 and the bipolar SN74AS8840 share the same architecture. Microcode for the 'AS8840 can be run on the 'ACT8841 if the additional control inputs to the 'ACT8841 are properly terminated. However, because the 'ACT8841 is a CMOS device with six additional control inputs, the 'AS8840 and the 'ACT8841 are not socket-compatible and cannot be used interchangably. A summary of the differences between the SN74AS8840 and the SN74ACT8841 is provided in the 'AS8840 and 'ACT8841 FUNCTIONAL COMPARISON at the end of the data sheet.

The SN74ACT8841 is characterized for operation from 0°C to 70°C.

Table 1. 'ACT8841 Pin Grid Allocation

NO.	PIN NAME	NO.	PIN NAME	NO.	PIN NAME	NO.	PIN NAME
A1	GND	C10	D31	H12	VCC	N7	CNTR13
A2	GND	C11	<u>OED6</u>	H13	LCLK	N8	CREADO
A3	D37	C12	VCC	H14	SELDS	N9	VCC
A4	D35	C13	GND	H15	CNTR3	N10	D0
A5	D33	C14	D23	J1	<u>OEC</u>	N11	D3
A6	WE	C15	D21	J2	CRWRITEO	N12	D6
A7	CRADR1	D1	D43	J3	CRWRITE1	N13	GND
A8	CNTR7	D2	D42	J4	GND	N14	D8
A9	CNTR4	D3	VCC	J12	GND	N15	D9
A10	<u>OED7</u>	D7	GND	J13	CNTR2	P1	GND
A11	D29	D8	VCC	J14	CNTR1	P2	GND
A12	D27	D9	GND	J15	CNTR0	P3	D56
A13	D25	D13	D22	K1	CRWRITE2	P4	D58
A14	GND	D14	D20	K2	<u>OED12</u>	P5	D60
A15	GND	D15	D19	K3	D48	P6	D62
B1	GND	E1	D45	K13	D15	P7	CNTR12
B2	GND	E2	D44	K14	D14	P8	CNTR15
B3	D39	E3	<u>OED10</u>	K15	<u>OED3</u>	P9	TP0
B4	D36	E13	<u>OED5</u>	L1	D49	P10	<u>OED0</u>
B5	D34	E14	D18	L2	D50	P11	D2
B6	<u>OED8</u>	E15	<u>D17</u>	L3	<u>OED13</u>	P12	D4
B7	CRADRO	F1	<u>OED11</u>	L13	OED2	P13	D7
B8	CRSRCE	F2	D46	L14	D12	P14	GND
B9	CNTR5	F3	D47	L15	D13	P15	GND
B10	D30	F13	D16	M1	D51	R1	GND
B11	D28	F14	<u>OED4</u>	M2	D52	R2	GND
B12	D26	F15	CRSEL3	M3	D54	R3	D57
B13	D24	G1	CNTR8	M7	GND	R4	D59
B14	GND	G2	CNTR9	M8	VCC	R5	D61
B15	GND	G3	CNTR10	M10	GND	R6	<u>OED15</u>
C1	D41	G4	GND	M13	VCC	R7	CNTR14
C2	D40	G12	GND	M14	D10	R8	CREAD1
C3	GND	G13	CRSEL2	M15	D11	R9	CREAD2
C4	D38	G14	CRSEL1	N1	D53	R10	TP1
C5	<u>OED9</u>	G15	CRSEL0	N2	D55	R11	D1
C6	D32	H1	CNTR11	N3	GND	R12	<u>OED1</u>
C7	VCC	H2	SELDS	N4	VCC	R13	D5
C8	CRCLK	H3	MSCLK	N5	<u>OED14</u>	R14	GND
C9	CNTR6	H4	VCC	N6	D63	R15	GND



POST OFFICE BOX 655012 • DALLAS, TEXAS 75265

SN74ACT8841
DIGITAL CROSSBAR SWITCH

PRODUCT PREVIEW

Table 2. 'ACT8841 Pin Functional Description

PIN NAME	PIN NO.	I/O	DESCRIPTION
CNTR0	J15		
CNTR1	J14		
CNTR2	J13		
CNTR3	H15		
CNTR4	A9		
CNTR5	B9		
CNTR6	C9		
CNTR7	A8	I/O	Control I/O. Inputs four control words to the control flip-flops on each CRCLK cycle. As outputs, the same addresses can be used to read the flip-flop settings.
CNTR8	G1		
CNTR9	G2		
CNTR10	G3		
CNTR11	H1		
CNTR12	P7		
CNTR13	N7		
CNTR14	R7		
CNTR15	P8		
CRADRO	B7	I	Control register address. Selects 16-bits of control flip-flops as a source/destination for outputs/inputs on CNTR0-CNTR15. (see Table 7)
CRADR1	A7		
CRCLK	C8	I	Control register clock. Clocks CNTR0-CNTR15 into the control flip-flops on low-to-high transition.
CREADO	N8		
CREAD1	R8	I	Selects one of eight banks of control flip-flops to read out on CNTR0-CNTR15 in 16-bit words addressed by CRADR1-CRADRO.
CREAD2	R9		
CRSEL0	G15		
CRSEL1	G14		
CRSEL2	G13	I	Selects one of ten control configurations.
CRSEL3	F15		
CRSRCE	B8	I	Load source select. When low selects CNTR inputs, when high selects DATA inputs.

6

SN74ACT8841

SN74ACT8841

DIGITAL CROSSBAR SWITCH

Table 2. 'ACT8841 Pin Functional Description (continued)

PIN NAME	PIN NO.	I/O	DESCRIPTION
CRWRITE0	J2		
CRWRITE1	J3	I	Destination select. Selects one of eight control banks. (see Table 4)
CRWRITE2	K1		
D0	N10		
D1	R11		
D2	P11		
D3	N11		
D4	P12		
D5	R13		
D6	N12		
D7	P13		
D8	N14		
D9	N15		
D10	M14		
D11	M15		
D12	L14		
D13	L15		
D14	K14		
D15	K13		
D16	F13	I/O	I/O data bits 0 through 31 (data bits 0 through 31 are the least significant half).
D17	E15		
D18	E14		
D19	D15		
D20	D14		
D21	C15		
D22	D13		
D23	C14		
D24	B13		
D25	A13		
D26	B12		
D27	A12		
D28	B11		
D29	A11		
D30	B10		
D31	C10		
D32	C6		
D33	A5	I/O	I/O data bits 32 through 35 (data bits 32 through 63 are the most significant half).
D34	B5		
D35	A4		

TEXAS
INSTRUMENTS

POST OFFICE BOX 655012 • DALLAS, TEXAS 75265

**SN74ACT8841
DIGITAL CROSSBAR SWITCH**

PRODUCT PREVIEW

6

SN74ACT8841

Table 2. 'ACT8841 Pin Functional Description (continued)

PIN NAME	PIN NO.	I/O	DESCRIPTION
D36	B4		
D37	A3		
D38	C4		
D39	B3		
D40	C2		
D41	C1		
D42	D2		
D43	D1		
D44	E2		
D45	E1		
D46	F2		
D47	F3		
D48	K3		
D49	L1		
D50	L2	I/O	I/O data bits 36 through 63 (data bits 32 through 63 are the most significant half).
D51	M1		
D52	M2		
D53	N1		
D54	M3		
D55	N2		
D56	P3		
D57	R3		
D58	P4		
D59	R4		
D60	P5		
D61	R5		
D62	P6		
D63	N6		
GND	A1		
GND	A2		
GND	A14		
GND	A15		
GND	B1		
GND	B2		
GND	B14		
GND	B15		
GND	C3		
GND	C13		
GND	D7		
GND	D9		
GND	G4		
GND	G12		Ground (all pins must be used).

TEXAS
INSTRUMENTS

POST OFFICE BOX 655012 • DALLAS, TEXAS 75265

SN74ACT8841

DIGITAL CROSSBAR SWITCH

Table 2. 'ACT8841 Pin Functional Description (continued)

PIN NAME	PIN NO.	I/O	DESCRIPTION
GND	J4		
GND	J12		
GND	M7		
GND	M10		
GND	N3		
GND	N13		
GND	P1		
GND	P2		Ground (all pins must be used).
GND	P14		
GND	P15		
GND	R1		
GND	R2		
GND	R14		
GND	R15		
LSCLK	H13	I	Clocks the least significant half of data inputs into the input registers on a low-to-high transition.
MSCLK	H3	I	Clocks the most significant half of data inputs into the input registers on a low-to-high transition.
OE _C	J1	I	Output enable for control flip-flops, active low
OE _{D0}	P10		
OE _{D1}	R12		
OE _{D2}	L13		
OE _{D3}	K15		
OE _{D4}	F14		
OE _{D5}	E13		
OE _{D6}	C11		
OE _{D7}	A10		
OE _{D8}	B6	I	Output enables for data nibbles, active low
OE _{D9}	C5		
OE _{D10}	E3		
OE _{D11}	F1		
OE _{D12}	K2		
OE _{D13}	L3		
OE _{D14}	N5		
OE _{D15}	R6		

TEXAS
INSTRUMENTS

POST OFFICE BOX 655012 • DALLAS, TEXAS 75265

Table 2. 'ACT8841 Pin Functional Description (concluded)

PIN NAME	PIN NO.	I/O	DESCRIPTION
SELDLSS	H14	I	When low, selects the stored, least significant data input to the main internal bus. When high, real-time data is selected.
SELDMSS	H2	I	When low, selects the stored, most significant data input to the main internal bus. When high, real-time data is selected.
TPO	P9		
TP1	R10	I	Test pins. High during normal operation. (see Table 9)
V _{CC}	C7		
V _{CC}	C12		
V _{CC}	D3		
V _{CC}	D8		
V _{CC}	H4		
V _{CC}	H12		
V _{CC}	M8		
V _{CC}	M13		
V _{CC}	N4		
V _{CC}	N9		
WE	A6	I	Write enable for control flip-flops, active low

overview

The 64 I/O pins of the 'ACT8841 are arranged in 16 nibble (four-bit) groups where each set of four pins serves as bidirectional inputs to and outputs from a nibble multiplexer. During a switching operation, each nibble passes four bits of either stored or real-time data to the main internal 64-bit data bus. Each output multiplexer will independently select one of the 16 nibbles from this 64-bit data bus.

Data nibbles are organized into two groups: the least significant half (D31-D0) and the most significant half (D63-D32). Stored versus real-time data inputs can be selected separately for the LSH and the MSH. Two clock inputs, LSCLK and MSCLK, are available to latch LSH and MSH data inputs, respectively, into the data register.

The pattern of output nibbles resulting from the switching operation is determined by a selectable control source, either one of eight banks of programmable control flip-flops or one of two hard-wired switching configurations. Inputs to the control flip-flops can be loaded either from the data bus or from control I/Os. A separate clock (CRCLK) is provided for loading the banks of control flip-flops.



POST OFFICE BOX 655012 • DALLAS, TEXAS 75265

SN74ACT8841

DIGITAL CROSSBAR SWITCH

logic symbol

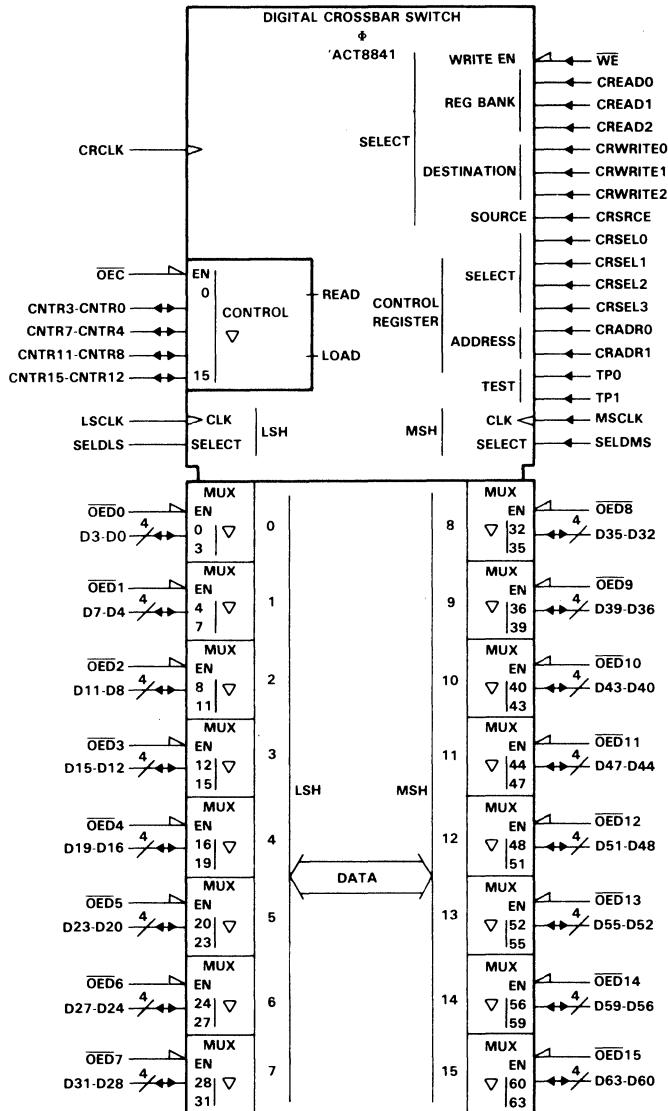


FIGURE 1

SN74ACT8841 DIGITAL CROSSBAR SWITCH

architecture

The 'ACT8841 digital crossbar switch has its 64 data I/Os arranged in 16 multiplexer logic blocks, as shown in Figure 2. Each nibble multiplexer logic block handles four bits of real-time input and four bits of stored-data input, and either input can be passed to the common data bus.

Two input multiplexer controls are provided to select between stored and real-time inputs. SELDLS controls input data selection for the LSH (D31-D0) of the 64-bit data input, and SELDMS for the MSH (D63-D32). The input register clocks, LSCLK and MSCLK, are grouped in the same way and are used to clock data into the registers in the multiplexer logic blocks. The 16 data input nibbles make up the 64 data bits on the internal main bus.

This common bus supplies 16 data nibbles to a 16-to-1 output multiplexer in each multiplexer logic block (see Figure 3). As determined by one of ten selectable control sources, the 16-to-1 output multiplexer selects a data nibble to send to the outputs via the three-state output driver.

Control of the input and output multiplexers determines the input-to-output pattern for the entire crossbar switch. Many different switching combinations can be set up by programming the control flip-flop configurations to determine the outputs from the 16-to-1 multiplexers.

For example, the switch can be programmed to broadcast one data input nibble through the other 15 nibbles (60 outputs). Conversely, a 15-to-1 nibble multiplexer can be configured by programming the switch to select and output a single data nibble from the 64-bit bus. Several examples are described in more detail in a later section.

SN74ACT8841

DIGITAL CROSSBAR SWITCH

functional block diagram

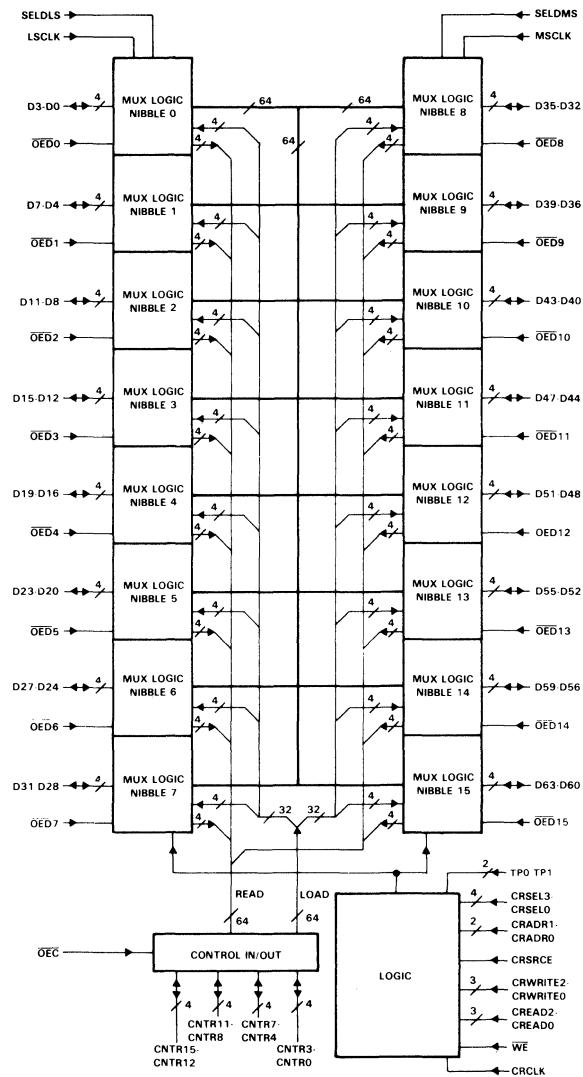


FIGURE 2

SN74ACT8841
DIGITAL CROSSBAR SWITCH

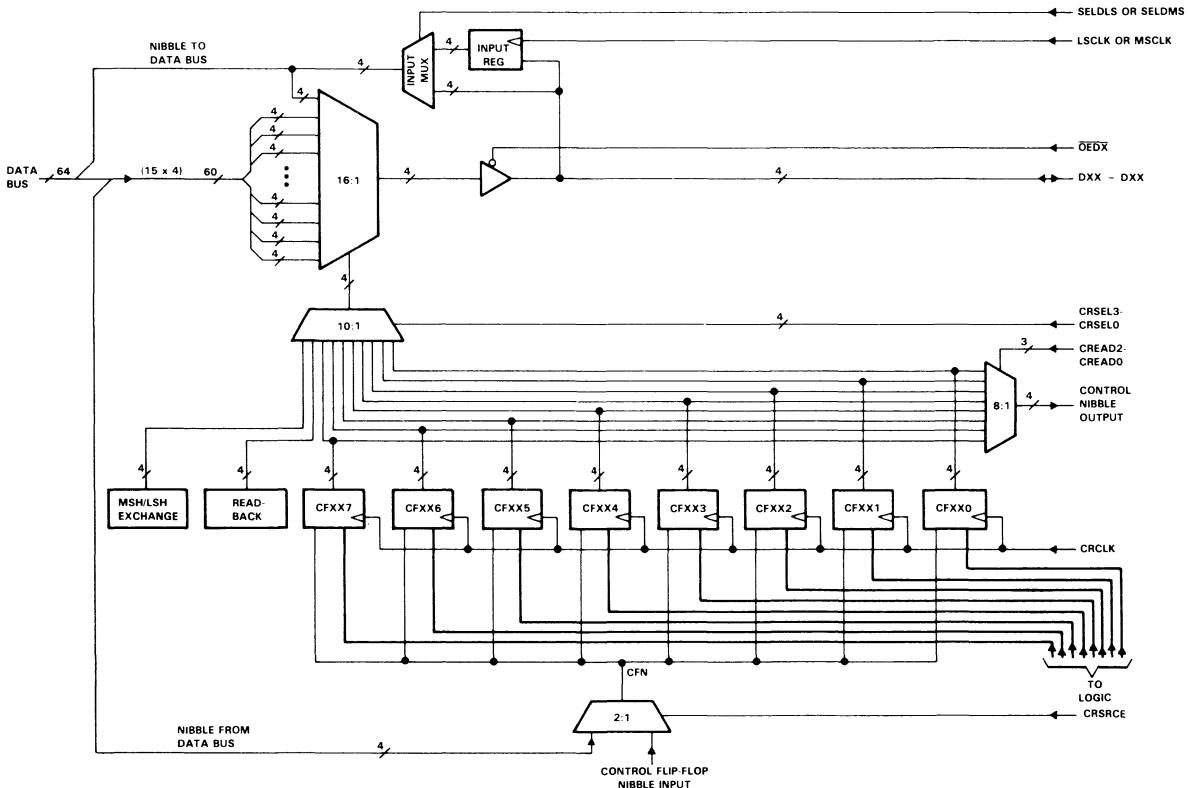


FIGURE 3. DATA NIBBLE MULTIPLEXER LOGIC



SN74ACT8841 DIGITAL CROSSBAR SWITCH

multiplexer logic group

There are 16 multiplexer logic blocks, one for each nibble. External data flows from four data I/O pins into a logic block. A block diagram of the multiplexer logic is shown in Figure 3. The data inputs are either clocked into the data register or passed directly to the main internal bus. The 64 bits of data from the main bus are presented to a 16-to-1 multiplexer, which selects the data nibble output.

Each of the 16 nibble multiplexer logic blocks contains eight control flip-flop (CF) groups, one for each of the control banks. A control bank stores one complete switching configuration. Each CF group consists of four D-type edge-triggered flip-flops. In Figure 3, the CF groups are shown as CFX_X0 to CFX_X7, where XX indicates the number of the nibble multiplexer logic group ($0 \leq X \leq 15$). CFX_X0 represents the 16 CF groups (one from each logic block) which make up flip-flop control bank 0, CFX_X1 the 16 CF groups in bank 1, etc.

In addition to the eight banks of programmable flip-flops, two hard-wired switching configurations can be selected. The MSH/LSH exchange directs the input nibbles from each half of the switch to the data outputs directly opposite. This switching pattern is shown in Table 3 below. For example, data input on D11-D8 is output on D43-D40, and data input on D43-D40 is output on D11-D8.

Table 3. MSH/LSH Exchange

LSH	MSH
D3-D0	↔ D35-D32
D7-D4	↔ D39-D36
D11-D8	↔ D43-D40
D15-D12	↔ D47-D44
D19-D16	↔ D51-D48
D23-D20	↔ D55-D52
D27-D24	↔ D59-D56
D31-D28	↔ D63-D60

The second hard-wired configuration, a read-back function, causes all 64 bit to be output on the same I/Os on which they were input. Neither of the hard-wired control configurations affects the contents of the control banks.

The control source select, CRSEL3-CRSEL0, determines which switching pattern is selected, as shown in Table 4.

Table 4. 16-to-1 Output Multiplexer Control Source Selects

CRSEL3	CRSEL2	CRSEL1	CRSEL0	CONTROL SOURCE SELECTED
L	L	L	L	Control bank 0 (programmable)
L	L	L	H	Control bank 1 (programmable)
L	L	H	L	Control bank 2 (programmable)
L	L	H	H	Control bank 3 (programmable)
L	H	L	L	Control bank 4 (programmable)
L	H	L	H	Control bank 5 (programmable)
L	H	H	L	Control bank 6 (programmable)
L	H	H	H	Control bank 7 (programmable)
H	X	X	L	MSH/LSH exchange*
H	X	X	H	Read-back (output echoes input)*

*Hard-wired switching configuration

X = don't care



POST OFFICE BOX 655012 • DALLAS, TEXAS 75265

control words

A CF group can store a four-bit control word (CFN3-CFN0) to select the output of the 16-to-1 multiplexer for that nibble port. One control word is loaded in each CF group. A total of 16 words, one per multiplexer logic block, are loaded in a bank to configure one complete switching pattern. Table 5 lists the control words and the input data each selects.

Each control word can be stored in a CF group and sent as an internal control signal to select the output of a 16-to-1 multiplexer in a nibble logic block. For example, any CF group loaded with the word "LHHH" will select the data input on D31-D28 as the outputs of the associated nibble. If all 16 CF groups in a bank were loaded with "LHHH," the same output (D31-D28) would be selected by the entire switch.

Table 5. 16-to-1 Output Multiplexer Control Words

INTERNAL SIGNALS				INPUT DATA SELECTED AS MULTIPLEXER OUTPUT
CFN3	CFN2	CFN1	CFN0	
L	L	L	L	D3-D0
L	L	L	H	D7-D4
L	L	H	L	D11-D8
L	L	H	H	D15-D12
L	H	L	L	D19-D16
L	H	L	H	D23-D20
L	H	H	L	D27-D24
L	H	H	H	D31-D28
H	L	L	L	D35-D32
H	L	L	H	D39-D36
H	L	H	L	D43-D40
H	L	H	H	D47-D44
H	H	L	L	D51-D48
H	H	L	H	D55-D52
H	H	H	L	D59-D56
H	H	H	H	D63-D60

loading control configurations

CRWRITE2-CRWRITE0 select which control bank is being loaded, as shown in Table 6.

Table 6. Control Flip-Flops Load Destination Select

CRWRITE2	CRWRITE1	CRWRITE0	DESTINATION
L	L	L	Control bank 0
L	L	H	Control bank 1
L	H	L	Control bank 2
L	H	H	Control bank 3
H	L	L	Control bank 4
H	L	H	Control bank 5
H	H	L	Control bank 6
H	H	H	Control bank 7

SN74ACT8841 DIGITAL CROSSBAR SWITCH

The control words for a bank can be loaded either 16 bits at a time on the control I/O pins (CNTR15-CNTR0) or all 64 bits at once on the data inputs (D63-D0). If the control load source select, CRSRCE, is high, the words are loaded from the data inputs. When CRSRCE = L, the CNTR inputs are used.

When a control bank is loaded from the data inputs, \overline{WE} , CRSRCE, CRWRITE2-CRWRITE0, and the control register clock CRCLK are used in combination to load all 16 control words (64 bits) in a single cycle. A MSH/LSH exchange like that shown in Table 3 is used to load the flip flops on a rising CRCLK clock edge. For example, data inputs D3-D0 go to the data bus and then to the CF group that selects the data outputs for D35-D32. CRWRITE2-CRWRITE0 select the control bank that is loaded (see Table 6).

The CNTR15-CNTR0 inputs can also be used to load the control banks. The bank is selected by CRWRITE2-CRWRITE0 (see Table 6). Four control words per CRCLK cycle can be input to the CF groups (CFXX) that make up the bank. The CF groups loaded are selected by CRADR1-CRADR0, as shown in Table 7. Four CRCLK cycles are needed to load an entire control bank.

Table 7. Loading Control Flip-Flops from CNTR I/Os

CRAD1	CRAD0	\overline{WE}	CRCLK	CF GROUPS LOADED BY CONTROL (CNTR) I/O NUMBERS			
				15-12	11-8	7-4	3-0
L	L	L	—	CF12	CF8	CF4	CF0
L	H	L	—	CF13	CF9	CF5	CF1
H	L	L	—	CF14	CF10	CF6	CF2
H	H	L	—	CF15	CF11	CF7	CF3
X	X	H	X	Inhibit write to flip-flops			

To read out the control settings, the same address signals can be used, except that no CRCLK signal is needed and \overline{OEC} is pulled low. CREAD2-CREAD0 select the bank to be read; the format is the same as for CRWRITE2-CRWRITE0, shown in Table 6.

Using the control I/Os to read the control bank settings can be valuable during debugging or diagnostics. Control settings are volatile and will be lost if the 'ACT8841 is powered off. An external program controlling switch operation may need to read the control bank settings so that it can save and restore the current switching configurations.

6

test pins

TP1-TPO test pins are provided for system testing. As Table 8 shows, these pins should be maintained high during normal operation. To force all outputs and I/Os low, low signals are placed on TP1-TPO and all output enables ($\overline{OED15-OED0}$ and \overline{OEC}). To force all outputs and I/Os high, TP1 and all output enables are pulled low, and TPO is driven high. When TPO is left low and a high signal is placed on TP1, all outputs on the 'ACT8841 are placed in a high-impedance state, isolating the chip from the rest of the system.

Table 8. Test Pin Inputs

TP1	TPO	$\overline{OED15-OED0}$	\overline{OEC}	RESULT
L	L	L	L	All outputs and I/Os forced low
L	H	L	L	All outputs and I/Os forced high
H	L	X	X	All outputs placed in a high-impedance state
H	H	X	X	Normal operation (default state)

examples

Most 'ACT8841 switch configurations are straightforward to program, involving few control signals and procedures to set up the control words in the banks of flip-flops. Control signals and procedures for loading and using control words are shown in the following examples.

broadcasting a nibble

Any of the 16 data input nibbles can be broadcast to the other 15 data nibbles for output. For ease of presentation, input nibble D63-D60 is used in this example. Example 1 presents the microcode sequence for loading flip-flop bank 0 and executing the nibble broadcast.

The low signal on CRSRCE selects CNTR15-CNTR0 as the input source, and the low signals on CRWRITE2-CRWRITE0 select flip-flop bank 0 as the destination. Table 5 shows that to select data on D63-D60 as the output nibble, the four bits in the control word CFN3-CFNO must be high; therefore the CNTR15-CNTR0 inputs are coded high. The four microcode instructions shown in Example 1 load the same control word from CNTR15-CNTR0 into all 16 CF groups of bank 0.

Once the control flip-flops have been loaded, the switch can be used to broadcast nibble D63-D60 as programmed. The microcode instruction to execute the broadcast is shown as the last instruction in Example 1. WE is held high and the data to be broadcast is input on D63-D60. The high signal on SELDMS selects a real-time data input for the broadcast. MSCLK and LSCLK (not shown) can be used to load the input registers if the input nibble is to be retained. No register clock signals are needed if the input data is not being stored.

The banks of control flip-flops not selected as a control source can be loaded with new control words or read out on CNTR15-CNTR0 while the switch is operating. For example, the MSH data inputs can be used to load flip-flop bank 1 of the LSH while bank 0 of the LSH is controlling data I/O.

PRODUCT REVIEW

SN74ACT8841

9

SN74ACT8841 DIGITAL CROSSBAR SWITCH

Example 1. Programming a Nibble Broadcast

INST. NO.	CRSRCE	CRWRITE2	CRWRITE1	CRWRITED	CRADR1	CRADRO	CNTR I/O NUMBERS				CRSEL3	CRSEL2	CRSEL1	CRSELO	WE	SELDMS	SELDS	QED15-QED0		DEC	CRCLK
							15-12	11-8	7-4	3-0											
1	0	0	0	0	0	0	1111	1111	1111	1111	X	X	X	X	0	X	X	XXXX XXXX XXXX XXXX	1	1	1
2	0	0	0	0	0	1	1111	1111	1111	1111	X	X	X	X	0	X	X	XXXX XXXX XXXX XXXX	1	1	1
3	0	0	0	0	1	0	1111	1111	1111	1111	X	X	X	X	0	X	X	XXXX XXXX XXXX XXXX	1	1	1
4	0	0	0	0	1	1	1111	1111	1111	1111	X	X	X	X	0	X	X	XXXX XXXX XXXX XXXX	1	1	1
5	X	X	X	X	X	X	XXXX	XXXX	XXXX	XXXX	0	0	0	0	1	1	X	1000 0000 0000 0000	1	None	

Comments

INST. NO.	COMMENT
1	Loads CF12, CF8, CF4, CFO of bank 0
2	Loads CF13, CF9, CF5, CF1 of bank 0
3	Loads CF14, CF10, CF6, CF2 of bank 0
4	Loads CF15, CF11, CF7, CF3 of bank 0
5	Selects bank 0 for switching control Selects real-time data inputs

Example 2. Programming an MSH/LSH Exchange on CNTR Inputs

INST. NO.	CRSRCE	CRWRITE2	CRWRITE1	CRWRITED	CRADR1	CRADRO	CNTR I/O NUMBERS				CRSEL3	CRSEL2	CRSEL1	CRSELO	WE	SELDMS	SELDS	QED15-QED0		DEC	CRCLK
							15-12	11-8	7-4	3-0											
1	0	1	1	1	0	0	0100	0000	1100	1000	X	X	X	X	0	X	X	XXXX XXXX XXXX XXXX	1	1	1
2	0	1	1	1	0	1	0101	0001	1101	1001	X	X	X	X	0	X	X	XXXX XXXX XXXX XXXX	1	1	1
3	0	1	1	1	1	0	0111	0011	1111	1011	X	X	X	X	0	X	X	XXXX XXXX XXXX XXXX	1	1	1
4	0	1	1	1	1	1	0111	0011	1111	1011	X	X	X	X	0	X	X	XXXX XXXX XXXX XXXX	1	1	1
5	X	X	X	X	X	X	XXXX	XXXX	XXXX	XXXX	0	1	1	1	0	0	0	0000 0000 0000 0000	1	None	

Comments

INST. NO.	COMMENT
1	Loads CF12, CF8, CF4, CFO of bank 7
2	Loads CF13, CF9, CF5, CF1 of bank 7
3	Loads CF14, CF10, CF6, CF2 of bank 7
4	Loads CF15, CF11, CF7, CF3 of bank 7
5	Selects bank 7 for switching control Selects registered data inputs

programming an MSH/LSH exchange

A second, more complicated example involves programming the switch to swap corresponding nibbles between the MSH and the LSH (first nibble in the LSH for first nibble in the MSH, and so on). This swap can be implemented using the hard-wired logic circuit selected when CRSEL3 is high and CRSEL0 is low. Programming this swap without using the MSH/LSH exchange logic requires loading a different control word into each mux logic block. This is described below for purposes of illustration.

Each nibble in one half, either LSH or MSH, selects as output the registered data from the corresponding nibble in the other half. The registered data from D35-D32 is to be output on D3-D0, the registered data from D3-D0 is output on D35-D32, and so on for the remaining nibbles. As shown in Table 4, the flip-flops for D3-D0 have to be set to 1000 and the D35-D32 inputs must be low. The CF groups and control words involved in this switching pattern are listed in Table 9.

Table 9. Control Words for an MSH/LSH Exchange

CF GROUP	CNTR INPUTS TO LOAD FLIP-FLOPS	CONTROL WORD LOADED	RESULTS
CF15	CNTR15-CNTR12	0111	D31-D28 → D63-D60
CF14		0110	D27-D24 → D59-D56
CF13		0101	D23-D20 → D55-D52
CF12		0100	D19-D16 → D51-D48
CF11	CNTR11-CNTR8	0011	D15-D12 → D47-D44
CF10		0010	D11-D8 → D43-D40
CF9		0001	D7-D4 → D39-D36
CF8		0000	D3-D0 → D35-D32
CF7	CNTR7-CNTR4	1111	D63-D60 → D31-D28
CF6		1110	D59-D56 → D27-D24
CF5		1101	D55-D52 → D23-D20
CF4		1100	D51-D48 → D19-D16
CF3	CNTR3-CNTR0	1011	D47-D44 → D15-D12
CF2		1010	D43-D40 → D11-D8
CF1		1001	D39-D36 → D7-D4
CF0		1000	D35-D32 → D3-D0

With this list of control words and the signals in Table 7, the 16-bit control inputs on CNTR15-CNTR0 can be arranged to load the control flip-flops in four cycles. Example 2 shows the microcode instructions for loading the control words and executing the exchange.

In Example 2, bank 7 of flip-flops is being programmed. Bank 7 is selected by taking CRWRITE2-CRWRITE0 high and leaving CRSRCE low (see Table 4) when the control words are loaded on CNTR15-CNTR0. With WE held low, the CRCLK is used to load the four sets of control words. Once the flip-flops are loaded, data can be input on D63-D0 and the programmed pattern of output selection can be executed. A microinstruction to select registered data inputs and bank 7 as the control source is shown as the last instruction in Example 2. The data must be clocked into the input registers, using LSCLK and MSCLK, before the last instruction is executed.

SN74ACT8841

DIGITAL CROSSBAR SWITCH

The control flip-flops could also have been loaded from the data input nibbles in one CRCLK cycle. Input nibbles from one half are mapped onto the control flip-flops of the other half. All control words to set up a switching pattern should be loaded before the bank of flip-flops is selected as control source. The microcode instructions to load bank 1 with the 16 control words in one cycle are presented in Example 3.

Example 3. Loading the MSH/LSH Exchange from Data Inputs

CRSRCE	CRWRITE2	CRWRITE1	CRWRITE0	WE	SELDMS	SELDSL	OE _D 15-OE _D 0	CRCLK
1	0	0	1	0	1	1	1111 1111 1111 1111	✓

These control nibbles may be loaded from the input as a 64-bit real-time input word or as two 32-bit words stored previously. To use stored control words, MSCLK and LSCLK are used to load the LSH and MSH input registers with the correct sequence of control nibbles. Whenever the flip-flops are loaded from the data inputs, all 64 bits of control data must be present when the CRCLK is used so that all control nibbles in a program are loaded simultaneously. Example 4 presents the three microcode instructions to load the MSH and LSH input registers and then to pass the registered data to flip-flop bank 2.

Example 4. Loading Control Flip-Flops from Input Registers

INST. NO.	CRSRCE	CRWRITE2	CRWRITE1	CRWRITE0	WE	SELDMS	SELDSL	OE _D 15- OE _D 0	CRCLK	MSCLK	LSCLK	COMMENTS
1	X	X	X	X	1	X	X	1	None	✓	None	Load inputs D63-D32
2	X	X	X	X	1	X	X	1	None	None	✓	Load inputs D31-D0
3	1	0	1	0	0	0	0	1	✓	None	None	Load control bank 2

The control words in a program can also be read back from the flip-flops using the CNTR outputs. Four instructions are necessary to read the 64 bits in a bank of flip-flops out on CNTR15-CNTR0. WE is held high and OE_C is taken low. No CRCLK signal is required. CREAD2-CREAD0 select bank 2 of flip-flops, and CRADR1-CRADR0 select in sequence the four addresses of the 16-bit words to be read out on the CNTR outputs. Example 5 shows the four microcode instructions.

Example 5. Reading Control Settings on CNTR Outputs

INST. NO.	CREAD2	CREAD1	CREAD0	OE _C	CRADR1	CRADR0	WE	CNTR I/O NUMBERS			COMMENT
								15-12	11-8	7-4	3-0
1	0	1	0	0	0	0	1	0100	0000	1100	1000
2	0	1	0	0	0	1	1	0101	0001	1101	1001
3	0	1	0	0	1	0	1	0110	0010	1110	1010
4	0	1	0	0	1	1	1	0111	0011	1111	1011



POST OFFICE BOX 655012 • DALLAS, TEXAS 75265

SN74ACT8841
DIGITAL CROSSBAR SWITCH

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 = 0 to 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	TEST CONDITIONS	V _{CC}	T _A = 25°C			MIN	NOM	MAX	UNIT
			MIN	TYP	MAX				
V _{CC}	Supply voltage		4.5	5.0	5.5	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				
d _{t/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			MIN	TYP	MAX	UNIT
			MIN	TYP	MAX				
V _{OH}	I _{OH} = -20 μA	4.5 V			4.4	V			
		5.5 V			5.4				
	I _{OH} = -8 mA	4.5 V	3.8	3.7					
		5.5 V	4.8	4.7					
V _{OL}	I _{OL} = 20 μA	4.5 V			0.1	V			
		5.5 V			0.1				
	I _{OL} = 8 mA	4.5 V	0.32	0.4					
		5.5 V	0.32	0.4					
I _{OZ}	V _O = V _{CC} or 0	5 V		±0.5		±0.5	μA		
I _I	V _I = V _{CC} or 0	5.5 V	0.1			±1	μA		
I _{CC}	V _I = V _{CC} or 0, I _O	5.5 V				100	μA		
C _I	V _I = V _{CC} or 0	5 V					pF		

[†]This is the increase in supply current for each input that is at one of the specified TTL voltage levels rather than 0 V or V_{CC}.

PRODUCT PREVIEW

6

SN74ACT8841

TEXAS
INSTRUMENTS

POST OFFICE BOX 655012 • DALLAS, TEXAS 75265

SN74ACT8841

DIGITAL CROSSBAR SWITCH

switching characteristics over recommended ranges of supply voltage and operating free-air temperature (unless otherwise noted)

PARAMETER	FROM	TO	MIN	TYP [†]	MAX	UNIT
t_{pd}	Data in	Data out		7	14	ns
	MSCLK, LSCLK			10	18	
	SELDMS, SELDLS			9	15	
	CRCLK			12	19	
	CRSEL3-CRSEL0			12	19	
	CREAD2-CREAD0			10	18	
	CRCLK			10	18	
	CRAD1, CRAD0			8	16	
t_{en}	TP1, TP0	All outputs		10	19	ns
	TP1, TP0	All outputs		10	15	
	\overline{OED}	Data out		7	12	
t_{dis}	\overline{OEC}	CNTRn		8	14	ns
	TP1, TP0	All outputs		10	15	
	\overline{OED}	Data out		5	8	
	\overline{OEC}	CNTRn		6	10	

[†]All typical values are at VCC = 5 V, TA = 25°C.

timing requirements over recommended ranges of supply voltage and operating free-air temperature (unless otherwise noted)

PARAMETER		MIN	MAX	UNIT
t_w	Pulse duration	LSCLK, MSCLK, CRCLK high or low	7	ns
	Data		7	
	CNTRn		7	
	SELDMS, SELDLS		9	
	CRADR1, CRAD0		8	
	CRSRCE, CRWRITE2-CRWRITED		8	
	LSCLK, MSCLK		10	
	\overline{WE}		8	
t_{su}	Setup time before CRCLK		7	ns
	Data		0	
	CNTRn		0	
	SELDMS, SELDLS		0	
	CRADR1, CRAD0		0	
	CRSRCE, CRWRITE		0	
t_h	Hold time after CRCLK	\overline{WE}	0	ns
	Data		0	
	CNTRn		0	
	SELDMS, SELDLS		0	
	CRADR1, CRAD0		0	
t_h	Hold time, data after LSCLK or MSCLK		0	ns
	\overline{WE}		0	

'AS8840 AND 'ACT8841 FUNCTIONAL COMPARISON

differences between the SN74AS8840 and the SN74ACT8841

The SN74AS8840 and the SN74ACT8841 digital crossbar switches essentially perform the same function. The SN74AS8840 and the SN74ACT8841 are based on the same 16-port architecture, differing in the number of control registers, power consumption, and pin-out.

One difference is in the number of programmable control flip-flop banks available to configure the switch. The 'AS8840 has two programmable control banks, while the 'ACT8841 has eight. Both have two selectable hard-wired switching configurations.

The increased number of control banks in the 'ACT8841 require six additional pins not found on the 'AS8840. These are: CRWRITE2, CRWRITE1, CREAD2, CREAD1, CRSEL3, and CRSEL2. CREAD and CRWRITE on the '8840 become CREAD0 and CRWRITE0 on the '8841. On the '8840, CRSEL1 selects the hardwired control functions when high. This function is performed by the CRSEL3 signal on the '8841. Therefore, CRSEL2 and CRSEL1 are actually the added signals.

The 'ACT8841 is a low-power CMOS device requiring only 5-V power. Because of its STL internal logic and TTL I/Os, the 'AS8840 requires both 2-V and 5-V power.

Both the 'AS8840 and the 'ACT8841 are in 156 pin grid-array packages, however, the two devices are not pin-for-pin compatible. Control signals were added to the 'ACT8841 and the 2-V VCC pins ('AS8840 only) were assigned other functions in the 'ACT8841.

changing 'AS8840 microcode to 'ACT8841 microcode

Since only six signals have been added to the 'ACT8841, changing existing 'AS8840 microcode to 'ACT8841 microcode is straight forward. CRSEL3 on the 'ACT8841 is functionally equivalent to CRSEL1 on the 'AS8840. CREAD2, CREAD1, CRWRITE2, CRWRITE1, CRSEL2, and CRSEL1 bits must be added. These can always be 0 if no additional control banks are needed. Additional control configurations can be stored by programming these bits.

All other signals in the 'AS8840 microcode remain the same when converting to 'ACT8841 microcode.

6

SN74ACT8841

Overview

1

SN74ACT8818 16-Bit Microsequencer

2

SN74ACT8832 32-Bit Registered ALU

3

SN74ACT8836 32- × 32-Bit Parallel Multiplier

4

SN74ACT8837 64-Bit Floating Point Processor

5

SN74ACT8841 Digital Crossbar Switch

6

SN74ACT8847 64-Bit Floating Point/Integer Processor

7

Support

8

Mechanical Data

9

7

SN74ACT8847

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.

