TEXAS
INSTRUMENTS

SN74ACT8800 Family

32-Bit CMOS Processor
Building Blocks

Data Manual

1988

SN74ACT8800 Family
32-Bit CMOS Processor
Building Blocks

Data Manual

TEXAS
INSTRUMENTS

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

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.

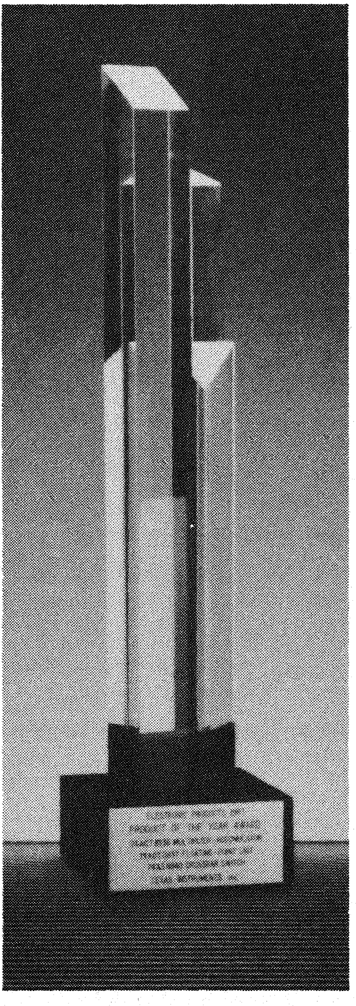
The *SN74ACT8800 Family* Data Manual contains design and specification data for all five devices previously listed and includes additional programming and operational information for the '8818, '8832, and '8837/'8847.

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 semicondutor product is available from your nearest TI field sales office, local authorized TI distributor, or by calling Texas Instruments at 1-800-232-3200.

Award Winners/1987



Electronic Products'

12th Annual Product of the Year Awards

Every year, the editors of *Electronic Products* select what we judge to be the best electronics products announced during the preceding 12 months. Our goal: to honor those products that have contributed most to help our engineer-readers do their jobs better.

We weigh several criteria as we sift through the thousands of products brought to our attention. Alone or in combination, each product must represent a significant advance in technology or its application, a decided innovation in design, or a substantial gain in price-performance ratio.

Alfred Rosenblatt

- **74ACT8836 Multiplier-Accumulator**
- **74ACT8837 Floating Point Unit**
- **74AS8840 Crossbar Switch**

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 μm 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 reliabilty, these bipolar devices use TI's proprietary Schottky Transistor Logic (STL) internal circuitry.

EPIC is a trademark of Texas Instruments Incorporated.

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
  - SN74AS8833 64-bit funnel shifter
  - SN74AS8834 64 × 40 register file
  - 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 μm 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

The SN74AS8833 64-bit-to-32-bit funnel shifter can increase overall speed in systems where multi-bit shift operations and field masking are frequently used. The device can perform logical, circular, and arithmetic shifts on 32-bit and 64-bit words, IEEE or IBM normalization, and field pack or extract operations. The 'AS8833 provides shift/mask/merge capability for graphics and data compression applications.

The SN74AS8834 is a high-speed, three-operand, 64-word by 40-bit register file. Designed to expand the 'ACT8832 register file, the 'AS8834 is an ideal temporary storage device for high-speed applications. Four address ports, two write and two read, operate independently to support MSH/LSH swap operations.

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.

SN74ACT8818
16-Bit Microsequencer

- Addresses Up to **64K** Locations of Microprogram Memory
- CLK-to-Y = **30 ns** (t_{pd})
- **Low-Power EPIC™** CMOS
- Addresses Selected from **Eight** Different Sources
- Performs Multiway Branching, Conditional Subroutine Calls, and Nested Loops
- Large **64-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.

Contents

Contents (Continued)

List of Illustrations

List of Tables

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

EPIC is a trademark of Texas Instruments Incorporated.

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.

Microprogramming 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

Texas Instruments Regional Technology Centers, staffed with systems-oriented engineers, offer a training course to assist users of TI's 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 LSI 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 the local Field Sales Engineer for current information or contact VLSI Systems Engineering at 214/997-3970.