7

SN74ACT8847

Contents

	<i>Page</i>
Overview	7-23
Understanding the 'ACT8847 Floating Point Unit	7-23
Microprogramming the 'ACT8847	7-23
Support Tools	7-24
Design Support	7-24
Design Expertise	7-24
'ACT8847 Logic Symbol	7-25
'ACT8847 Pin Descriptions	7-26
'ACT8847 Specifications	7-35
'ACT8847 Load Circuit	7-43
SN74ACT8847 64-Bit Floating Point Unit	7-50
Introduction	7-50
Major Architectural Features	7-50
Data Flow in Pipelined Architectures	7-52
Control Architectures for High-Speed Microprogrammed Architectures	7-54
Microprogram Control of an 'ACT8847 FPU Subsystem	7-57
'ACT8847 Data Formats	7-57
Status Outputs	7-60
SN74ACT8847 Architecture	7-60
Overview	7-60
Pipeline Controls	7-62
Temporary Input Register	7-64
RA and RB Input Registers	7-65
Configuration Controls	7-65
Clock Mode Settings	7-66
Operand Selection	7-68
C Register	7-70
Pipelined ALU	7-72
Pipelined Multiplier	7-73
Data Output Controls	7-74
Parity Checker/Generator	7-75
Master/Slave Comparator	7-75
Status and Exception Generation	7-77

Contents (Continued)

	<i>Page</i>
Microprogramming the 'ACT8847	7-82
Control Inputs	7-82
Rounding Modes	7-84
FAST and IEEE Modes	7-84
Handling of Denormalized Numbers	7-84
Stalling the Device	7-86
Reset	7-86
Test Pins	7-86
Independent ALU Operations	7-87
Independent Multiplier Operations	7-93
Chained Multiplier/ALU Operations	7-96
Sample Independent ALU Microinstructions	7-98
Sample Independent Multiplier Microinstructions	7-106
Sample Chained Mode Microinstructions	7-126
Instruction Timing	7-134
Exception and Status Handling	7-136
'ACT8847 Reference Guide	7-139
Instruction Inputs	7-139
Input Configuration	7-144
Operand Source Select	7-144
Pipeline Control	7-145
Round Control	7-145
Status Output Selection	7-146
Test Pin Control	7-146
Miscellaneous Control Inputs	7-147
Glossary	7-147
SN74ACT8847 Application Notes	7-148
Sum of Products and Product of Sums	7-148

Contents (Continued)

	<i>Page</i>
Matrix Operations	7-151
Representation of Variables	7-151
Sample Matrix Transformation	7-152
Microinstructions for Sample Matrix Manipulation	7-158
Chebyshev Routines for the SN74ACT8847 FPU	7-162
Introduction	7-162
Overview of Chebyshev's Expansion Method	7-163
Format for the Remainder of the Application Note	7-165
References	7-166
Cosine Routine Using Chebyshev's Method	7-166
Steps Required to Perform the Calculation	7-166
Algorithms for the Three Steps	7-167
Required System Intervention	7-168
Number of 'ACT8847 Cycles Required to Calculate Cosine(x)	7-168
Listing of the Chebyshev Constants (c's)	7-168
Pseudocode Table for the Cosine(x) Calculation	7-169
Microcode Table for the Cosine(x) Calculation	7-172
Sine Routine Using Chebyshev's Method	7-174
Steps Required to Perform the Calculation	7-174
Algorithms for the Three Steps	7-174
Required System Intervention	7-175
Number of 'ACT8847 Cycles Required to Calculate Sine(x)	7-175
Listing of the Chebyshev Constants (c's)	7-175
Pseudocode Table for the Sine(x) Calculation	7-176
Microcode Table for the Sine(x) Calculation	7-179

Contents (Continued)

	<i>Page</i>
Tangent Routine Using Chebyshev's Method	7-181
Steps Required to Perform the Calculation	7-181
Algorithms for the Three Steps	7-181
Required System Intervention	7-183
Number of 'ACT8847 Cycles Required to Calculate Tangent(x)	7-183
Listing of the Chebyshev Constants (c's)	7-183
Pseudocode Table for the Tangent(x) Calculation	7-184
Microcode Table for the Tangent(x) Calculation	7-188
ArcSine and ArcCosine Routine Using Chebyshev's Method	7-192
Steps Required to Perform the Calculation	7-192
Algorithms for the Three Steps	7-192
Required System Intervention	7-194
Number of 'ACT8847 Cycles Required to Calculate ArcSine(x) and ArcCosine(x)	7-194
Listing of the Chebyshev Constants (c's)	7-194
Pseudocode Table for the ArcSine(x) and ArcCosine Calculation	7-195
Microcode Table for the ArcSine(x) and ArcCosine(x) Calculation	7-198
ArcTangent Routine Using Chebyshev's Method	7-201
Steps Required to Perform the Calculation	7-201
Algorithms for the Three Steps	7-202
Required System Intervention	7-203
Number of 'ACT8847 Cycles Required to Calculate Arctangent(x)	7-203
Listing of the Chebyshev Constants (c's)	7-204
Pseudocode Table for the ArcTangent(x) Calculation	7-205
Microcode Table for the ArcTangent(x) Calculation	7-210

Contents (Concluded)

	<i>Page</i>
Exponential Routine Using Chebyshev's Method	7-214
Steps Required to Perform the Calculation	7-214
Algorithms for the Three Steps	7-215
Required System Intervention	7-216
Number of 'ACT8847 Cycles Required to Calculate Exp(x)	7-216
Listing of the Chebyshev Constants (c's)	7-216
Pseudocode Table for the Exp(x) Calculation	7-217
Microcode Table for the Exp(x) Calculation	7-220
High-Speed Vector Math and 3-D Graphics	7-223
Introduction	7-223
SN74ACT8837 and SN74ACT8847 Floating Point Units	7-223
Mathematical Processing Applications	7-227
Graphics Applications	7-227
Vector Arithmetic	7-228
Computational Operations on Data Vectors	7-229
Compare Operations on Data Vectors	7-239
Graphics Applications	7-243
Creating a 3-D Image	7-244
Three-Dimensional Coordinate Transforms	7-247
Three-Dimensional Clipping	7-250
Summary of Graphics Systems Performance	7-263

7

SN74ACT8847

List of Illustrations

<i>Figure</i>	<i>Page</i>
1 Load Circuit	7-43
2 Timing Diagram for: SP ALU → DP MULT → DP ALU → CONVERT DP TO SP	7-44
3 Timing Diagram for: DP ALU → DP MULT → DP ALU	7-46
4 Timing Diagram for: SP ((Scalar * Vector) + Vector)	7-48
5 High Level Block Diagram	7-51
6 Multiply/Accumulate Operation	7-52
7 Example of Fully Pipelined Operation	7-53
8 PLA Control Circuit Example	7-54
9 Microprogrammed Architecture	7-55
10 Microprogrammed Architecture with Address Register	7-56
11 IEEE Single-Precision Format	7-58
12 IEEE Double-Precision Format	7-58
13 'ACT8847 Detailed Block Diagram	7-61
14 Pipeline Controls	7-63
15 Input Register Control	7-65
16 Operand Selection Multiplexer	7-69
17 C Register Timing	7-71
18 Functional Diagram for ALU	7-72
19 Functional Diagram for Multiplier	7-73
20 Y Output Control	7-75
21 Example of Master/Slave Operation	7-76
22 Status Output Control	7-79
23 Exception Detect Mask Logic	7-81
24 Single-Precision Independent ALU Operation, All Registers Disabled (PIPES2-PIPES0 = 111, CLKMODE = X)	7-98
25 Single-Precision Independent ALU Operation, Input Registers Enabled (PIPES2-PIPES0 = 110, CLKMODE = X)	7-99
26 Single-Precision Independent ALU Operation, Input and Output Registers Enabled (PIPES2-PIPES0 = 010, CLKMODE = X)	7-100

List of Illustrations (Continued)

<i>Figure</i>		<i>Page</i>
27	Single-Precision Independent ALU Operation, All Registers Enabled (PIPES2-PIPES0 = 000, CLKMODE = X)	7-101
28	Double-Precision Independent ALU Operation, All Registers Disabled (PIPES2-PIPES0 = 111, CLKMODE = 0)	7-102
29	Double-Precision Independent ALU Operation, Input Registers Enabled (PIPES2-PIPES0 = 110, CLKMODE = 0)	7-103
30	Double-Precision Independent ALU Operation, Input and Output Registers Enabled (PIPES2-PIPES0 = 010, CLKMODE = 1)	7-104
31	Double-Precision Independent ALU Operation, All Registers Enabled (PIPES2-PIPES0 = 000, CLKMODE = 0)	7-105
32	Single-Precision Independent Multiplier Operation, All Registers Disabled(PIPES2-PIPES0 = 111, CLKMODE = X)	7-106
33	Single-Precision Independent Multiplier Operation, Input Registers Enabled (PIPES2-PIPES0 = 110, CLKMODE = X)	7-107
34	Single-Precision Independent Multiplier Operation, Input and Output Registers Enabled (PIPES2-PIPES0 = 010, CLKMODE = X)	7-108
35	Single-Precision Independent Multiplier Operation, All Registers Enabled (PIPES2-PIPES0 = 000, CLKMODE = X)	7-109
36	Double-Precision Independent Multiplier Operation, All Registers Disabled (PIPES2-PIPES0 = 111, CLKMODE = 0)	7-110
37	Double-Precision Independent Multiplier Operation, Input Registers Enabled (PIPES2-PIPES0 = 110, CLKMODE = 1)	7-111
38	Double-Precision Independent Multiplier Operation, Input and Output Registers Enabled (PIPES2-PIPES0 = 010, CLKMODE = 0)	7-112

List of Illustrations (Continued)

<i>Figure</i>		<i>Page</i>
39	Double-Precision Independent Multiplier Operation, All Registers Enabled (PIPES2-PIPES0 = 000, CLKMODE = 0)	7-113
40	Single-Precision Floating Point Division, Input Registers Enabled (PIPES2-PIPES0 = 110, CLKMODE = X)	7-114
41	Single-Precision Floating Point Division, Input and Pipeline Registers Enabled (PIPES2-PIPES0 = 100, CLKMODE = X)	7-114
42	Single-Precision Floating Point Division, Input and Output Registers Enabled (PIPES2-PIPES0 = 010, CLKMODE = X)	7-115
43	Single-Precision Floating Point Division, All Registers Enabled (PIPES2-PIPES0 = 000, CLKMODE = X)	7-115
44	Double-Precision Floating Point Division, Input Registers Enabled (PIPES2-PIPES0 = 110, CLKMODE = 0)	7-116
45	Double-Precision Floating Point Division, Input and Pipeline Registers Enabled (PIPES2-PIPES0 = 100, CLKMODE = 0)	7-116
46	Double-Precision Floating Point Division, Input and Output Registers Enabled (PIPES2-PIPES0 = 010, CLKMODE = 1)	7-117
47	Double-Precision Floating Point Division, All Registers Enabled (PIPES2-PIPES0 = 000, CLKMODE = 1)	7-117
48	Integer Division, Input Registers Enabled (PIPES2-PIPES0 = 100, CLKMODE = X)	7-118
49	Integer Division, Input and Pipeline Registers Enabled (PIPES2-PIPES0 = 100 CLKMODE = X)	7-118
50	Integer Division, Input and Output Registers Enabled (PIPES2-PIPES0 = 010, CLKMODE = X)	7-119
51	Integer Division, All Registers Enabled (PIPES2-PIPES0 = 000, CLKMODE = X)	7-119
52	Single-Precision Floating Point Square Root, Input Registers Enabled (PIPES2-PIPES0 = 110, CLKMODE = X)	7-120

SN74ACT8847

7

List of Illustrations (Continued)

<i>Figure</i>		<i>Page</i>
53	Single-Precision Floating Point Square Root, Input and Pipeline Registers Enabled (PIPES2-PIPES0 = 100, CLKMODE = X)	7-120
54	Single-Precision Floating Point Square Root, Input and Output Registers Enabled (PIPES2-PIPES0 = 010, CLKMODE = X)	7-121
55	Single-Precision Floating Point Square Root, All Registers Enabled (PIPES2-PIPES0 = 000, CLKMODE = X)	7-121
56	Double-Precision Floating Point Square Root, Input Registers Enabled (PIPES2-PIPES0 = 110, CLKMODE = 1)	7-122
57	Double-Precision Floating Point Square Root, Input and Pipeline Registers Enabled (PIPES2-PIPES0 = 100, CLKMODE = 0)	7-122
58	Double-Precision Floating Point Square Root, Input and Output Registers Enabled (PIPES2-PIPES0 = 010, CLKMODE = 1)	7-123
59	Double-Precision Floating Point Square Root, All Registers Enabled (PIPES2-PIPES0 = 000, CLKMODE = 0)	7-123
60	Integer Square Root, Input Registers Enabled (PIPES2-PIPES0 = 110, CLKMODE = X)	7-124
61	Integer Square Root, Input and Pipeline Registers Enabled (PIPES2-PIPES0 = 100, CLKMODE = X)	7-124
62	Integer Square Root, Input and Output Registers Enabled (PIPES2-PIPES0 = 010, CLKMODE = X)	7-125
63	Integer Square Root, All Registers Enabled (PIPES2-PIPES0 = 000, CLKMODE = X)	7-125
64	Single-Precision Chained Mode Operation, All Registers Disabled (PIPES2-PIPES0 = 111, CLKMODE = X)	7-126
65	Single-Precision Chained Mode Operation, Input Registers Enabled (PIPES2-PIPES0 = 110, CLKMODE = 1)	7-127
66	Single-Precision Chained Mode Operation, Input and Output Registers Enabled (PIPES2-PIPES0 = 010, CLKMODE = X)	7-128

List of Illustrations (Concluded)

<i>Figure</i>		<i>Page</i>
67	Single-Precision Chained Mode Operation, All Registers Enabled (PIPES2-PIPES0 = 000, CLKMODE = X)	7-129
68	Double-Precision Chained Mode Operation, All Registers Disabled (PIPES2-PIPES0 = 111, CLKMODE = 0)	7-130
69	Double-Precision Chained Mode Operation, Input Registers Enabled (PIPES2-PIPES0 = 110, CLKMODE = 1)	7-131
70	Double-Precision Chained Mode Operation, Input and Output Registers Enabled (PIPES2-PIPES0 = 010, CLKMODE = 0)	7-132
71	Double-Precision Chained Mode Operation, All Registers Enabled (PIPES2-PIPES0 = 000, CLKMODE = 0)	7-133
72	Sequence of Matrix Operations	7-154
73	Resultant Matrix Transformation	7-161
74	SN74ACT8837 Floating Point Unit	7-225
75	SN74ACT8847 Floating Point Unit	7-226
76	Creating a 3-D Image	7-245
77	View Volume	7-246
78a	Model of Procedure for Creating a 3-D Graphic	7-247
78b	Model of Creating and Transforming a 3-D Graphic	7-247
79	Viewing Pyramid Showing Six Clipping Planes	7-251

7

SN74ACT8847

List of Tables

<i>Table</i>	<i>Page</i>
1 'ACT8847 Pin Grid Allocation	7-27
2 'ACT8847 Pin Functional Description	7-28
3 Sum of Products Calculation	7-51
4 IEEE Floating Point Number Representations	7-59
5 Pipeline Controls (PIPES2-PIPES0)	7-64
6 Double-Precision Input Data Configuration Modes	7-66
7 Single-Precision Input Data Configuration Mode	7-66
8a Double-Precision CREG + PREG Using CLKMODE = 0, PIPES = 010	7-67
8b Double-Precision CREG + PREG Using CLKMODE = 1, PIPES = 010	7-67
9a Double-Precision PREG + RB Using CLKMODE = 0, PIPES = 010	7-68
9b Double-Precision PREG + RB Using CLKMODE = 1, PIPES = 010	7-68
10 Multiplier Input Selection	7-68
11 ALU Input Selection	7-70
12 Independent ALU Operations	7-73
13 Independent Multiplier Operations	7-74
14 Comparison Status Outputs	7-77
15 Status Outputs	7-78
16 Status Output Selection (Chained Mode)	7-80
17 Control Inputs	7-83
18 Rounding Modes	7-84
19 Handling Wrapped Multiplier Outputs	7-85
20 Test Pin Control Inputs	7-86
21 Independent ALU Operations, Single Floating Point Operand (I10 = 0, I9 = 0, I7 = 0, I6 = 0)	7-88
22 Independent ALU Operations, Single Integer Operand (I10 = 0, I9 = 1, I6 = 0)	7-89
23 Independent ALU Operations, Two Floating Point Operands (I10 = 0, I9 = 0, I5 = 0)	7-91
24 Independent ALU Operations, Two Integer Operands (I10 = 0, I9 = 1, I6 = 0)	7-91
25 Loading the Exception Detect Mask Register	7-92

List of Tables (Continued)

<i>Table</i>	<i>Page</i>
26 NOP Instruction	7-92
27 Independent Multiplier Operations	7-94
28 Independent Multiply Operations Selected by I4-I2 (I10 = 0, I6 = 1, I5 = 0)	7-95
29 Independent Divide/Square Root Operations Selected by I4-I2 (I10 = 0, I6 = 1, I5 = 1)	7-95
30 Chained Multiplier/ALU Operations (I10 = 1)	7-97
31 Number of Clocks Required to Complete an Operation	7-134
32 NOPs Inserted to Guarantee that Double-Precision Results Remain Valid for Two Clock Cycles (PIPES2-PIPES0 = 000)	7-135
33 Independent ALU Operations, Single Floating Point Operand	7-139
34 Independent ALU Operations, Two Floating Point Operands	7-140
35 Independent ALU Operations, One Integer Operand	7-140
36 Independent ALU Operations, Two Integer Operands	7-141
37 Independent Floating Point Multiply Operations	7-141
38 Independent Floating Point Divide/Square Root Operations	7-141
39 Independent Integer Multiply/Divide/Square Root Operations	7-142
40 Chained Multiplier/ALU Floating Point Operations	7-142
41 Chained Multiplier/ALU Integer Operations	7-143
42 Double-Precision Input Data Configuration Modes	7-144
43 Multiplier Input Selection	7-144
44 ALU Input Selection	7-145
45 Pipeline Controls (PIPES2-PIPES0)	7-145
46 Rounding Modes	7-145
47 Status Output Selection (Chained Mode)	7-146
48 Test Pin Control Inputs	7-146
49 Miscellaneous Control Inputs	7-147
50 Pseudocode for Fully Pipelined Double-Precision Sum of Products (CLKMODE = 0, CONFIG1-CONFIG0 = 10, PIPES2-PIPES0 = 000)	7-149

List of Tables (Continued)

<i>Table</i>		<i>Page</i>
51 Pseudocode for Fully Pipelined Double-Precision Product of Sums (CLKMODE = 0, CONFIG1-CONFIG0 = 10, PIPES2-PIPES0 = 000)		7-150
52 Single-Precision Matrix Multiplication (PIPES2-PIPES0 = 010)		7-159
53 Microinstructions for Sample Matrix Multiplication		7-160
54 Fully Pipelined Single-Precision Sum of Products (PIPES2-PIPES0 = 000)		7-162
55 Cycle Count and Execution Speed for the Seven Chebyshev Functions		7-163
56 Pseudocode for Chebyshev Cosine Routine (PIPES2-PIPES0 = 010, RND1-RND0 = 00)		7-169
57 Pseudocode for Chebyshev Sine Routine (PIPES2-PIPES0 = 010, RND1-RND0 = 00)		7-176
58 Pseudocode for Chebyshev Tangent Routine (PIPES2-PIPES0 = 010, RND1-RND0 = 00)		7-184
59 Pseudocode for Chebyshev ArcSine and ArcCosine Routine (PIPES2-PIPES0 = 010, RND1-RND0 = 00)		7-195
60 Pseudocode for Chebyshev ArcTangent Routine (PIPES2-PIPES0 = 010, RND1-RND0 = 00)		7-205
61 Pseudocode for Chebyshev Exponential Routine (PIPES2-PIPES0 = 010, RND1-RND0 = 00)		7-217
62 Data Flow for Pipelined Single-Precision Vector Add, N = 6		7-229
63 Program Listing for Pipelined Single-Precision Vector Add, N = 6		7-230
64 Data Flow for Pipelined Single-Precision Vector Multiply, N = 6		7-230
65 Program Listing for Pipelined Single-Precision Vector Multiply, N = 6		7-231
66 Data Flow for Unpiped Single-Precision Vector Multiply, N = 6		7-231
67 Data Flow for Pipelined Single-Precision Sum of Products, N = 8		7-232

List of Tables (Continued)

<i>Table</i>	<i>Page</i>
68 Program Listing for Pipelined Single-Precision Sum of Products, N = 8	7-233
69 Data Flow for Unpiped Single-Precision Sum of Products, N = 8	7-233
70 Data Flow for 'ACT8837 Pipelined Single-Precision Vector Divide, N = 1	7-235
71 Program Listing for 'ACT8837 Pipelined Single-Precision Vector Divide, N = 1	7-236
72 Data Flow for 'ACT8837 Pipelined Single-Precision Interleaved Vector Divide, N = 2	7-236
73 Data Flow for 'ACT8837 Unpiped Single-Precision Interleaved Vector Divide, N = 1	7-237
74 Data Flow for 'ACT8847 Pipelined Single-Precision Vector Divide	7-238
75 Program Listing for 'ACT8847 Pipelined Single-Precision Vector Divide	7-238
76 Data Flow for Pipelined Single-Precision Vector MAX	7-240
77 Data Flow for Pipelined Single-Precision Interleaved Vector MAX/MIN	7-240
78 Program Listing for Pipelined Single-Precision Interleaved Vector MAX/MIN	7-241
79 Data Flow for Unpiped Single-Precision Vector MAX	7-241
80 Data Flow for Pipelined Single-Precision List MAX	7-242
81 Program Listing for Pipelined Single-Precision List MAX	7-243
82 Partial Data Flow for Product of [X, Y, Z, W] and General Transform Matrix	7-248
83 Partial Data Flow for Product of [X, Y, Z, W] and Reduced Transform Matrix	7-249
84 Data Flow for Clipping a Line Segment Against the Z = N Plane	7-253
85 Program Listing for Clipping a Line Segment Against the Z = N Plane	7-254
86 Data Flow for Clipping a Line Segment Against the Z = N Plane	7-255

List of Tables (Concluded)

<i>Table</i>		<i>Page</i>
87	Data Flow for Computing t ₁ , t ₂ , s ₁ , and s ₂ Using an SN74ACT8837	7-257
88	Program Listing for Three-Processor Clip to Compute t ₁ , t ₂ , s ₁ , and s ₂	7-258
89	A > B Comparison Function Table	7-259
90	Data Flow for Accept/Reject Testing	7-260
91	Data Flow for the X Processor	7-262
92	Program Listing for the X Processor	7-262
93	Summary of Graphics Systems Performance	7-263
94	Available Options for Graphic System Designs	7-263

7

SN74ACT8847

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

Boston

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

Northern California

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

Chicago

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

Dallas

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

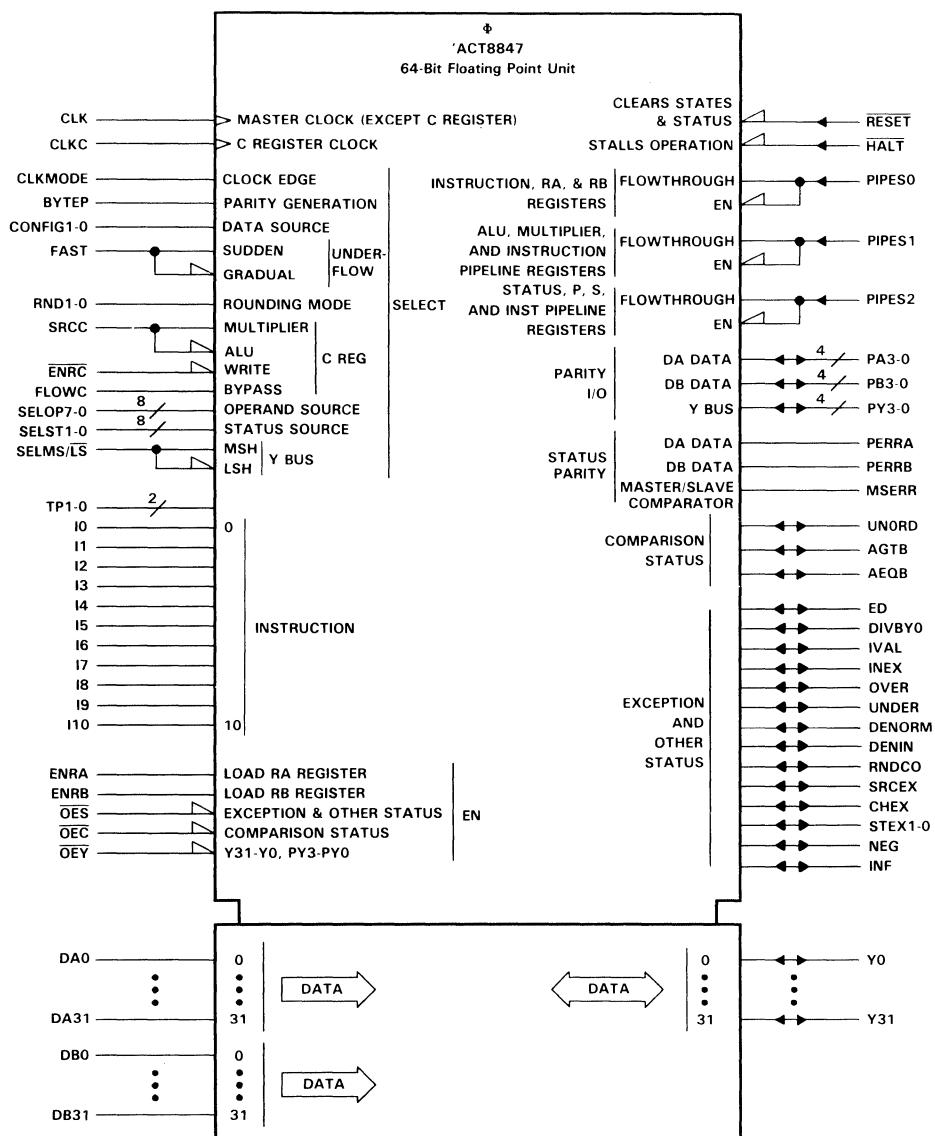
Southern California

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)**

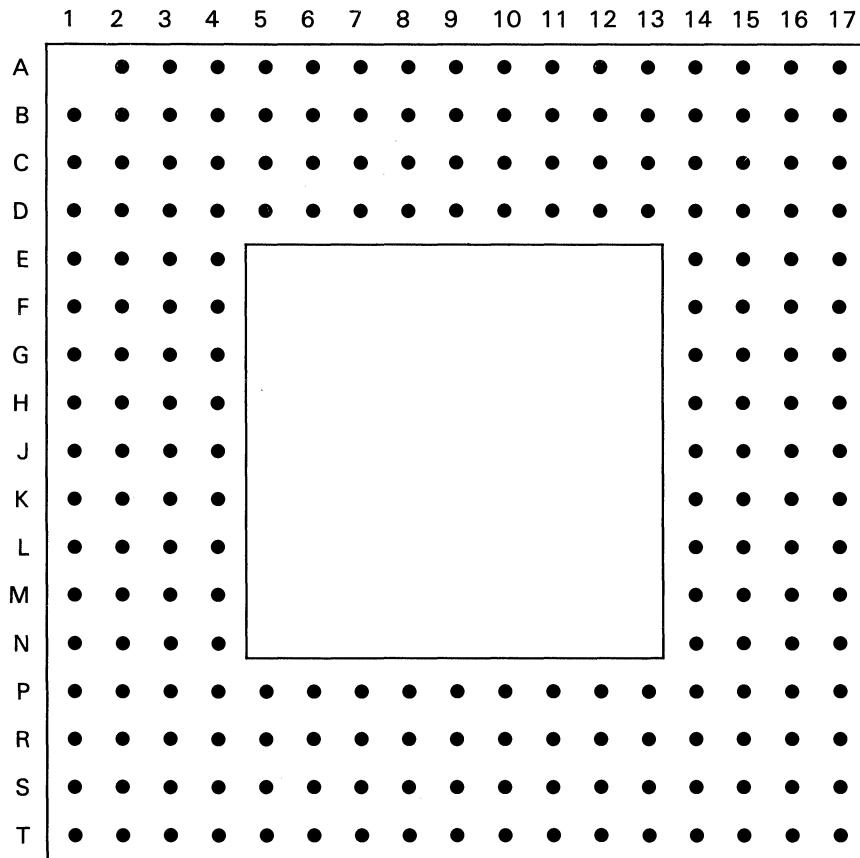


Table 1. 'ACT8847 Pin Grid Allocation

PIN NO.	PIN NAME	PIN NO.	PIN NAME	PIN NO.	PIN NAME	PIN NO.	PIN NAME
A1 missing	C2 Y0	E3 FAST	J15 FLOWC	P1 ENRC	S1 NC		
A2 INF	C3 Y3	E4 GND	J16 SRCC	P2 PIPES0	S2 PBO		
A3 Y5	C4 Y6	E14 GND	J17 BYTEP	P3 RESET	S3 DBO		
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 OEY	L2 SELOP7	P13 DA8	S13 DA6		
A14 PY3	C15 DENIN	F17 OES	L3 CLK	P14 DA12	S14 DA10		
A15 IVAL	C16 SRCEX	G1 I7	L4 VCC	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 VCC	L16 DA31	R1 PIPES1	T1 NC		
B2 Y2	D3 Y1	G14 VCC	L17 PA0	R2 HALT	T2 PB3		
B3 Y4	D4 GND	G15 OEC	M1 ENRB	R3 PB2	T3 DB3		
B4 Y7	D5 VCC	G16 SELMS/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 VCC	H2 I9	M14 VCC	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 VCC	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 VCC	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 SELOPO	N15 DA24	R16 DA20	T16 NC		
B17 DIVBY0	E1 I2	J4 VCC	N16 DA25	R17 DA21	T17 NC		
C1 PERRB	E2 IO	J14 VCC	N17 DA26				

Table 2. 'ACT8847 Pin Functional Description

PIN NAME	PIN NO.	I/O/Z [†]	DESCRIPTION
DATA BUS SIGNALS (96 PINS)			
DA0	P11		
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	I	DA 32-bit input data bus. Data can be latched in a 64-bit temporary register or loaded directly into an input register
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		
DB1	P5		
DB2	R4		
DB3	T3		
DB4	S4		
DB5	P6	I	DB 32-bit input data bus. Data can be latched in a 64-bit temporary register or loaded directly into an input register.
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	NO.	I/O/Z [†]	DESCRIPTION
DATA BUS SIGNALS (96 PINS)			
DB11	S5		
DB12	S6		
DB13	T6		
DB14	R7		
DB15	S7		
DB16	P8		
DB17	T7		
DB18	R8		
DB19	S8		
DB20	T8		
DB21	P9	I	DB32-bit input data bus. Data can be latched in a 64-bit temporary register or loaded directly into an input register.
DB22	R9		
DB23	S9		
DB24	T9		
DB25	T10		
DB26	S10		
DB27	R10		
DB28	P10		
DB29	T11		
DB30	S11		
DB31	R11		
Y0	C2		
Y1	D3		
Y2	B2		
Y3	C3		
Y4	B3		
Y5	A3		
Y6	C4		
Y7	B4		
Y8	A4		
Y9	C5		
Y10	B5	I/O/Z	32-bit Y output data bus
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	NO.	I/O/Z [†]	DESCRIPTION
DATA BUS SIGNALS (96 PINS)			
Y21	A9		
Y22	B9		
Y23	C9		
Y24	A10		
Y25	B10		
Y26	C10	I/O	32-bit Y output data bus
Y27	A11		
Y28	B11		
Y29	A12		
Y30	C11		
Y31	B12		
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	L17		
PA1	K15		
PA2	K16		
PA3	K17		
PB0	S2		
PB1	P4		
PB2	R3		
PB3	T2		
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	A13		
PY1	C12		
PY2	B13		
PY3	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	N1	I	Select data sources for RA and RB registers from DA bus, DB bus and temporary register
CONFIG1	N2	I	
ENRA	M2	I	When high, enables loading of RA register on a rising clock edge if the RA register is not disabled (see PIPES0 below).
ENRB	M1	I	When high, enables loading of RB register on a rising clock edge if the RB register is not disabled (see PIPES0 below).
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).
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	E2		
I1	D1		
I2	E1		
I3	F2		
I4	G3		
I5	F1	I	Instruction inputs
I6	G2		
I7	G1		
I8	H3		
I9	H2		
I10	H1		
OE _C	G15	I	Comparison status output enable. Active low.
OE _S	F17	I	Exception status and other status output enable. Active low.
OE _Y	F16	I	Y bus output enable. Active low.
PIPES0	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	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.
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/L ^S	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	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 than 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)			
VCC	D5		
VCC	D8		
VCC	D11		
VCC	D14		
VCC	G4		
VCC	G14	I	5-V supply voltage pins
VCC	J4		
VCC	J14		
VCC	L4		
VCC	M14		
GND	D4		
GND	D6		
GND	D7		
GND	D9		
GND	D10		
GND	D12		
GND	D13		
GND	E4		
GND	E14	I	Ground pins. NOTE: All ground pins should be used and connected.
GND	F4		
GND	F14		
GND	H4		
GND	H14		
GND	K4		
GND	K14		
GND	L14		
GND	M4		
NC	A17		
NC	S1		
NC	T1		
NC	T16		
NC	T17		No internal connection. Pins should be left floating.

[†]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 _{CC}	V
V _{IL} Low-level input voltage	0	0.8	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 _{CC}	V
V _O Output voltage	0	V _{CC}	V _{CC}	V
dt/dv Input transition rise or fall rate	0	15	15	ns/V
T _A Operating free-air temperature	0	70	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		UNIT
					MIN	MAX	
1	t_{pd1}	DA/DB/Inst	Y OUTPUT	111		\dagger	ns
2	t_{pd2}	INPUT REG	Y OUTPUT	110		70	ns
		INPUT REG	STATUS	110		70	ns
3	t_{pd3}	PIPELN REG	Y OUTPUT	10X		48	ns
		PIPELN REG	STATUS	10X		48	ns
4	t_{pd4}	OUTPUT REG	Y OUTPUT	0XX		20	ns
		OUTPUT REG	STATUS	0XX		20	ns
5	t_{pd5}	SELMS/ \overline{LS}	Y OUTPUT	XXX		18	ns
6	t_{pd6}	CLK \uparrow	Y OUTPUT INVALID	all but 111	3.0		ns
7	t_{pd7}	CLK \uparrow	STATUS INVALID	all but 111	3.0		ns
8	t_{pd8}	SELMS/ \overline{LS}	Y OUTPUT INVALID	XXX	1.5		ns
9	t_{d1}^{\ddagger}	CLK \uparrow	CLK \uparrow	010	56		ns
10	t_{d2}^{\ddagger}	CLK \uparrow	CLK \uparrow	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	$t_d - 0^{\$}$	ns
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		UNIT
			MIN	MAX	
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}	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}	RESET after CLK↑	XX0	6	

CLK/RESET requirements

	PARAMETER	SN74ACT8847-30		UNIT
		MIN	MAX	
t_w	Pulse duration	CLK high	10	ns
		CLK low	10	
		RESET	10	

switching characteristics

NO.	PARAMETER	FROM (INPUT)	TO (OUTPUT)	PIPELINE CONTROLS PIPES2-PIPES0	SN74ACT8847-40		UNIT
					MIN	MAX	
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}^{\ddagger}	CLK↑	CLK↑	010	72		ns
10	t_{d2}^{\ddagger}	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	$t_d - 0^{\$}$	
12	t_{en1}	OEY	Y OUTPUT	XXX		16	ns
13	t_{en2}	OEC, \overline{OES}	STATUS	XXX		16	
14	t_{dis1}	OEY	Y OUTPUT	XXX		16	
15	t_{dis2}	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	SN74ACT8847-40		UNIT
			Pipes2-Pipes0	MIN	MAX	
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}	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}	RESET after CLK↑	XX0	6		

CLK/RESET requirements

PARAMETER		SN74ACT8847-40		UNIT
		MIN	MAX	
t_w	Pulse duration	CLK high	15	ns
		CLK low	15	
		RESET	12	

switching characteristics

NO.	PARAMETER	FROM (INPUT)	TO (OUTPUT)	PIPELINE CONTROLS PIPES2-PIPES0	SN74ACT8847-50		UNIT
					MIN	MAX	
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/L \bar{S}	Y OUTPUT	XXX		24	ns
6	t_{pd6}	CLK \uparrow	Y OUTPUT INVALID	all but 111	3.0		ns
7	t_{pd7}	CLK \uparrow	STATUS INVALID	all but 111	3.0		ns
8	t_{pd8}	SELMS/L \bar{S}	Y OUTPUT INVALID	XXX	1.5		ns
9	t_{d1}^{\ddagger}	CLK \uparrow	CLK \uparrow	010	100		ns
10	t_{d2}^{\ddagger}	CLK \uparrow	CLK \uparrow	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

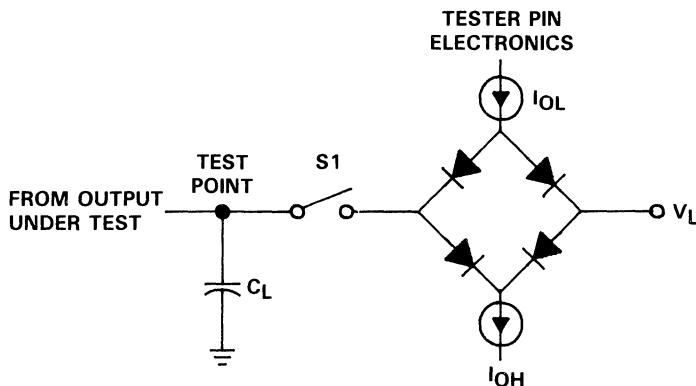
NO.	PARAMETER	PIPELINE CONTROLS PIPES2-PIPES0	SN74ACT8847-50		UNIT
			MIN	MAX	
16	t_{su1}	Inst/control before CLK↑	XX0	16	ns
17	t_{su2}	DA/DB before CLK↑	XX0	16	
18	t_{su3}	DA/DB before 2nd CLK↑ (DP)	XX1	75	
19	t_{su4}	CONFIG1-0 before CLK↑	XX0	18	
20	t_{su5}	SRCC before CLKC↑	XXX	16	
21	t_{su6}	RESET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

CLK/RESET requirements

	PARAMETER	SN74ACT8847-50		UNIT
		MIN	MAX	
t_w	Pulse duration	CLK high	15	ns
		CLK low	15	
		RESET	15	

'ACT8847 Load Circuit

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



TIMING PARAMETER		C_L^{\dagger}	I_{OL}	I_{OH}	V_L	S_1
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

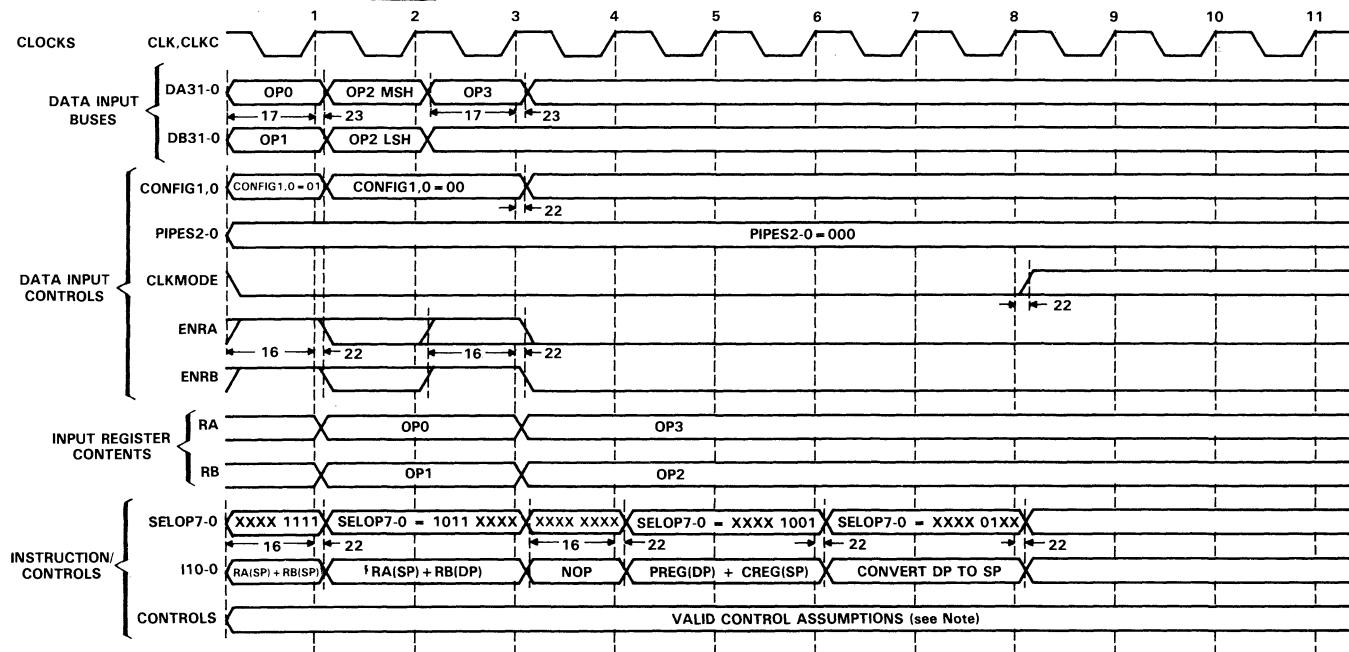
$\dagger C_L$ includes probe and test fixture capacitance.

NOTE: All input pulses are supplied by generators having the following characteristics:
 $PRR \leq 1 \text{ MHz}$, $Z_O = 50 \Omega$, $t_r \leq 6 \text{ ns}$, $t_f \leq 6 \text{ ns}$.

Figure 1. Load Circuit

SN74ACT8847

7444



NOTES: Assume the following mixed precision operation.

Single precision $OP0 + OP1 = RA + RB \rightarrow SUM1 \rightarrow CREG$, where OP0 is SP and OP1 is SP.

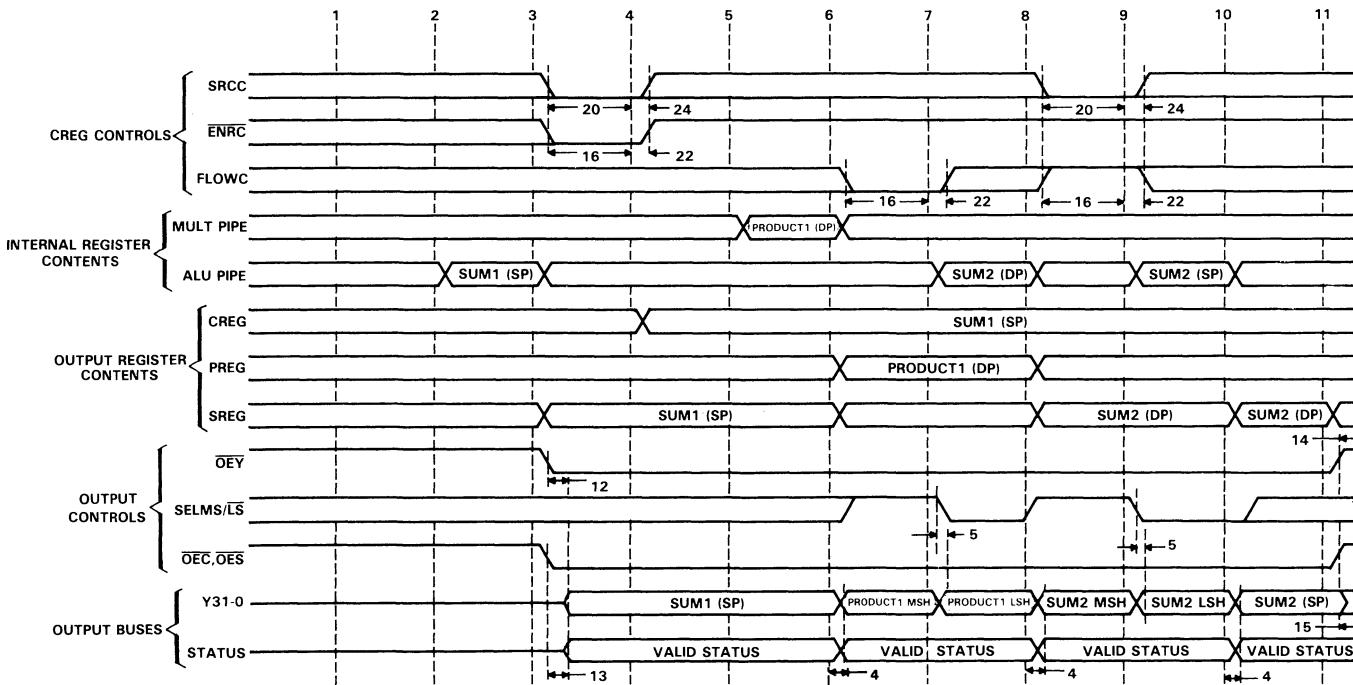
Mixed precision $OP2 * OP2 = RA * RB \rightarrow PRODUCT1$, where OP3 is SP and OP2 is DP.

NOP (must be inserted).

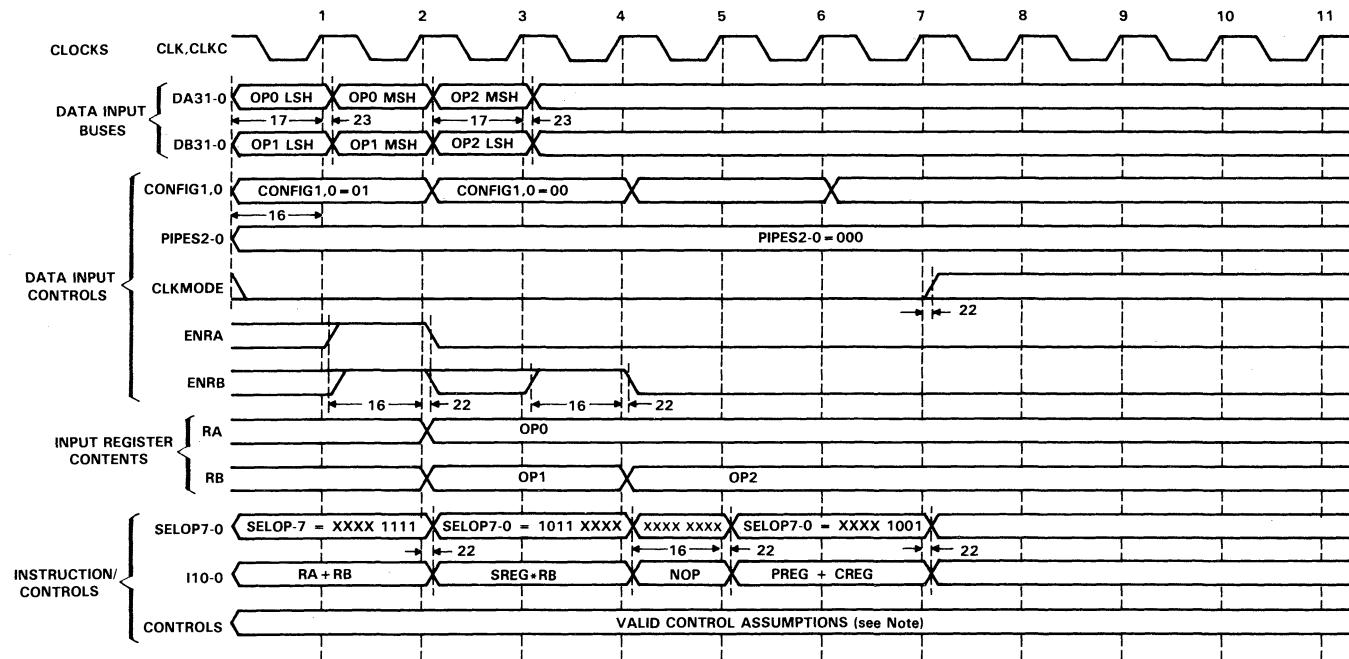
Mixed precision $(OP3 * OP2) + (OP0 + OP1) = PREG + CREG \rightarrow SUM2 (DP)$, and then convert to SP.

Assume valid control signals for FAST, HALT = 1, PIPES2,0 = 000 (fully pipelined mode), 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



SN74ACT8847



NOTES: Assume the following double precision operation.

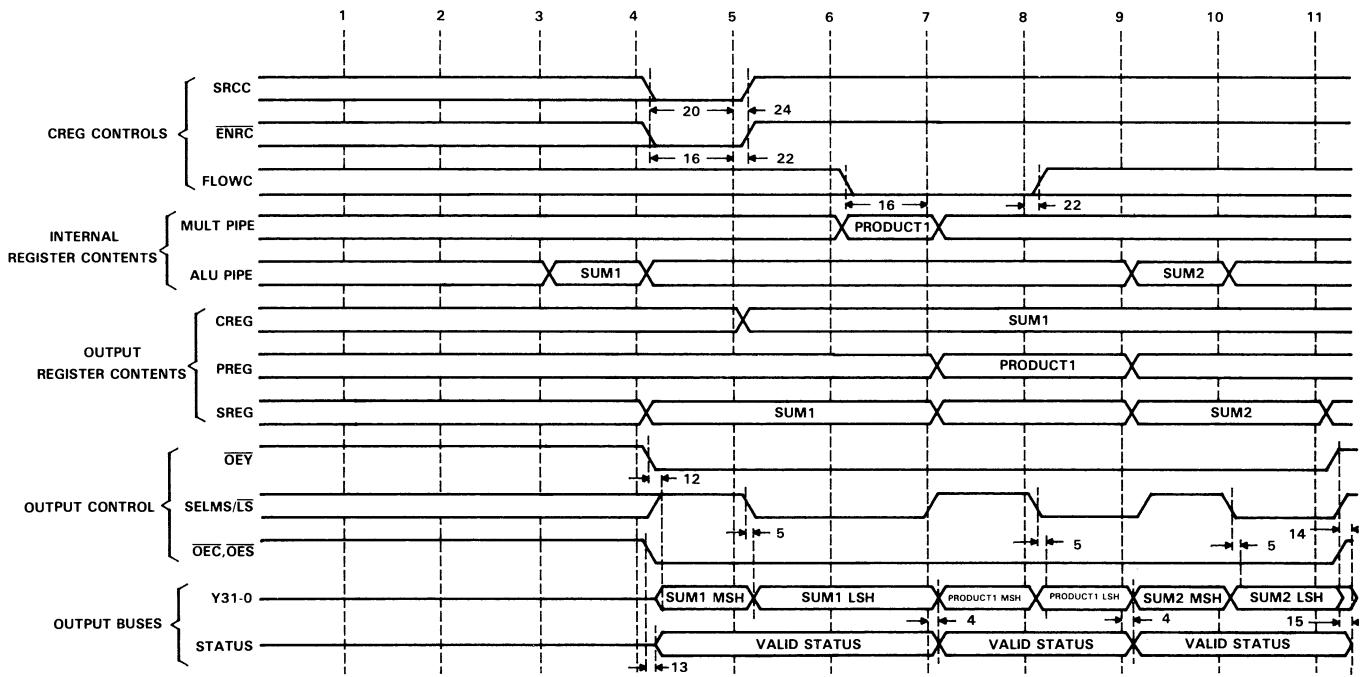
$$\text{OPO} + \text{OP1} = \text{RA} + \text{RB} \rightarrow \text{SUM1} \rightarrow \text{CREG}$$

$$(\text{OPO} + \text{OP1}) * \text{OP2} = \text{SREG} * \text{RB} \rightarrow \text{PRODUCT1}$$

$$[(\text{OPO} + \text{OP1}) * \text{OP2}] + (\text{OPO} + \text{OP1}) = \text{PREG} + \text{CREG} \rightarrow \text{SUM2}$$

Assume valid control signals for FAST, HALT=1, PIPES2,0=000 (fully pipelined mode), 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.

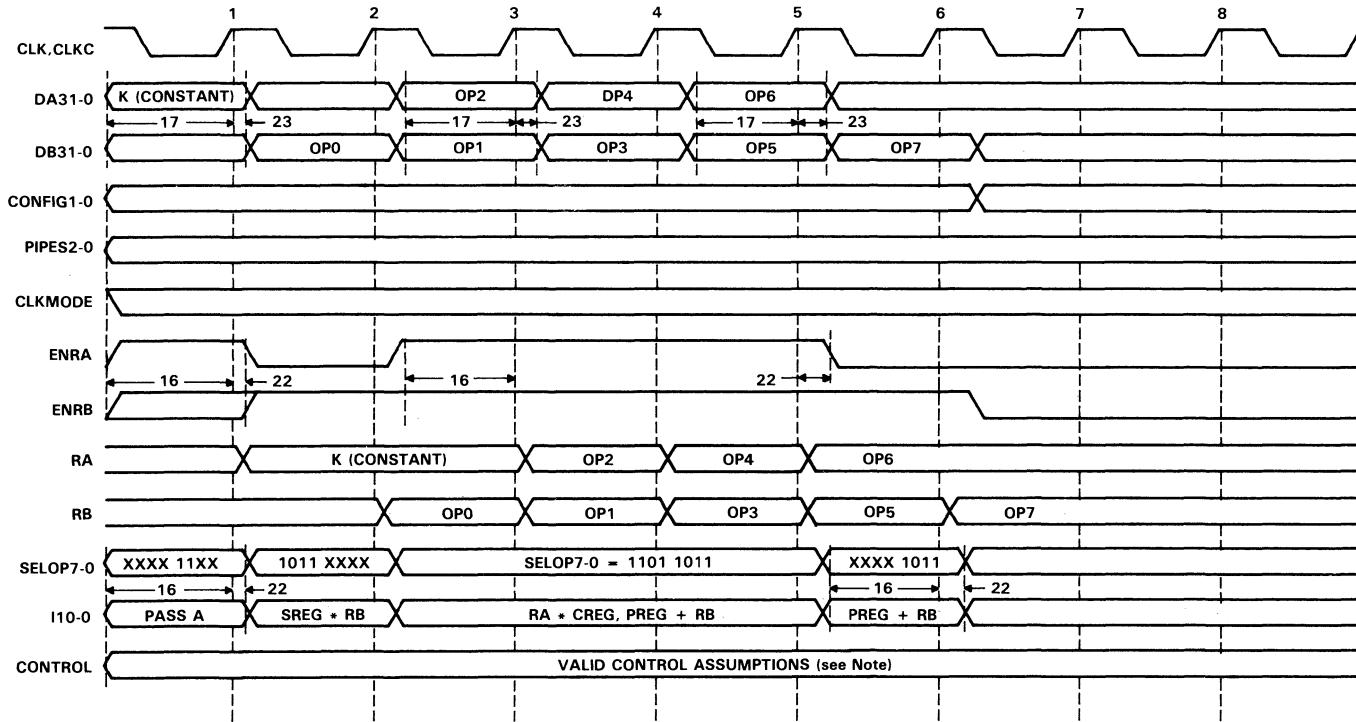
$$OPO + OP1 = RA + RB \rightarrow SUM1 \rightarrow CREG$$

$$(OPO + OP1) * OP2 = SREG * RB \rightarrow PRODUCT1$$

$$[(OPO + OP1) * OP2] + (OPO + OP1) = PREG + CREG \rightarrow SUM2$$

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

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

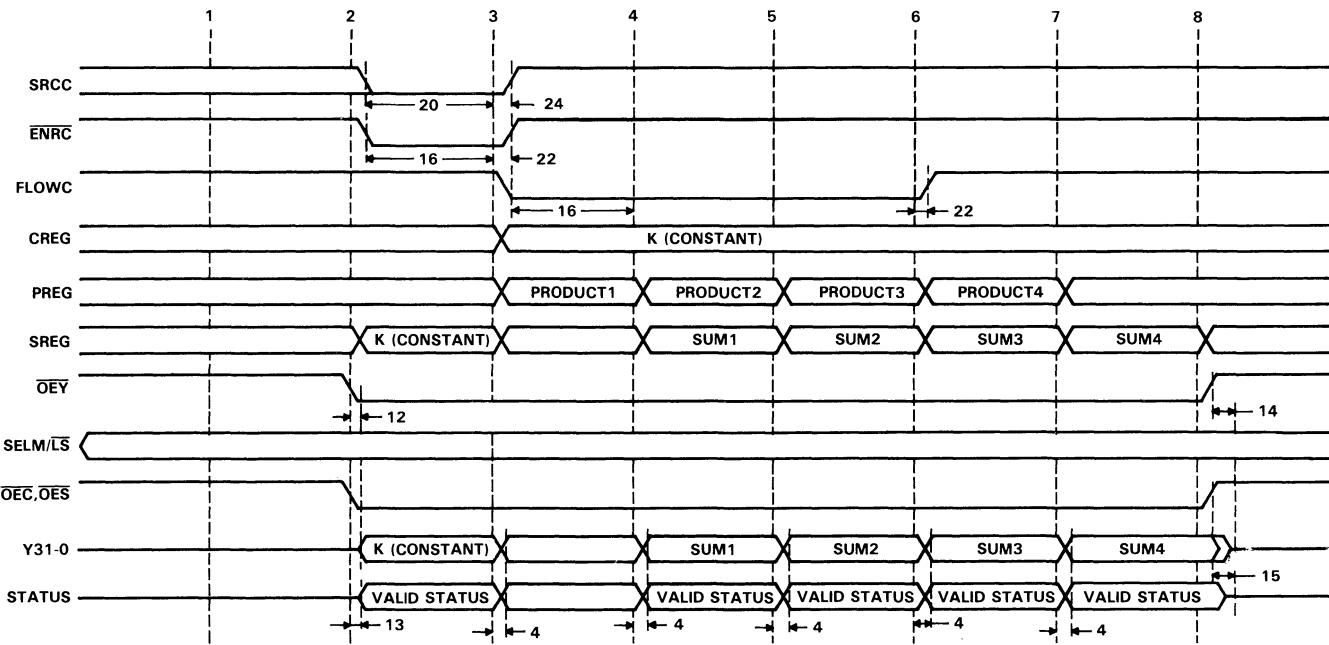


NOTES: Assume the following single precision operations.

$$\begin{aligned}
 & (K * OP0) + OP1 = PRODUCT1 + OP1 \rightarrow SUM1 \\
 & (K * OP2) + OP3 = PRODUCT2 + OP3 \rightarrow SUM2 \\
 & (K * OP4) + OP5 = PRODUCT3 + OP5 \rightarrow SUM3 \\
 & (K * OP6) + OP7 = PRODUCT4 + OP7 \rightarrow SUM4
 \end{aligned}$$

Assume valid control signals for FAST, HALT = 1, PIPES2-0 = 010, 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.

$$(K * OP_0) + OP_1 = \text{PRODUCT}_1 + OP_1 \rightarrow \text{SUM}_1$$

$$(K * OP_2) + OP_3 = \text{PRODUCT}_2 + OP_3 \rightarrow \text{SUM}_2$$

$$(K * OP_4) + OP_5 = \text{PRODUCT}_3 + OP_5 \rightarrow \text{SUM}_3$$

$$(K * OP_6) + OP_7 = \text{PRODUCT}_4 + OP_7 \rightarrow \text{SUM}_4$$

Assume valid control signals for **FAST**, **HALT** = 1, **PIPES2-0** = 010, **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.

7

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.

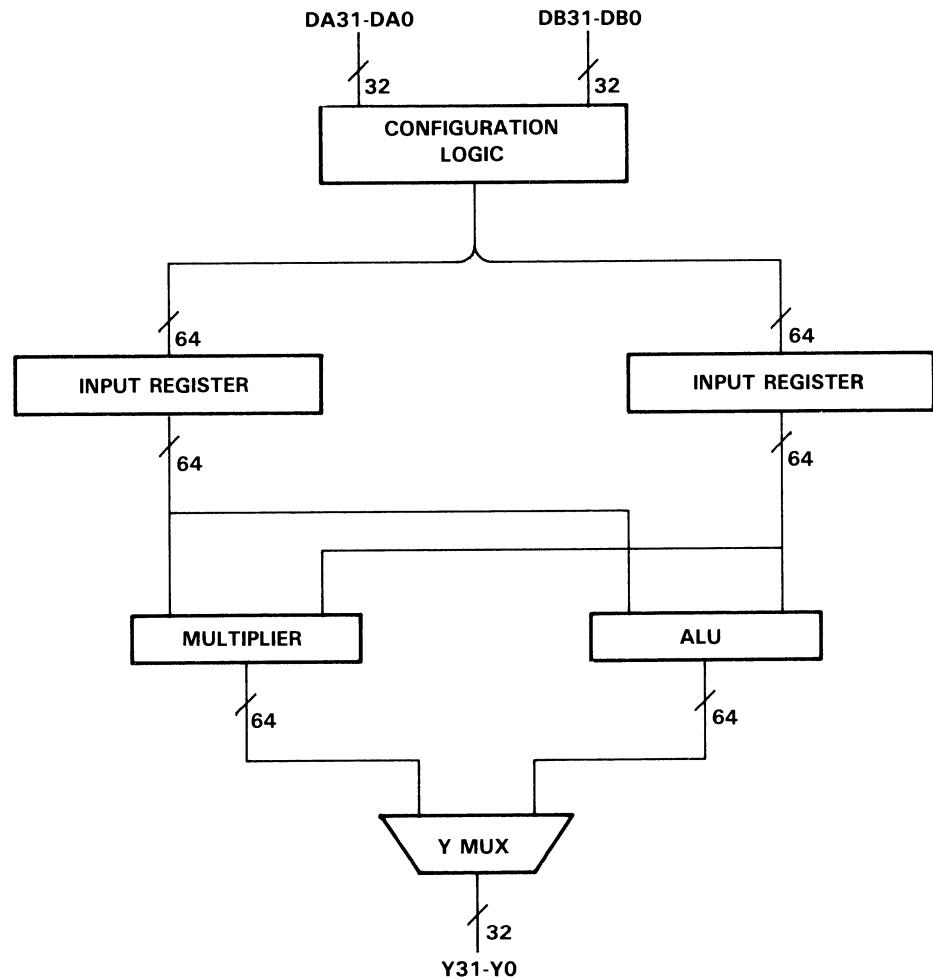


Figure 5. High Level Block Diagram

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.

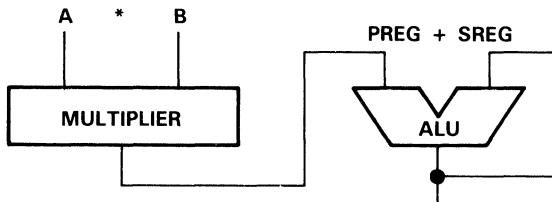


Figure 6. Multiply/Accumulate Operation

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.

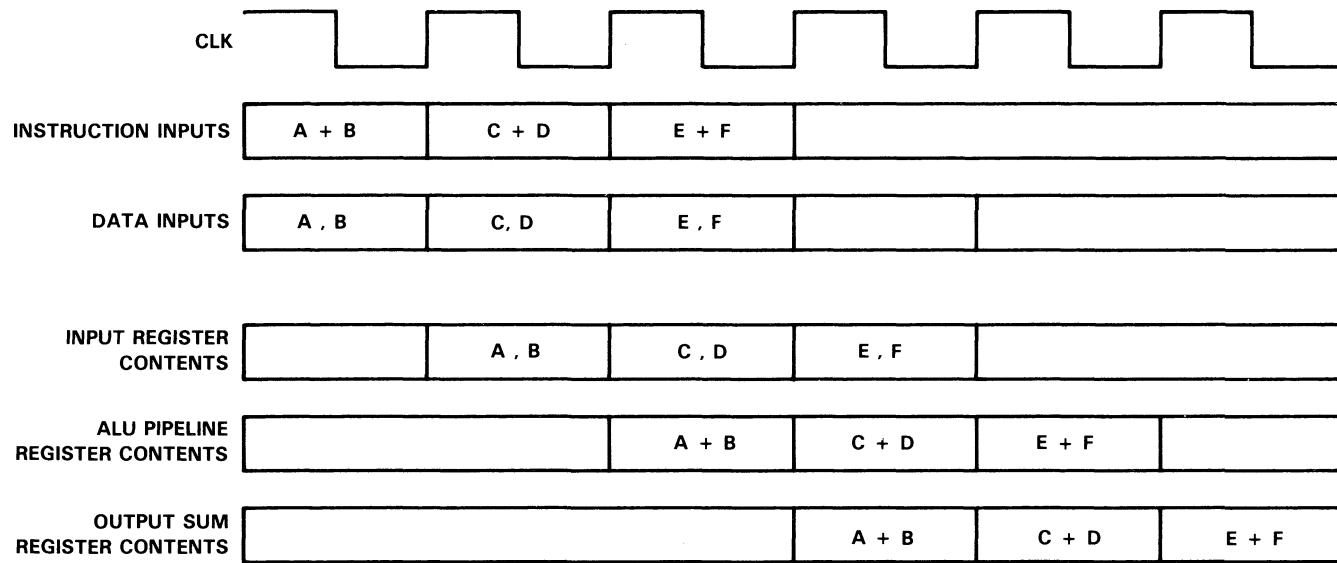


Figure 7. Example of Fully Pipelined Operation

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).

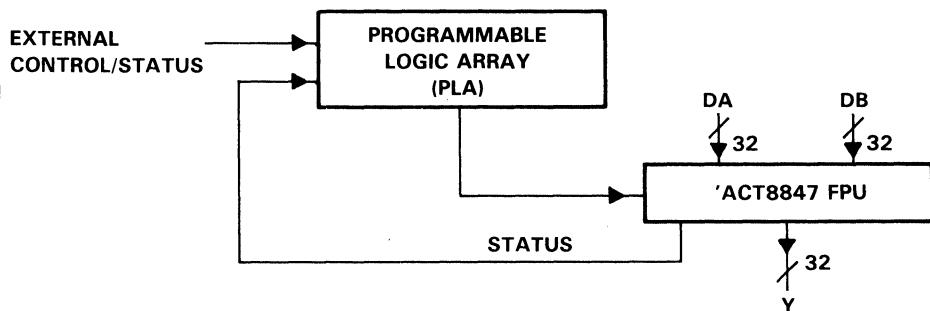


Figure 8. PLA Control Circuit Example

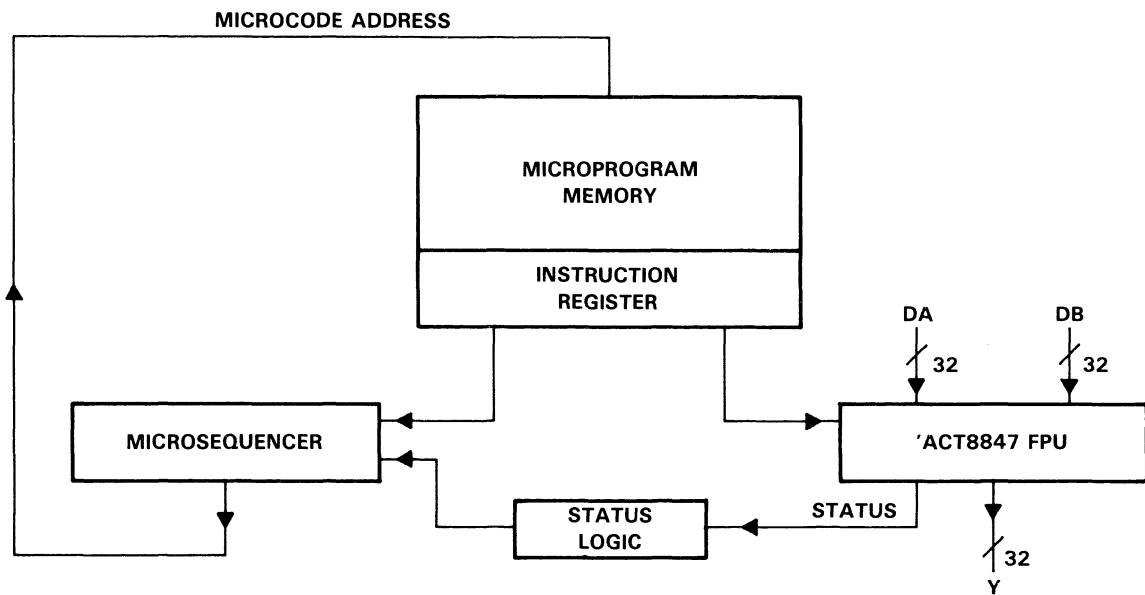


Figure 9. Microprogrammed Architecture

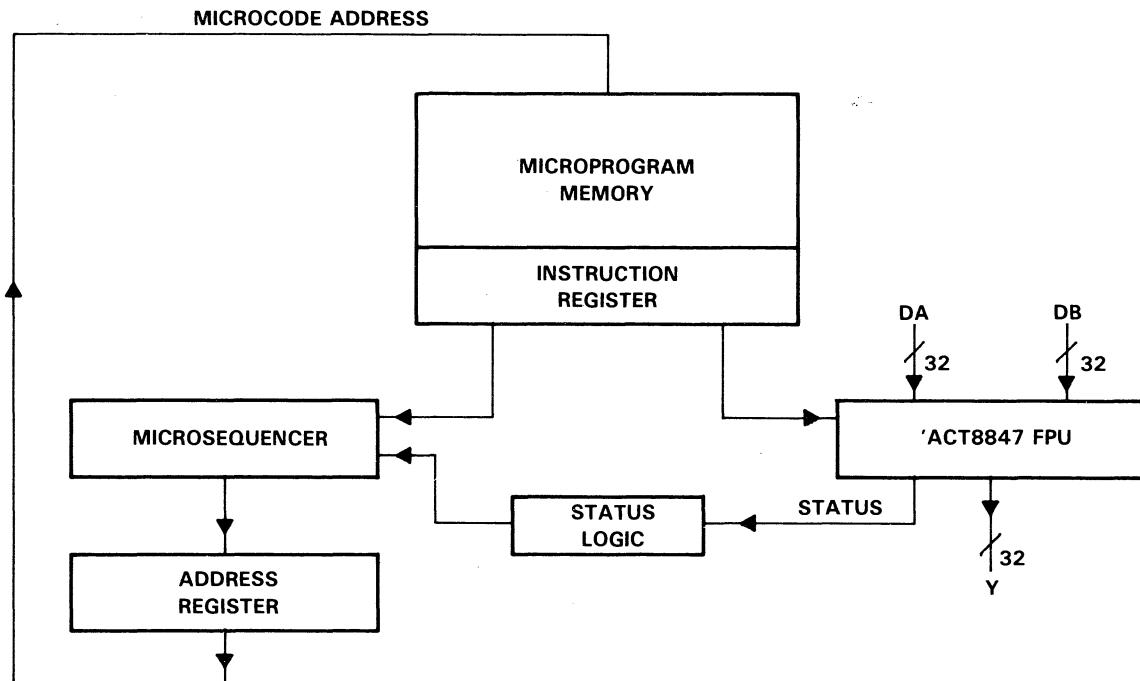


Figure 10. Microprogrammed Architecture with Address Register

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.

Micropogram Control of an 'ACT8847 FPU Subsystem

A micropogram 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 micropogram 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:

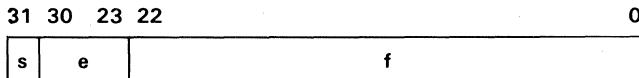
1.fraction

The representation of a normalized floating point number is:

$$(-1)^S \cdot 1.f \cdot 2^{(e-\text{bias})}$$

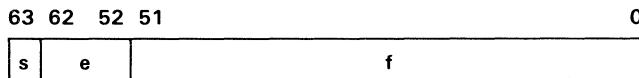
where the bias is either 127 for single-precision operands or 1023 for double-precision operands.

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:

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

7

SN74ACT8847

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)		FRACTION (f) (BINARY)	HIDDEN BIT	VALUE OF NUMBER REPRESENTED	
	SP (HEX)	DP (HEX)			SP (DECIMAL)†	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	$(-1)^s (\text{infinity})$	$(-1)^s (\text{infinity})$
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- μ m 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).

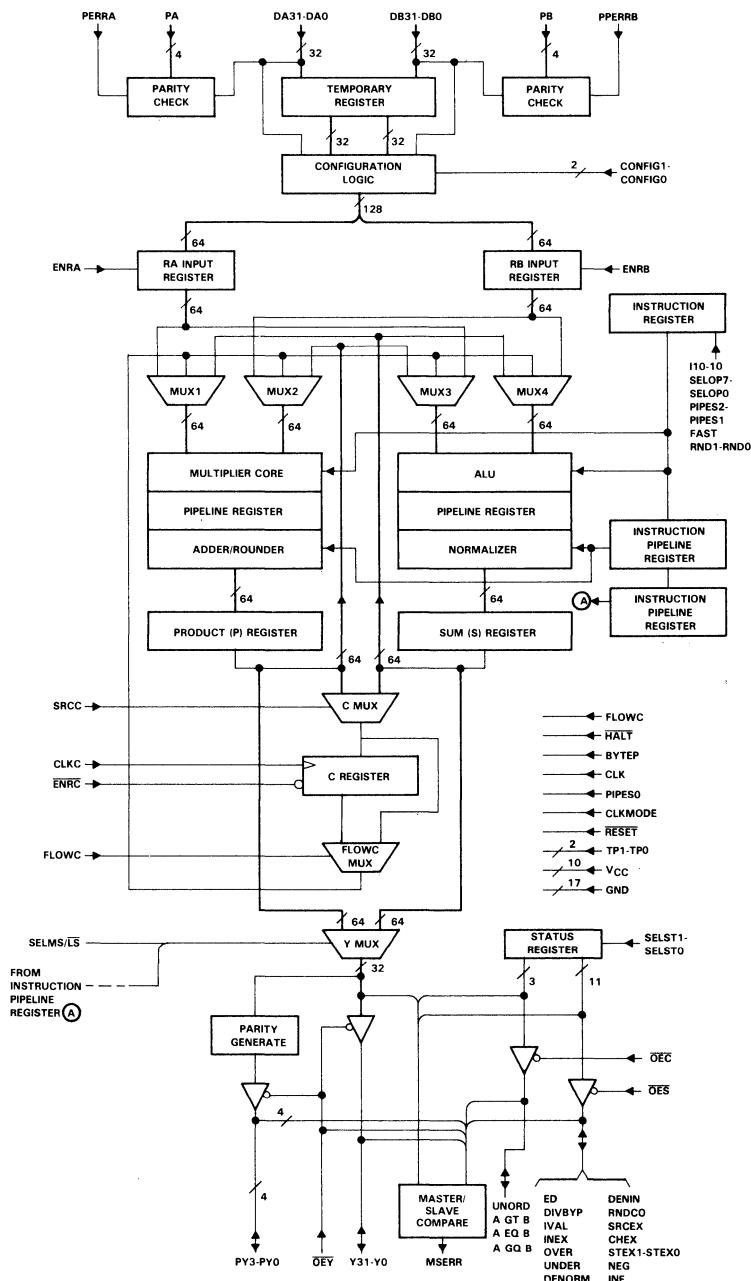


Figure 13. 'ACT8847 Detailed Block Diagram'

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.

7

SN74ACT8847

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.

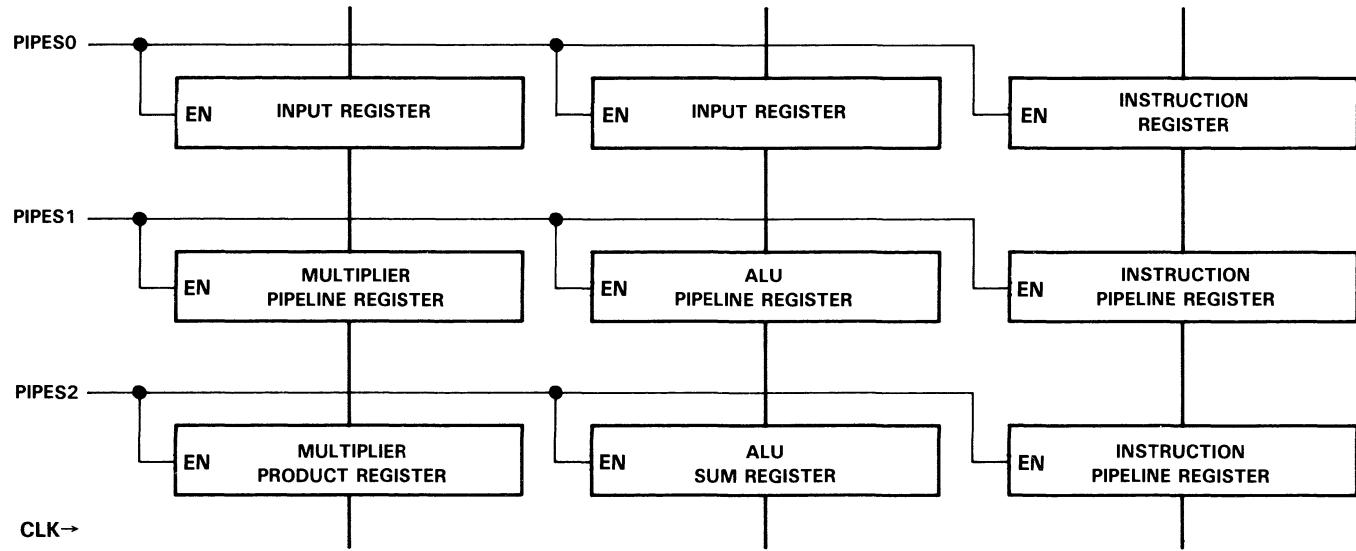


Figure 14. Pipeline Controls

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.

S7N74ACT8847

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. PIPESO 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.

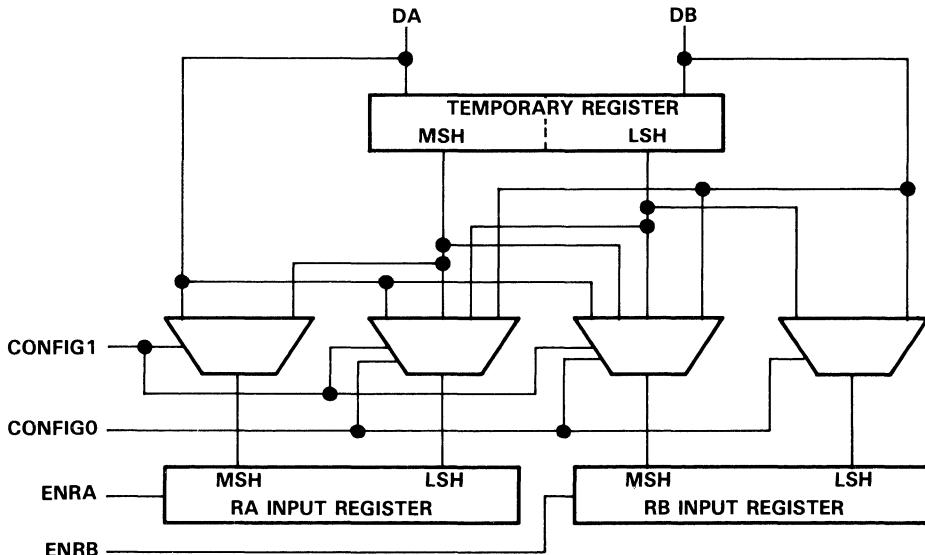


Figure 15. Input Register Control

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		NOTE
		CONFIG1	CONFIG0	
DA	DB	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-SELOPO 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.

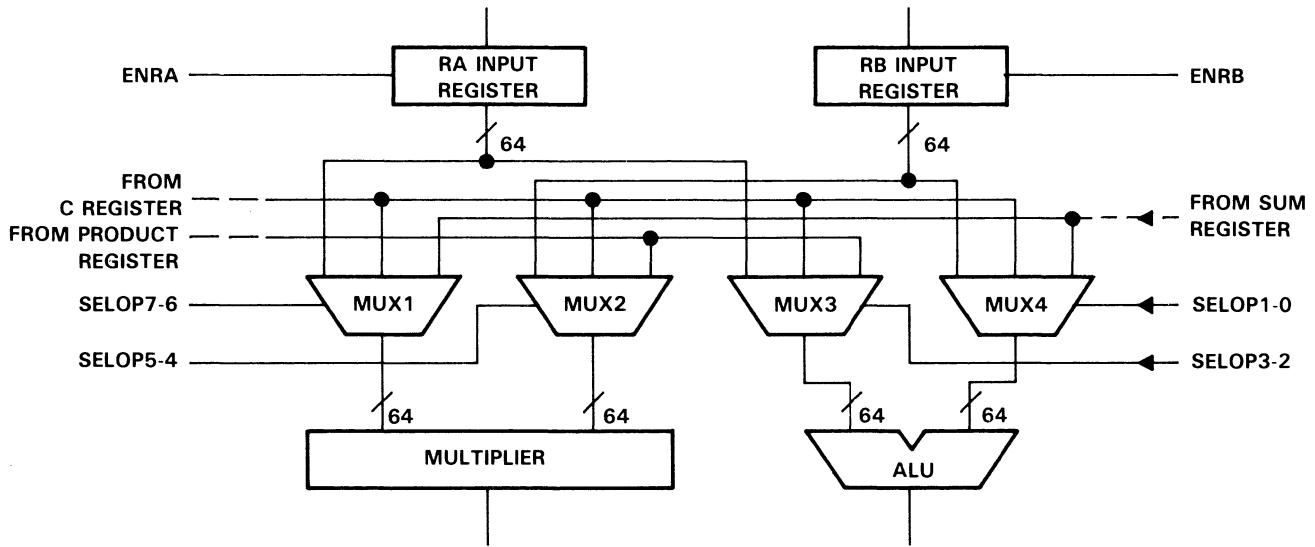


Figure 16. Operand Selection Multiplexer

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-SELOPO = 11111111).

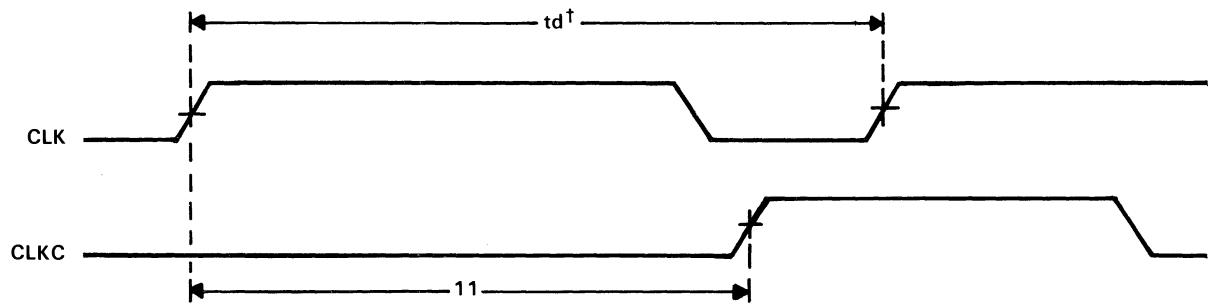
Selection of operands also interacts with the selected operation in the ALU or the multiplier. ALU operations with one operand are performed only on the A operand (with the exception of the Pass B operation). Also, depending on the instruction selected, the B operand may optionally be forced to zero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

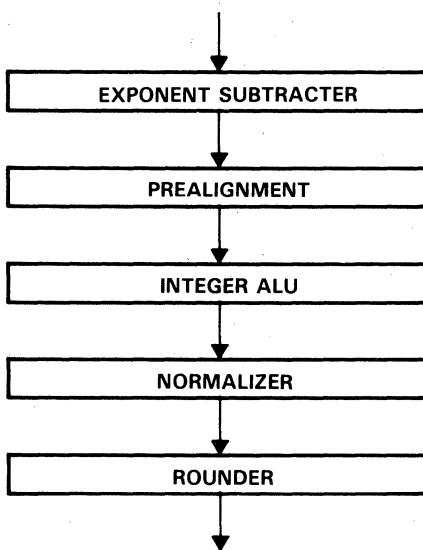


Figure 18. Functional Diagram for ALU

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.

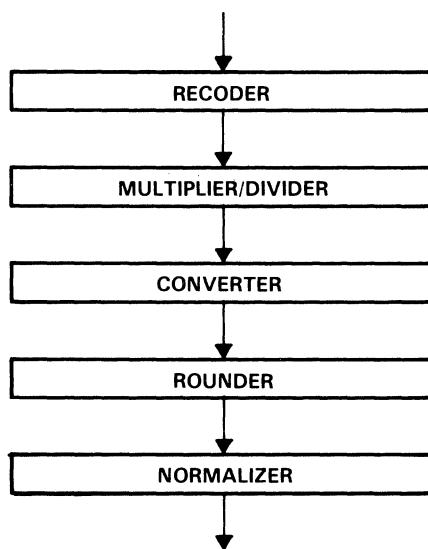


Figure 19. Functional Diagram for Multiplier

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/LS signal in the middle of the clock period.

When a single-precision result is output, the SELMS/LS signal has no effect. The SELMS/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 OEY must be low. OEY is an asynchronous signal.

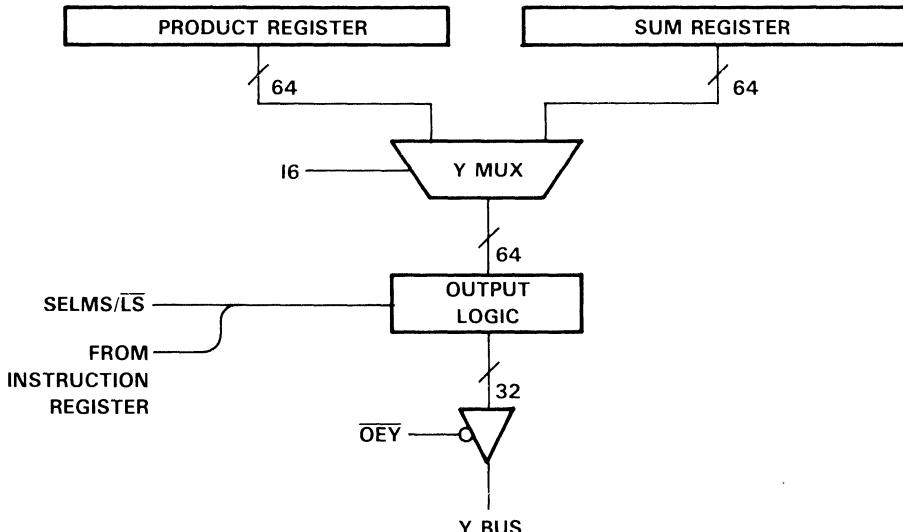


Figure 20. Y Output Control

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.

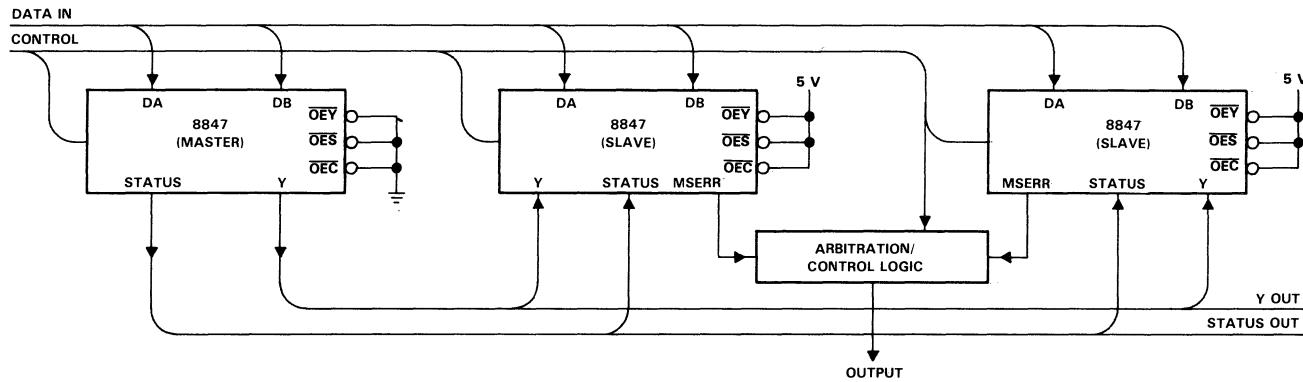


Figure 21. Example of Master/Slave Operation

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 ($\text{PIPES2} = 0$), 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, $\overline{\text{OEC}}$ for comparison status and $\overline{\text{OES}}$ 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 operation, i.e., PASS (-A) with unsigned integer operand.
DIVBYO	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.

7

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.

SN74ACT8847

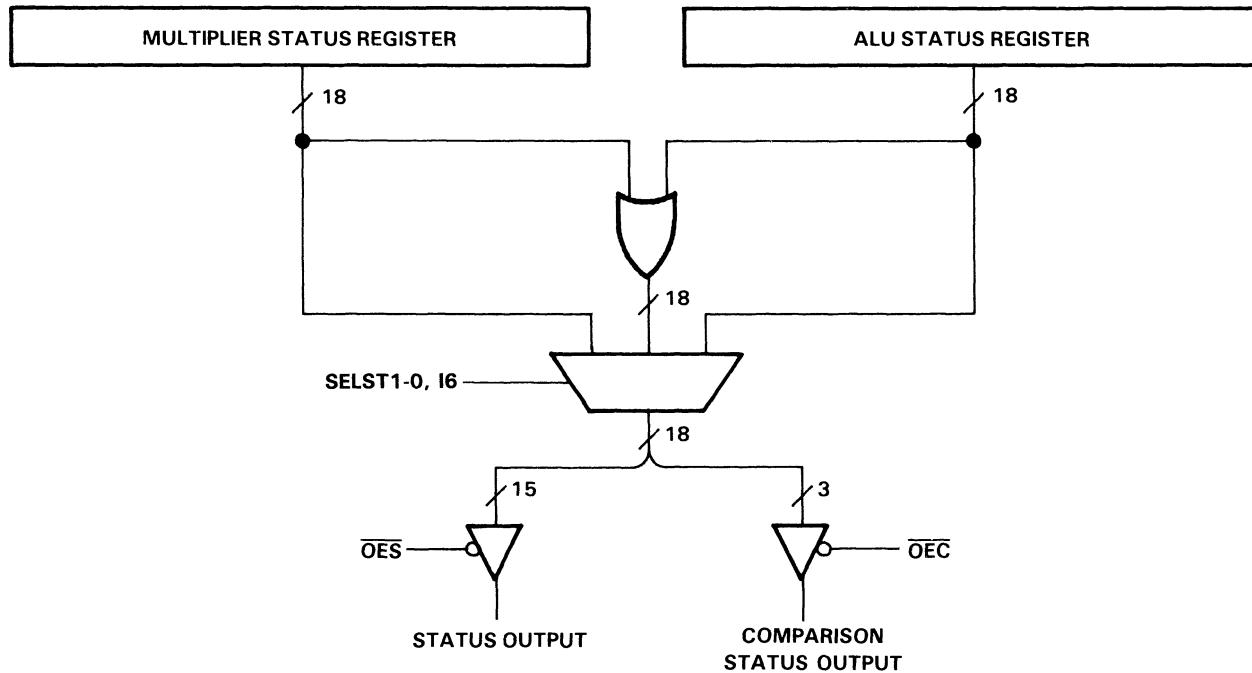


Figure 22. Status Output Control

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.

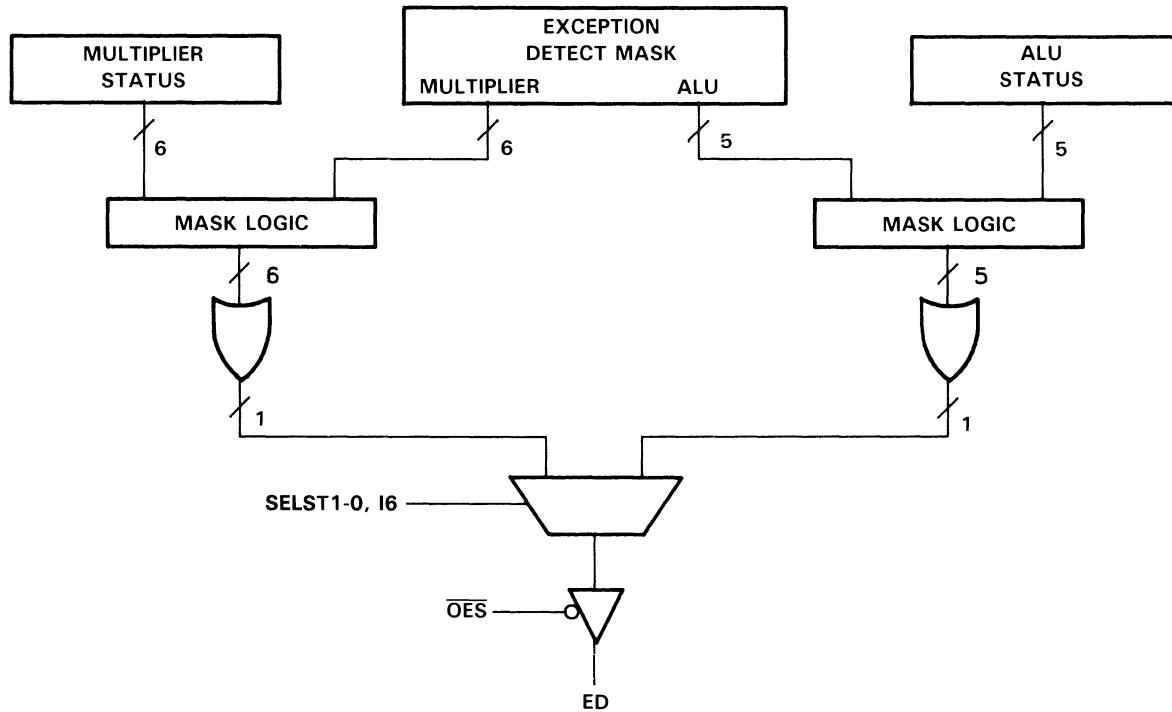


Figure 23. Exception Detect Mask Logic



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
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
HALT	No effect	Stalls device operation but does not affect registers, internal states, or status. C register loading is not disabled	7-85
OEC	Disables compare pins	Enables compare pins	7-77
OES	Disables status outputs	Enables status outputs	7-77
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
RESET	No effect	Clears internal states, status, internal pipeline registers, and exception disable register. Does not affect other data registers.	7-86
RND1-RNDO	See Table 18 (Rounding Mode Control)	See Table 18 (Rounding Mode Control)	7-84
SELOP7-SELOPO	See Tables 10 and 11 (Multiplier/ALU operand selection)	See Tables 10 and 11 (Multiplier/ALU operand selection)	7-68
SELMS/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-TPO	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 HALT signal. Bringing the 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 HALT signal is set high.

Stalling the device does not stall the C register. If 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 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 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 HALT signal is set high again and normal operation resumes.