'ACT8818 Pin Grid Allocation

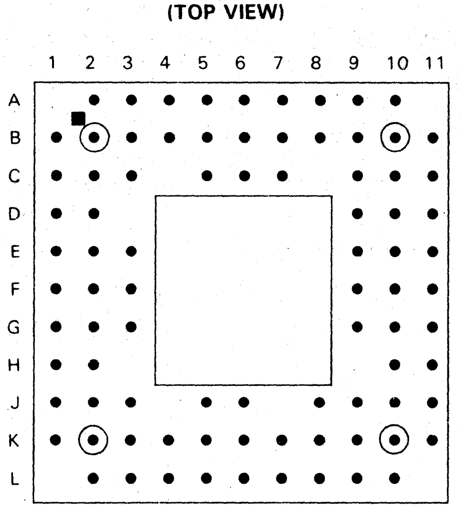


Figure 1. 'ACT8818 . . . . . GC Package

Table 1. 'ACT8818 Pin Grid Allocation

PIN		PIN		PIN		PIN	
NO.	**NAME**	**NO.**	**NAME**	**NO.**	**NAME**	**NO.**	**NAME**
A2	RC2	C2	RC0	F3	$\overline{\text{RBOE}}$	J10	S1
A3	Y1	C3	GND	F9	B0	J11	STKWRN/RER
A4	Y3	C5	GND	F10	B1	K1	DRB0
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	V_{CC}	G3	GND	K5	DRA10
A9	Y13	C11	$\overline{\text{RE}}$	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	$\overline{\text{CC}}$	K11	S2
B5	Y4	E1	DRB9	H11	ZEROUT	L2	DRA15
B6	$\overline{\text{YOE}}$	E2	DRB10	J1	DRB2	L3	DRA13
B7	Y9	E3	DRB11	J2	DRB1	L4	DRA11
B8	Y12	E9	$\overline{\text{INT}}$	J3	V_{CC}	L5	DRA9
B9	Y14	E10	B3	J5	GND	L6	DRA8
B10	Y15	E11	B2	J6	$\overline{\text{RAOE}}$	L7	DRA6
B11	ZEROIN	F1	DRB7	J8	DRA1	L8	DRA4
C1	DRB14	F2	DRB8	J9	GND	L9	DRA2
						L10	OSEL

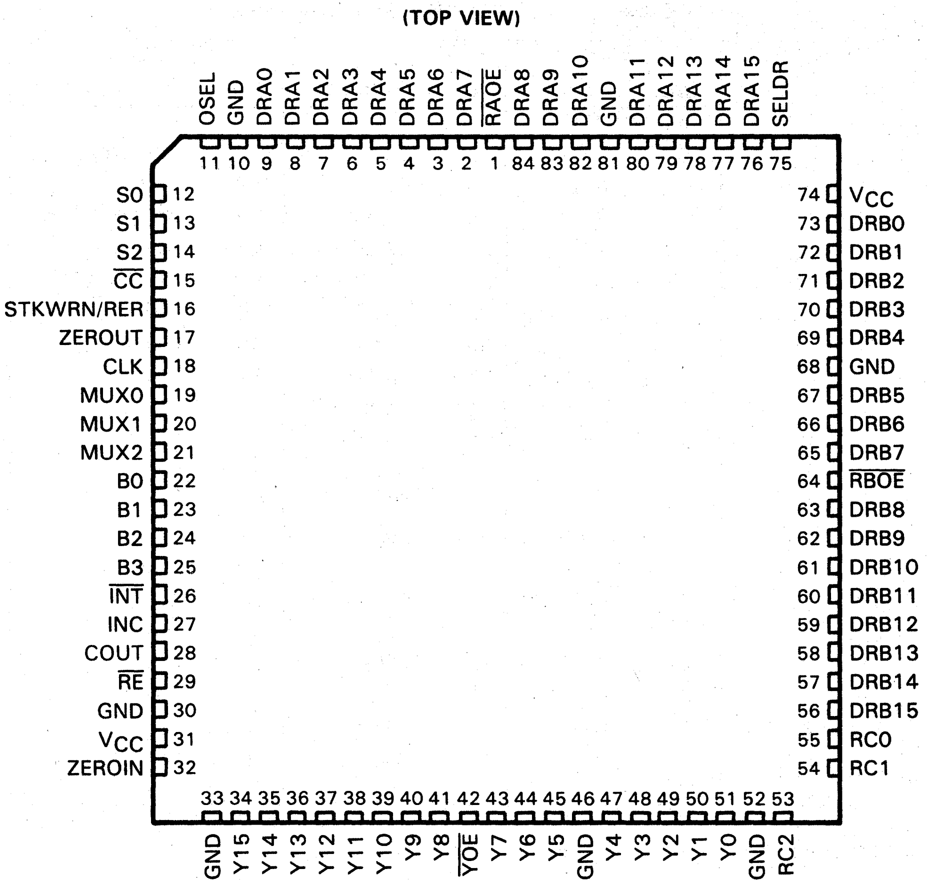


Figure 2. 'ACT8818 . . . FN Package

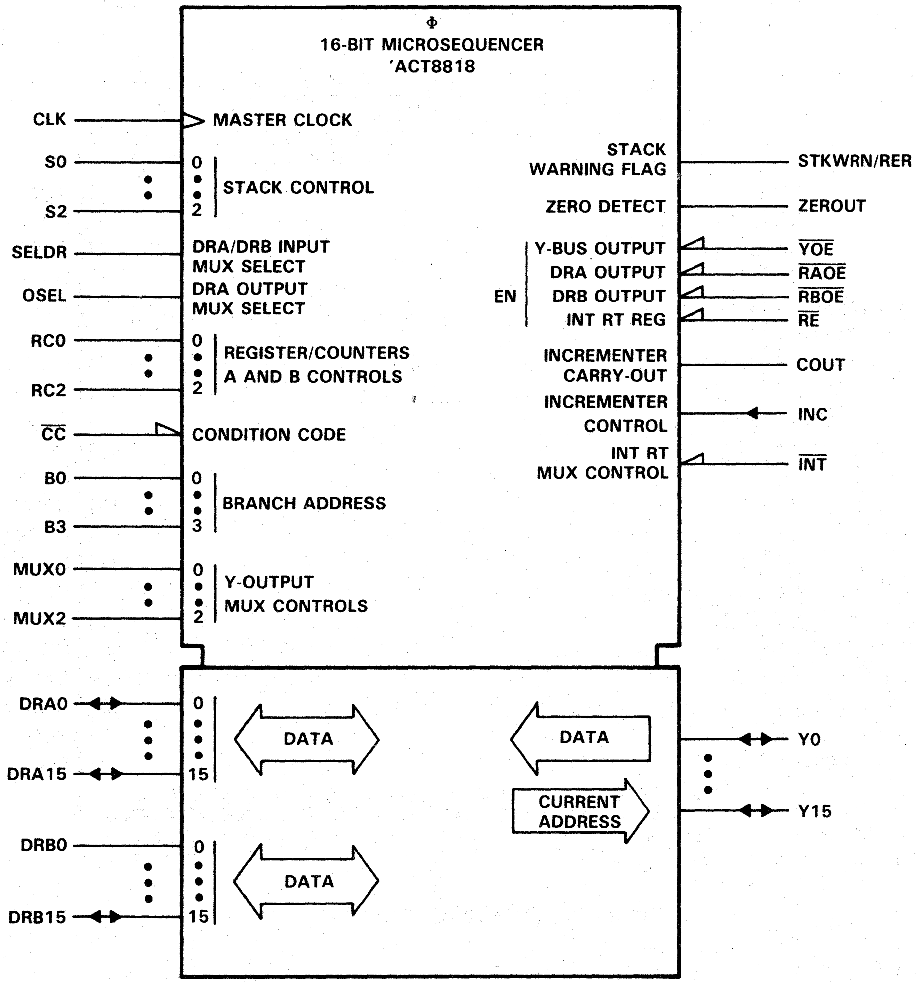


Figure 3. 'ACT8818 . . . Logic Symbol

Table 2. 'ACT8818 Pin Functional Description

PIN NAME	GC NO.	FN NO.	I/O	DESCRIPTION
B0	F9	22	I	Input bits for branch addressing (see Table 3)
B1	F10	23		
B2	E11	24		
B3	E10	25		
CLK	G9	18		System clock
COUT	D10	28	0	Incrementer carry-out. Goes high when an attempt is made to increment microprogram counter beyond addressable micromemory.
$\overline{CC}$	H10	15	I	Condition code
DRA0	K9	9	I/O	Bidirectional DRA data port. Outputs data from stack or register/counter A ($\overline{RAOE} = 0$) or inputs external data ($\overline{RAOE} = 1$).
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		
DRB0	K1	73	I/O	Bidirectional DRB data port. Outputs data from register/counter B ($\overline{RBOE} = 0$) or inputs external data
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		