RESET

The 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 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
O = 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 I10	OPERAND FORMAT/PRECISION			OUTPUT SOURCE I6	OPERAND TYPE I5	ALU OPERATION	
	I9	I8	I7			I4-10	RESULT
0 = Not Chained	1 = Integer	0 1	0 = SP 2's complement 1 = SP unsigned integer	0 = ALU result	1 = Single Operands	00000 00001 00010 00101 01000 01001 01101	Pass A operand Pass (-A) operand Negate A operand (1's complement) Pass B operand Shift A operand left logical [†] Shift A operand right logical [†] 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	A + B
									01	A - B
									10	Compare A, B
									11	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 (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

7

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
011000000000	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 I10	OPERAND FORMAT/PRECISION			OUTPUT SOURCE I6	MULTIPLY/ DIVIDE I5	ABSOLUTE VALUE A I4 [†]	ABSOLUTE VALUE B/ DIV/SQRT I3 [†]	NEGATE RESULT I2 [†]	WRAP A I1 [†]	WRAP B I0 [†]
	I9	I8	I7 [‡]		I5	I4 [†]	I3 [†]			
0 = Not chained	0 = floating point	0 = A(SP) 1 = A(DP)	0 = B(SP) 1 = B(DP)	1 = Multi- plier 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	0 = SP 2's complement 1 = SP unsigned integer		1 = Div/SQRT	0 = A 1 = A	0 = Div 1 = SQRT			

[†]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)

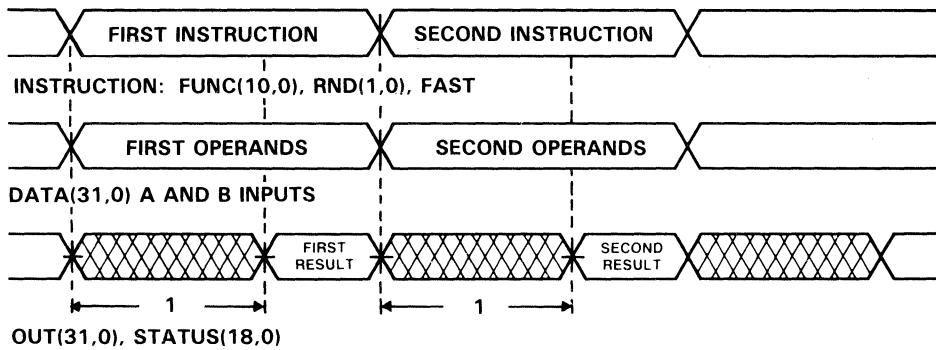
CHAINED OPERATION I10	OPERAND FORMAT/PRECISION			OUTPUT SOURCE I6	ADD ZERO I5	MULTIPLY BY ONE I4	NEGATE ALU RESULT I3 [†]	NEGATE MULTIPLIER RESULT I2 [†]	ALU OPERATIONS	
	I9	I8	I7						I1-I0	RESULT
1 = Chained	0 = floating point	0 = A(SP) 1 = A(DP)	0 = B(SP) 1 = B(DP)	0 = ALU result	0 = Normal operation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

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



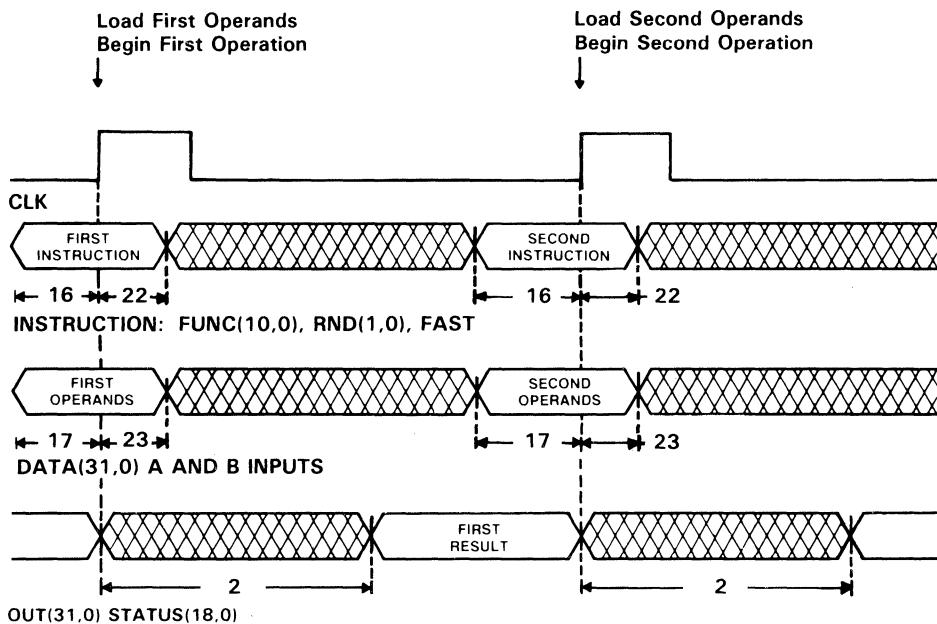
Sample Independent ALU Microinstructions

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/LS = X, OEH = 0, OEC = OES = 0, RESET = HALT = 1, TP1-0 = 11

**Figure 24. Single-Precision Independent ALU Operation, All Registers Disabled
(PIPS2-PIPS0 = 111, CLKMODE = X)**



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

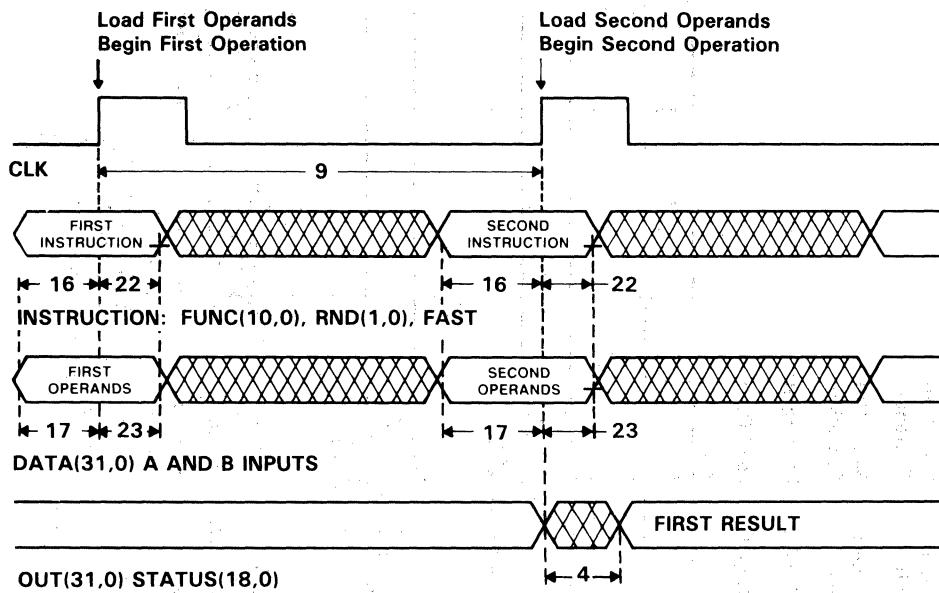
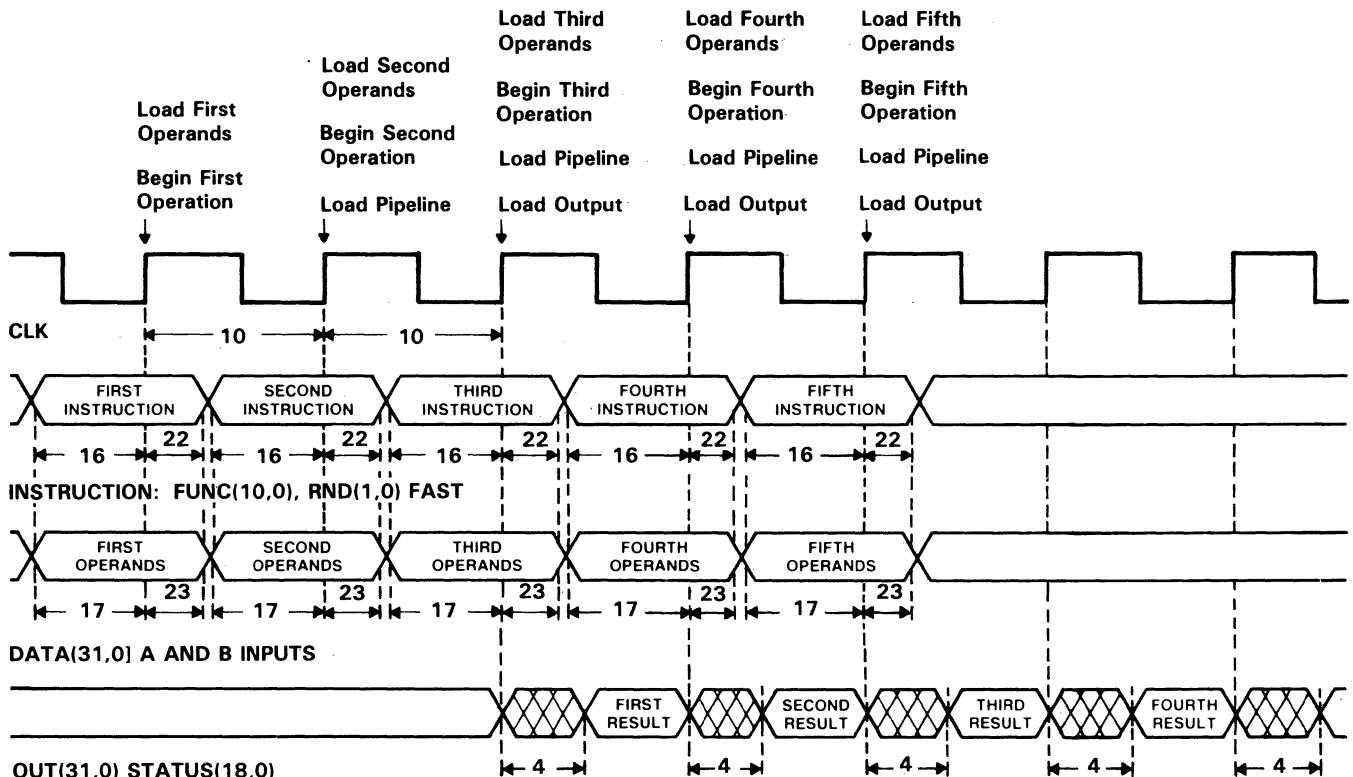


Figure 26. Single-Precision Independent ALU Operation, Input and Output Registers Enabled (PIPES2-PIPES0 = 010, CLKMODE = X)



NOTE: Assume PIPES2-0 = 000, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, SELMS/LS = X, OEH = 0, OEC = OES = 0, RESET = HALT = 1, TP1-0 = 11

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

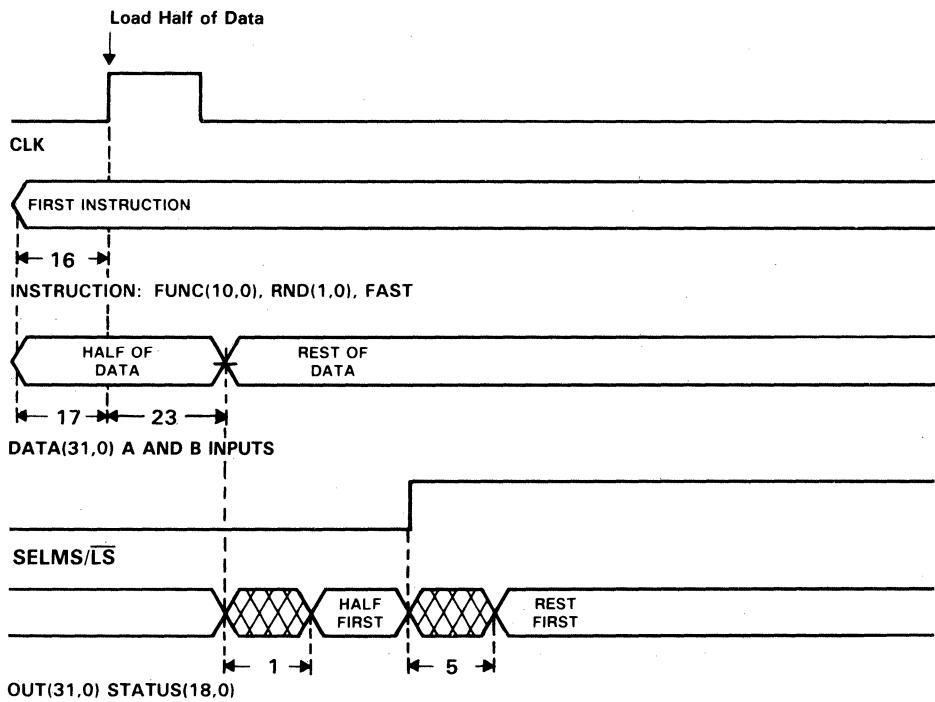


Figure 28. Double-Precision Independent ALU Operation, All Registers Disabled (PIPES2-PIPES0 = 111, CLKMODE = 0)

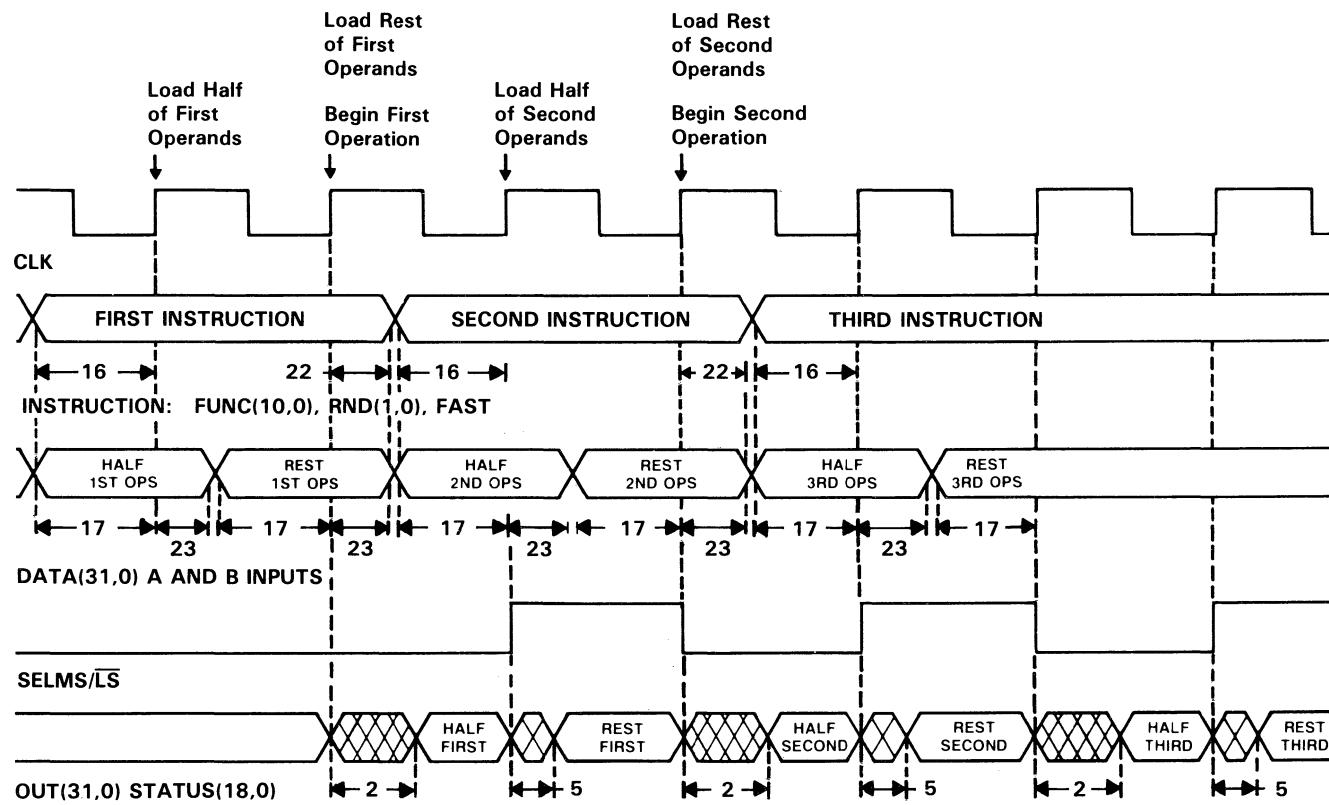
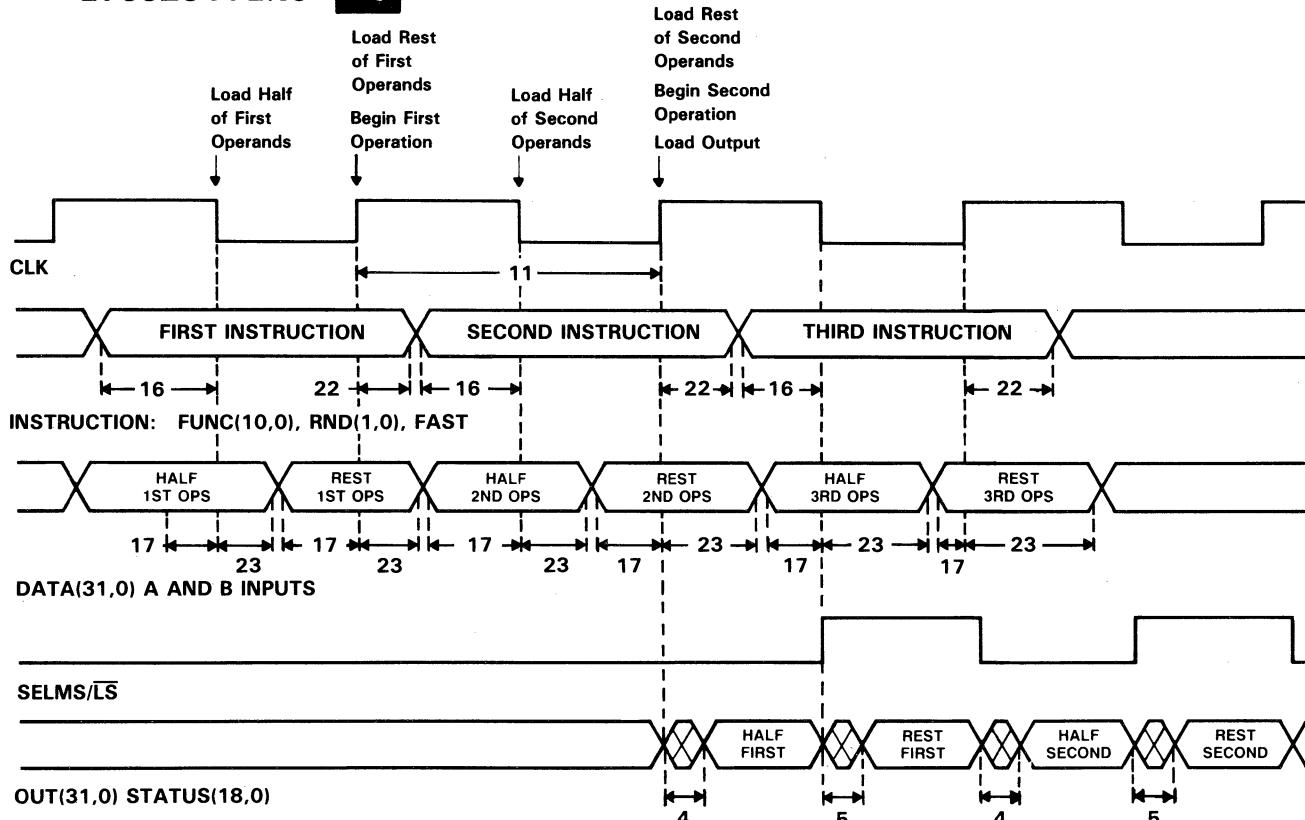
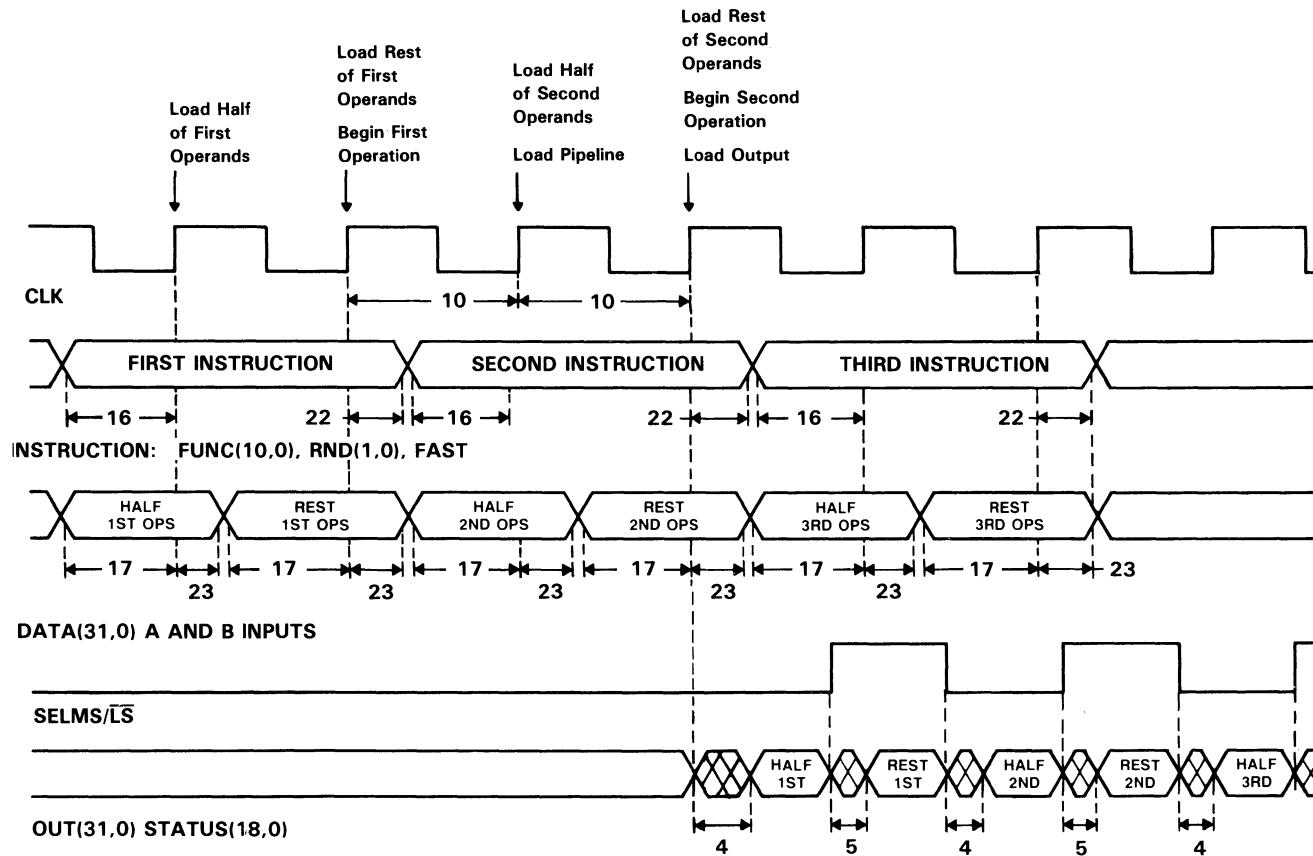


Figure 29. Double-Precision Independent ALU Operation, Input Registers Enabled
(PIPES2-PIPES0 = 110, CLKMODE = 0)



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

Figure 30. Double-Precision Independent ALU Operation, Input and Output Registers Enabled (PIPS2-PIPS0 = 010, CLKMODE = 1)



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

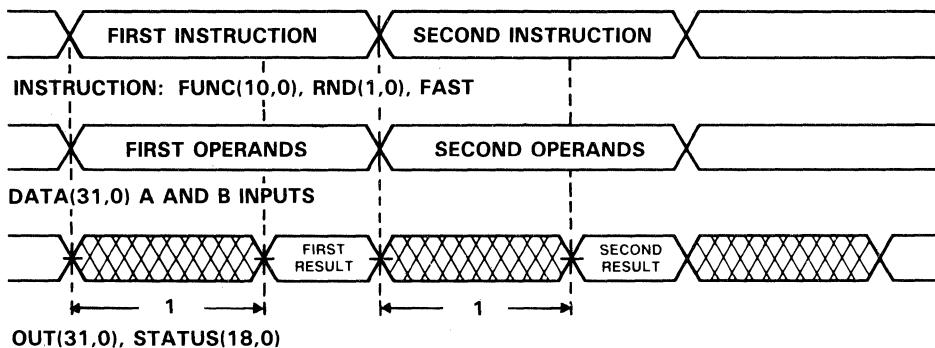
Figure 31. Double-Precision Independent ALU Operation, All Registers Enabled
(PIPES2-PIPES0 = 000, CLKMODE = 0)

SN74ACT8847



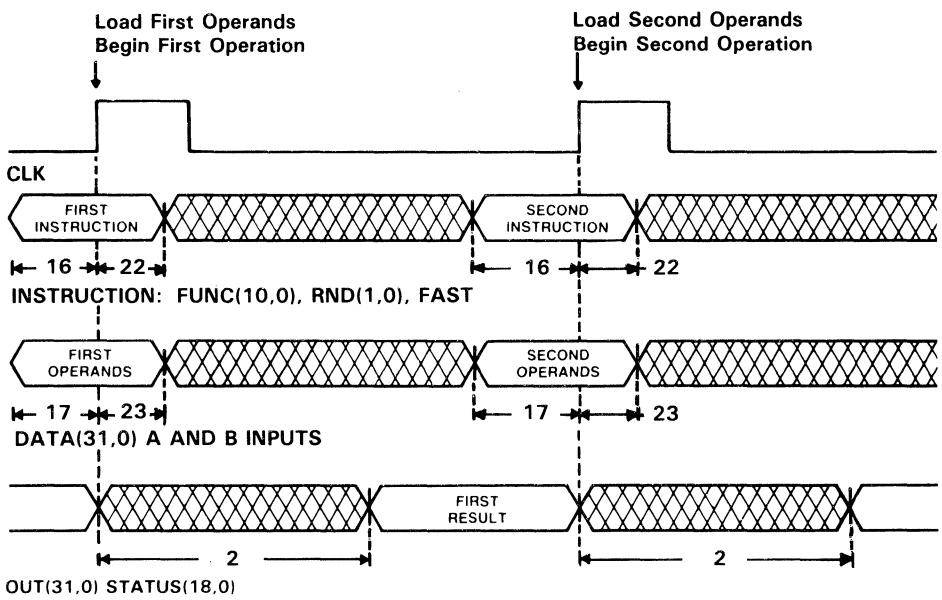
Sample Independent Multiplier Microinstructions

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.



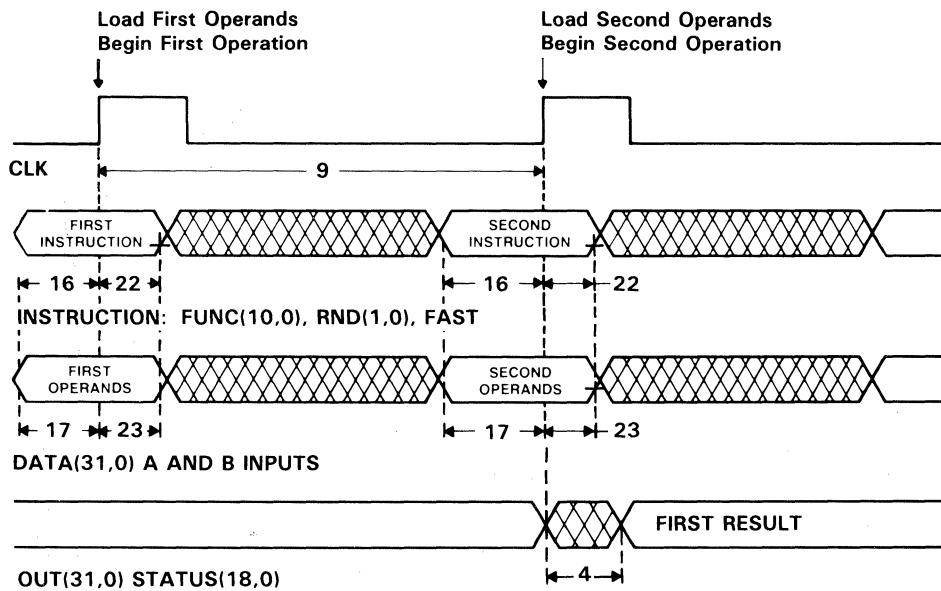
NOTE: Assume PIPES2-0 = 111, CONFIG1-0 = 01, ENRA = X, ENRB = X, SELMS/LSX, OEY = 0, OEC = OES = 0, RESET = HALT = 1 TP1-0 = 11

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



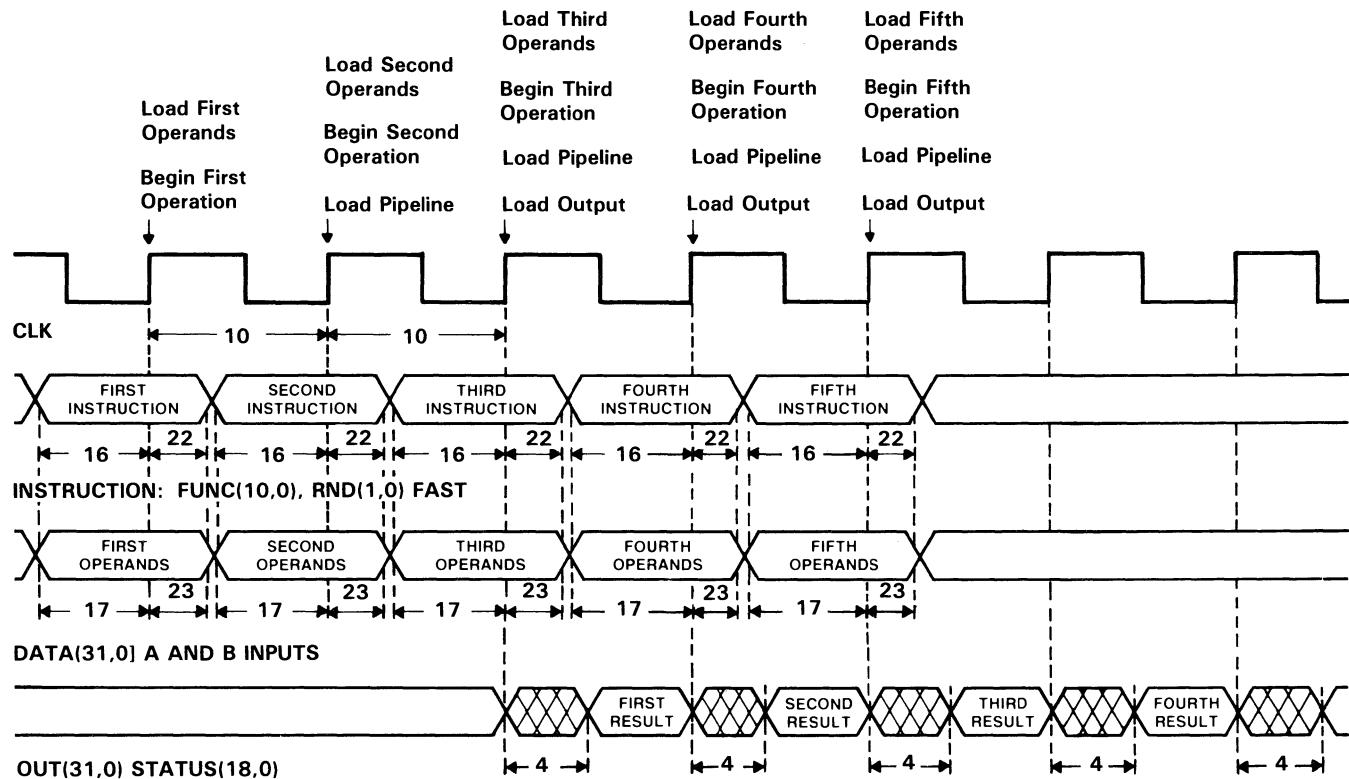
NOTE: Assume PIPES2-0 = 110, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, SELMS/LS = X, OEY = 0, OEC = OES = 0, RESET = HALT = 1 TP1-0 = 11

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



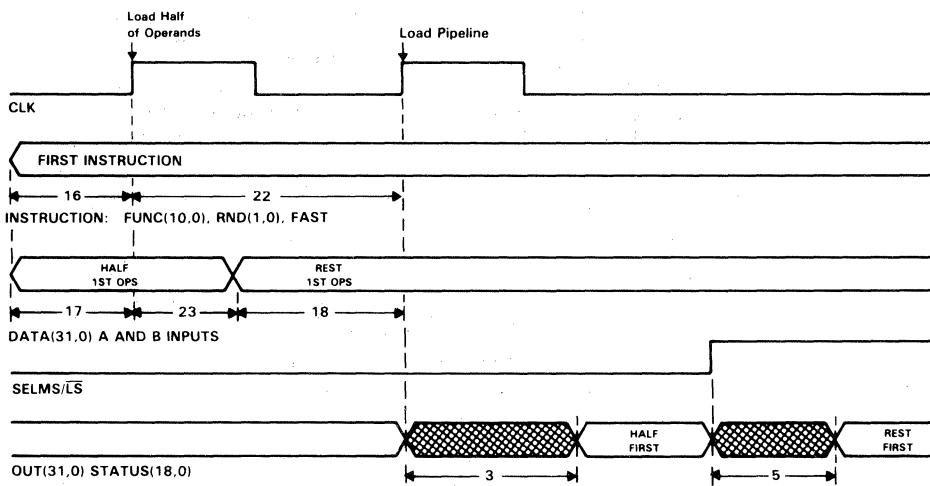
NOTE: Assume PIPES2-0 = 010, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, SELMS/LS = X, OEY = 0, OEC = OES = 0, RESET = HALT = 1 TP1-0 = 11

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



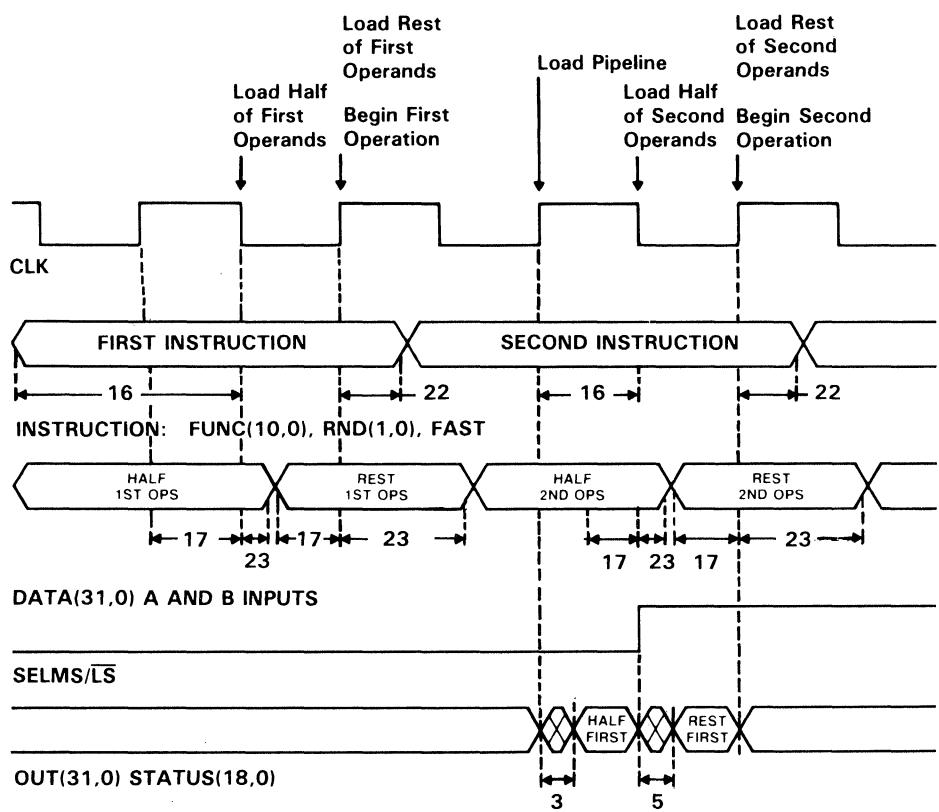
NOTE: Assume PIPES2-0 = 000, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, SELMS/LS = X, OEY = 0, OEC = OES = 0, RESET = HALT = 1, TP1-0 = 11

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



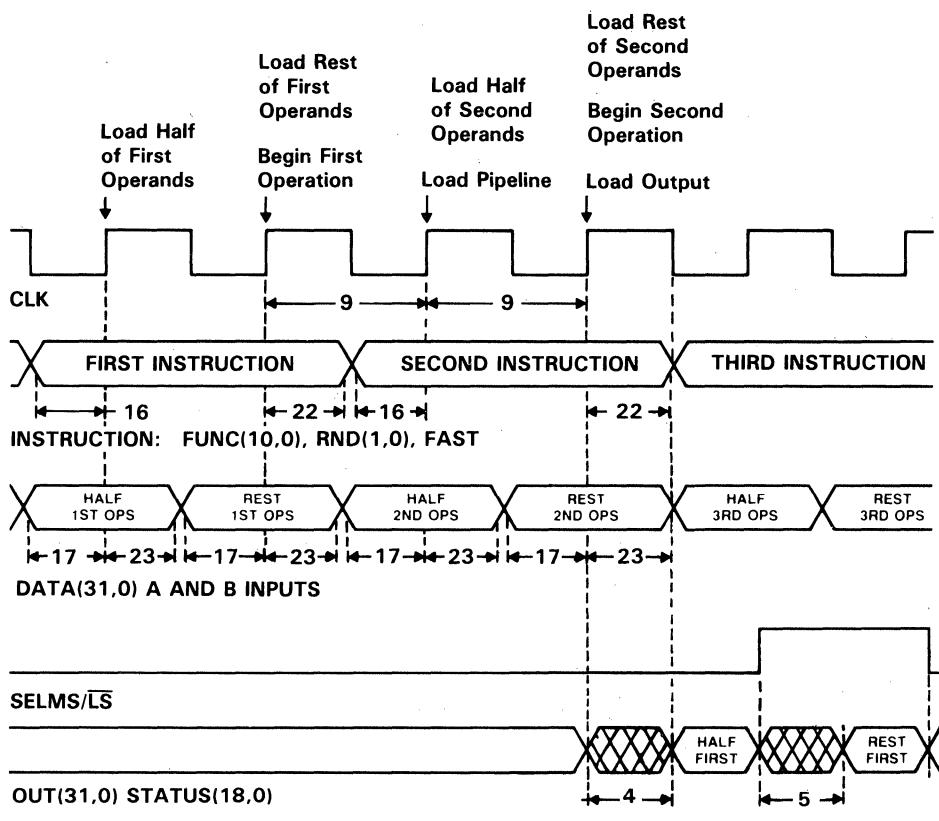
NOTE: Assume PIPES2-0 = 111, CLKMODE = 0, CONFIG1-0 = 11, ENRA = X, ENRB = X, OEW = 0, OEC = OES = 0, RESET = HALT = 1, TP1-0 = 11

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



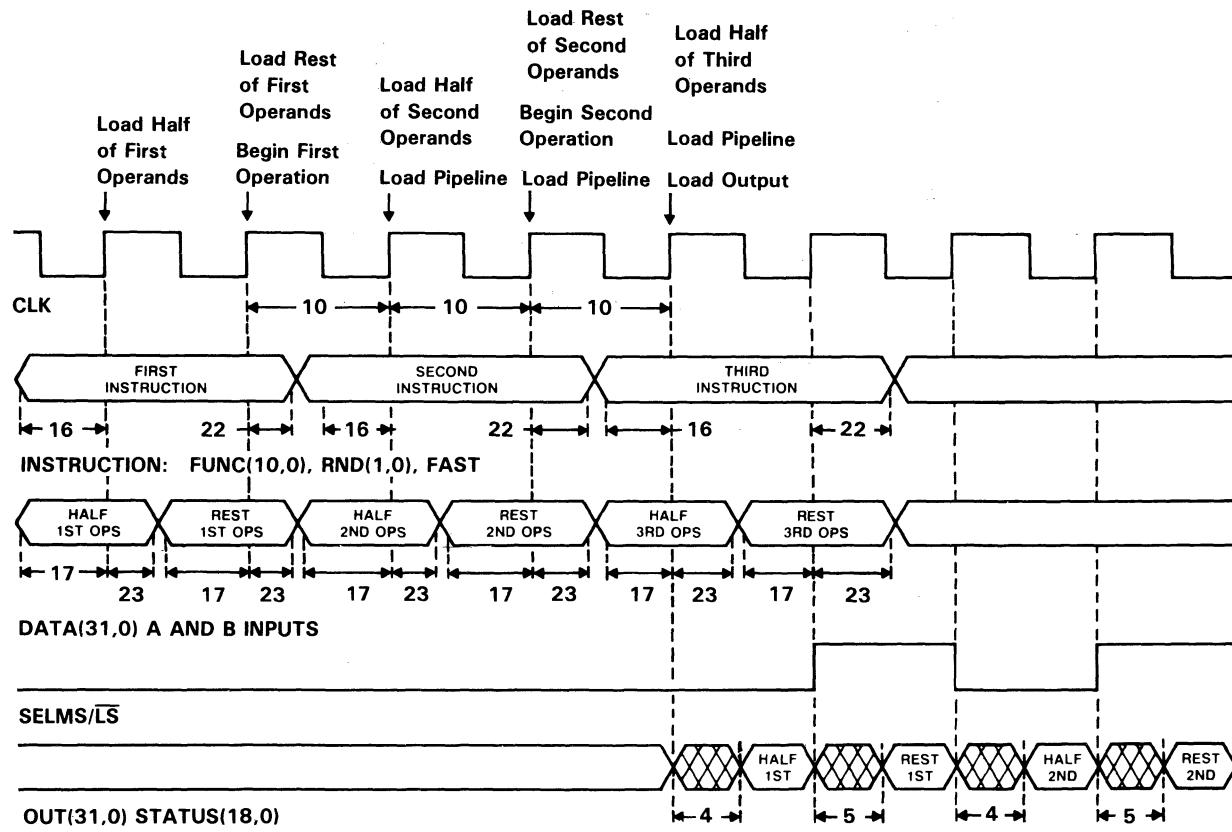
NOTE: Assume PIPES2-0 = 110, CONFIG1-0 = 11, ENRA = 1, ENRB = 1, OEY = 0, OEC = OES = 0, RESET = HALT = 1, TP1-0 = 11

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, OEY = 0, OEC = OES = 0, RESET = HALT = 1, TP1-0 = 11

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

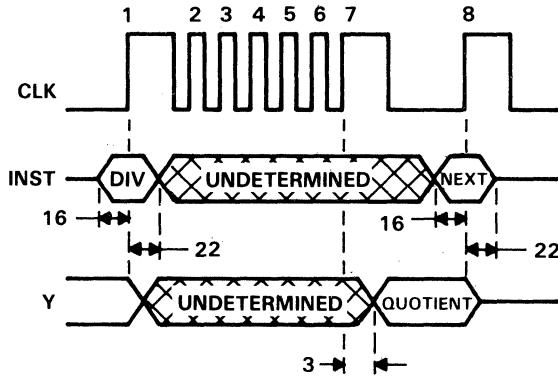


NOTE: Assume PIPES2-0 = 000, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, OEY = 0, OEC = 0, OES = 0, RESET = HALT = 1, TP1-0 = 11

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

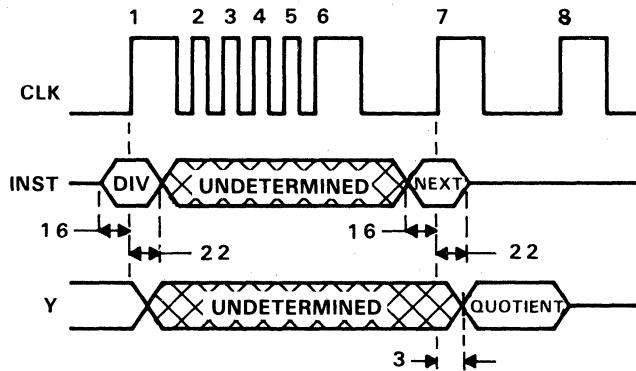
SN74ACT8847





NOTE: Assume PIPESEL2-0 = 110, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, SELMS/LS = X, OEV = 0, OEC = OES = 0, RESET = HALT = 1, TP1-0 = 11

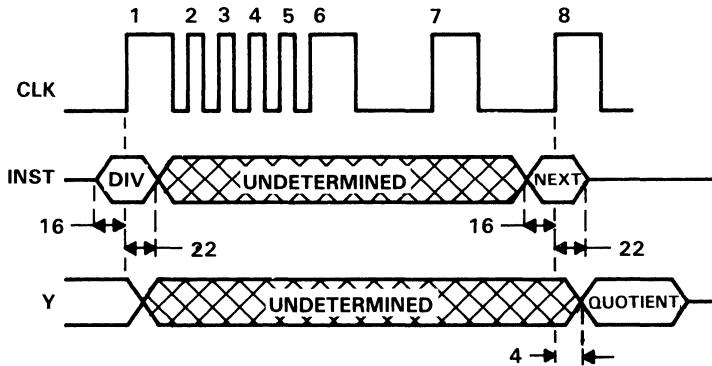
**Figure 40. Single-Precision Floating Point Division
(PIPESEL2-PIPESEL0 = 110, CLKMODE = X)**



7

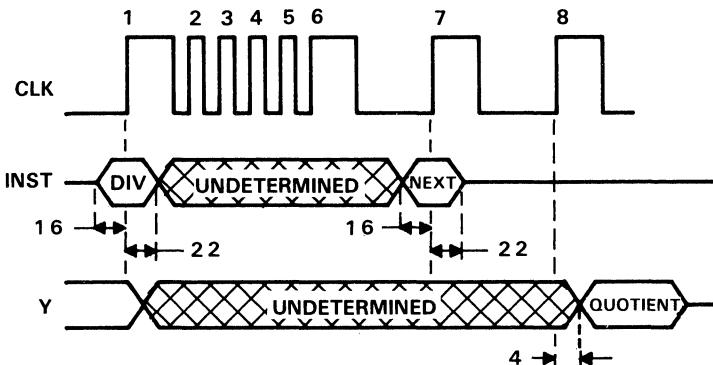
NOTE: Assume PIPESEL2-0 = 100, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, SELMS/LS = X, OEV = 0, OEC = OES = 0, RESET = HALT = 1, TP1-0 = 11

**Figure 41. Single-Precision Floating Point Division
(PIPESEL2-PIPESEL0 = 100, CLKMODE = X)**



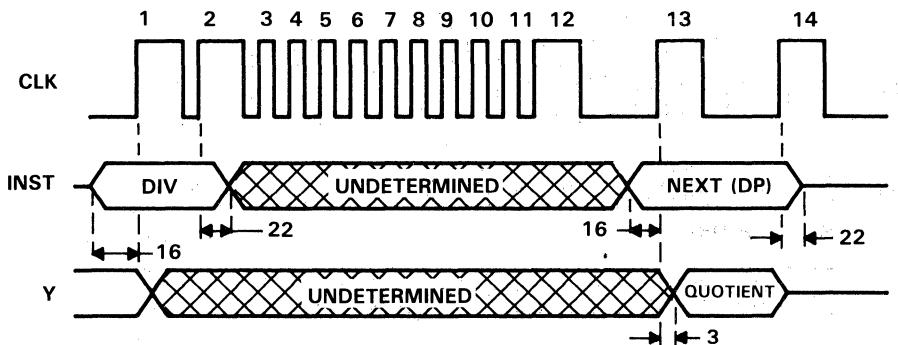
NOTE: Assume PIPES2-0 = 010, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, SELMS/LS = X, OEY = 0, OEC = OES = 0, RESET = HALT = 1, TP1-0 = 11

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



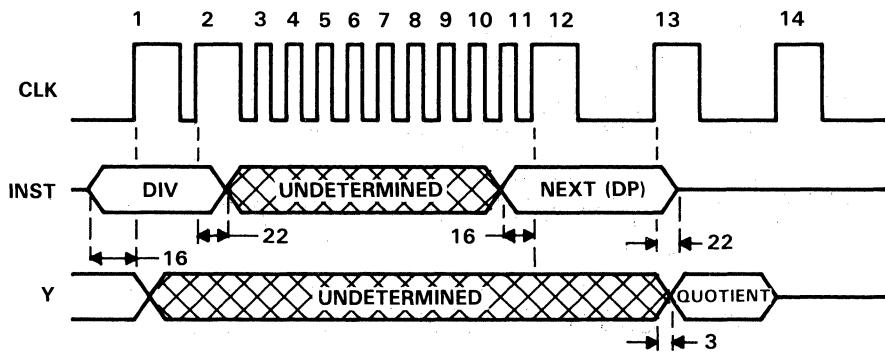
NOTE: Assume PIPES2-0 = 000, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, SELMS/LS = X, OEY = 0, OEC = OES = 0, RESET = HALT = 1, TP1-0 = 11

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



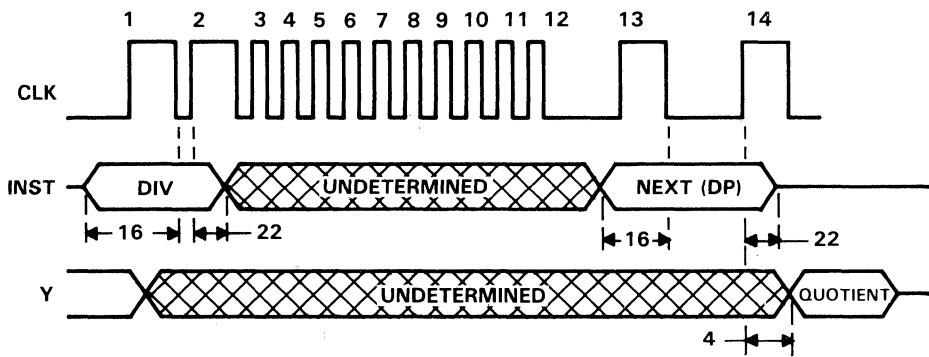
NOTE: Assume PIPES2-0 = 110, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, $\overline{OEY} = 0$, $\overline{OEC} = \overline{OES} = 0$,
 RESET = HALT = 1, TP1-0 = 11

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



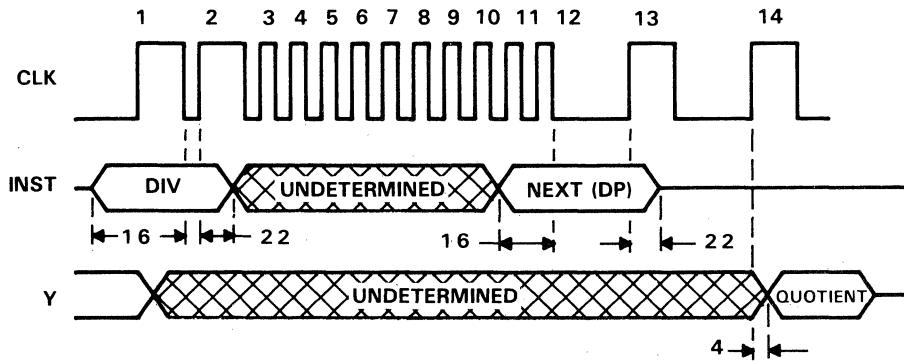
NOTE: Assume PIPES2-0 = 100, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, $\overline{OEY} = 0$, $\overline{OEC} = \overline{OES} = 0$,
 RESET = HALT = 1, TP1-0 = 11

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



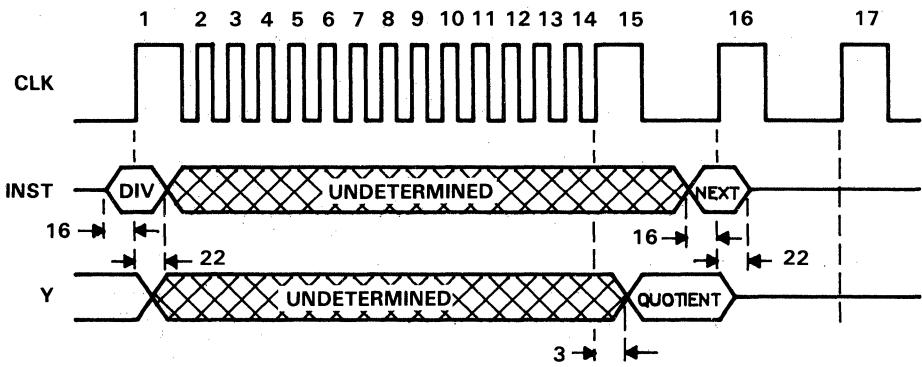
NOTE: Assume PIPES2-0 = 010, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, SELMS/LS = X, OEY = 0, OEC = OES = 0, RESET = HALT = 1, TP1-0 = 11

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



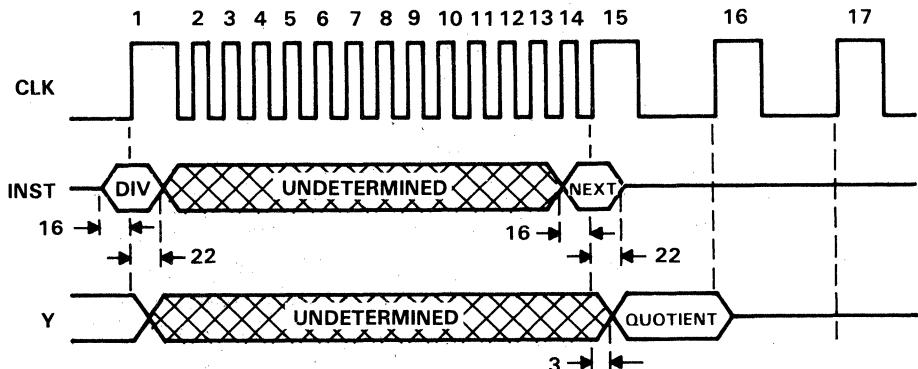
NOTE: Assume PIPES2-0 = 000, CONFIG1-0 = 00, ENRA = 1, ENRB = 1, OEY = 0, OEC = OES = 0, RESET = HALT = 1, TP1-0 = 11

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



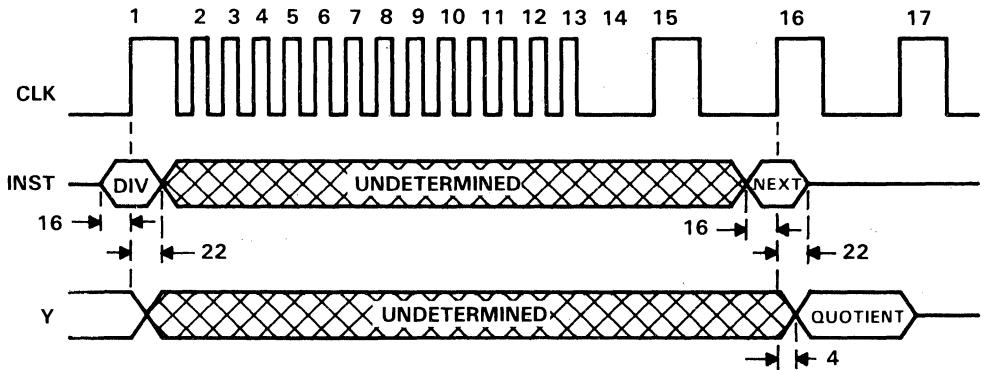
NOTE: Assume PIPES2-0 = 110, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, SELMS/LS = X, OEY = 0, OEC = OES = 0, RESET = HALT = 1, TP1-0 = 11. The result appears in the SREG.