Table 2. 'ACT8818 Pin Functional Description (Continued)

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

Table 2. 'ACT8818 Pin Functional Description (Concluded)

PIN NAME	GC NO.	FN NO.	I/O	DESCRIPTION
Y0	B3	51	I/O	Bidirectional Y data port
Y1	A3	50		
Y2	B4	49		
Y3	A4	48		
Y4	B5	47		
Y5	A5	45		
Y6	A6	44		
Y7	C6	43		
Y8	A7	41		
Y9	B7	40		
Y10	C7	39		
Y11	A8	38		
Y12	B8	37		
Y13	A9	36		
Y14	B9	35		
Y15	B10	34		
$\overline{\text{YOE}}$	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	COUT	
t_{pd}	$\overline{CC}$	23						
	CLK	27 30†	 23†	24	16	25		
	DRA15-DRA0	23						ns
	DRB15-DRB0	22						
	MUX2-MUX0	22						
	RC2-RC0	26	18					
	S2-S0	25		19				
	B3-B0	19						
	OSEL	25		20				
	ZEROIN	25						
	SELDR	23						
	INC						20	
	Y						16	
t_{en}	$\overline{YOE}$	16						ns
	$\overline{RAOE}$			18				
	$\overline{RBOE}$				17			
t_{dis}	$\overline{YOE}$	14						ns
	$\overline{RAOE}$			13				
	$\overline{RBOE}$				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-DRB0	RCB	7		
		INT RT	11		
	INC	MPC	7		
	$\overline{INT}$	Stack	7		
	RC2-RC0	Stack	15		
		RCA, RCB	6		
		INT RT	16		
	S2-S0	Stack	13		
		INT RT	13		
	OSEL	Stack	12		
		INT RT	13		
	B3-B0	Stack	8		
		INT RT	14		
	SELDR	Stack	10		
		INT RT	10		
	ZEROIN	Stack	14		
		INT RT	13		
	Y	MPC	6		
	$\overline{RE}$	INT RT (CLK)	7		
	MUX2-MUX0	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, $\overline{\text{RAOE}}$ and $\overline{\text{RBOE}}$; and $\overline{\text{INT}}$, 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 $\overline{\text{CC}}$, 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 $\overline{\text{CC}}$ 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, $\overline{\text{CC}}$ will generally be controlled by the output of a status multiplexer. In this case, whether $\overline{\text{CC}}$ 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 $\overline{INT}$)	No effect
$\overline{CC}$	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.
$\overline{INT}$[‡]	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
$\overline{RAOE}$	DRA output disabled (high-Z)	DRA output enabled
$\overline{RBOE}$	DRB output disabled (high-Z)	DRB output enabled
RC2-RC0	See Table 6	See Table 6
$\overline{RE}$	Hold interrupt return register contents	Load address on Y bus to interrupt return register
S2-S0	See Table 5	See Table 5
SELDR	Selects DRA/DRB external data as inputs to DRA/DRB buses	Selects RCA (OSEL low) or stack (OSEL high) to DRA bus, RCB to DRA bus
$\overline{YOE}$	Y output disabled (high-Z)	Y output enabled
ZEROIN	Sets ZERO 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 $\overline{YOE}$, which allows interrupt hardware to force an interrupt vector on the microaddress bus.

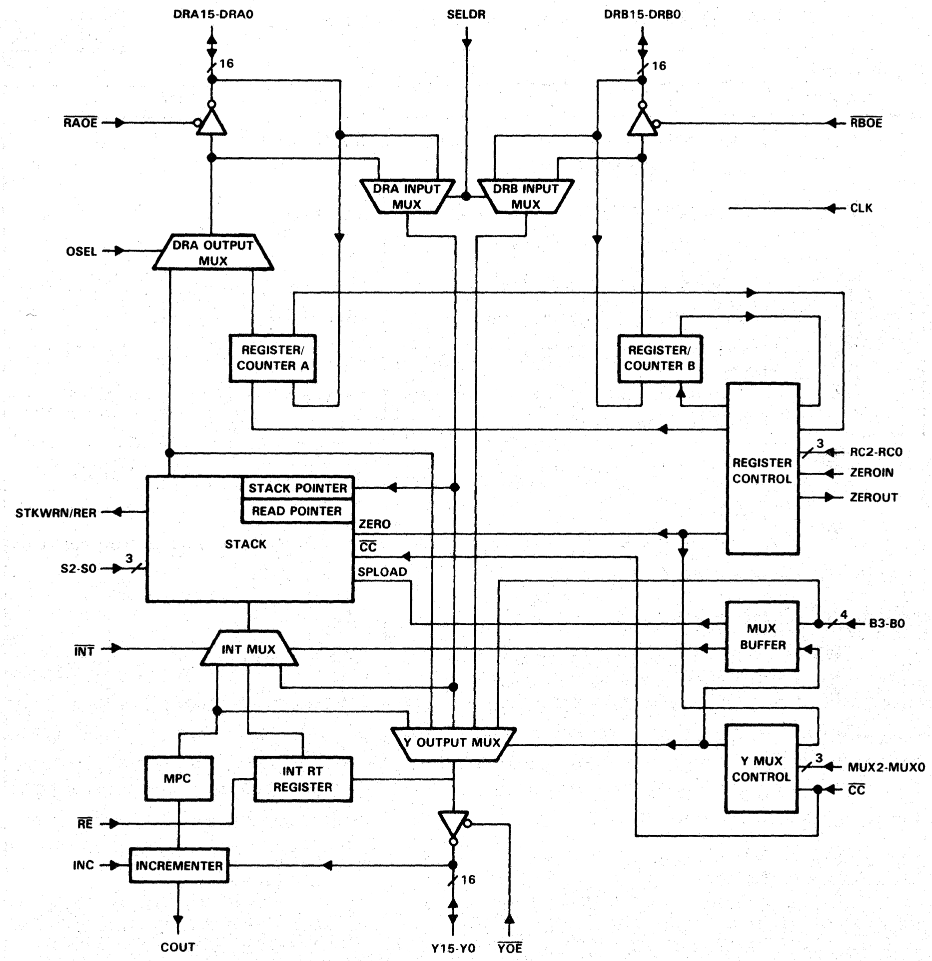


Figure 4. 'ACT8818 Functional Block Diagram

Status from the 'ACT8818 is provided by ZEROUT, 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.

Microprogram 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 (0-67) 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 63. 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 second condition in which the STKWRN/RER signal goes high is to indicate that the last location has been popped from the stack and the stack is empty. The user may be protected from attempting to pop an empty stack by monitoring STKWRN/RER before pop operations. A high level at this pin signifies that the last address has been removed from the stack (SP = 0). This condition remains until an address is pushed onto the stack and the stack pointer is incremented to one.

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 $\overline{\text{INT}}$ low and perform a push to put the contents of the interrupt return register on the stack.

Microprogramming the 'ACT8818

Microprogramming is unlike programming monolithic processors for several reasons. First, the width of the microinstuction 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 ← DRA
BR A'	Y ← DRA'
BR B	Y ← DRB
BR B'	Y ← DRB'
BR S	Y ← STK
CALL A	Y ← DRA; STK ← MPC; SP ← SP + 1
CALL B	Y ← DRB; STK ← MPC; SP ← SP + 1
CALL A'	Y ← DRA'; STK ← MPC; SP ← SP + 1
CALL B'	Y ← DRB'; STK ← MPC; SP ← SP + 1
CALL S	Y ← STK; STK ← MPC; SP ← SP + 1
CLR SP, RP	SP ← 0; RP ← 0
CONT/RPT	Y ← MPC + 1 if INC = H; Y ← MPC 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 ← SP − 1
PUSH	STK ← MPC; SP ← SP + 1
RCA	Register/counter A
RCB	Register/counter B
READ	Y ← STK; RP ← RP − 1
RESET	Y ← 0; SP ← 0; RP ← 0
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.

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 continues 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. Continue/repeat instructions are summarized in Table 7 below.