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



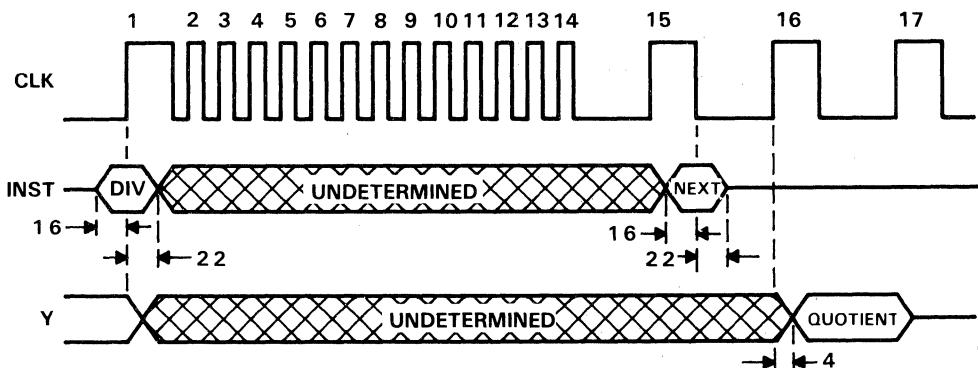
NOTE: Assume PIPES2-0 = 100, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, SELMS/LS = X, OEY = 0, OEC = OES = 0, RESET = HALT = 1, TP1-0 = 11. The result appears in the SREG.

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



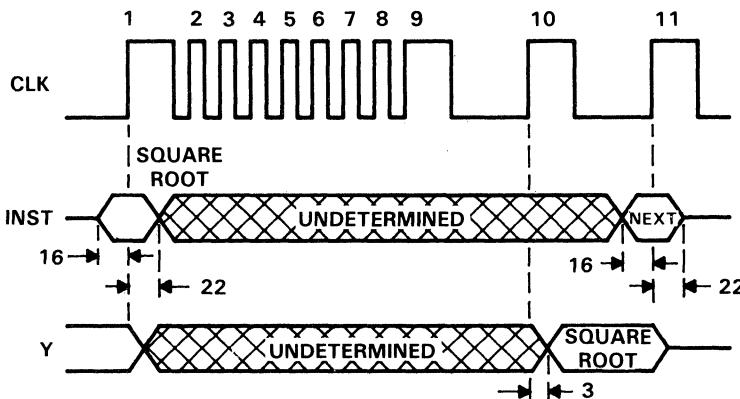
NOTE: Assume PIPES2-0 = 010, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, SELMS/LS = X, OEY = 0, OEC = OES = 0, RESET = HALT = 1, TP1-0 = 11. The result appears in the SREG.

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



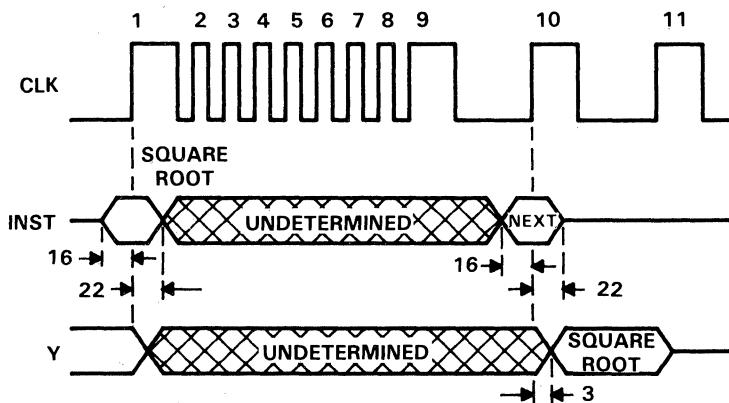
NOTE: Assume PIPES2-0 = 000, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, SELMS/LS = X, OEY = 0, OEC = OES = 0, RESET = HALT = 1, TP1-0 = 11. The result appears in the SREG.

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



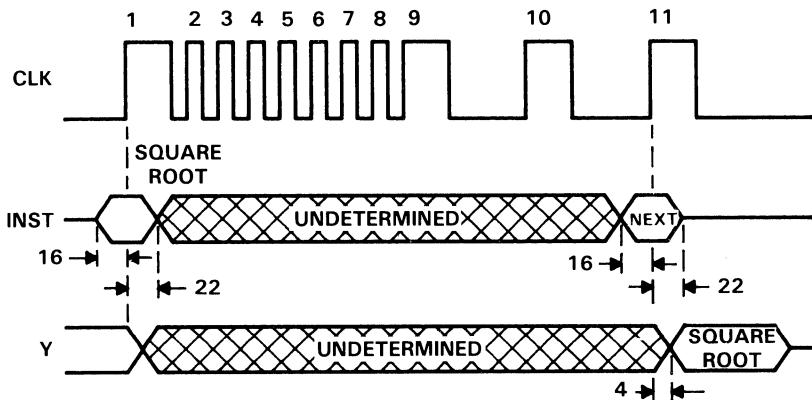
NOTE: Assume PIPES2-0 = 110, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, SELMS/LS = X, OEY = 0, OEC = OES = 0, RESET = HALT = 1, TP1-0 = 11

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



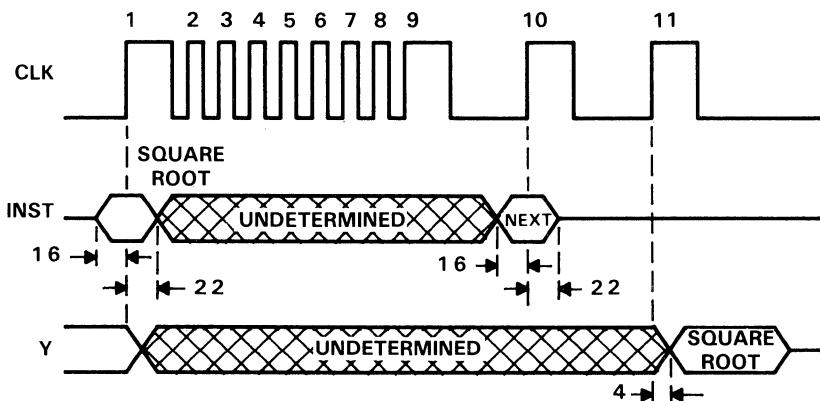
NOTE: Assume PIPES2-0 = 110, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, SELMS/LS = X, OEY = 0, OEC = OES = 0, RESET = HALT = 1, TP1-0 = 11

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



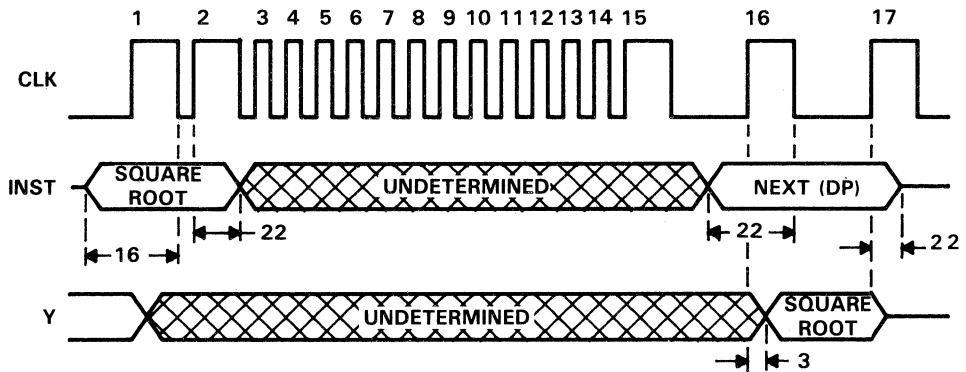
NOTE: Assume PIPES2-0 = 010, CONFIG1-0 = 01, ENRA = 1, SELMS/LS = X, OEV = 0,
OEC = OES = 0, RESET = HALT = 1, TP1-0 = 11

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



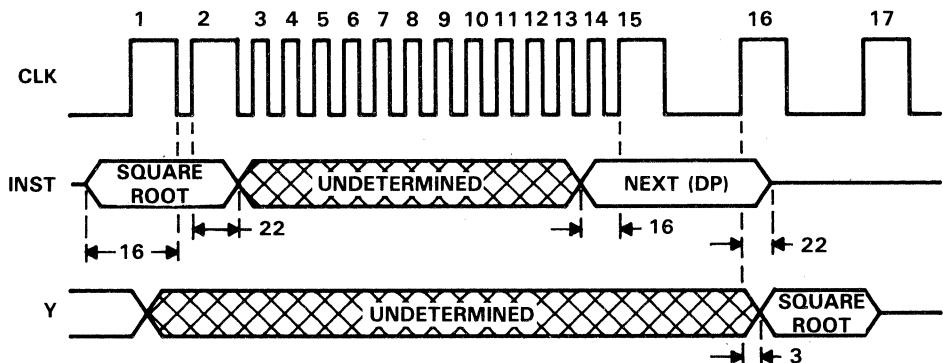
NOTE: Assume PIPES2-0 = 000, CONFIG1-0 = 00, ENRA = 1, SELMS/LS = X, OEV = 0,
OEC = OES = 0, RESET = HALT = 1, TP1-0 = 11

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



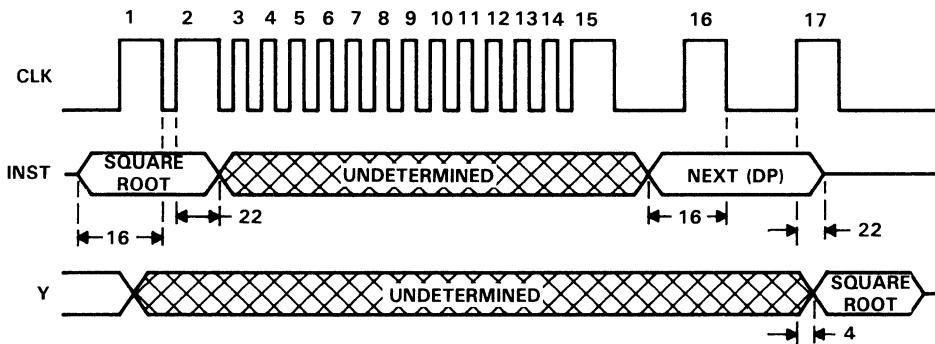
NOTE: Assume PIPES2-0 = 110, CONFIG1-0 = 11, ENRA = 1, \overline{OEY} = 0, \overline{OEC} = \overline{OES} = 0
 RESET = HALT = 1, TP1-0 = 11

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



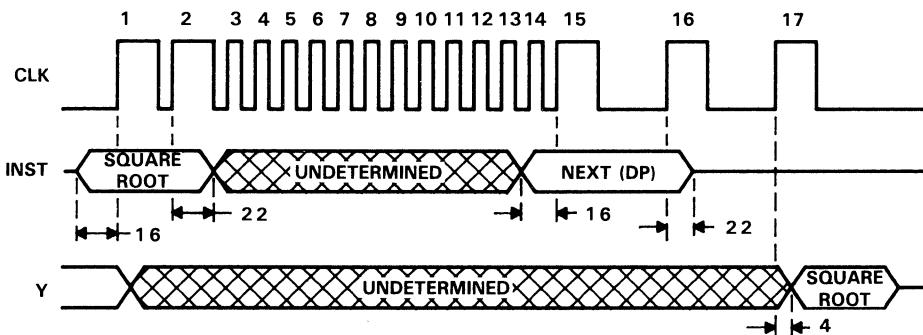
NOTE: Assume PIPES2-0 = 100, CONFIG1-0 = 01, ENRA = 1, \overline{OEY} = 0, \overline{OEC} = \overline{OES} = 0,
 RESET = HALT = 1, TP1-0 = 11

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



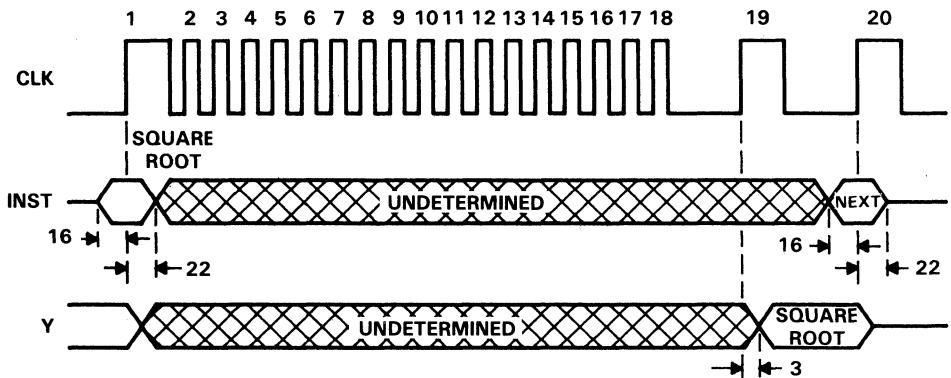
NOTE: Assume PIPES2-0 = 010, CONFIG1-0 = 10, ENRA = 1, \overline{OEY} = 0, \overline{OEC} = \overline{OES} = 0, RESET = HALT = 1, TP1-0 = 11

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



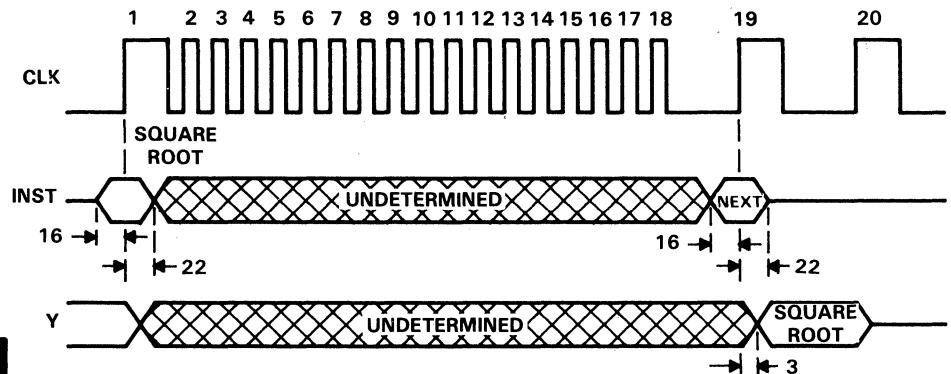
NOTE: Assume PIPES2-0 = 000, CONFIG1-0 = 00, ENRA = 1, \overline{OEY} = 0, \overline{OEC} = \overline{OES} = 0, RESET = HALT = 1, TP1-0 = 11

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



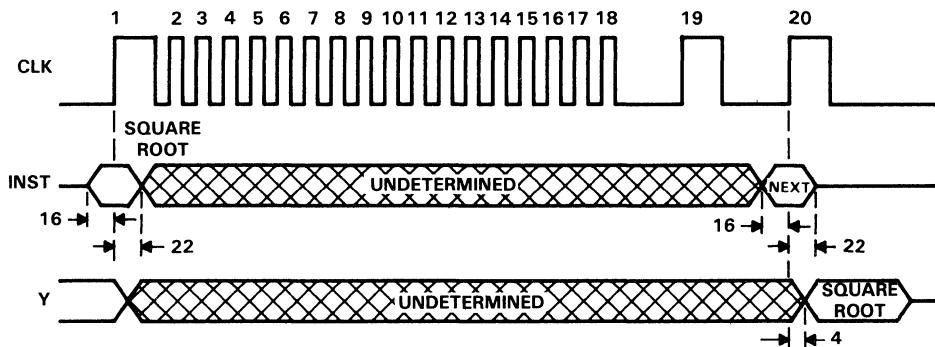
NOTE: Assume PIPES2-0 = 110, CONFIG1-0 = 01, ENRA = 1, SELMS/LS = X, OEY = 0, OEC = OES = 0, RESET = HALT = 1, TP1-0 = 11. The result appears in the SREG.

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



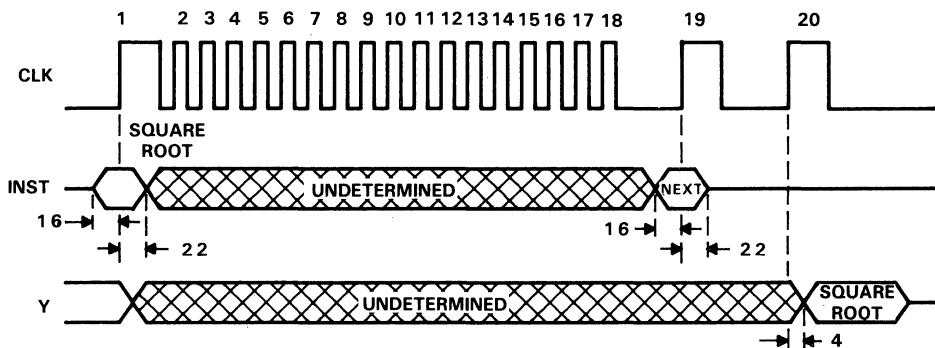
NOTE: Assume PIPES2-0 = 100, CONFIG1-0 = 00, ENRA = 1, SELMS/LS = X, OEY = 0, OEC = OES = 0, RESET = HALT = 1, TP1-0 = 11. The result appears in the SREG.

**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/LS = X, OEV = 0, OEC = OES = 0, RESET = 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)**

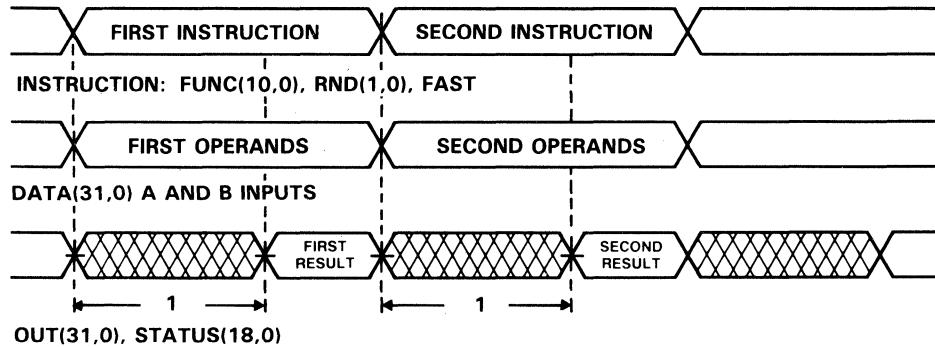


NOTE: Assume PIPES2-0 = 000, CONFIG1-0 = 00, ENRA = 1, SELMS/LS = X, OEV = 0, OEC = OES = 0, RESET = HALT = 1, TP1-0 = 11. The result appears in the SREG.

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

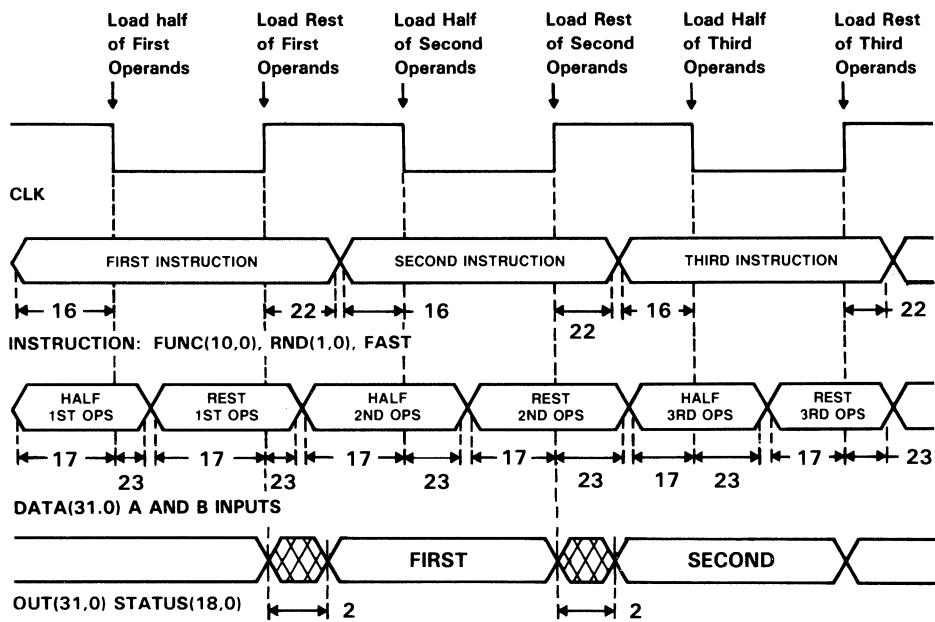
Sample Chained Mode Microinstructions

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/LS, OEY = 0, OEC = OES = 0, RESET = 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/LS = X, OEY = 0, OEC = OES = 0, RESET = HALT = 1, TP1-0 = 11

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

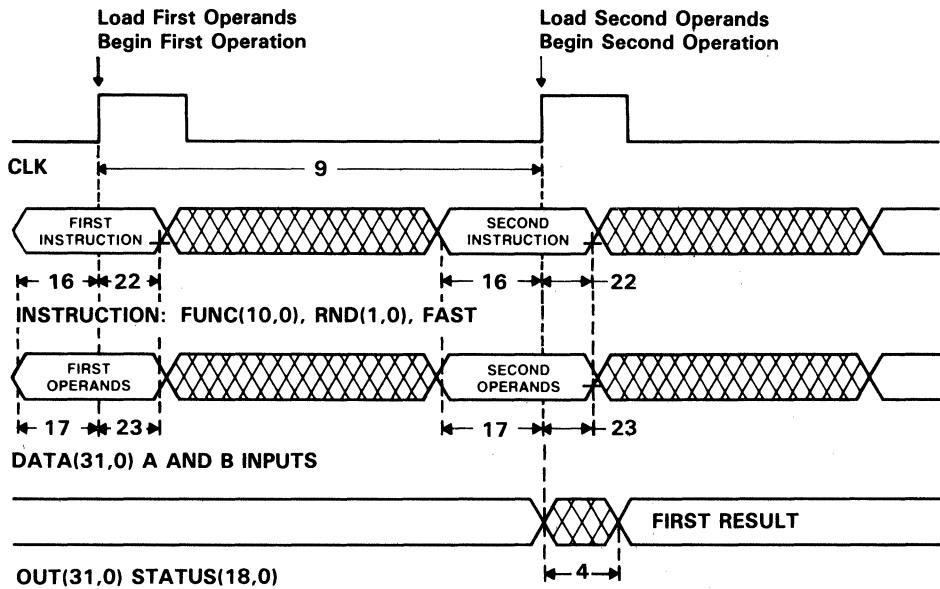
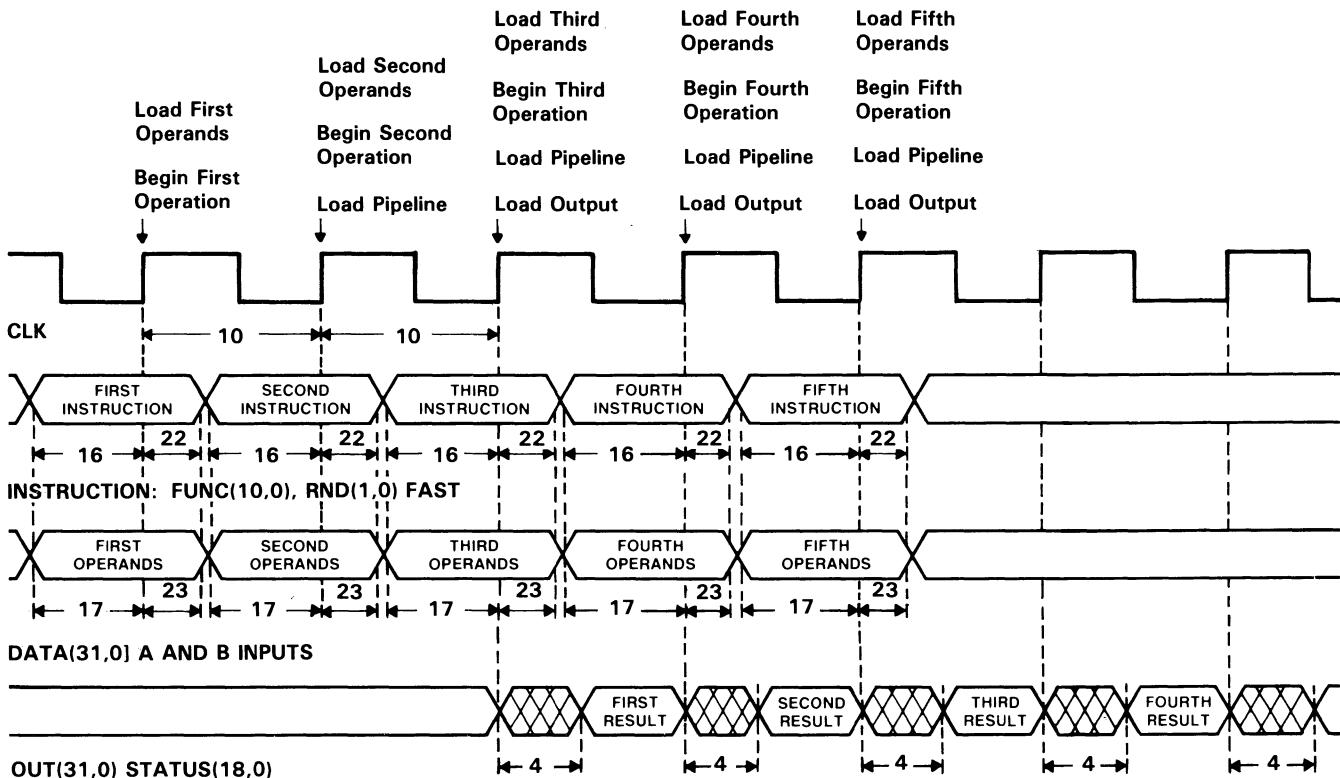
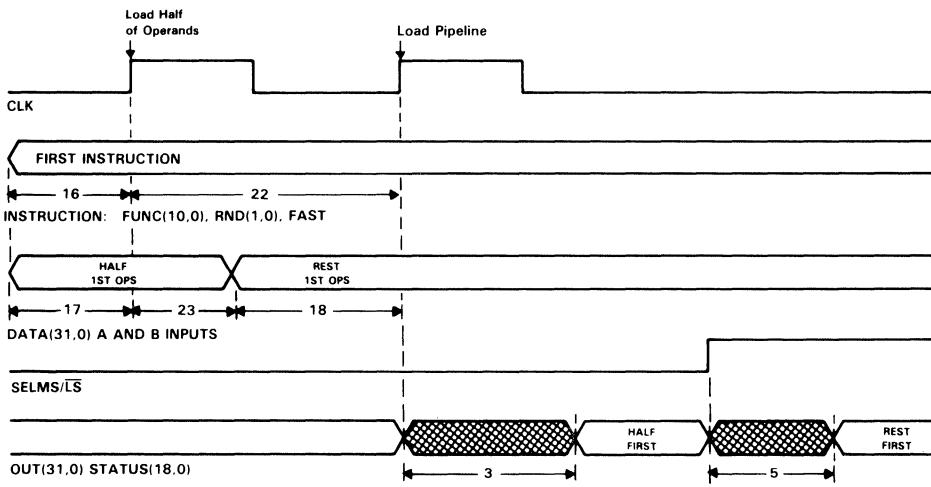


Figure 66. Single-Precision Chained Mode Operation, Input and Output Registers Enabled (PIPES2-PIPES0 = 010, CLKMODE = X)



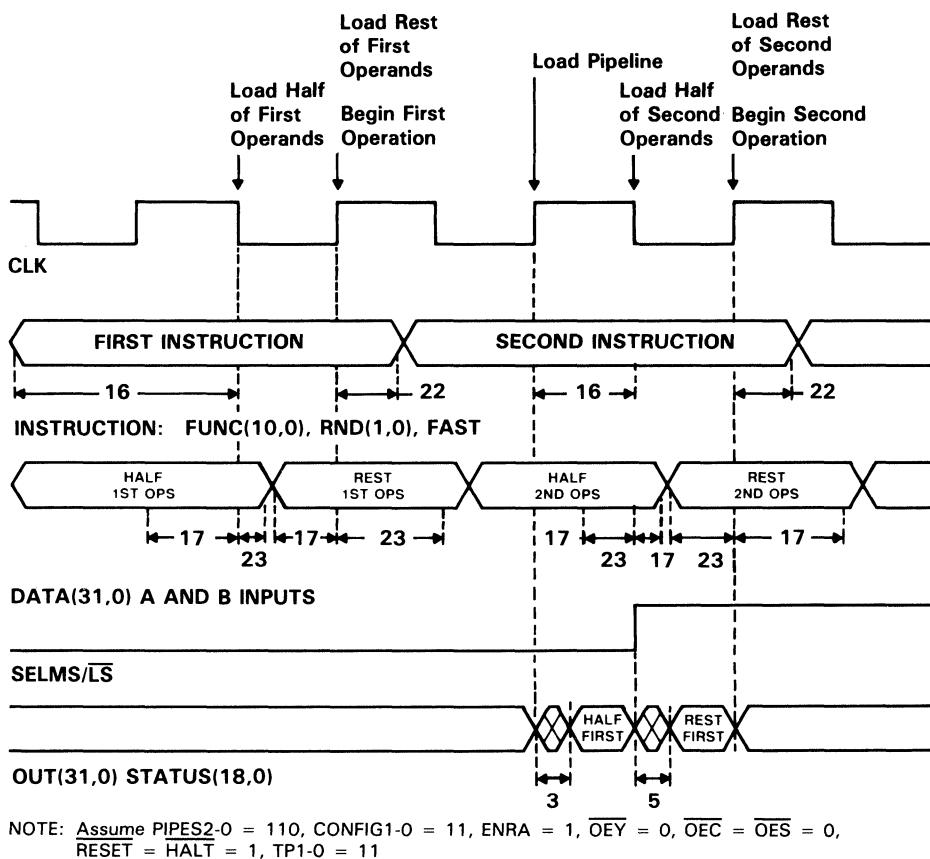
NOTE: Assume PIPES2-0 = 000, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, SELMS/LS = X, OEY = 0, OEC = OES = 0, RESET = HALT = 1, TP1-0 = 11

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

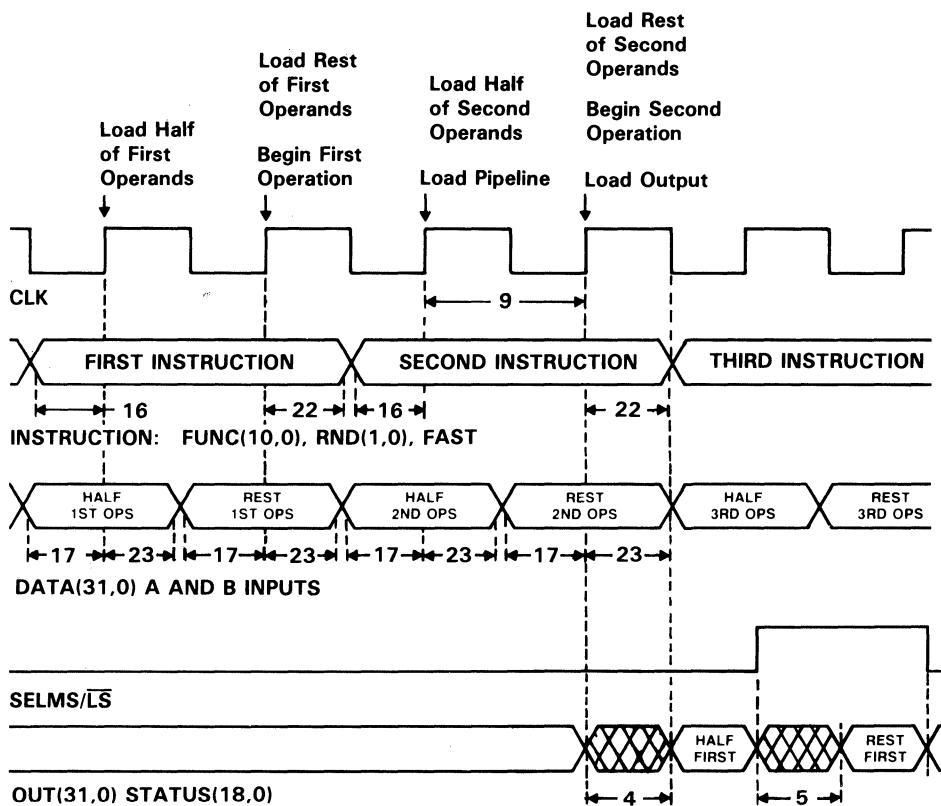


NOTE: Assume PIPES2-0 = 111, CONFIG1-0 = 11, ENRA = 1, ENRB = 1, OEY = 0, OEC = OES = 0, RESET = HALT = 1, TP1-0 = 11

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

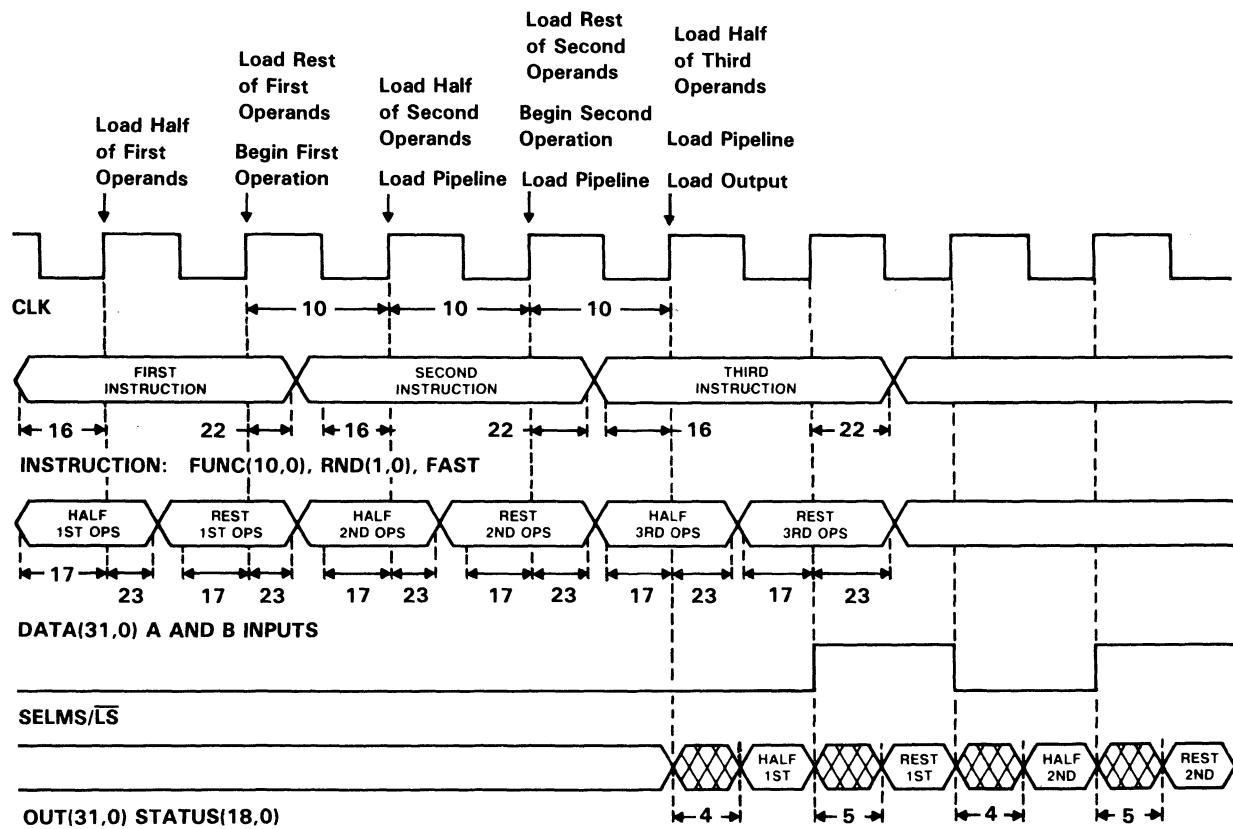


**Figure 69. Double-Precision Chained Mode Operation, Input Registers Enabled
(PIPESES2-PIPESES0 = 110, CLKMODE = 1)**



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

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

SN74ACT8847



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 been 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.

$$\begin{aligned}+0 + (-0) &= +0 \\+0 + (+0) &= +0 \\-0 + (-0) &= -0 \\-0 + (+0) &= +0 \\+0 - (-0) &= +0 \\+0 - (+0) &= +0 \\-0 - (-0) &= +0 \\-0 - (+0) &= -0\end{aligned}$$

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 DIVBYO flag set.

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

$$\begin{aligned}\pm \text{infinity multiplied by } \pm 0 \\ \pm \text{infinity divided by } \pm 0 \\ +\text{infinity} + (-\text{infinity}) \\ -\text{infinity} + (+\text{infinity}) \\ +\text{infinity} - (+\text{infinity}) \\ -\text{infinity} - (-\text{infinity})\end{aligned}$$

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

'ACT8847 Reference Guide

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 0 = A (SP) 1 = A (DP)
Move A operand (pass without NaN detect or status flags active)	00x x01x 0100	I7 selects precision of B operand and must equal I8.
Pass B operand	00x x01x 0101	I4 selects absolute value of a operand: 0 = A 1 = A
Convert from floating point to floating point (adjusts precision of input: SP → DP, DP → SP) [‡]	00x x01x 0110	During integer to floating point conversion, A is not allowed as a result.
Floating point to unsigned integer conversion [†]	00x x01x 0111	
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	
Add A + B	00x x001 1x00	
Subtract A - B	00x x000 0x01	
Subtract A - B	00x x001 0x01	
Subtract A - B	00x x000 1x01	
Subtract A - B	00x x001 1x01	
Compare A, B	00x x000 0x10	
Compare A , B	00x x001 0x10	
Compare A, B	00x x000 1x10	
Compare A , B	00x x001 1x10	
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: 0 = Single-precision 2's complement
Negate A operand (1's complement)	010 xx10 0010	1 = Single-precision unsigned integer
Pass B operand	010 xx10 0101	I8 must equal I7
Shift left logical [†]	010 xx10 1000	
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	
Subtract A - B	010 x000 0001	x = Don't Care
Compare A, B	010 x000 0010	I7 selects format of A and B operands:
Subtract B - A	010 x000 0011	0 = Single-precision 2's complement
Logical AND A, B	010 x000 1000	1 = Single-precision unsigned integer
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
Multiply -(A * B)	00x x100 01xx	I8 selects A operand precision (0 = SP, 1 = DP)
Multiply A * B	00x x100 10xx	I7 selects B operand precision (0 = SP, 1 = DP)
Multiply -(A * B)	00x x100 11xx	I1 selects A operand format (0 = Normal, 1 = Wrapped)
Multiply A * B	00x x101 00xx	I0 selects B operand format (0 = Normal, 1 = Wrapped)
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
SQRT A	00x x110 1xxx	I8 selects A operand precision and I7 selects B operand precision (0 = SP, 1 = DP)
Divide A / B	00x x111 0xxx	I2 negates multiplier result (0 = Normal, 1 = Negated)
SQRT A	00x x111 1xxx	I1 selects A operand format and I0 selects B operand format (0 = Normal, 1 = Wrapped)

[†]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	010 x100 0000	x = Don't care
Divide A / B	010 x110 0000	I7 selects operand format:
SQRT A	010 x110 1000	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 SOURCE	INSTRUCTION INPUTS I10-I0	NOTES
MULTIPLIER	ALU			
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 SOURCE	INSTRUCTION INPUTS I10-I0	NOTES
MULTIPLIER	ALU			
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	
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	x = Don't Care I7 selects format of A and B operands: 0 = SP 2's complement 1 = SP unsigned integer

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			
CONFIG1	CONFIG0	DATA LOADED INTO TEMP REGISTER ON FIRST CLOCK AND RA/RB REGISTERS ON SECOND CLOCK [†]		DATA LOADED INTO RA/RB REGISTERS ON SECOND CLOCK	
		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-SELOPO 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	SELOPO	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-RNDO select the rounding mode as shown in Table 46.

Table 46. Rounding Modes

RND1- RNDO	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-TPO as shown below.

Table 48. Test Pin Control Inputs

TP1- TPO	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
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 FLOW_C	Places device in FAST mode Causes output value to bypass C register and appear on C register output bus.	Places device in IEEE mode No effect
HALT	No effect	Stalls device operation but does not affect registers, internal states, or status
OEC OES OEY RESET	Disables compare pins Disables status outputs Disables Y bus No effect	Enables compare pins Enables status outputs Enables Y bus Clears internal states, status, internal pipeline registers, and exception disable register. Does not affect other data registers.
SELMS/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 $0 * \infty$ 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.