Table 7. Continue/Repeat Encodings

MUX2-MUX0	S2-S0	OSEL	$\overline{CC}$ = H
LHL	LLH	X	CONT/RPT
LHL	LHL	X	CONT/RPT: POP
LHL	HLL	X	CONT/RPT: PUSH
LHL	HHH	0	CONT/RPT
LHL	HHH	1	CONT/RPT: READ
LHH	LLH	X	CONT/RPT
LHH	LHL	X	CONT/RPT: POP
LHH	HLL	X	CONT/RPT: PUSH
LHH	HHH	0	CONT/RPT
LHH	HHH	1	CONT/RPT: READ
HHL	LLH	X	CONT/RPT
HHL	LHL	X	CONT/RPT: POP
HHL	LHH	X	CONT/RPT
HHL	HLL	X	CONT/RPT: PUSH
HHL	HHH	0	CONT/RPT
HHL	HHH	1	CONT/RPT: READ
HHH	LLH	X	CONT/RPT
HHH	LHL	X	CONT/RPT: POP
HHH	LHH	X	CONT/RPT
HHH	HLL	X	CONT/RPT: PUSH
HHH	HHH	0	CONT/RPT
HHH	HHH	1	CONT/RPT: READ

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 microprogram, 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. Branch instructions are summarized in Table 8, below. Call (Branch and Push MPC) instructions and Return (Branch to Stack and Pop) instructions are discussed in later sections.

Table 8. Branch Encodings

MUX2-MUX0	S2-S0	OSEL	$\overline{CC}$ = H
LLL	LLH	X	BR A
LLL	LHL	X	BR A: POP
LLL	HHH	0	BR A
LLL	HHH	1	BR A: READ
LLH	LLH	X	BR B
LLH	LHL	X	BR B: POP
LLH	HHH	0	BR B
LLH	HHH	1	BR B: READ
HLL	LLH	X	BR B
HLL	LHL	X	BR B: POP
HLL	LHH	X	BR B
HLL	HHH	0	BR B
HLL	HHH	1	BR B: READ
HLH	LLH	X	BR B′ (16-way)
HLH	LHL	X	BR B′ (16-way) : POP
HLH	LHH	X	BR B′ (16-way)
HLH	HHH	0	BR B′ (16-way)
HLH	HHH	1	BR B′ (16-way): READ
LLL	LLH	X	BR S: CLR SP/RP
LLL	LHL	X	BR S

Table 8. Branch Encodings (Continued)

MUX2-MUX0	S2-S0	OSEL	$\overline{CC}$ = H
LLL	HLL	X	BR S
LLL	HHH	0	BR S
LLL	HHH	1	BR S: READ
LLH	LLH	X	BR S: CLR SP/RP
LLH	LHL	X	BR S
LLH	HLL	X	BR S
LLH	HHH	0	BR S
LLH	HHH	1	BR S: READ
LHL	LLH	X	BR S: CLR SP/RP
LHL	LHL	X	BR S
LHL	HLL	X	BR S
LHL	HHH	0	BR S
LHL	HHH	1	BR S: READ
LHH	LLH	X	BR S: CLR SP/RP
LHH	LHL	X	BR S
LHH	HLL	X	BR S
LHH	HHH	0	BR S
LHH	HHH	1	BR S: READ
HLL	LLH	X	BR A: CLR SP/RP
HLL	LHL	X	BR A
HLL	LHH	X	BR A: POP
HLL	HLL	X	BR A
HLL	HHH	0	BR A
HLL	HHH	1	BR A: READ
HLH	LLH	X	BR A′ (16-way): CLR SP/RP
HLH	LHL	X	BR A′ (16-way)
HLH	LHH	X	BR A′ (16-way): POP
HLH	HLL	X	BR A′ (16-way)
HLH	HHH	0	BR A′ (16-way)
HLH	HHH	1	BR A′ (16-way): READ
HHL	LLH	X	BR A: CLR SP/RP
HHL	LHL	X	BR A
HHL	LHH	X	BR A: POP
HHL	HLL	X	BR A
HHL	HHH	0	BR A
HHL	HHH	1	BR A: READ

Table 8. Branch Encodings (Concluded)

MUX2-MUX0	S2-S0	OSEL	$\overline{CC}$ = H
HHH	LLH	X	BR B: CLR SP/RP
HHH	LHL	X	BR B
HHH	LHH	X	BR B: POP
HHH	HLL	X	BR B
HHH	HHH	0	BR B
HHH	HHH	1	BR B: READ

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. Conditional branching codes are listed in Table 9. Call (Branch and Push MPC) instructions and Return (Branch to Stack and Pop) instructions are discussed in later sections.