7

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.

SN74ACT8847

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

SN74ACT8847

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			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:

$$\frac{dx_1}{dt} = x_2, \dots, \frac{dx_{(n-1)}}{dt} = x_n$$

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

$$\frac{d}{dt} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} b_{11} & \dots & b_{1n} \\ b_{21} & \dots & b_{2n} \\ \vdots & & \vdots \\ b_{n1} & \dots & b_{nn} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix}$$

$$\text{or, } \dot{\bar{X}} = a_{\bar{X}} + b_{\bar{U}}$$

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

$$\dot{x}_1 = (a_{11} * x_1 + \dots + a_{1n} * x_n) + (b_{11} * u_1 + \dots + b_{1n} * u_n)$$

$$\text{where } \dot{x}_1 = dx_1/dt.$$

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 $n \times n$ matrix A multiplied by an $n \times n$ matrix X yields an $n \times n$ matrix C whose elements c_{ij} are given by this equation:

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

For the c_{ij} elements to be calculated by the 'ACT8847, the corresponding elements a_{jk} 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:

$$T = \begin{bmatrix} a & b & c & : & d \\ e & f & g & : & h \\ i & j & k & : & l \\ \dots & \dots & \dots & : & \dots \\ m & n & o & : & p \end{bmatrix}$$

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

$$\begin{array}{c|c} & : \\ & : 3 \\ 3 \times 3 & : x \\ & : 1 \\ \dots & \dots \dots \\ 1 \times 3 & : 1 \times 1 \end{array}$$

7

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:

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

translation to origin rotation about origin translation back to initial position

where R may be expressed as:

$$R = \begin{bmatrix} n_1^2 + (1-n)^2 \cos\phi & n_1n_2(1-\cos\phi) + n_3\sin\phi & n_1n_3(1-\cos\phi) - n_2\sin\phi & 0 \\ n_1n_2(1-\cos\phi) - n_3\sin\phi & n_2^2 + (1-n_2)^2 \cos\phi & n_2n_3(1-\cos\phi) + n_1\sin\phi & 0 \\ n_1n_3(1-\cos\phi) + n_2\sin\phi & n_2n_3(1-\cos\phi) - n_1\sin\phi & n_3^2 + (1-n_3)^2 \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

and $n_1 = q_1/(q_1^2 + q_2^2 + q_3^2)^{1/2}$ = direction cosine for x-axis of rotation

$n_2 = q_2/(q_1^2 + q_2^2 + q_3^2)^{1/2}$ = direction cosine for y-axis of rotation

$n_3 = q_3/(q_1^2 + q_2^2 + q_3^2)^{1/2}$ = direction cosine for z-axis of rotation

$\bar{n} = (n_1 \ n_2 \ n_3)$ = unit vector for \bar{Q}

\bar{Q} = vector defining axis of rotation = $[q_1 \ q_2 \ q_3]$

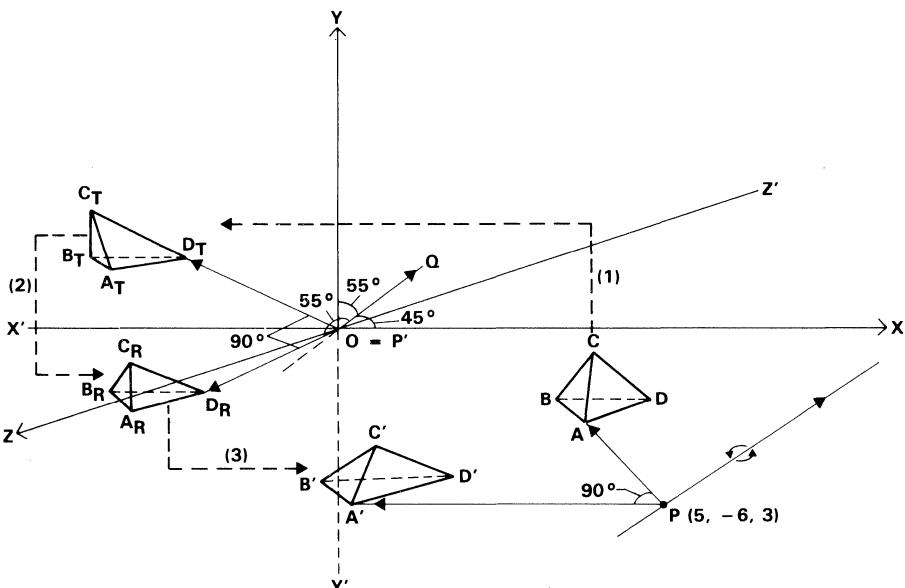
ϕ = the rotation angle about \bar{Q}

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 $[n_1, n_2, n_3]$, 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 θ is 90 degrees (see Figure 72). The rotation matrix [R] becomes

2	-3	3	1
1	-2	2	1
2	-1	2	1
2	-2	2	1

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



- (1) THIS ARROW DEPICTS THE FIRST TRANSLATION
- (2) THIS ARROW 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:

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

$$\begin{bmatrix} 2 & -3 & 3 & 1 \\ 1 & -2 & 2 & 1 \\ 2 & -1 & 2 & 1 \\ 2 & -2 & 2 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -5 & 6 & -3 & 1 \end{bmatrix}$$

translation
to origin

$$\begin{bmatrix} 0.750 & 1.140 & 0.112 & 0 \\ -0.274 & 0.250 & 1.22 & 0 \\ 1.112 & -0.513 & 0.500 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

rotation about origin

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 5 & -6 & 3 & 1 \end{bmatrix}$$

translation
back to
initial
position

(3)

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:

$$\begin{array}{c}
 \begin{array}{|c c c c|} \hline
 2 & -3 & 3 & 1 \\ \hline
 1 & -2 & 2 & 1 \\ \hline
 2 & -1 & 2 & 1 \\ \hline
 2 & -2 & 2 & 1 \\ \hline
 \end{array}
 \quad
 \begin{array}{|c c c c|} \hline
 1 & 0 & 0 & 0 \\ \hline
 0 & 1 & 0 & 0 \\ \hline
 0 & 0 & 1 & 0 \\ \hline
 -5 & 6 & -3 & 1 \\ \hline
 \end{array}
 \quad
 \begin{array}{|c c c c|} \hline
 (2-5) & (-3+6) & (3-3) & 1 \\ \hline
 (1-5) & (-2+6) & (2-3) & 1 \\ \hline
 (2-5) & (-1+6) & (2-3) & 1 \\ \hline
 (2-5) & (-2+6) & (2-3) & 1 \\ \hline
 \end{array}
 \\
 = \\
 \begin{array}{c}
 \text{translation} \\
 \text{to origin} \\
 \hline
 \end{array}
 \quad
 \begin{array}{c}
 \text{vertices of translated} \\
 \text{tetrahedron} \\
 \hline
 \end{array}
 \\
 \\
 \begin{array}{c}
 = \\
 \begin{array}{|c c c c|} \hline
 -3 & +3 & 0 & 1 \\ \hline
 -4 & +4 & -1 & 1 \\ \hline
 -3 & +5 & -1 & 1 \\ \hline
 -3 & +4 & -1 & 1 \\ \hline
 \end{array}
 \quad
 \begin{array}{c}
 \text{AT} \\
 \text{BT} \\
 \text{CT} \\
 \text{DT} \\
 \hline
 \end{array}
 \end{array}
 \end{array}$$

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:

7

SN74ACT8847

$$\begin{array}{c}
 \begin{array}{|c c c c|} \hline
 -3 & 3 & 0 & 1 \\ \hline
 -4 & 4 & -1 & 1 \\ \hline
 -3 & 5 & -1 & 1 \\ \hline
 -3 & 4 & -1 & 1 \\ \hline
 \end{array}
 \quad
 \begin{array}{|c c c c|} \hline
 0.750 & 1.140 & 0.112 & 0 \\ \hline
 -0.274 & 0.250 & 1.22 & 0 \\ \hline
 1.112 & -0.513 & 0.500 & 0 \\ \hline
 0 & 0 & 0 & 1 \\ \hline
 \end{array}
 \quad
 \begin{array}{|c c c c|} \hline
 -3.072 & -2.670 & 3.324 & 1 \\ \hline
 -5.208 & -3.047 & 3.932 & 1 \\ \hline
 -4.732 & -1.657 & 5.264 & 1 \\ \hline
 -4.458 & -1.907 & 4.044 & 1 \\ \hline
 \end{array}
 \\
 = \\
 \begin{array}{c}
 \text{rotation about origin} \\
 \hline
 \end{array}
 \quad
 \begin{array}{c}
 \text{rotated coordinates} \\
 \hline
 \end{array}
 \end{array}$$

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:

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

For example, the coordinates may be computed as follows:

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

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

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

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

7

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

The other rotated vertices are computed in a similar manner:

$$\begin{aligned} B' &= [-5.208 \quad -3.047 \quad 3.932 \quad 1] \\ C' &= [-4.732 \quad -1.657 \quad 5.264 \quad 1] \\ D' &= [-4.458 \quad -1.907 \quad 4.044 \quad 1] \end{aligned}$$

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

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

Table 53. Microinstructions for Sample Matrix Multiplication

C C C		S	
L O O P P	S S	M	S S
K N N I I	E E	S	B E E R
M F F P P	L L	R R F E E S /	Y L L E H
O I I E E	O O	N N A N N R	<u>0 0 0 T S S S A T T</u>
I I D G G S S	P P	D D S R R C L E E E E T T E L P P	
10-0 E 1-0 2-0	7-0	1-0 T A B C S Y C S P	1-0 T T T 1-0
000 0100 0000	0 01 0101111 xxxx	00 0 1 1 x x x x x x x xx	1 1 11
100 0110 0000	0 01 0101111 xxxx	00 0 1 1 x x x x x x x xx	1 1 11
100 0000 0000	0 01 0101111 1010	00 0 1 1 x x x x x x x xx	1 1 11
100 0000 0000	0 01 0101111 1010	00 0 1 1 x x x x x x x xx	1 1 11
100 0000 0000	0 01 0101111 1010	00 0 1 1 x x x x x x x xx	1 1 11
100 0110 0000	0 01 0101111 xxxx	00 0 1 1 x x x x x x x xx	1 1 11
100 0000 0000	0 01 0101111 1010	00 0 1 1 x x x x x x x xx	1 1 11
100 0000 0000	0 01 0101111 1010	00 0 1 1 x x x x x x x xx	1 1 11
100 0000 0000	0 01 0101111 1010	00 0 1 1 x x x x x x x xx	1 1 11
100 0110 0000	0 01 0101111 xxxx	00 0 1 1 x 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:

0
0
0
1

The h-scalars h_1 , h_2 , h_3 , and h_4 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.

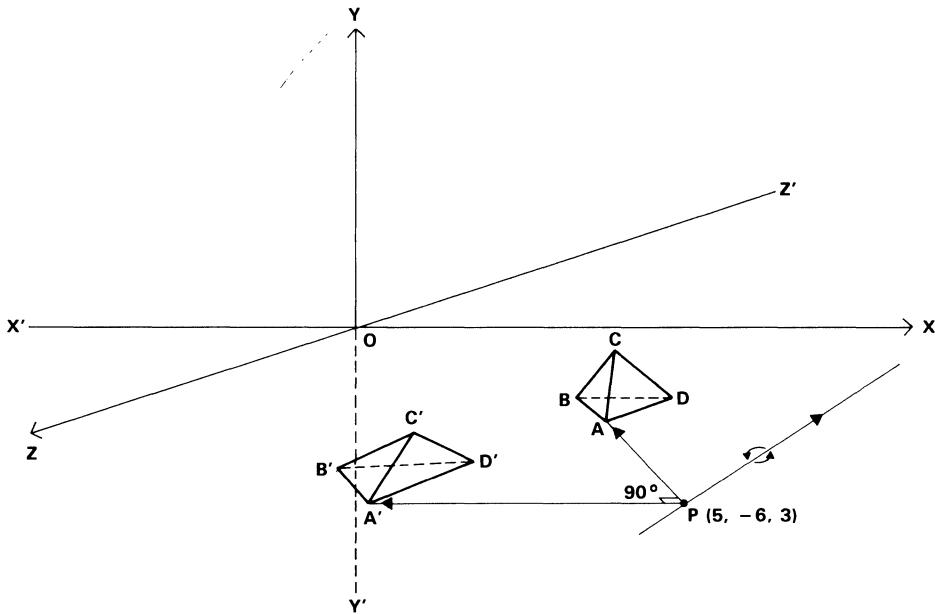


Figure 73. Resultant Matrix Transformation

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, x_A and y_A .

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:

$$\begin{array}{lll}
 s1 = p1 + 0 & s5 = p5 + p7 & s9 = p10 + p12 \\
 s2 = p1 + p3 & s6 = p6 + p8 & x_A = p1 + p2 + p3 + p4 \\
 s3 = p2 + p4 & s7 = p9 + 0 & y_A = p5 + p6 + p7 + p8 \\
 s4 = p5 + 0 & s8 = p9 + p11 &
 \end{array}$$

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	
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:

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

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

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:

$$\begin{aligned} T_{r+1}(x) - 2xT_r(x) + T_{r-1}(x) &= 0, \\ T_0(x) &= 1, \\ T_1(x) &= x \end{aligned}$$

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,

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

7

Rearranging the expression, by grouping powers of x ,

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

SN74ACT8847

Next make the following substitutions:

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

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 comparison 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 relevant '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 ← Cseries_cos
← (((((c8*x + c7)*x + c6)*x + c5)*x +
c4)*x + c3)*x + c2)*x + c1)*x + c0
```

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 $\leftarrow X * (2.0/\pi)$	2.0/pi entered as a constant
T2 $\leftarrow T1 + 2.0$	
T3 $\leftarrow 0.25 * T2$ and	CREG $\leftarrow T1$, T3 and T4 result
T4 $\leftarrow 1.0 - CREG$	from a chained instruction
T5 $\leftarrow \text{INT}(T3)$	round controls set to truncate
T6 $\leftarrow 4 * T5$	CREG $\leftarrow T4$
T7 $\leftarrow \text{DOUBLE}(T6)$	convert from integer to double
T8 $\leftarrow T7 + CREG$	
CMP (1.0, T8)	
If (1.0 > T8)	CREG $\leftarrow T8$
Then T9 $\leftarrow 2.0 - CREG$	T9 is X3 in Step 1, must
Else T9 $\leftarrow CREG$	be stored externally
	CREG $\rightarrow T9$
T10 $\leftarrow CREG * CREG$	
T11 $\leftarrow T10 * 2.0$	
T12 $\leftarrow T11 - 1.0$	T12 is X4 in Step 1, the input to the core routine

Step 2 perform the core calculation:

T13 $\leftarrow c_8 * CREG$	
T14 $\leftarrow T13 + c_7$	CREG $\leftarrow T12$
T15 $\leftarrow T14 * CREG$	
T16 $\leftarrow T15 + c_6$	
T17 $\leftarrow T16 * CREG$	
T18 $\leftarrow T17 + c_5$	
T19 $\leftarrow T18 * CREG$	
T20 $\leftarrow T19 + c_4$	
T21 $\leftarrow T20 * CREG$	
T22 $\leftarrow T21 + c_3$	
T23 $\leftarrow T22 * CREG$	
T24 $\leftarrow T23 + c_2$	
T25 $\leftarrow T24 * CREG$	
T26 $\leftarrow T25 + c_1$	
T27 $\leftarrow T26 * CREG$	
T28 $\leftarrow T27 + c_0$	

Step 3 perform the postprocessing:

Cosine(X) $\leftarrow T28 * T9$

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.

```
c8 = 3D19D46B7D4C8F32  
c7 = BD962909C5C01ED6  
c6 = 3E0D53517735F927  
c5 = BE7CC930FD0ADA9D  
c4 = 3EE3E0AF61F7677F  
c3 = BF41E5FDEF25C403  
c2 = 3F92A9FB40C119ED  
c1 = BFD23B03366AA0C9  
c0 = 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 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	c ₆	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	P14	Output MSH of answer
46			2.0	S6	0	NOP			P14		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 $1/2 \pi$.

F	3FF921FB	54442D18	F	0	0	_	2	0	3	FF	1	1	1	0	1C0	0	0	0	0	3	3	1	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
F	3FF00000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	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
F	3FD00000	00000000	F	0	1	_L	2	1	3	BD	1	1	0	0	581	0	0	1	0	3	3	1	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
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	1A3	1	0	0	0	3	3	1	0	0
F	00000000	00000004	F	0	1	_L	2	0	1	BF	1	1	0	0	240	0	0	0	0	3	3	1	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
F	00000000	00000000	F	0	0	_	2	1	3	F6	1	1	1	0	180	0	0	0	0	3	3	1	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
F	00000000	00000000	F	0	0	_L	2	0	3	FF	1	1	0	0	300	0	0	0	0	3	3	1	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
F	00000000	00000000	F	0	0	_	2	1	3	5F	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0
F	40000000	00000000	F	0	0	_L	2	0	3	EF	1	1	0	0	1C0	0	0	0	0	3	3	1	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
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	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
F	3D19D46B	7D4C8F32	F	0	1	_	2	1	3	BF	1	1	1	0	1C0	0	0	0	0	3	3	1	0	0
F	00000000	00000000	F	0	0	_L	2	0	3	FB	1	1	0	0	180	0	0	0	0	3	3	1	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

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

P A	D A	D B	P B	E N	C N	P L	C K	C P	S N	Ā L	H K	E L	F C	I O	R S	F C	S T	B L	S S	T L	S Y	S L	ĀĀĀĀ		
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	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 ← Cseries_sin
← (((((c8*x*x + c7)*x + c6)*x + c5)*x +
c4)*x + c3)*x + c2)*x + c1)*x + c0
```

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 ← c8*CREG          CREG ← T11
T13 ← T12 + c7
T14 ← T13*CREG
T15 ← T14 + c6
T16 ← T15*CREG
T17 ← T16 + c5
T18 ← T17*CREG
T19 ← T18 + c4
T20 ← T19*CREG
T21 ← T20 + c3
T22 ← T21*CREG
T23 ← T22 + c2
T24 ← T23*CREG
T25 ← T24 + c1
T26 ← T25*CREG
T27 ← T26 ← c0
```

Step 3 perform the postprocessing:

```
Sine(X) ← 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.

7

Listing of the Chebyshev Constants (c's)

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

```
c8 = 3D19D46B7D4C8F32
c7 = BD962909C5C01ED6
c6 = 3E0D53517735F927
c5 = BE7CC930FD0ADA9D
c4 = 3EE3E0AF61F7677F
c3 = BF41E5FDEF25C403
c2 = 3F92A9FB40C119ED
c1 = BFD23B03366AA0C9
c0 = 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	c ₃	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.

P	D	D	P	E	E	C	P	C	C	S	\bar{R}	\bar{H}	E	F	I	R	F	S	B	S	T	S	\bar{O}	\bar{O}	\bar{O}		
A	A	B	B	N	N	L	I	L	O	E	E	A	N	L	N	N	A	R	Y	E	E	E	E	E	E		
							A	B	K	P	K	N	L	S	L	C	O	S	D	S	C	T	L	S	L	Y	S
							C	E	M	F	O	E	T	W	T	T	C	E	S	T	Y						
							S	O	I	P	T		C	R		P	T										
							D	G																			
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	1AO	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)

P A	D A	D B	P B	E N	E N	C L	P K	C P	C K	S L	R S	H L	E E	F A	I N	R N	F A	S R	B Y	S Y	T E	T C	S T	S L	S Y	S C	S T	S L	S Y	S C	S T	S L	S Y	S C			
					</																																

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 $X5 \cdot X3$.

```
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 ← Cseries_tan
← (((((((((c14)*x + c13)*x + c12)*x + c11)*x + c10)*x +
c9)*x + c8)*x + c7)*x + c6)*x + c5)*x + c4)*x + c3)*x +
c2)*x + c1)*x + c0
```

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

7

SN74ACT8847

T9 \leftarrow CREG * CREG
T10 \leftarrow T9 * 2.0
T11 \leftarrow T10 - 1.0

CREG \leftarrow T8

T11 is X4 in Step 1, the
input to the core routine

Step 2 perform the core calculation:

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

CREG \leftarrow T11

7

Step 3 perform the postprocessing:

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

SN74ACT8847

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.

```
c14 = 3D747D842210CC35  
c13 = 3DA1D66636043991  
c12 = 3DCCD078F52B3A73  
c11 = 3DF938F9CDDFF864  
c10 = 3E2620430E99B5B7  
c9 = 3E535C2C953CE515  
c8 = 3E80F07AFC099D7F  
c7 = 3EADA4D789EB45C4  
c6 = 3ED9F03D4C51A771  
c5 = 3F06B236DE4D014C  
c4 = 3F33DBFB01B3F415  
c3 = 3F6160DE701F3A53  
c2 = 3F8E70A18736FC10  
c1 = 3FBAEA2653199611  
c0 = 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 $1/3 \pi$.

F	3FF0C152	382D7365	F	0	0	_	2	0	3	FF	1	1	1	0	1C0	0	0	0	0	3	3	1	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
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	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
F	3FD00000	00000000	F	0	1	_	2	1	3	BF	1	1	0	0	1C0	0	0	1	0	3	3	1	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
F	00000000	00000000	F	1	0	_	2	0	3	FB	1	1	1	0	1A3	1	0	0	0	3	3	1	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
F	00000000	00000000	F	0	0	_	2	1	3	FB	1	1	1	0	1A2	0	0	0	0	3	3	1	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
F	00000000	00000000	F	0	0	_	2	1	3	FE	1	1	1	0	182	0	0	0	0	3	3	1	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
F	40000000	00000000	F	0	1	_	2	1	3	F7	1	1	1	0	183	0	0	0	0	3	3	1	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
F	40000000	00000000	F	0	0	_	2	0	3	EF	1	1	0	0	1C0	0	0	0	0	3	3	1	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
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	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
F	3D747D84	2210CC35	F	0	1	_	2	1	3	BF	1	1	1	0	1C0	0	0	0	0	3	3	1	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
F	3DA1D666	36043991	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	0	0

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

P A	D A	D B	P B	E N	E N	C L	P K	C K	C N	S L	R S	H L	E C	F O	I T	R D	F S	B C	S T	T L	S S	\bar{S} \bar{T}	\bar{S} \bar{L}	\bar{S} \bar{Y}	\bar{S} \bar{C}
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	OE99B5B7	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

SN74ACT8847

7

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

F	00000000	00000000	F	0	0	—	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	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
F	3F33DBFB	01B3F415	F	0	1	—	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	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
F	00000000	00000000	F	0	0	—	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	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
F	00000000	00000000	F	0	0	—	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	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
F	3F8E70A1	8736FC10	F	0	1	—	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	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
F	00000000	00000000	F	0	0	—	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	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
F	00000000	00000000	F	0	0	—	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	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
F	3FEC14B2	675B10BA	F	0	1	—	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	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
F	00000000	00000000	F	0	0	—	2	0	3	FF	1	1	1	0	180	0	0	0	0	3	3	1	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
F	3FF00000	00000000	F	0	0	—	2	0	3	FF	1	1	1	0	1E0	0	0	0	0	3	3	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
F	00000000	00000000	F	0	0	—	2	0	3	FF	1	1	1	0	300	0	0	0	0	3	3	1	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
F	00000000	00000000	F	0	0	—	2	0	3	FF	1	1	1	0	300	0	0	0	0	3	3	1	0	0

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

P A	D A	D B	P B	E N	E N	C L	P K	C N	C L	S L	\bar{R} C	\bar{H} O	E E	F A	I N	R N	F A	S R	B S	S T	T S	\bar{O} \bar{O}	\bar{O} \bar{O}			
F	00000000	F	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	F	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	F	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	F	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	F	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	F	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	F	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	F	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	F	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	F	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	F	00000000	F	0	0	—	2	0	3	FF	1	1	1	0	300	0	0	0	0	3	3	1	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.

$$X2 \leftarrow C_{\text{series_asin\&acos}}$$

$$\begin{aligned} &\leftarrow (((((((((c18*x + c17)*x + c16)*x + \right. \\ &c15*x + c14)*x + c13)*x + c12)*x + c11)*x + c10)*x + \\ &c9)*x + c8)*x + c7)*x + c6)*x + c5)*x + c4)*x + c3)*x + \\ &c2)*x + c1)*x + c0 \end{aligned}$$

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

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

$$X3 \leftarrow X2 * \text{SQRT}(2.0)$$

$$\text{ArcSine}(X) \leftarrow X3 * X$$

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

Algorithms for the Three Steps

7

Step 1 perform the preprocessing:

$$\begin{aligned} T1 &\leftarrow X * X \\ T2 &\leftarrow 4.0 * T1 \\ T3 &\leftarrow T2 - 1 \end{aligned}$$

T3 is X1 in Step 1, the input to the core routine

SN74ACT8847

Step Two perform the core calculation:

```
T4 ← c18*CREG  
T5 ← T4 + c17  
T6 ← T5*CREG  
T7 ← T6 + c16  
T8 ← T7*CREG  
T9 ← T8 + c15  
T10 ← T9*CREG  
T11 ← T10 + c14  
T12 ← T11*CREG  
T13 ← T12 + c13  
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  
CREG ← T3
```

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.

```
c18 = 3DA4A49F8CCD9E73  
c17 = 3DC05DFE52AAD200  
c16 = 3DCCF31E26F94C8D  
c15 = 3DE86CDA3C8CAEB0  
c14 = 3E0768D9F4E950EA  
c13 = 3E2383A37598FC80  
c12 = 3E403E4B2F65F0DE  
c11 = 3E5BAFC8245ABDF8  
c10 = 3E77E3333AFF1AB4  
c9 = 3E94E3A4D4220C9C  
c8 = 3EB296DD4C084ACB  
c7 = 3ED0E913F5F9D496  
c6 = 3EEFA74E896F8FA8  
c5 = 3FOEC76B7832DBB6  
c4 = 3F2F978698C8B2E4  
c3 = 3F519B1087542073  
c2 = 3F7696895FFC05A0  
c1 = 3FA375CA61D2988C  
c0 = 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				

SN74ACT8847



SN74ACT8847

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	c4	1	SR49*CR49					S15		
50			4.0	c4	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 overwriting 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

SN74ACT8847



SN74ACT8847

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 $1/(SQRT(2.0))$.

P	D	D	P	E	E	C	P	C	C	S	R	H	E	F	I	R	F	S	B	S	T	S	\bar{O}	\bar{O}	\bar{O}	
A	A	B	B	N	N	L	I	L	O	E	E	A	N	L	N	N	A	R	Y	E	E	E	E	E	E	
				A	B	K	P	K	N	L	S	L	C	O	S	D	S	C	T	L	S	L	Y	S	C	
				C	E	M	F	O	E	T	W	T	T	C	E	S	T	Y								
				S	O	I	P	T		C	R			P	T											
				D	G																					

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)

P A	D A	D B	P B	E N	E N	C L	C K	C P	S L	R I	H L	E E	F C	I N	R N	F A	S R	B Y	S Y	T C	T C	S L	S T	O L	O Y	O S	O C	
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 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)

F	3F0EC76B	7832DBB6	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	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
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	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
F	00000000	00000000	F	0	0	_	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	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
F	3F519B10	87542073	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	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
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	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
F	00000000	00000000	F	0	0	_	2	1	3	9F	1	1	1	0	1C0	0	0	0	0	3	3	1	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
F	3FA375CA	61D2988C	F	0	1	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	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
F	00000000	00000000	F	0	0	_	2	0	3	FB	1	1	1	0	180	0	0	0	0	3	3	1	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
F	3FE6A09E	667F3BCD	F	0	1	_	2	1	3	BF	1	1	1	0	1C0	0	0	0	0	3	3	1	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
F	00000000	00000000	F	1	0	_	2	0	3	EF	1	1	1	0	1C0	0	0	0	0	3	3	1	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
F	00000000	00000000	F	0	0	_	2	0	3	FF	1	1	1	0	300	0	0	0	0	3	3	1	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
F	00000000	00000000	F	0	0	_	2	0	3	FF	1	1	1	0	300	0	0	0	0	3	3	1	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

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 ← 1.0/X
  Else X1 ← X
X2 ← 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 ← Cseries_atan
  ← (((((((((c19*x + c18)*x + c17)*x + c16)*x + c15)*x +
    c14)*x + c13)*x + c12)*x + c11)*x + c10)*x + c9)*x
    + c8)*x + c7)*x + c6)*x + c5)*x + c4)*x + c3)*x + c2)*x
    + c1)*x + c0
```

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 ← X3*X1
If X > 1.0
  Then ArcTangent(X) ← pi/2 - X4
Else If X < -1.0
  Then ArcTangent(X) ← -pi/2 - X4
  Else ArcTangent(X) ← X4
```

Algorithms for the Three Steps

Step 1 perform the preprocessing:

```
If |X| > 1.0
    Then T1 ← 1.0/X
        T2 ← T1 * 2.0
        T3 ← T2 * CREG
        T4 ← T3 - 1.0
Else T1 ← X
    T2 ← T1 * 2.0
    T3 ← T2 * T1
    T4 ← T3 - 1.0
```

T1 is X1 in Step 1, must be stored externally
CREG ← T1

Step 2 perform the core calculation:

```
T5 ← c19 * CREG
T6 ← T5 + c18
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 + c5
T33 ← T32*CREG
T34 ← T33 + c4
T35 ← T34*CREG
T36 ← T35 + c3
T37 ← T36*CREG
T38 ← T37 + c2
T39 ← T38*CREG
T40 ← T39 + c1
T41 ← T40*CREG
T42 ← T41 + c0

```

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 preprocessing 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.

c₁₉ = BDC4D6CC6308553F
c₁₈ = 3DDFFD56FCFD2315
c₁₇ = BDE880782D99D071
c₁₆ = 3E0409670CB71218
c₁₅ = BE237C8239249B77
c₁₄ = 3E3F1358EC1D6AC0
c₁₃ = BE587CD25F4AFBED
c₁₂ = 3E73D2388B0B8A86
c₁₁ = BE9028E921CA6A94
c₁₀ = 3EAA814997A38D4E
c₉ = BEC5EDAD9A21FE5F
c₈ = 3EE256E57BA07FAE
c₇ = BEFF171F48FDF707
c₆ = 3F1ACFA9F95CA0DF
c₅ = BF37A8464221D994
c₄ = 3F558DF7A83283C9
c₃ = BF749B3E2E433683
c₂ = 3F955A300BFB8078
c₁ = BFBA1494C19FADD4
c₀ = 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				In cycles 16a and 17 a output P1 and store it for use in cycle 79
19a			2.0	X	0	CR19*PR19			P2a				
16b	2.0 MSH	2.0 LSH	1.0	X	0	RA17*RB17							If the reciprocal of X was

SN74ACT847

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 cycle 16b through 19b
18b	X MSH	X LSH	X	X	0	RA19*PR19	RA17*RB17						
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	c14	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).

P	D	D	P	E	E	C	P	C	C	S	\bar{R}	\bar{H}	E	F	I	R	F	S	B	S	T	S	\bar{O}	\bar{O}	
A	A	B	B	N	N	L	I	L	O	E	E	A	N	L	N	N	A	R	Y	E	E	E	E	E	
				A	B	K	P	K	N	L	S	L	C	O	S	D	S	C	T	L	S	L	Y	S	
				C	E	M	F	O	E	T	W	T			T	C	E	S	T	Y					
				S	O	I	P	T		C	R				P	T									
				D	G																				
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	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	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)

P	D	D	P	E	E	C	P	C	C	S	\bar{R}	\bar{H}	E	F	I	R	F	S	B	S	T	S	\bar{O}	\bar{O}	\bar{O}	
A	A	B	B	N	N	L	I	L	O	E	E	A	N	L	N	N	A	R	Y	E	E	E	E	E	E	
				A	B	K	P	K	N	L	S	L	C	O	S	D	S	C	T	L	S	L	Y	S	C	
				C	E	M	F	O	E	T			W	T		T	C	E	S	T	Y					
				S	O	I	P	T			C	R				P	T									
				D	G																					
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	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	3E040967	OCB71218	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)

P A	D A	D B	P B	E N	E N	C L	P K	C N	C L	S L	R C	H O	E S	F O	I T	R D	F S	S C	B T	S T	T S	S L	S Y	\bar{O}	\bar{O}	\bar{O}
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	00000000	00000000	F	0	0	_	2	1	3	9F	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)

P	D	D	P	E	E	C	P	C	C	S	\bar{R}	\bar{H}	E	F	I	R	F	S	B	S	T	S	\bar{O}	\bar{O}	\bar{O}
A	A	B	B	N	N	L	I	L	O	E	E	A	N	L	N	N	A	R	Y	E	E	E	E	E	E
				A	B	K	P	K	N	L	S	L	C	O	S	D	S	C	T	L	S	L	Y	S	
				C	E	M	F	O	E	T	W	T	T	C	E	S	T	Y							
				S	O	I	P	T		C	R			P	T										
				D	G																				
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	OBFB8078	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 ← X*log2e  
X2 ← TRUNC(X1)  
EX ← 1024 + X2  
N ← EX - 1023  
X3 ← DOUBLE(N)  
X4 ← 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 ← Cseries_exp  
← (((((((c11*x + c10)*x + c9)*x + c8)*x + c7)*x + c6)*x +  
c5)*x + c4)*x + c3)*x + c2)*x + c1)*x + c0
```

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 ← EX SLL by 20 bits  
Y bus ← X6  
DA bus ← Y bus  
Exp(X) ← X5 * X6
```

Algorithms for the Three Steps

Step 1 perform the preprocessing:

T1 $\leftarrow X \cdot \log_2 e$	log ₂ e entered as a constant
T2 $\leftarrow \text{INT}(T1)$	round controls set to truncate
T3 $\leftarrow 1024 + T2$	T3 is EX in Step 1, must be stored externally, CREG $\leftarrow T1$
T4 $\leftarrow T3 - 1023$	makes T4 available to A2 MUX
T5 $\leftarrow 1 \cdot T4$	convert from integer to double
T6 $\leftarrow \text{DOUBLE}(T5)$	
T7 $\leftarrow T6 - \text{CREG}$	
T8 $\leftarrow 2.0 \cdot T7$	
T9 $\leftarrow T8 - 1.0$	T9 is X4 in Step 1, the input to the core routine

Step 2 perform the core calculation:

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

Step 3 perform the postprocessing:

T32 \leftarrow T3 SLL by 20 bits
Y bus \leftarrow T32

DA bus \leftarrow Y bus (= T32)

Exp(X) \leftarrow T32*CREG

Shift T3 20 bits left
Output and then Input T32

CREG \leftarrow T31

Two cycles required to
input both halves of T32

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.

c₁₁ = BD45A7FC05D3B501
c₁₀ = 3D957BFD2DBF487C
c₉ = BDE351B821AC16D5
c₈ = 3E2F5B0E17440879
c₇ = BE769E51EE631E87
c₆ = 3EBC8D7530548DD5
c₅ = BEFEE4FD234A4926
c₄ = 3F3BDB696E8987AC
c₃ = BF741839EB88156E
c₂ = 3FA5BE298ADF0369
c₁ = BFCF5E46537AB906
c₀ = 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 ₂ e MSH	Log ₂ e LSH	X	Log ₂ e	0	RA2*RB2							
3			X	Log ₂ e	0	DP2I(PR4)	RA2+RB2						Double-precision → integer
4			X	Log ₂ e	0	DP2I(PR4)			P1				
5	1024		1024	Log ₂ e	0	RA5+SR5				P1	S1		
6	-1023		-1023	Log ₂ e	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						

SN74ACT8847



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	c1	0	PR45 + RB45	SR43*CR43						
45	c0 MSH	c0 LSH	-1023	c0	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.

P	D	D	P	E	E	C	P	C	C	S	\bar{R}	\bar{H}	E	F	I	R	F	S	B	S	T	S	\bar{O}	\bar{O}	\bar{O}			
A	A	B	B	N	N	L	I	L	O	E	E	A	N	L	N	N	A	R	Y	E	E	E	E	E	E			
							A	B	K	P	K	N	L	S	L	C	O	S	D	S	C	T	L	S	L	Y	S	C
							C	E	M	F	O	E	T	W	T	T	C	E	S	T	Y							
							S	O	I	P	T		C	R			P	T										
							D	G																				
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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	0		

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

P A	D A	D B	P B	E N	E N	C L	P K	C P	C K	S L	R S	H E	E A	F N	I N	R N	F A	S R	B Y	S E	T E	T E	S E	O E	O E	O E		
				A	B	K	P	K	N	L	S	L	C	O	S	D	S	C	T	L	S	L	Y	S	C			
				C	E	M	F	O	E	T	W	T	T	C	E	T	C	E	S	T	Y							
				S	O	I	P	T		C	R			P	T													
				D	G																							
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	3EBBC8D75	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	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	00000000	00000000	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	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	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	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	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	00000000	00000000	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)

P	D	D	P	E	E	C	P	C	C	S	\bar{R}	\bar{H}	E	F	I	R	F	S	B	S	T	S	\bar{O}	\bar{O}	\bar{O}	
A	A	B	B	N	N	L	I	L	O	E	E	A	N	L	N	N	A	R	Y	E	E	E	E	E	E	
				A	B	K	P	K	N	L	S	L	C	O	S	D	S	C	T	L	S	L	Y	S		
				C	E	M	F	O	E	T	W	T	T	C	E	S	T	Y								
				S	O	I	P	T		C	R				P	T										
				D	G																					

F	00000409	000000014	F	1	1	—	2	0	1	FF	1	1	1	0	228	0	0	0	3	3	1	0	0	0
F	00000000	000000000	F	0	1	—	2	0	3	FF	1	1	0	0	300	0	0	0	3	3	1	0	0	0
F	40900000	000000000	F	0	0	—	2	0	2	DF	1	1	1	0	1C0	0	0	0	3	3	1	0	0	0
F	00000000	000000000	F	1	0	—	2	0	2	DF	1	1	1	0	1C0	0	0	0	3	3	1	0	0	0
F	00000000	000000000	F	0	0	—	2	0	3	FF	1	1	1	0	180	0	0	0	3	3	1	0	0	0
F	00000000	000000000	F	0	0	—	2	0	3	FF	1	1	1	0	300	0	0	0	3	3	1	0	0	0
F	00000000	000000000	F	0	0	—	2	0	3	FF	1	1	1	0	300	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 ("unpipled" 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.

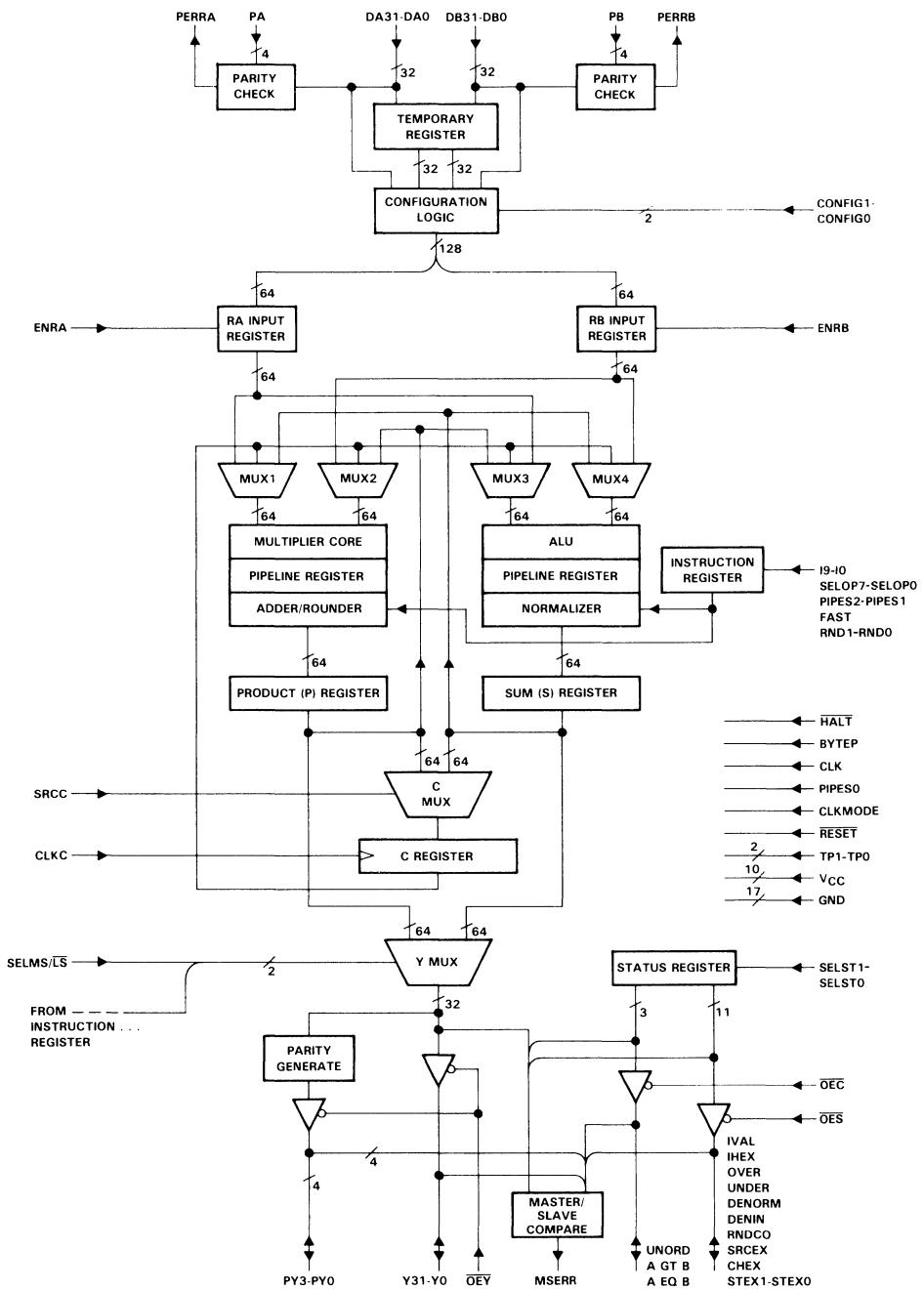


Figure 74. SN74ACT8837 Floating Point Unit

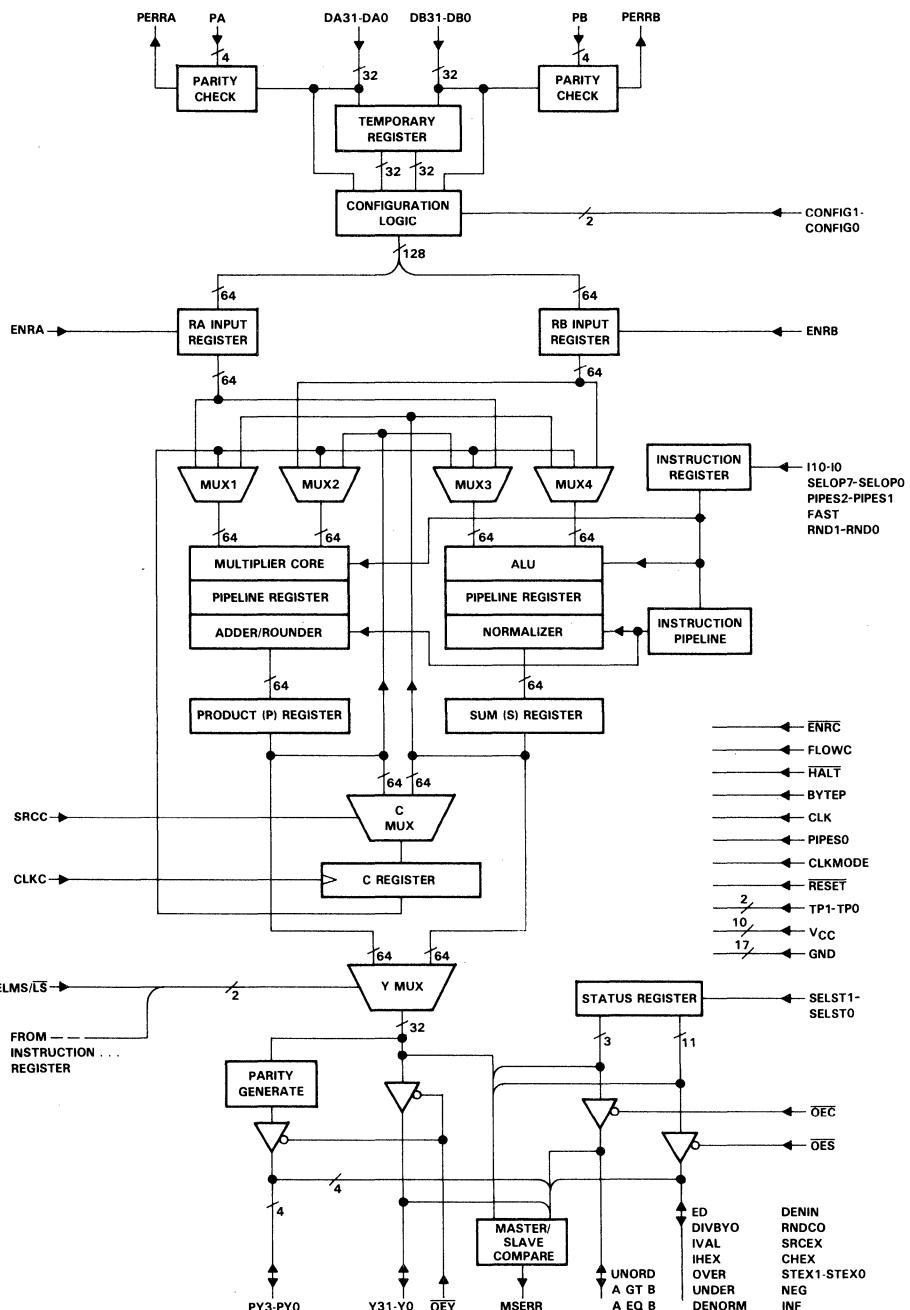


Figure 75. SN74ACT8847 Floating Point Unit

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:

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

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

Graphics Applications

7

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.

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

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

7

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,

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

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;	$Y \leftarrow S$	ADD(RA,RB)	
2.	LOAD RA, RB;	$Y \leftarrow S$	ADD(RA,RB)	
3.	LOAD RA, RB;	$Y \leftarrow S$	ADD(RA,RB)	
.	.	.		
6.	LOAD RA, RB;	$Y \leftarrow S$	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			$A1 \times B1$	$A2 \times B2$	$A3 \times B3$	$A4 \times B4$	$A5 \times B5$	$A6 \times B6$
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 \leftarrow P		MULT(RA,RB)
2. LOAD RA, RB; Y \leftarrow P		MULT(RA,RB)
3. LOAD RA, RB; Y \leftarrow P		MULT(RA,RB)
.		
6. LOAD RA, RB; Y \leftarrow 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 \times B1	A2 \times B2	A3 \times B3	A4 \times B4	A5 \times B5	A6 \times 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 $S_j = P_j + S_{j-2}$ (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.

7

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. Similarly, 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	C ← P	MULT(RA,RB)
2.	LOAD RA, RB		MULT(RA,RB)
3.	LOAD RA, RB		MULT(RA,RB)
4.	LOAD RA, RB;		MULT(RA,RB)
5.	LOAD RA, RB		MULT(RA,RB)
6.	LOAD RA, RB		MULT(RA,RB)
7.	LOAD RA, RB		MULT(RA,RB)
8.	LOAD RA, RB		MULT(RA,RB)
9.			MULT(RA,RB)
10.			MULT(RA,RB)
11.			
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.

Unpipel 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 Unpipel 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 'ACT8847 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.

SN74ACT8837 Vector Divide

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		$B \times X0$		X0		X1		$B \times X1$		
C										
Y										
CLK	1	2	3	4	5	6	7	8	9	10

RA		B								
RB				A						
S	T1			X2		T2				
P	X1		X2	$B \times X2$	$A \times X2$			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.	$Y \leftarrow P$		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_1 , and the approximation for $1/D$ is denoted by X_1 . $T_i = (2 - B \times W_i)$, 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			$B \times W_0$	$D \times X_0$	W0	X0	W1	X1	$B \times W_1$	$D \times X_1$
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	$B \times W_2$	$D \times X_2$	$A \times W_2$	$C \times X_2$	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 $[12 \times \text{CEILING}(N/2) + 2]$ 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		$B \times X0$	X0	X1	$B \times X1$	X1	X2	$B \times X2$	$A \times X2$
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 \leftarrow P . .		DIVIDE .
7. LOAD RA, RB; Y \leftarrow P . .		DIVIDE .
13. LOAD RA, RB; Y \leftarrow 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 $[6(N \times M) + 2]$ cycles in pipelined mode. In unpiped mode, the equation is $7(N \times M)$.

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,

$$Y_i = \text{MAX/MIN}(A_i, B_i), \quad 1 \leq i \leq N.$$

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_j) to output. Similarly, if A ≤ B, AGTB and I6 are low, and the ALU result (B_j) is output. The circuitous route taken by A_j 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_j = MAX/MIN(A_j, B_j) and Z_j = MAX/MIN(C_j, D_j).

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_j (and C_j) 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 $[6 \times N \times \text{CEILING}(M/2) + 4]$ 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 $\text{CLK} = 2, 5, \text{etc.}$, and used at $\text{CLK} = 3, 6, \text{etc.}$, as the I6 bit of the instruction. I6 selects either the multiplier or ALU result to output to the Y bus at $\text{CLK} = 4, 7, \text{etc.}$

The program computes the vector comparison of M pairs of vectors of length N in $[3 \times M \times (N + 1)]$ 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,

$$Z = \text{MAX/MIN}(A_i), \quad 1 \leq i \leq N.$$

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 $\text{CLK} = 4N - 2$, 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. $C \leftarrow P/S$		
6. LOAD RA	COMPARE(RA,C)	
7. LOAD RA	ADD(C,0)	MULT(RA,1)
8.		
9. $C \leftarrow P/S$		
REPEAT STEPS 6 THROUGH 9 UNTIL STEP $4N-2$ IS REACHED, THEN:		
$4N - 2$	$Y \leftarrow S$	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.

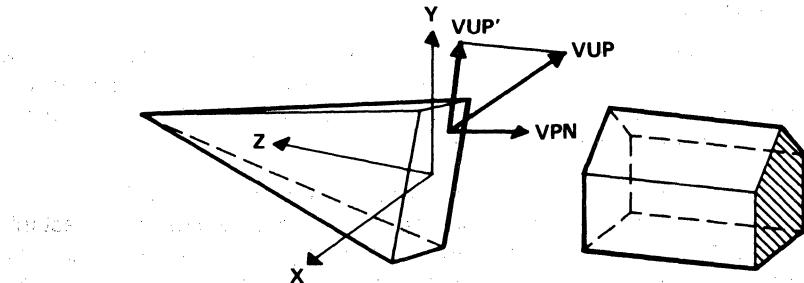


Figure 76a. In a sequence of transformations, the world coordinate positions for the house are transformed into the normalized viewing coordinate system (also called the eye system). For clarity, the house is pictured outside the view column. Also shown are the direction vectors VUP (view up), VPN (view normal), and VUP' (the projection of VUP parallel to VUN onto the view plane).

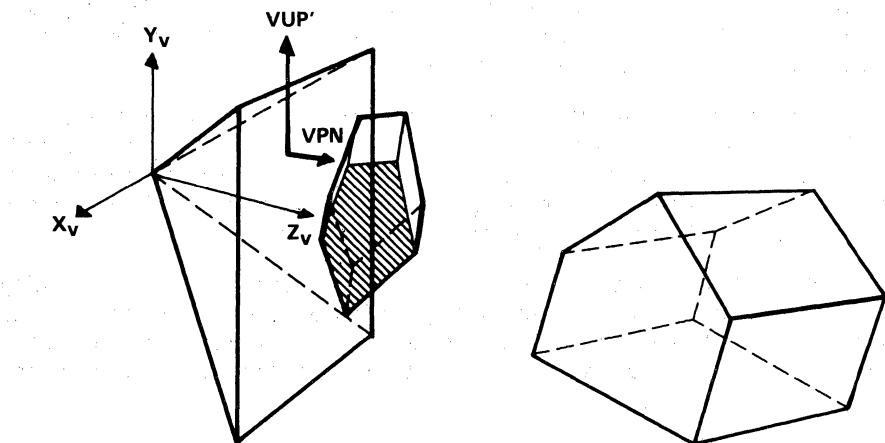


Figure 76b. After a series of translations, rotations, and shearing and scaling operations, the view volume becomes the canonical perspective projection view volume, which is a truncated pyramid with apex at the origin, and the house has been transformed from the world to the viewing coordinate system.

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."

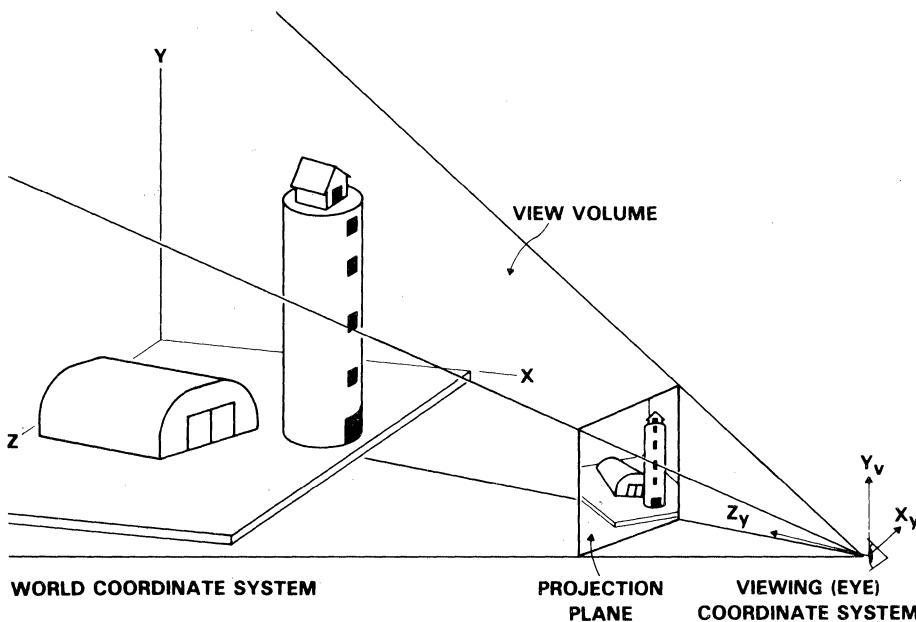


Figure 77. View Volume

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."

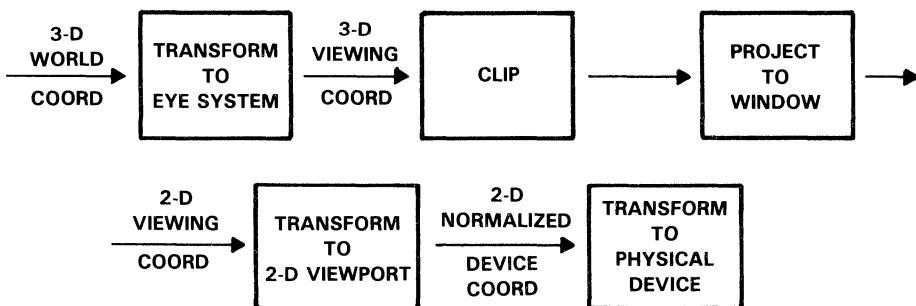


Figure 78a. Model of Procedure for Creating a 3-D Graphic

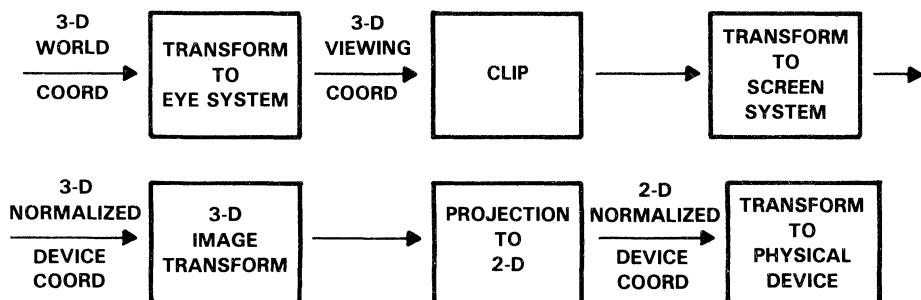


Figure 78b. Model for Creating and Transforming a 3-D Image

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:

$$[X'_h, Y'_h, Z'_h, W'_h] = [X_h, Y_h, Z_h, W_h] \times \begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{bmatrix}$$

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:

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

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	X	Y	Z	X	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 $1 \leq i \leq 3$, $P1(i) = X \times A_{1i}$, $P2(i) = Y \times A_{2i}$, etc. $S1(i) = P1(i) + A4i$, $S2(i) = P2(i) + P3(i)$, 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.

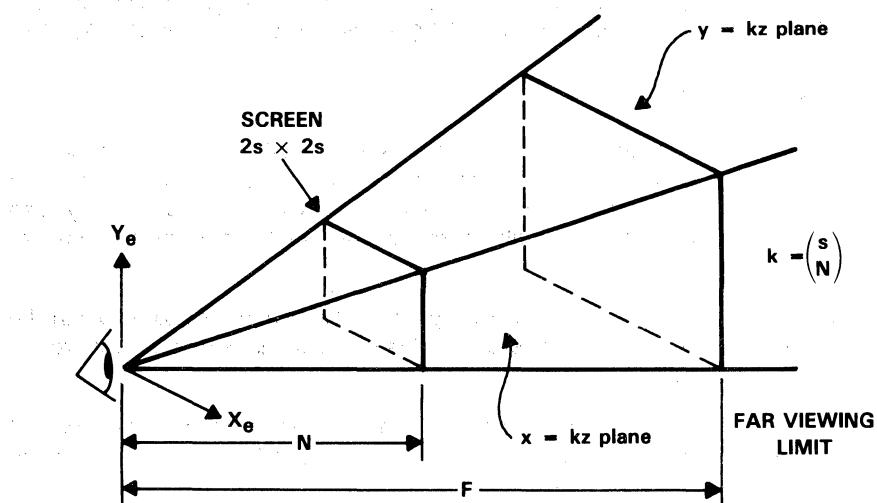


Figure 79. Viewing Pyramid Showing Six Clipping Planes

Consider the case of clipping the line defined by the points $P_1 = (X_1, Y_1, Z_1)$ and $P_2 = (X_2, Y_2, Z_2)$ against the $Z = N$ plane. First computed are $(Z_1 - N)$ and $(Z_2 - N)$. 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 Z_1Z_2 , and therefore of P_1P_2 as a whole, lies on the P_1 side of the $Z = N$ plane, is computed as follows:

$$t = (Z_1 - N)/(Z_1 - Z_2).$$

In general, the value of the parameter is derived as described in Newman and Sproull,¹ using the following equations of the line: $X = X_1 + (X_2 - X_1)u$; $Y = Y_1 + (Y_2 - Y_1)u$; $Z = Z_1 + (Z_2 - Z_1)u$. These equations are each inserted into the corresponding plane equation. In the current example, $N = Z_1 + (Z_2 - Z_1)t$.

Since N is between Z_1 and Z_2 , t is always positive, and the signs of $Z_1 - N$ and $Z_2 - N$ are used to determine which end to clip. If $Z_1 - N$ is negative, the P_1 end is clipped, using the value of t to determine the delta in X_1 and Y_1 . The coordinates for the new endpoint of the shortened line segment are given by:

$$X_1' = X_1 + (X_2 - X_1) \times t, Y_1' = Y_1 + (Y_2 - Y_1) \times t, Z_1' = N.$$

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 $Z_1 < N$, $t_1 = (Z_1 - N)/(Z_1 - Z_2)$; otherwise, $t_1 = 0$. If $Z_2 < N$, $t_2 = (Z_2 - N)/(Z_1 - Z_2)$; 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 $(Z_i - N)$ is negative, $H'_i = 2H_i = 2(Z_i - N)$; otherwise, $H'_i = 0$. Hence, in a straightforward manner, a suitable numerator for t_i has been computed, regardless of the sign of $(Z_i - N)$. This approach avoids resorting to an "if/then" decision to compute t_i .

To scale the denominator to the numerator, $D = 2(Z_1 - Z_2)$ 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 (X_1, Y_1, Z_2) and (X_2, Y_2, Z_2) 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, $t_1 = |(H_1 - |H_1|)|/D$. Also, $d = Z_1 - Z_2$, $H_1 = Z_1 - N$, $H'_1 = H_1 - |H_1|$, $H_2 = Z_2 - N$, $H'_2 = H_2 - |H_2|$, R_i = successive approximations for $1/d$, $T_i = (2 - d \times R_i)$, and $R_{(i+1)} = T_i \times R_i$.

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	17

CHAIN	Y	Y	Y	Y	Y	Y	Y	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)					
			xt1	xt1	xt1			xt1			xt2	xt2	xt2			
C		t1	t1	t2		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	ADD (P, RB)	MULT(RA, C)
24. LOAD RB Y←S	ADD (P, RB)	MULT(RA, C)
25. LOAD RB Y←S	ADD (P, RB)	MULT(RA, C)
26. Y←S	ADD (P, RB)	
27.		
28.		

In pipelined mode, computing $(Z_1 - Z_2)$ takes 2 cycles. This value is passed off-chip and used to get the first approximation to $0.5/(Z_1 - Z_2)$ 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 H_1' and H_2' are interleaved with the divide algorithm and are completed before it.

$(X_2 - X_1)$, $(Y_2 - Y_1)$, and $(Z_2 - Z_1)$ are also computed during the divide. The values of t_1 and t_2 are ready in steps 18 and 19. New values of X_1 , X_2 , Y_1 , Y_2 , Z_1 , and Z_2 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, (X_1', Y_1', Z_1') and (X_2', Y_2', Z_2') , 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	$X_2 - X_1$	H_1'	H_2'			$Y_2 - Y_1$			
P										1/D		t1	t2	
C					H1	H2					1/D	t1		STEPS 20 THRU 28
Y			d			$X_2 - X_1$	H_1'	H_2'			$Y_2 - Y_1$			
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.

7

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 $P_1 = (X_1, Y_1, Z_1)$ and $P_2 = (X_2, Y_2, Z_2)$, to perform the entire clipping operation on all six planes at once, the following two six-tuples must be computed:

$$Q = (W_1 + X_1, W_1 - X_1, W_1 + Y_1, W_1 - Y_1, Z_1 - N, F - Z_1) = (Q_1, Q_2, \dots),$$

$$R = (W_2 + X_2, W_2 - X_2, W_2 + Y_2, W_2 - Y_2, Z_2 - N, F - Z_2) = (R_1, R_2, \dots),$$

where $W_1 = KZ_1$ and $W_2 = KZ_2$.

Consider the case where $X_1 < -W_1$. Then, $W_1 + X_1 < 0$; i.e., $Q_1 < 0$. In general, a negative element of Q indicates that P_1 is on the invisible side of one of the clipping planes, while a negative element of R indicates the same for P_2 . To clip the line, the six parameters t_i for clipping the P_1 end and the six parameters s_j for clipping the P_2 end are computed. Here, $t_{j20} = 20Q_j/(Q_j - R_j)$ and $s_j = R_j/(R_j - Q_j)$. (Again, the equations of the line as described in Newman and Sproull are used).

For example, to find the value t_1 for clipping P_1 to the $X = -W = -KZ$ plane, the following equation is used:

$$X_1 + (X_2 - X_1)t_1 = -K[Z_1 + (Z_2 - Z_1)t_1].$$

Solving for t_1 ,

$$t_1 = (X_1 + W_1)/[(X_1 + W_1) - (X_2 + W_2)] = Q_1 / (Q_1 - R_1).$$

In general, $t_i = Q_i/(Q_i - R_i)$. Similarly, $s_j = R_j/(R_j - Q_j)$.

To actually carry out the computations of t_i and s_j , 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,

$$Q'_i = 2 \times Q_i \text{ if } Q_i < 0, \text{ and } Q'_i = 0 \text{ otherwise.}$$

$$R'_i = 2 \times R_i \text{ if } R_i < 0, \text{ and } R'_i = 0 \text{ otherwise.}$$

Next calculated is $t_i = Q'_i/[2(Q_i - R_i)]$ and $s_j = R'_j/[2(R_j - Q_j)]$, followed by $T_1 = \text{MAX}(t_i)$ and $T_2 = 1 - \text{MAX}(s_j)$. The P_1 end is clipped using T_1 and the P_2 end is clipped using T_2 .

In an '8837 three-processor parallel system, in which each processor is given the task of computing two t_i and two s_j values, computing the Q'_i and R'_j values takes 14 cycles, with the values of $Q_i - R_j$ computed by step 13. The six divides, $0.5/(Q_i - R_j)$, 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 $1/D2 = 0.5/d2$, $d_i = Q_i - R_i$.

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.

7

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 .

SN74ACT8847

Table 89. A > B Comparison Function Table

Sign Q_j	Sign R_j	Sign $A = -Q_j \times R_j $	Sign $B = Q_j \times R_j$	$A > B$	$A = B$
-	-	+	-	T	F
-	+	+	+	F	T
+	-	-	-	F	T
+	+	-	+	F	F

The $A > B$ status provides the needed AND function of the sign bits of Q_j and R_j . 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 $A = (Q_j \times |R_j|)$ and $B = (|Q_j| \times R_j)$ gives the logical AND of the complement of the sign bits. It is TRUE when both Q_j and R_j 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 $W = K \times Z$. After $-W_1$ and $-W_2$ are computed, a sequence of comparison operations are performed, summarized as follows:

- with X_1 in RB and $-W_1$ in P, $P > RB$ (i.e., $-W_1 > X_1$)
- with X_1 in RA and $-W_1$ in C, $RA > |C|$ (i.e., $X_1 > W_1$)
- with Y_1 in RB and $-W_1$ in C, $C > RB$
- with Y_1 in RA, $RA > |C|$ comparison
- with Z_1 in RB and N in RA, $RA > RB$ (i.e. $N > Z_1$)
- with Z_1 in RA and F in RB, $RA > RB$ (i.e., $Z_1 > F$).

These six operations are carried out in successive cycles and then repeated for (X_2, Y_2, Z_2) . 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 $Z' = Z \times a \times (b + Z)$. (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, $Q = (1 + X_1, 1 - X_1)$ is computed and Q' is formed, where $Q'_j = Q_j - |Q_j|$. i.e.,

$$Q'_1 = 2(1 + X_1), \text{ if } (1 + X_1) < 0; Q'_1 = 0 \text{ otherwise.}$$

$$Q'_2 = 2(1 - X_1), \text{ if } (1 - X_1) < 0; Q'_2 = 0 \text{ otherwise.}$$

7

At least one of Q'_j will be zero; the other will be negative. Hence, $\text{MIN}(Q'_1, Q'_2) = Q'_1 + Q'_2 = [(1 + X_1) - |1 + X_1|] + [(1 - X_1) - |1 - X_1|]$. Therefore, $\text{MIN}(Q'_1, Q'_2) = (1 - |X_1|) - |1 - |X_1||$. So, $t = |(m_1 - |m_1|)| / 2d$ and $s = |(m_2 - |m_2|)| / 2d$, where $m_j = 1 - |X_j|$, and $d = X_1 - X_2$. 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:

$$\begin{aligned} X_1' &= X_1 + (X_2 - X_1)T, \\ X_2' &= X_2 - (X_2 - X_1)S. \end{aligned}$$

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	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)	
5.		Y←S	ADD (C,- C)	MULT (RA,RB)
6.			ADD (2,-P)	MULT (S,1)
7.		Y←S	ADD (C,- C)	MULT (S,P)
8.				
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 second 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.

7

SN74ACT8847

Overview**1****SN74ACT8818 16-Bit Microsequencer****2****SN74ACT8832 32-Bit Registered ALU****3****SN74ACT8836 32- × 32-Bit Parallel Multiplier****4****SN74ACT8837 64-Bit Floating Point Processor****5****SN74ACT8841 Digital Crossbar Switch****6****SN74ACT8847 64-Bit Floating Point/Integer Processor****7****Support****8****Mechanical Data****9**

8

Support

Support

8

Support

8

Support

Design Support for TI's SN74ACT8800 Family

TI's '8800 32-bit processor family is supported by a variety of tools developed to aid in design evaluation and verification. These tools will streamline all stages of the design process, from assessing the operation and performance of an individual device to evaluating a total system application. The tools include functional models, behavioral models, microcode development software, as well as the expertise of TI's VLSI Logic applications group.

Functional Evaluation Models Aid in Device Evaluation

Many design decisions can easily be made and evaluated before hardware or board prototypes are needed, using functional evaluation software models. The result is shortened design cycles and lower design costs.

Texas Instruments offers functional evaluation models for many of the devices in the '8800 family. These models are written in Microsoft C® and can be used in stand-alone mode or as callable functions.

These models are designed to provide insight into the operation of the devices by allowing the designer to write microcode and run it through the model. This allows the designer to select the device that best executes a specific application and provides a head start in evaluating programming performance.

The models correctly represent device timing in clock cycles, measured from the input of control and data to the output of results and status. Hence, initial performance estimates for a particular design can be made by relating the number of clock cycles required for an operation to the typical ac timing data for the device.

Behavioral Simulation Models Simplify System Debugging

System simulation with behavioral models can further shorten design time and ease design effort. The behavioral simulation models that support TI's '8800 chip set have the timing-control and error-handling capability to perform thorough PCB and system simulation. These models decrease the time spent in debugging and reduce the number of required prototype runs.

Users of system simulation models report a reduction by more than half in the number of prototype runs typically required to produce the highest-quality system. This savings in time reduces costs and gets the product to market as much as several months earlier than could be done using traditional methods.

Behavioral models for TI's '8800 family are written at the functional behavioral level and, therefore, are faster and easier to use and take up less disk space than some other types of simulation models. This higher efficiency means a simulation run can include more IC models and yet require less CPU time than an equivalent simulation using other types of models.

These behavioral simulation models also provide explicit error messages that can help in the debugging process. For example, if a design violates a device set-up time, the model explains, via an error message, what type of violation occurred, at what point it occurred in the simulation run, and specifically which part's set-up time was violated. Then, the model continues on with the run as if no violation occurred, saving time rather than crashing the run at every error.

In other words, an expert debugger is built right into the simulation.

The models are available with commercial and military timing and interact with a variety of simulators.

Behavioral Models for TI's '8800 Family are Easily Obtained

Texas Instruments has been working closely with both Quadtree Software Corporation and Logic Automation Incorporated to produce software behavioral simulation models of many of its VLSI devices. Since accuracy is key to solving design problems, we've provided Quadtree and Logic Automation with test patterns for most of our devices to ensure each model passes the same set of test vectors as does the actual silicon device.

Quadtree offers a library of Designer's Choice™ full-functional behavioral models of Texas Instruments '8800 32-bit processor building block devices.

Logic Automation Smartmodel™ library contains many Texas Instruments products, including devices from the '8800 chip set.

These companies may be contacted directly at the addresses below. General information about behavioral model support for the '8800 family may be obtained by calling Texas Instruments at (214) 997-5402.

LOGIC AUTOMATION INCORPORATED P.O. Box 310 Beaverton, OR 97075 (503) 690-6900	QUADTREE SOFTWARE CORPORATION 1170 Route 22 East Bridgewater, NJ 08807 (201) 725-2272
---	---

'8800 SDB Design Kit

TI offers an '8800 Software Development Board (SDB) Design Kit as an evaluation and training tool. The '8800 SDB kit uses a range of software development tools to allow users to evaluate performance and write microprograms for several of the '8800 building blocks. Using the SDB, microcode can be developed earlier in a system's design cycle so that code development parallels, rather than follows, prototype design.

The '8800 SDB Design Kit consists of a combination of specially developed hardware, software, and documentation including:

- The '8800 Software Development Board Assembly
- The '8800 SDB User's Guide
- Floppy disk with MS-DOS™ software tools written in Microsoft C, several example microprograms, and demo programs. Source code is included.
- Microcode definition files for use with HILEVEL, STEP Engineering, and Texas Instruments microcode development tools.

Built on a PC/AT card occupying a single slot, the '8800 SDB contains an 'ACT8818 microsequencer, 'ACT8832 registered ALU, and an 'ACT8847 floating point/integer processor, along with 32 K by 128 bits of microcode memory, and 32 K by 32 bits of local data memory. A block diagram of the '8800 SDB is detailed in Figure 8-1. The board operates under an MS-DOS environment.

The SDB Design Kit complements other '8800 family development tools such as functional evaluation and behavioral simulation models. It actually provides the next step beyond simulators. System code can be executed in a realtime environment that includes conditional branching, on-board data memory, and single-step/breakpoint facilities.

For additional technical information, contact VLSI System Engineering at (214) 997-3970. For ordering information, please call your local field sales representative.

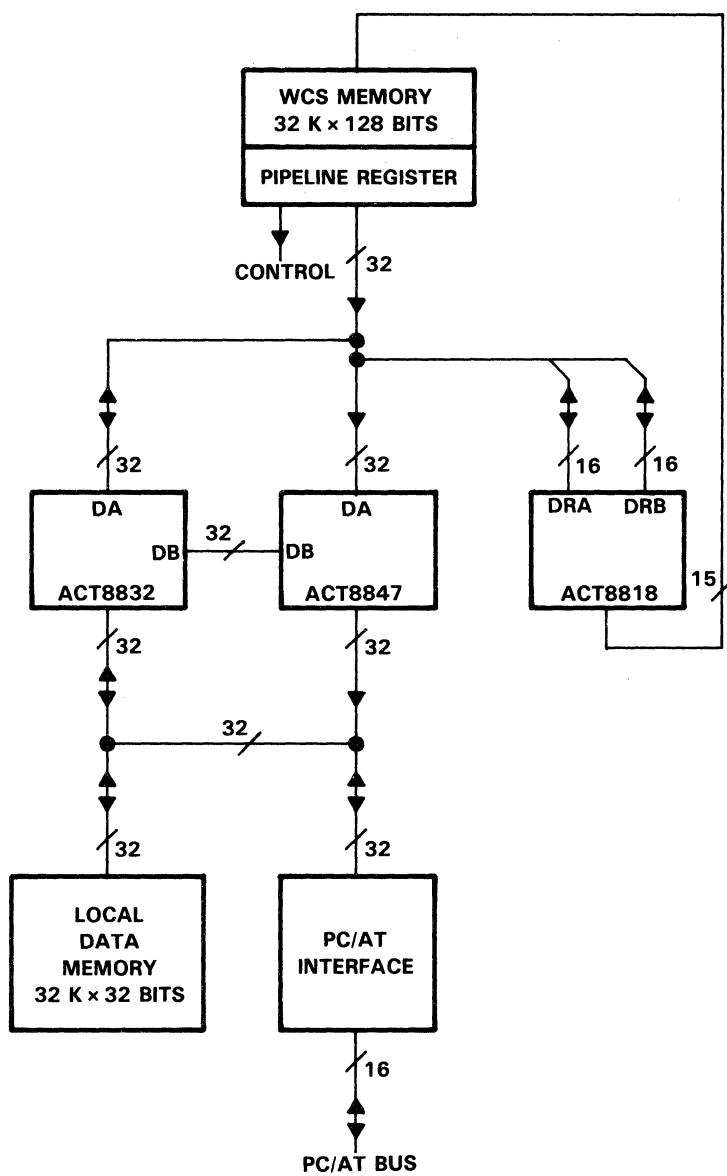


Figure 8-1. '8800 SDB Block Diagram

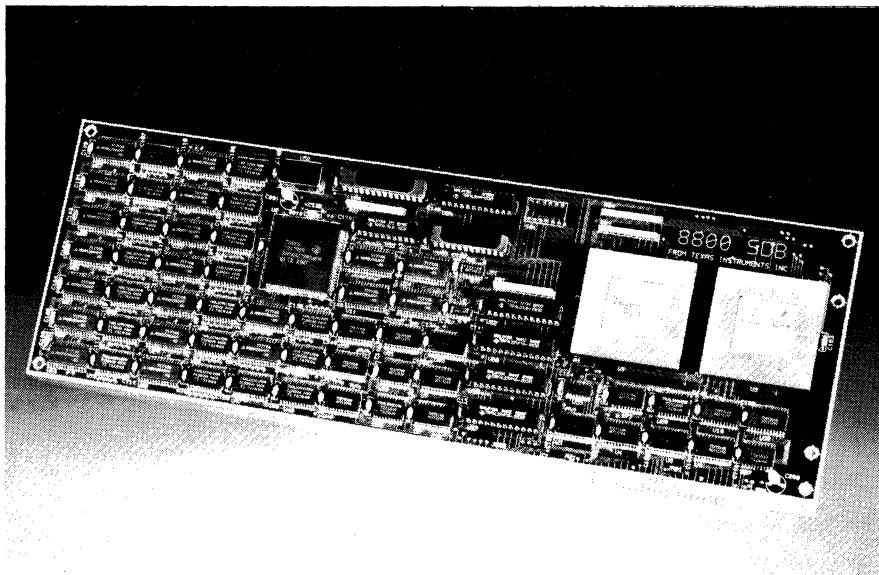
Program Code Generation Using the TI Meta Assembler

The TI Meta Assembler (TIM) provides the means to create object microcode files and to support listings for programs that execute in architectures without standard instruction sets. The end-product of TIM is an absolute object code module in suitable format for downloading to PROM programmers or to the emulator memories of development systems. TIM is fully compatible with some other assemblers as well.

Systems Expertise is a Phone Call Away

Texas Instruments VLSI Logic applications group is available to help designers analyze TI's high-performance VLSI products, such as the '8800 32-bit processor family. The group works directly with designers to provide ready answers to device-related questions and also prepares a variety of applications documentation.

The group may be reached in Dallas, at (214) 997-3970.



8

Support

Overview

1

SN74ACT8818 16-Bit Microsequencer

2

SN74ACT8832 32-Bit Registered ALU

3

SN74ACT8836 32- × 32-Bit Parallel Multiplier

4

SN74ACT8837 64-Bit Floating Point Processor

5

SN74ACT8841 Digital Crossbar Switch

6

SN74ACT8847 64-Bit Floating Point/Integer Processor

7

Support

8

Mechanical Data

9

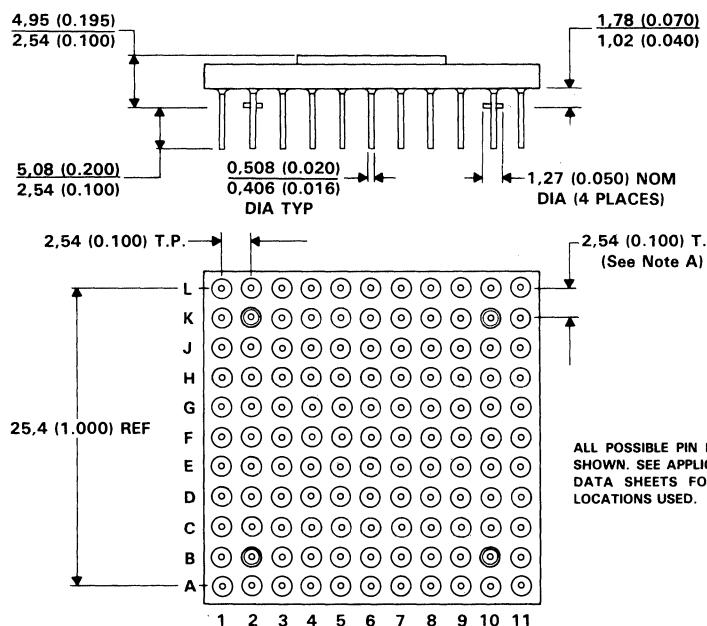
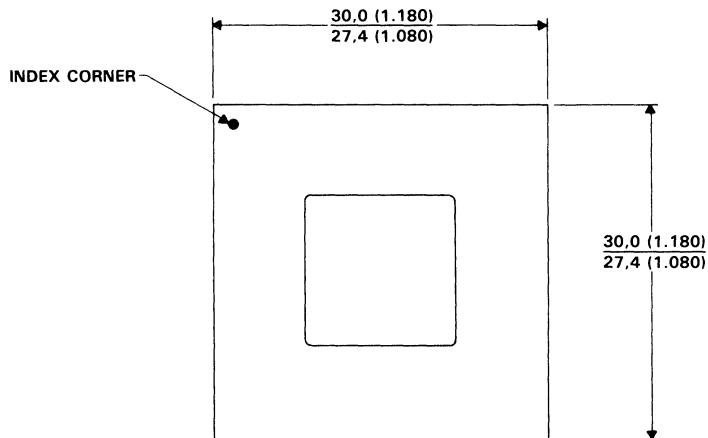
Mechanical Data

Mechanical Data

SN74ACT8818	11×11 GC PACKAGE
SN74ACT8832	17×17 GB PACKAGE
SN74ACT8836	15×15 GB PACKAGE
SN74ACT8837	17×17 GB PACKAGE
SN74ACT8841	15×15 GB PACKAGE
SN74ACT8847	17×17 GA PACKAGE

Mechanical Data

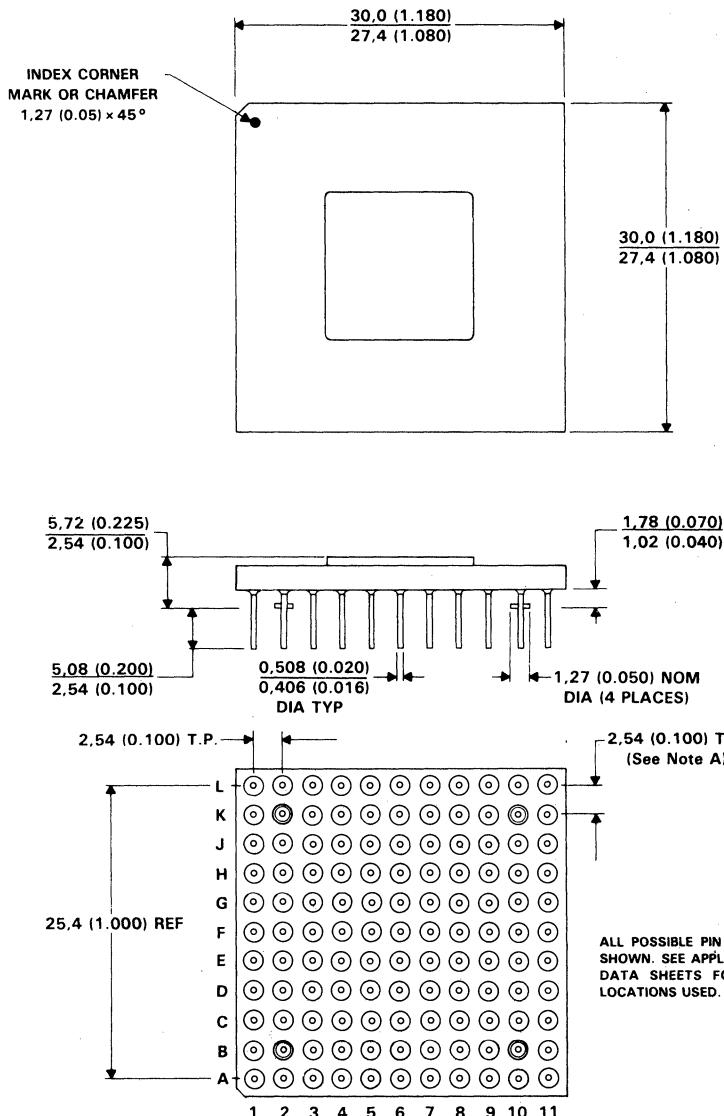
11 × 11 GB pin grid array ceramic package



NOTE A: Pins are located within 0.13 (0.005) radius of true position relative to each other at maximum material condition and within 0.381 (0.051) radius relative to the center of the ceramic.

ALL LINEAR DIMENSIONS ARE IN MILLIMETERS AND PARENTHETICALLY IN INCHES

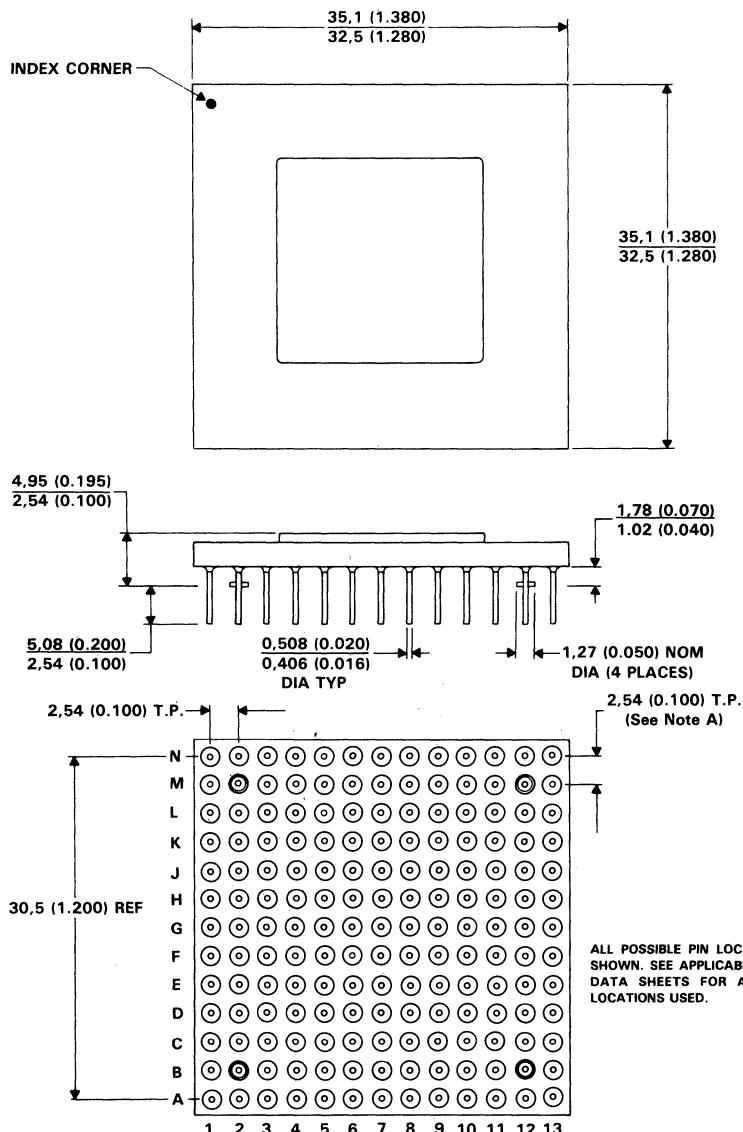
11 × 11 GC pin grid array ceramic package



NOTE A: Pins are located within 0.13 (0.005) radius of true position relative to each other at maximum material condition and within 0.381 (0.051) radius relative to the center of the ceramic.

ALL LINEAR DIMENSIONS ARE IN MILLIMETERS AND PARENTHETICALLY IN INCHES

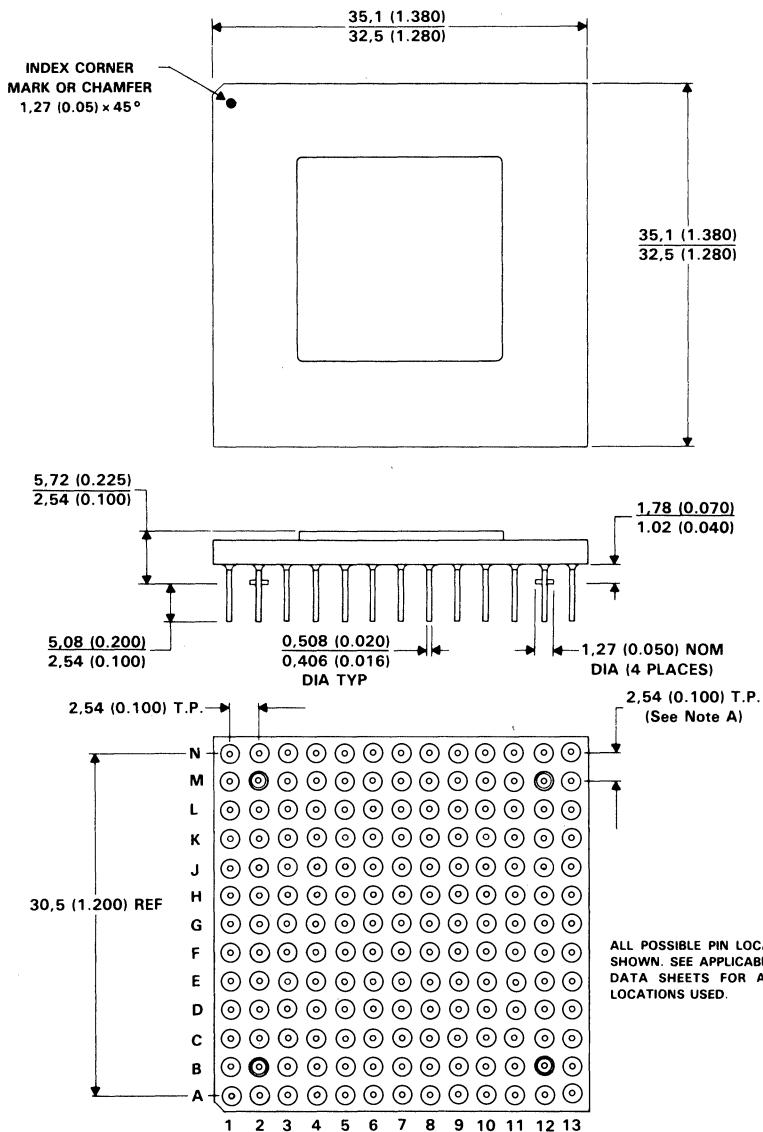
13 × 13 GB pin grid array ceramic package



NOTE A: Pins are located within 0.13 (0.005) radius of true position relative to each other at maximum material condition and within 0.381 (0.051) radius relative to the center of the ceramic.

ALL LINEAR DIMENSIONS ARE IN MILLIMETERS AND PARENTHETICALLY IN INCHES

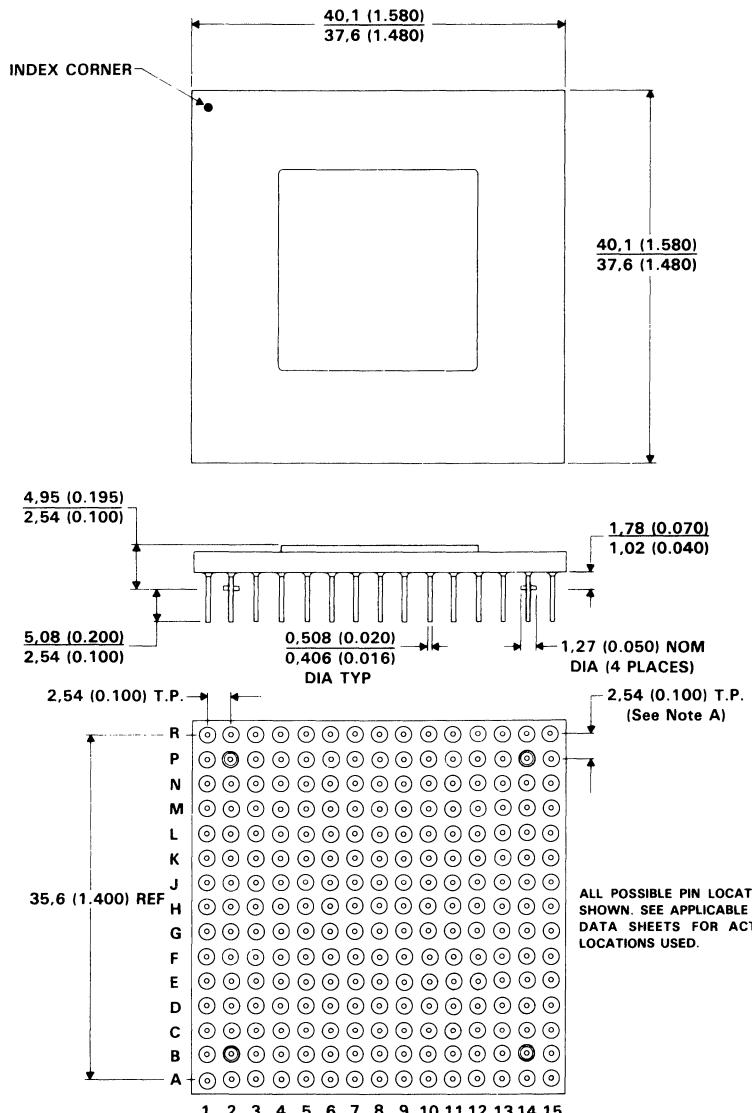
13 × 13 GC pin grid array ceramic package



NOTE A: Pins are located within 0.13 (0.005) radius of true position relative to each other at maximum material condition and within 0.381 (0.051) radius relative to the center of the ceramic.