Table 9. Conditional Branch Encodings

MUX2-MUX0	S2-S0	OSEL	$\overline{CC}$ = L	$\overline{CC}$ = H
LLL	LLH	X	BR S: CLR SP/RP	BR A
LLL	LHL	X	BR S	BR A: POP
LLL	HLL	X	BR S	CALL A
LLL	HHH	0	BR S	BR A
LLL	HHH	1	BR S: READ	BR A: READ
LLH	LLH	X	BR S: CLR SP/RP	BR B
LLH	HHH	0	BR S	BR B
LLH	LHL	X	BR S	BR B: POP
LLH	HLL	X	BR S	CALL B
LLH	HHH	1	BR S: READ	BR B: READ
LHL	LLH	X	BR S: CLR SP/RP	CONT/RPT

Table 9. Conditional Branch Encodings (Concluded)

MUX2-MUX0	S2-S0	OSEL	$\overline{\text{CC}}$ = L	$\overline{\text{CC}}$ = H
LHL	LHL	X	BR S	CONT/RPT: POP
LHL	HLL	X	BR S	CONT/RPT: PUSH
LHL	HHH	0	BR S	CONT/RPT
LHL	HHH	1	BR S: READ	CONT/RPT: READ
LHH	LLH	X	BR S: CLR SP/RP	CONT/RPT
LHH	LHL	X	BR S	CONT/RPT: POP
LHH	HLL	X	BR S	CONT/RPT: PUSH
LHH	HHH	0	BR S	CONT/RPT
LHH	HHH	1	BR S: READ	CONT/RPT: READ
HLL	LLH	X	BR A: CLR SP/RP	BR B
HLL	LHL	X	BR A	BR B: POP
HLL	LHH	X	BR A: POP	BR B
HLL	HLL	X	BR A	CALL B
HLL	HHH	0	BR A	BR B
HLL	HHH	1	BR A: READ	BR B: READ
HLH	LLH	X	BR A′ (16-way): CLR SP/RP	BR B′ (16-way)
HLH	LHL	X	BR A′ (16-way)	BR B′ (16-way): POP
HLH	LHH	X	BR A′ (16-way): POP	BR B′ (16-way)
HLH	HLL	X	BR A′ (16-way)	CALL B′ (16-way)
HLH	HHH	0	BR A′ (16-way)	BR B′ (16-way)
HLH	HHH	1	BR A′ (16-way): READ	BR B′ (16-way): READ
HHL	LLH	X	BR A: CLR SP/RP	CONT/RPT
HHL	LHL	X	BR A	CONT/RPT: POP
HHL	LHH	X	BR A: POP	CONT/RPT
HHL	HLL	X	BR A	CONT/RPT: PUSH
HHL	HHH	0	BR A	CONT/RPT
HHL	HHH	1	BR A: READ	CONT/RPT: READ
HHH	LLH	X	BR B: CLR SP/RP	CONT/RPT
HHH	LHL	X	BR B	CONT/RPT: POP
HHH	LHH	X	BR B: POP	CONT/RPT
HHH	HLL	X	BR B	CONT/RPT: PUSH
HHH	HHH	0	BR B	CONT/RPT
HHH	HHH	1	BR B: READ	CONT/RPT: READ

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 10. 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 10. 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
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
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: 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	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 10. Decrement and Branch on Nonzero Encodings (Concluded)

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 11 and 12.

Table 11. Call Encodings without Register Decrements

MUX2-MUX0	S2-S0	OSEL	$\overline{CC}$ = 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 12. Call Encodings with Register Decrements

MUX2-MUX0	S2-S0	OSEL	$\overline{\text{CC}}$ = L		$\overline{\text{CC}}$ = 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 $\overline{\text{CC}}$ 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 13 and 14.

Table 13. Return Encodings without Register Decrements

MUX2-MUX0	S2-S0	OSEL	$\overline{\text{CC}}$ = L	$\overline{\text{CC}}$ = 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 14. Return Encodings with Register Decrements

MUX2-MUX0	S2-S0	OSEL	$\overline{\text{CC}}$ = L		$\overline{\text{CC}}$ = 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

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 $\overline{\text{CC}}$ 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 $\overline{\text{YOE}}$ 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 $\overline{\text{INT}}$ 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), this example uses an instruction in Table 7 with CONT/RPT in the instruction column and no stack operation. 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), this example uses an instruction in Table 7 with CONT/RPT: POP in the instruction column. 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), this example uses an instruction in Table 7 with CONT/RPT: PUSH in the instruction column. 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