ALL LINEAR DIMENSIONS ARE IN MILLIMETERS AND PARENTHETICALLY IN INCHES

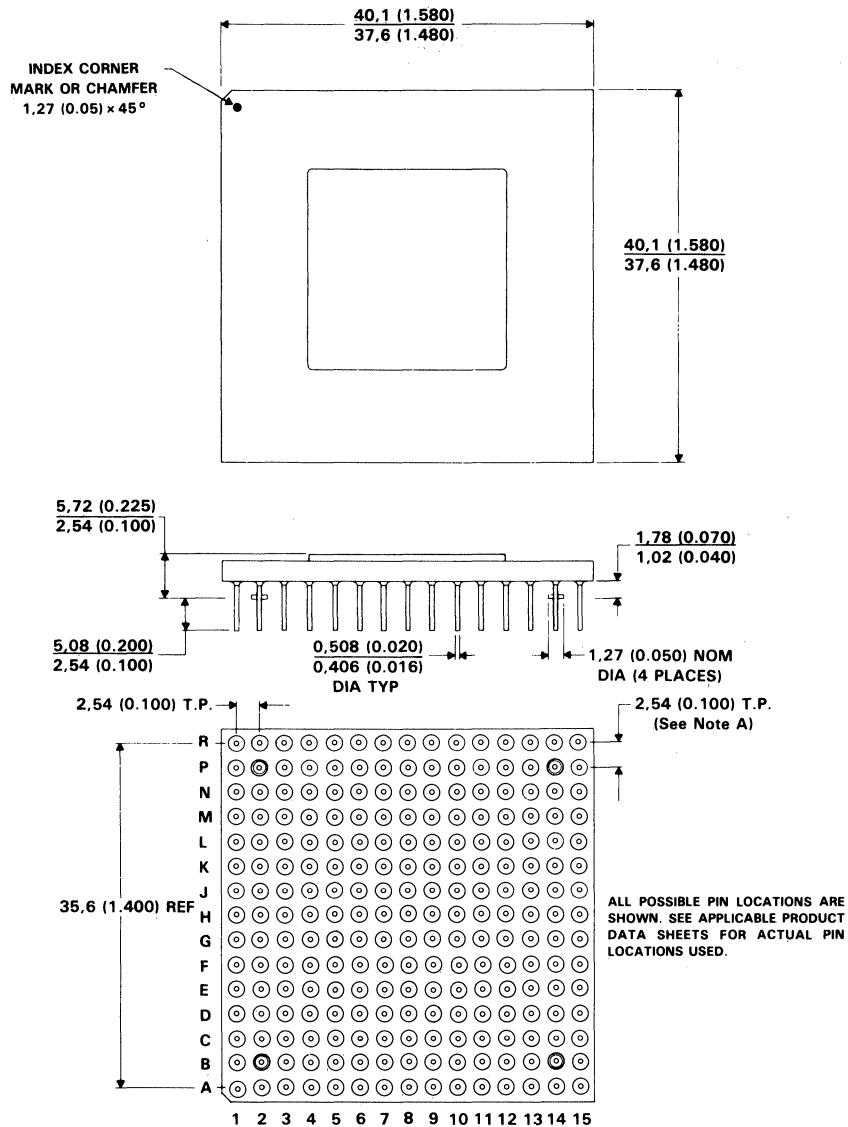
15 × 15 GB pin grid array ceramic package



NOTE A: Pins are located within 0.13 (0.005) radius of true position relative to each other at maximum material condition and within 0.381 (0.051) radius relative to the center of the ceramic.

ALL LINEAR DIMENSIONS ARE IN MILLIMETERS AND PARENTHETICALLY IN INCHES

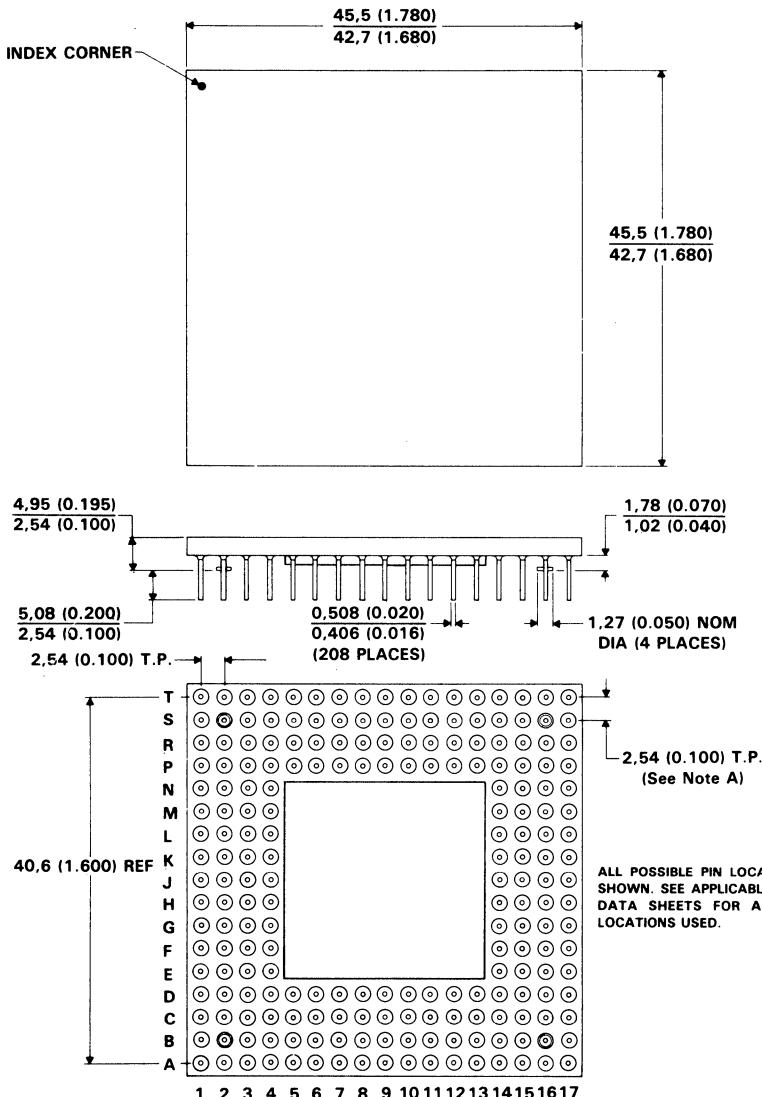
15 × 15 GC pin grid array ceramic package



NOTE A: Pins are located within 0.13 (0.005) radius of true position relative to each other at maximum material condition and within 0.381 (0.015) radius relative to the center of the ceramic.

ALL LINEAR DIMENSIONS ARE IN MILLIMETERS AND PARENTHETICALLY IN INCHES

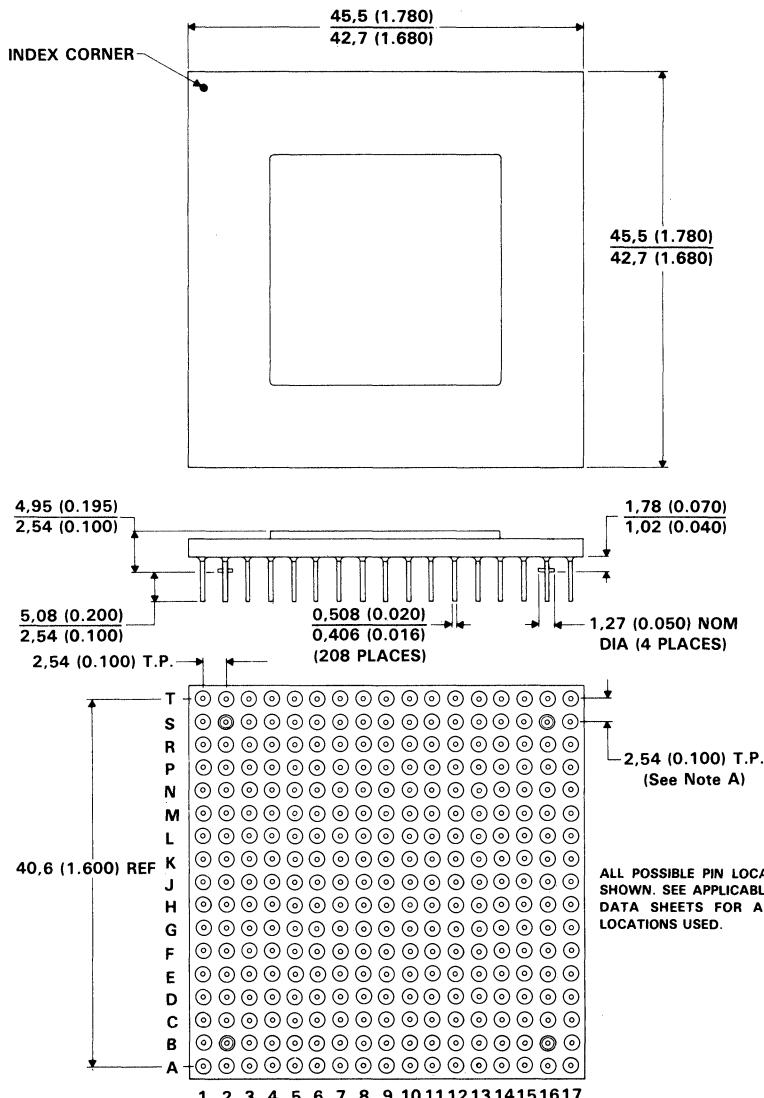
17 × 17 GA pin grid array ceramic package



NOTE A: Pins are located within 0.13 (0.005) radius of true position relative to each other at maximum material condition and within 0.381 (0.051) radius relative to the center of the ceramic.

ALL LINEAR DIMENSIONS ARE IN MILLIMETERS AND PARENTHETICALLY IN INCHES

17 × 17 GB pin grid array ceramic package



NOTE A: Pins are located within 0.13 (0.005) radius of true position relative to each other at maximum material condition and within 0.381 (0.051) radius relative to the center of the ceramic.

ALL LINEAR DIMENSIONS ARE IN MILLIMETERS AND PARENTHETICALLY IN INCHES

TI Sales Offices

ALABAMA: Huntsville (205) 837-7530.

ARIZONA: Phoenix (602) 995-1007; Tucson (602) 292-2640.

CALIFORNIA: Irvine (714) 660-1200; Roseville (916) 785-9208;

San Diego (619) 278-9601; Santa Clara (408) 980-9000;

Torrance (213) 217-7010; Woodland Hills (818) 704-7759.

COLORADO: Aurora (303) 368-8000.

CONNECTICUT: Wallingford (203) 269-0074.

FLORIDA: Altamonte Springs (305) 260-2116; Ft. Lauderdale (305) 973-8502; Tampa (813) 885-7411.

GEORGIA: Norcross (404) 662-7900.

ILLINOIS: Arlington Heights (312) 640-2925.

INDIANA: Carmel (317) 573-6400; Ft. Wayne (219) 424-5174.

IOWA: Cedar Rapids (319) 395-9550.

KANSAS: Overland Park (913) 451-4511.

MARYLAND: Columbia (301) 964-2003.

MASSACHUSETTS: Waltham (617) 895-9100.

MICHIGAN: Farmington Hills (313) 553-1569;

Grand Rapids (616) 957-4200.

MINNESOTA: Eden Prairie (612) 828-9300.

MISSOURI: St. Louis (314) 569-7600.

NEW JERSEY: Iselin (201) 750-1050.

NEW MEXICO: Albuquerque (505) 345-2555.

NEW YORK: East Syracuse (315) 463-9291;

Pittsfield (716) 385-6770;

Poughkeepsie (914) 473-2900.

NORTH CAROLINA: Charlotte (704) 527-0933;

Raleigh (919) 876-2725.

OHIO: Beachwood (216) 464-6100;

Beaver Creek (513) 427-6200.

OREGON: Beaverton (503) 643-6758.

PENNSYLVANIA: Blue Bell (215) 825-9500.

PUERTO RICO: Hato Rey (809) 753-8700.

TENNESSEE: Johnson City (615) 461-2192.

TEXAS: Austin (512) 250-7655;

Houston (713) 778-5592;

Richardson (214) 680-5082;

San Antonio (512) 496-1779.

UTAH: Murray (801) 266-8972.

WASHINGTON: Redmond (206) 881-3080.

WISCONSIN: Brookfield (414) 782-2899.

CANADA: Nepean, Ontario (613) 726-1970;

Richmond Hill, Ontario (416) 884-9181;

St. Laurent, Quebec (514) 336-1860.

TI Regional Technology Centers

CALIFORNIA: Irvine (714) 660-8105;

Santa Clara (408) 748-2220;

GEORGIA: Norcross (404) 662-7945.

ILLINOIS: Arlington Heights (312) 640-2909.

MASSACHUSETTS: Waltham (617) 895-9196.

TEXAS: Richardson (214) 680-5066.

CANADA: Nepean, Ontario (613) 726-1970.

TI Distributors

TI AUTHORIZED DISTRIBUTORS

Arrow/Kierulff Electronics Group

Arrow (Canada)

Future Electronics (Canada)

GRS Electronics Co., Inc.

Hall-Mark Electronics

Marshall Industries

Newark Electronics

Schweber Electronics

Tlme Electronics

Wyle Laboratories

Zeus Components

- OBSOLETE PRODUCT ONLY -

Rochester Electronics, Inc.

Newburyport, Massachusetts

(508) 462-9332

MARYLAND: Arrow/Kierulff (301) 995-6002;

Hall-Mark (301) 988-9800; Marshall (301) 235-9464;

Schweber (301) 840-5900; Zeus (301) 997-1118.

MASSACHUSETTS: Arrow/Kierulff (508) 658-0900;

Hall-Mark (508) 657-0902; Marshall (508) 659-0810;

Schweber (617) 275-5100; Time (617) 532-6200;

Wyle (617) 273-7300; Zeus (617) 863-8800.

MICHIGAN: Detroit: Arrow/Kierulff (313) 462-2290;

Hall-Mark (313) 462-1205; Marshall (313) 525-5850;

Newark (313) 967-0600; Schweber (313) 525-8100;

Grand Rapids: Arrow/Kierulff (616) 243-0912.

MINNESOTA: Arrow/Kierulff (612) 830-1800;

Hall-Mark (612) 941-2600; Marshall (612) 559-2211;

Schweber (612) 941-5280.

MISSOURI: St. Louis: Arrow/Kierulff (314) 567-6888;

Hall-Mark (314) 291-5350; Marshall (314) 291-4650;

Schweber (314) 739-0526.

NEW HAMPSHIRE: Arrow/Kierulff (603) 666-6968;

Schweber (603) 625-2250.

NEW JERSEY: Arrow/Kierulff (201) 539-0000;

(609) 596-8000; GRS Electronics (609) 964-8560;

Hall-Mark (201) 575-4415; (201) 882-9773;

(609) 235-1900; Marshall (201) 882-0320;

(609) 234-9170; Schweber (201) 227-7880.

NEW MEXICO: Arrow/Kierulff (505) 243-4566.

NEW YORK, Long Island:

Arrow/Kierulff (516) 231-1009; Hall-Mark (516) 737-0600;

Marshall (516) 273-2424; Schweber (516) 334-7474;

Zeus (914) 937-7400;

Rochester: Arrow/Kierulff (716) 427-0300;

Marshall (716) 427-0300; Marshall (716) 235-7620;

Schweber (716) 424-2222;

Syracuse: Marshall (607) 798-1611.

NORTH CAROLINA: Arrow/Kierulff (919) 876-3132;

(919) 725-9711; Hall-Mark (919) 872-0712;

Marshall (919) 878-9882; Schweber (919) 876-0000.

OHIO: Cleveland: Arrow/Kierulff (216) 248-3990;

Hall-Mark (216) 349-4632; Marshall (216) 248-1788;

Schweber (216) 464-2970;

Columbus: Hall-Mark (614) 886-3313;

Dayton: Arrow/Kierulff (513) 435-5563;

Marshall (513) 898-4480; Schweber (513) 439-1800.

OKLAHOMA: Arrow/Kierulff (918) 252-5737;

Schweber (918) 622-8003.

OREGON: Arrow/Kierulff (503) 645-6456;

Marshall (503) 644-5050; Wyle (503) 640-6000.

PENNSYLVANIA: Arrow/Kierulff (216) 856-7000;

(215) 928-1800; GRS Electronics (215) 922-7037;

Marshall (412) 963-0441; Schweber (215) 441-0600,

(412) 963-6804.

TEXAS: Austin: Arrow/Kierulff (512) 835-4180;

Hall-Mark (512) 258-8848; Marshall (512) 837-1991;

Schweber (512) 339-0088; Wyle (512) 834-9957;

Dallas: Arrow/Kierulff (214) 380-6464;

Houston: Arrow/Kierulff (214) 525-5010;

Marshall (214) 561-5010; Wyle (214) 235-9593;

Zeus (214) 783-7010;

El Paso: Marshall (915) 593-0706;

Houston: Arrow/Kierulff (713) 530-4700;

Hall-Mark (713) 781-6100; Marshall (713) 895-9200;

Schweber (713) 784-3600; Wyle (713) 879-9953.

UTAH: Arrow/Kierulff (801) 973-6913;

Hall-Mark (801) 972-1008; Marshall (801) 485-1551;

Wyle (801) 974-9953.

WASHINGTON: Arrow/Kierulff (206) 575-4420;

Marshall (206) 486-5747; Wyle (206) 881-1150.

WISCONSIN: Arrow/Kierulff (414) 792-0150;

Hall-Mark (414) 797-7844; Marshall (414) 797-8400;

Schweber (414) 784-9020.

CANADA: Calgary: Future (403) 235-5225;

Edmonton: Future (403) 438-2658;

Montreal: Arrow Canada (514) 735-5511;

Futura: Arrow Canada (514) 693-1010;

Ottawa: Arrow Canada (613) 226-6903;

Quebec City: Arrow Canada (416) 817-7500;

Toronto: Arrow Canada (416) 672-7769;

Future (416) 638-4771; Marshall (416) 674-2161;

Vancouver: Arrow Canada (604) 251-2996;

Future (604) 294-1166.

Customer Response Center

TOLL FREE: (800) 232-3200

OUTSIDE USA: (214) 995-6111

(8:00 a.m. - 5:00 p.m. CST)

A-189

TEXAS
INSTRUMENTS

TI Worldwide Sales Offices

ALABAMA: Huntsville: 500 Wynn Drive, Suite 514, Huntsville, AL 35805, (205) 837-7530.

ARIZONA: Phoenix: 8825 N. 23rd Ave., Phoenix, AZ 85021, (602) 995-1007; Tucson: 818 W. Miracle Mile, Suite 43, Tucson, AZ 85705, (602) 292-2640.

CALIFORNIA: Irvine: 17891 Cartwright Dr., Irvine, CA 92714, (714) 856-6500; San Jose: Sirena Gate Plaza, Roseville, CA 95747, (619) 785-3208; San Diego: 4333 View Ridge Ave., Suite 100, San Diego, CA 92123, (619) 278-9601; Santa Clara: 5355 Bessie Ross Dr., Santa Clara, CA 95054, (408) 880-9000; Torrance: 690 Knox St., Torrance, CA 90504, (310) 217-7010; Woodland Hills: 21220 Bellanca St., Woodland Hills, CA 91367, (800) 704-7759.

COLORADO: Aurora: 1400 S. Potomac Ave., Suite 101, Aurora, CO 80012, (303) 368-8000.

CONNECTICUT: Wallingford: 9 Barnes Industrial Park Rd., Barnes Industrial Park, Wallingford, CT 06492, (203) 269-0074.

FLORIDA: Altamonte Springs: 370 S. North Lake Blvd., Altamonte Springs, FL 32717, (305) 260-2116; Ft. Lauderdale: 3260 N.W. 62nd St., Ft. Lauderdale, FL 33309, (305) 973-8502; Tampa: 4803 Georgia Dr., Suite 390, Tampa, FL 33634, (813) 885-7411.

GEORGIA: Norcross: 5515 Spalding Drive, Norcross, GA 30092, (404) 662-7900.

ILLINOIS: Arlington Heights: 515 W. Algonquin, Arlington Heights, IL 60005, (312) 640-2925.

INDIANA: Ft. Wayne: 2020 Inwood Dr., Ft. Wayne, IN 46815, (219) 424-1747; Carmel: 550 Congressional Dr., Carmel, IN 46032, (317) 573-8400.

IOWA: Cedar Rapids: 373 Collins Rd. NE, Suite 201, Cedar Rapids, IA 52402, (319) 395-9550.

KANSAS: Overland Park: 7300 College Blvd., Lighton Plaza, Overland Park, KS 66210, (913) 451-4511.

MARYLAND: Columbia: 8815 Centre Park Dr., Columbia MD 21045, (301) 964-2003.

MASSACHUSETTS: Waltham: 950 Winter St., Waltham, MA 02154, (617) 895-9100.

MICHIGAN: Farmington Hills: 33737 W. 12 Mile Rd., Farmington Hills, MI 48018, (313) 553-1569; Grand Rapids: 3075 Orchard Vista Dr. S.E., Grand Rapids, MI 49506, (616) 957-4200.

MINNESOTA: Eden Prairie: 11000 W. 78th St., Eden Prairie, MN 55314, (612) 828-9300.

MISSOURI: St. Louis: 11816 Borman Drive, St. Louis, MO 63146, (314) 569-7600.

NEW JERSEY: Iselin: 485E Route 1 South, Parkway Towers, Iselin, NJ 08830 (201) 750-1050.

NEW MEXICO: Albuquerque: 2820-D Broadhurst Pkwy NE, Albuquerque, NM 87107, (505) 345-2555.

NEW YORK: East Syracuse: 6365 Colamer Dr., East Syracuse, NY 13057, (315) 463-9291; Melville: 2805 Highwoods Blvd., P.O. Box 2936, Melville, NY 11747, (516) 454-6800; Pittsford: 2851 Clover St., Pittsford, NY 14534, (716) 385-6770; Poughkeepsie: 385 South Rd., Poughkeepsie, NY 12501, (914) 473-2900.

NORTH CAROLINA: Charlotte: 8 Woodlawn Green, Woodlawn Rd., Charlotte, NC 28210, (704) 527-0933; Raleigh: 2805 Highwoods Blvd., Suite 100, Raleigh, NC 27625, (919) 876-2725.

OHIO: Beachwood: 23775 Commerce Park Rd., Beachwood, OH 44122, (216) 464-6100; Beavercreek: 4200 Colonel Glenn Hwy., Beavercreek, OH 45431, (513) 427-6200.

OREGON: Beaverton: 6700 SW 105th St., Suite 110, Beaverton, OR 97007, (503) 643-6758.

PENNSYLVANIA: Blue Bell: 670 Sentry Pkwy., Blue Bell, PA 19422, (215) 825-9500.

PUERTO RICO: Hato Rey: Mercantil Plaza Bldg., Suite 505, Hato Rey, PR 00918, (809) 753-8700.

TENNESSEE: Johnson City: Erwin Hwy., P.O. Drawer, 1255, Johnson City, TN 37605 (615) 461-2192.

TEXAS: Austin: 12501 Research Blvd., Austin, TX 78759, (512) 765-7655; Richardson: 1001 E. Cypress Creek Rd., Richardson, TX 75081, (214) 680-5028; Houston: 9100 Southwest Freway, Suite 250, Houston, TX 77074, (713) 778-6592; San Antonio: 1000 Central Parkway South, San Antonio, TX 78232, (512) 496-1779.

UTAH: Murray: 5201 South Green St., Suite 200, Murray, UT 84123, (801) 226-8972.

WASHINGTON: Redmond: 5101 148th NE, Bldg B, Suite 107, Redmond, WA 98052, (206) 881-3080.

WISCONSIN: Brookfield: 450 N. Sunny Slope, Suite 150, Brookfield, WI 53005, (414) 782-2899.

CANADA: Nepean: 301 Morris Drive, Mallorn Center, Nepean, Ontario, K2B 8R9; Richmond Hill: 2999 Steeles Ave. E., Richmond Hill L4C1B1, Ontario, Canada, (416) 884-9181; St. Laurent: Ville St. Laurent Quebec, 9460 Trans Canada Hwy., St. Laurent, Quebec, Canada H4S1R9, (416) 336-1860.

ARGENTINA: Texas Instruments Argentina Viamonte 1119, 1053 Capital Federal, Buenos Aires, Argentina, 5417-748-3699.

AUSTRALIA (& NEW ZEALAND): Texas Instruments Australia Ltd.: 6-10 Talavera Rd., North Ryde (Sydney), New South Wales, Australia 2113, 2-4-6, (02) 856-1000; King's Road, Melbourne, Victoria, Australia 3000, 3 - 267-4677; 171 Phillip Highway, Elizabeth, South Australia 5112, 8 + 255-2065.

AUSTRIA: Texas Instruments G.m.b.H.: Industriestrasse B/16, A-2345 Brunn/Bergbirge, 2326-846210.

BELGIUM: Texas Instruments N.V., Belgium S.A.: 11, Avenue Jules Baudetstraat 11, 1140 Brussels, Belgium, (02) 242-3080.

BRAZIL: Texas Instruments Eletronicos do Brasil Ltda.: Rue Paes Leme, 5247-A Andar Pinheiros, 05424 Sao Paulo, Brazil, 0815-6166.

DENMARK: Texas Instruments A/S, Mairleundvej 46E, 2730 Herlev, Denmark, 2 - 91 74 00.

FINLAND: Texas Instruments Finland OY: Ahvenisto 3, P.O. Box 81, ESPOO, Finland, (90) 0-461-422.

FRANCE: Texas Instruments France: Paris Office, BP 67-8-1 Avenue Marne-Stalingrad, 92140 Villey-Villacoublay, cedex 1 (3) 30 70 1003.

GERMANY: Federal Republic of Germany: Texas Instruments Deutschland GmbH, Hegewegstrasse 1, 8050 Freising, 8161-80-4591; Kurfuerstendamm 195/196, 1000 Berlin 15, 30-B82-7365; III, Hagen 43/Kibelstrasse, 19, 4300 Essen, 201-24250; Kirchhorsterstrasse 2, 3000 Hannover 51, 511+648021; Maybachstrasse 11, 7302 Ostfildern 2-Nelingen, 711-34030.


**TEXAS
INSTRUMENTS**

HONG KONG: Texas Instruments Hong Kong Ltd., 8th Floor, World Shipping Ctr., 7 Canton Rd., Kowloon, Hong Kong, (852) 3-7351223.

IRELAND: Texas Instruments (Ireland) Limited: 7/8 Harcourt Street, Stillorgan, County Dublin, Eire, I 781677.

ITALY: Texas Instruments Italia S.p.A. Divisione Semiconductor: Viale Europa, 40, 20093 Cologne Monzese (Milano), (02) 253001; Via Castello della Magliana, 38, 00148 Roma, (06) 5222651; Via Amendola, 17, 40100 Bologna, (051) 554004.

JAPAN: Tokyo Marketing/Sales (Headquarters): Texas Instruments Japan Ltd., MS Shibaura Bldg., 9F, 4-1-1, Shibaura, Minato-ku, Tokyo, Japan; Nitto-ishi-ku, Osaka 541, Japan, 06-294-1881; Daini Toyota West Bldg., 7F, 10-27 Meiki 4-chome, Nakamura-ku, Nagoya 450, 052-883-8691; Daichi Seimeji Bldg., 6F, 3-1-2, Ichigaya-cho, Toshima-ku, Tokyo 171, 03-326-9200; 3-762-2-5471; Daichi Olympic, Tachikawa Bldg., 6F, 1-25-12 Akabono-cho, Tachikawa 190, Tokyo, 0425-27-6426; Matsumoto Bldg., 6F, 2-11 Fushimi 1-chome, Matsumoto 390, Nagano-ken, 0263-33-1080; Yokohama-Higashiku KN 6F, 2-1-2, 2-1-2, Yokohama, Kanagawa 220, 045-322-6741; Nihon Seimeji Kyoto Yaska Bldg., 5F, 8-2-2 Higashiohikobori, Nishinotoin-Higashiru, Shioaku, Shimogyo-ku, Kyoto 600, 075-341-7713; 2597-1 Aza Harada, Oaza Yasaka, Kitaky 817, Oita, 097-3211; Mine Plant, 2350 Kihara Miho-mura, Inashigakubo 300-04, Ibaraki-ken, 0298-85-2541.

KOREA: Texas Instruments Korea Ltd., 28th Fl., Trade Tower, #158 Samseung-Dong, Kangnam-ku, Seoul, Korea 2-551-2810.

MEXICO: Texas Instruments de Mexico S.A.: Alfonso Reyes 115, Col. Hipódromo Condesa, Mexico, D.F., Mexico 06120, 525/525-3860.

MIDDLE EAST: Texas Instruments: No. 13, 1st Floor P.O. Box 200, Diplomatic Area, P.O. Box 26235, Manama Bahrain; Arabian Gulf, 973+274681.

NETHERLANDS: Texas Instruments Holland B.V., 19 Hoofdweg, 1100 AZ Amsterdam-Zuidoost, Holland 20 + 5602911.

NORWAY: Texas Instruments Norway A/S, PB106, Reftast 0585, Oslo 5, Norway, (2) 155-0590.

PEOPLES REPUBLIC OF CHINA: Texas Instruments China Inc., Beijing Representative Office, 7-05 Civic Center, 19 Jianguomenwai Dajie, Beijing, China, (861) 5002235, Ext. 3750.

PHILIPPINES: Texas Instruments Asia Ltd.: 14th Floor, Ba-Leonto Bldg., Paseo de Roxas, Makati, Metro Manila, Philippines, 618-60-31.

PORTUGAL: Texas Instruments Equipamento Eletronico, Lda.: Rue Eng. Frederico Ulrich, 2650 Moreira Da Maia, 4470 Maia, Portugal, 2149-1003.

SINGAPORE (& INDIA, INDONESIA, MALAYSIA, THAILAND): Texas Instruments Singapore (PTE) Ltd., Asia Pacific Division, 101 Thompson Rd., 01-001, United Square, Singapore 1130, 350-8100.

SPAIN: Texas Instruments Espana, S.A.: C/Jose Lazaro Galdiano No. 6, Madrid 2036, 1/458-14-58.

SWEDEN: Texas Instruments International Trade Corporation, Sverigesplatsen 5-164-93, Stockholm, Sweden, 8-752-5800.

SWITZERLAND: Texas Instruments, Inc., Reidstrasse 1-74, 8000 Zurich, Switzerland, 01-740-2250.

TAIWAN: Texas Instruments Supply Co., 9th Floor Bank Tower, 205 Tun Hwa N.Rd., Taipei, Taiwan, Republic of China, 2 + 713-9311.

UNITED KINGDOM: Texas Instruments Limited: Manton Lane, Bedford, MK41 7PA, England, 0234 270111.



Printed in U.S.A.
689-20

SCSS006B



SN74ACT8800 Family

***32-Bit CMOS Processor
Building Blocks***

ERRATA

1989

***SN74ACT8800 Family
32-Bit CMOS Processor
Building Blocks***

Errata



IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to or to discontinue any semiconductor product or service identified in this publication without notice. TI advises its customers to obtain the latest version of the relevant information to verify, before placing orders, that the information being relied upon is current.

TI warrants performance of its semiconductor products to current specifications in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Unless mandated by government requirements, specific testing of all parameters of each device is not necessarily performed.

TI assumes no liability for TI applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

Copyright © 1989, Texas Instruments Incorporated

Printed in U.S.A.

ERRATA

TO THE SN74ACT8800 FAMILY DATA MANUAL (SCSS006B)

JUNE 1989 REVISIONS

These errata pages contain corrections to the following specifications:

1. Switching Characteristics, pg. 7-37
2. Setup and Hold Times, pg. 7-38
3. CLK/RESET Requirements, pg 7-38
4. Switching Characteristics, pg. 7-39
5. Switching Characteristics, pg. 7-41.

If you should have any further questions or concerns, contact your nearest TI field sales office, local authorized TI distributor, or the TI Customer Response Center at 1-800-232-3200.

- Page 7-37 — Replace the switching characteristics with the following:

switching characteristics

NO.	PARAM-ETER	FROM (INPUT)	TO (OUTPUT)	PIPELINE CONTROLS PIPES2-PIPES0	SN74ACT8847-30		UNIT
					MIN	MAX	
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		54	ns
		PIPELN REG	STATUS	10X		54	
4	t_{pd4}	OUTPUT REG	Y OUTPUT	0XX		20	ns
		OUTPUT REG	STATUS	0XX		20	
5	t_{pd5}	SELMS/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/LS	Y OUTPUT INVALID	XXX	1.5		ns
9	t_{d1}	CLK↑	CLK↑	010 w/o feedback	56		ns
		CLK↑	CLK↑	010 w/feedback‡	56		
		CLK↑	CLK↑	010 W/FLOWC§	66		
10	t_{d2}	CLK↑	CLK↑	000 w/o feedback	30		ns
		CLK↑	CLK↑	000 w/feedback‡	30		
		CLK↑	CLK↑	000 W/FLOWC§	36		
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	$t_d-0¶$	ns
12	t_{en1}	OEY	Y OUTPUT	XXX		15	ns
13	t_{en2}	ŒC, OES	STATUS	XXX		15	
14	t_{dis1}	OEY	Y OUTPUT	XXX		15	
15	t_{dis2}	ŒC, OES	STATUS	XXX		15	

† This parameter no longer tested and will be deleted on next Data Manual revision.

‡ Applies to all feedback cases except where operands are fed back using FLOWC to bypass C register. (Please see Figure 13 for feedback paths).

§ Operands are fed back using FLOWC to bypass the C register.

¶ t_d is the clock cycle period.

- Page 7-38 — Replace the setup and hold times with the following:

setup and hold times

NO.	PARAMETER	PIPELINE CONTROLS PIPES2-PIPES0	SN74ACT8847-30		UNIT
			MIN	MAX	
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	12	
21	t_{su6}	RESET before CLK↑	XX0	12	
22	t_{h1}	Inst/control after CLK↑	XXX	3	ns
23	t_{h2}	DA/DB after CLK↑	XXX	4	
24	t_{h3}	SRCC after CLKC↑	XXX	1	
25	t_{h4}	RESET after CLK↑	XX0	6	

- Page 7-38 — Replace the CLK/RESET requirements with the following:

CLK/RESET requirements

t _w	PARAMETER	SN74ACT8847-30		UNIT
		MIN	MAX	
t _w	Pulse duration	CLK high	10	ns
		CLK low	10	
		RESET	10	

- Page 7-39 — Replace the switching characteristics with the following:

switching characteristics

NO.	PARAMETER	FROM (INPUT)	TO (OUTPUT)	PIPELINE CONTROLS PIPES2-PIPES0	SN74ACT8847-40		UNIT
					MIN	MAX	
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 w/o feedback	72		ns
		CLK↑	CLK↑	010 w/feedback‡	72		
		CLK↑	CLK↑	010 W/FLOWC§	84		
10	t_{d2}	CLK↑	CLK↑	000 w/o feedback	40		ns
		CLK↑	CLK↑	000 w/feedback‡	40		
		CLK↑	CLK↑	000 W/FLOWC§	47		
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	$t_d-0\ddagger$	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.

‡Applies to all feedback cases except where operands are fed back using FLOWC to bypass C register. (Please see Figure 13 for feedback paths).

§Operands are fed back using FLOWC to bypass the C register.

¶ t_d is the clock cycle period.

- Page 7-41 – Replace the switching characteristics with the following:

switching characteristics

NO.	PARAM-ETER	FROM (INPUT)	TO (OUTPUT)	PIPELINE CONTROLS PIPES2-PIPES0	SN74ACT8847-50		UNIT
					MIN	MAX	
1	t_{pd1}	DA/DB/Inst	Y OUTPUT	111		\dagger	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 w/o feedback	100		ns
		CLK↑	CLK↑	010 w/feedback \ddagger	100		
		CLK↑	CLK↑	010 W/FLOWC \S	117		
10	t_{d2}	CLK↑	CLK↑	000 w/o feedback	50		ns
		CLK↑	CLK↑	000 w/feedback \ddagger	50		
		CLK↑	CLK↑	000 W/FLOWC \S	60		
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	$t_d-0\ddagger$	ns
12	t_{en1}	\overline{OEY}	Y OUTPUT	XXX		20	ns
13	t_{en2}	OEC, OES	STATUS	XXX		20	
14	t_{dis1}	\overline{OEY}	Y OUTPUT	XXX		20	
15	t_{dis2}	OEC, OES	STATUS	XXX		20	

\dagger This parameter no longer tested and will be deleted on next Data Manual revision.

\ddagger Applies to all feedback cases except where operands are fed back using FLOWC to bypass C register. (Please see Figure 13 for feedback paths).

\S Operands are fed back using FLOWC to bypass the C register.

\ddagger t_d is the clock cycle period.

TI Sales Offices

ALABAMA: Huntsville (205) 837-7530.

ARIZONA: Phoenix (602) 995-1007;
Tucson (602) 292-2640.

CALIFORNIA: Irvine (714) 660-1200;
Roseville (916) 655-9900;
San Diego (619) 279-9601;
Santa Clara (408) 980-9000;
Torrance (213) 217-7010;
Woodland Hills (818) 704-7759.

COLORADO: Aurora (303) 368-8000.

CONNECTICUT: Wallingford (203) 269-0074.

FLORIDA: Altamonte Springs (305) 260-2116;
Ft. Lauderdale (305) 973-8502;
Tampa (813) 885-7411.

GEORGIA: Norcross (404) 662-7900.

ILLINOIS: Arlington Heights (312) 640-2925.

INDIANA: Carmel (317) 573-6400;
Ft. Wayne (219) 424-5174.

IOWA: Cedar Rapids (319) 395-9550.

KANSAS: Overland Park (913) 451-4511.

MARYLAND: Columbia (301) 964-2003.

MASSACHUSETTS: Waltham (617) 895-9100.

MICHIGAN: Farmington Hills (313) 553-1569;
Grand Rapids (616) 957-4200.

MINNESOTA: Eden Prairie (612) 828-9300.

MISSOURI: St. Louis (314) 569-7600.

NEW JERSEY: Iselin (201) 750-1050.

NEW MEXICO: Albuquerque (505) 345-2555.

NEW YORK: East Syracuse (315) 463-9291;

Melville (516) 454-6600;

Pittsford (716) 385-6770;

Poughkeepsie (914) 473-2900.

NORTH CAROLINA: Charlotte (704) 527-0933;
Raleigh (919) 876-2725.

OHIO: Beachwood (216) 464-6100;

Beaver Creek (513) 427-6200.

OREGON: Beaverton (503) 643-6758.

PENNSYLVANIA: Blue Bell (215) 825-9500.

PUERTO RICO: Hato Rey (809) 753-8700.

TENNESSEE: Johnson City (615) 461-2192.

TEXAS: Austin (512) 250-7655;

Houston (713) 778-5592;

Richardson (214) 680-5082;

San Antonio (512) 496-1779.

UTAH: Murray (801) 266-8972.

WASHINGTON: Redmond (206) 881-3080.

WISCONSIN: Brookfield (414) 782-2899.

CANADA: Nepean, Ontario (613) 726-1970;

Richmond Hill, Ontario (416) 884-9181;

St. Laurent, Quebec (514) 336-1860.

TI Regional Technology Centers

CALIFORNIA: Irvine (714) 660-8105;

Santa Clara (408) 748-2220;

GEORGIA: Norcross (404) 662-7945.

ILLINOIS: Arlington Heights (312) 640-2909.

MASSACHUSETTS: Waltham (617) 895-9196.

TEXAS: Richardson (214) 680-5066.

CANADA: Nepean, Ontario (613) 726-1970.

TI Distributors

TI AUTHORIZED DISTRIBUTORS

Arrow/Kierulff Electronics Group
Arrow (Canada)
Future Electronics (Canada)
GRS Electronics Co., Inc.
Hall-Mark Electronics
Marshall Industries
Newark Electronics
Schweber Electronics
Time Electronics
Wyle Laboratories
Zeus Components

- OBSOLETE PRODUCT ONLY -
Rochester Electronics, Inc.
Newburyport, Massachusetts
(508) 462-9332

ALABAMA: Arrow/Kierulff (205) 837-6955;
Hall-Mark (205) 837-8700; Marshall (205) 881-9235;
Schweber (205) 895-0480.

ARIZONA: Arrow/Kierulff (602) 437-0750;
Hall-Mark (602) 437-1200; Marshall (602) 496-0290;
Schweber (602) 431-0030; Wyle (602) 866-2888.

CALIFORNIA: Los Angeles/Orange County:
Arrow/Kierulff (818) 701-7500, (714) 838-5422;
Hall-Mark (818) 773-4500, (714) 669-4100;
Marshall (818) 407-0101, (818) 459-5500;
(714) 458-5393; Schweber (818) 880-9686;
(714) 863-0200, (213) 320-8090; Wyle (818) 880-9000,
(714) 863-0200; Zeit (714) 921-9000; (818) 889-3838;
Sacramento: Hall-Mark (916) 624-9781;
Marshall (916) 635-3700; Schweber (916) 364-0222;
Wyle (916) 638-5273.

San Diego: Arrow/Kierulff (619) 565-4800;
Hall-Mark (619) 268-1201; Marshall (619) 578-9600;
Schweber (619) 450-0454; Wyle (619) 565-9171;

Hall-Mark (408) 252-0900; Marshall (408) 942-4600;
Schweber (408) 432-7711; Wyle (408) 727-2500;
Zeus (408) 998-5173.

COLORADO: Arrow/Kierulff (303) 790-4444;
Hall-Mark (303) 790-1652; Marshall (303) 451-9383;
Schweber (303) 799-0258; Wyle (303) 457-9953.

CONNECTICUT: Arrow/Kierulff (203) 265-7741;
Hall-Mark (203) 271-2844; Marshall (203) 265-3822;
Schweber (203) 264-4700.

FLORIDA: Ft. Lauderdale:
Arrow/Kierulff (305) 429-0200; Hall-Mark (305) 971-9280;
Orlando: Arrow/Kierulff (407) 323-0252;
Hall-Mark (407) 830-5855; Marshall (407) 767-8585;
Schweber (407) 331-7555; Zeus (407) 365-3000;
Tampa: Hall-Mark (813) 530-4543;
Marshall (813) 576-1399; Schweber (813) 541-5100.

GEORGIA: Arrow/Kierulff (404) 449-8252;
Hall-Mark (404) 447-8000; Marshall (404) 923-5750;
Schweber (404) 449-9170.

ILLINOIS: Arrow/Kierulff (312) 250-0500;
Hall-Mark (312) 860-3800; Marshall (312) 490-0155;
Newark (312) 784-5100; Schweber (312) 364-3750.

INDIANA: Indianapolis: Arrow/Kierulff (317) 243-9353;
Hall-Mark (317) 872-8855; Marshall (317) 297-0483;
Schweber (317) 843-1030.

IOWA: Arrow/Kierulff (319) 395-7230;

Schweber (319) 373-1417.

KANSAS: Kansas City: Arrow/Kierulff (913) 541-9542;
Hall-Mark (913) 888-4747; Marshall (913) 492-3121;
Schweber (913) 492-2922.

MARYLAND: Arrow/Kierulff (301) 995-6002;
Hall-Mark (301) 988-9800; Marshall (301) 235-9464;
Schweber (301) 840-5900; Zeus (301) 997-1118.

MASSACHUSETTS: Arrow/Kierulff (508) 658-0900;
Hall-Mark (508) 667-0202; Marshall (508) 658-0810;
Wyle (617) 275-5100; Time (617) 532-6200;
Zeus (617) 873-7300; Schweber (617) 863-8800.

MICHIGAN: Detroit: Arrow/Kierulff (313) 462-2290;
Hall-Mark (313) 462-1525; Marshall (313) 525-5850;
Newark (313) 967-9200; Schweber (313) 525-8100;
Grand Rapids: Arrow/Kierulff (616) 243-0912.

MINNESOTA: Arrow/Kierulff (612) 830-1800;
Hall-Mark (612) 941-2600; Marshall (612) 559-2211;
Schweber (612) 941-5280.

MISSOURI: St. Louis: Arrow/Kierulff (314) 567-6888;
Hall-Mark (314) 291-5350; Marshall (314) 291-4650;
Schweber (314) 739-0526.

NEW HAMPSHIRE: Arrow/Kierulff (603) 668-6968;
Schweber (603) 625-2250.

NEW JERSEY: Arrow/Kierulff (201) 538-0900;
(609) 596-8000; GRS Electronics (609) 964-8560;
Hall-Mark (201) 575-4415, (201) 882-9773;
(609) 235-1900; Marshall (201) 882-0320;
(609) 234-9170; Schweber (201) 227-7880.

NEW MEXICO: Arrow/Kierulff (505) 243-4566.

NEW YORK: Long Island:
Arrow/Kierulff (516) 231-1009; Hall-Mark (516) 737-0600;
Marshall (516) 273-2424; Schweber (513) 334-7474;
Zeus (914) 937-7400.

North Carolina: Arrow/Kierulff (919) 876-3132;
(919) 725-8711; Hall-Mark (919) 872-0712;
Marshall (919) 878-9882; Schweber (919) 876-0000.

OHIO: Cleveland: Arrow/Kierulff (216) 248-3990;
Hall-Mark (216) 349-4632; Marshall (216) 248-1788;
Schweber (216) 248-2974; Zeit (216) 888-3313;

Detroit: Arrow/Kierulff (313) 345-5563;
Marshall (513) 898-4480; Schweber (513) 439-1800.
OKLAHOMA: Arrow/Kierulff (918) 252-7537;
Schweber (918) 622-0003.

OREGON: Arrow/Kierulff (503) 645-6456;
Marshall (503) 644-0500; Wyle (503) 640-6000.

PENNSYLVANIA: Arrow/Kierulff (215) 472-8600;
(215) 928-1800; GRS Electronics (215) 922-7037;
Marshall (412) 963-0441; Schweber (215) 441-0600,
(412) 963-6804.

TEXAS: Austin: Arrow/Kierulff (512) 835-4180;
Hall-Mark (512) 258-8848; Marshall (512) 837-1991;
Schweber (512) 339-0088; Wyle (512) 834-9957;

Dallas: Arrow/Kierulff (214) 380-6464;
Hall-Mark (214) 553-4300; Marshall (214) 233-5200;
Schweber (214) 551-5010; Wyle (214) 235-9953;
(214) 783-7011.

Erl Paso: Marshall (915) 593-0706;
Houston: Arrow/Kierulff (713) 530-4700;
Hall-Mark (713) 781-6100; Marshall (713) 859-9200;
Schweber (713) 784-3600; Wyle (713) 879-9553.

UTAH: Arrow/Kierulff (801) 973-6913;
Hall-Mark (801) 972-1008; Marshall (801) 485-1551;
Wyle (801) 974-9953.

WASHINGTON: Arrow/Kierulff (206) 575-4420;
Marshall (206) 486-5747; Wyle (206) 881-1150.

WISCONSIN: Arrow/Kierulff (414) 792-0150;
Hall-Mark (414) 797-7844; Marshall (414) 797-8400;
Schweber (414) 784-9020.

CANADA: Calgary: Future (403) 235-5325;
Montreal: Arrow Canada (438) 428-2858;
Montreal: Arrow Canada (514) 735-5511;
Future (514) 694-7710;

Ottawa: Arrow Canada (613) 226-6903;
Quebec City: Arrow Canada (418) 871-7500;
Toronto: Arrow Canada (416) 672-7769;
Future (416) 438-4710; Marshall (416) 674-2161;
Vancouver: Arrow Canada (604) 291-2966;
Future (604) 294-1166.


**TEXAS
INSTRUMENTS**

Customer Response Center

TOLL FREE: (800) 232-3200

OUTSIDE USA: (214) 995-6611
(8:00 a.m. - 5:00 p.m. CST)