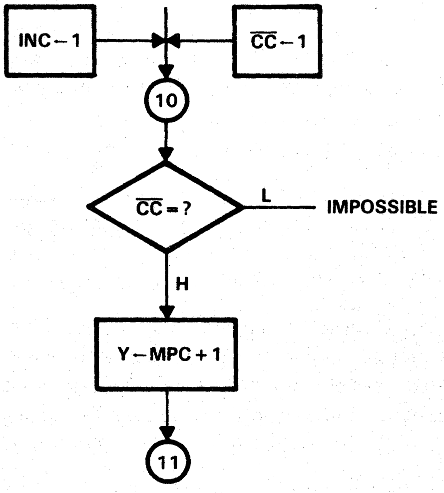


Figure 5. Continue

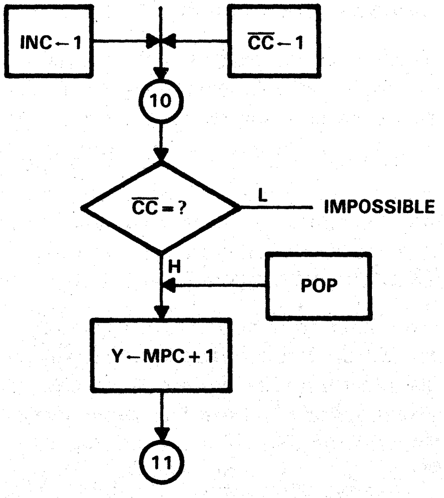


Figure 6. Continue and Pop

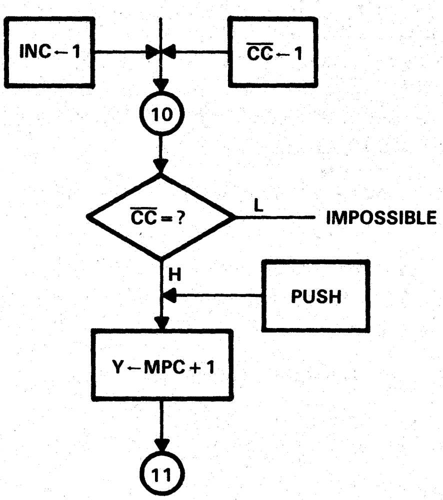


Figure 7. Continue and Push

Branch (Example 1)

To Branch from address 10 to address 20, this example uses a BR A instruction from the $\overline{CC}$ = H column of Table 8. $\overline{CC}$ 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)

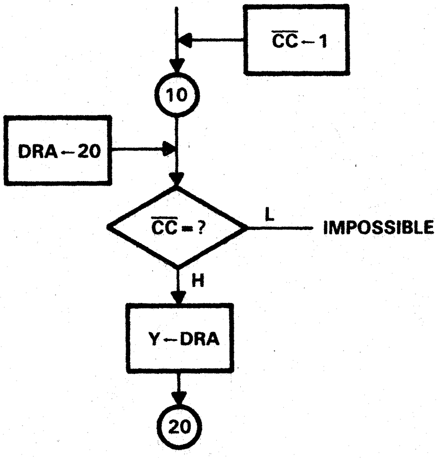
To Branch from address 10 to address 20, this example uses a BR A instruction from the $\overline{CC}$ = L column of Table 8. $\overline{CC}$ is programmed low in the previous instruction; as a result, a ZERO test follows the condition code test in Instruction 10. To ensure that a ZERO = H condition will not occur, registers should not be decremented during this instruction.

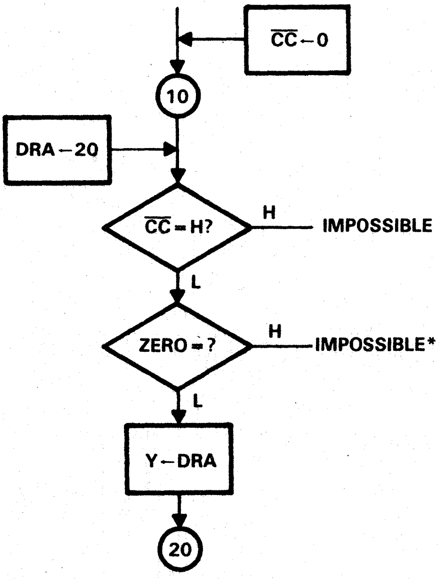
Address	Instruction	MUX2-MUX0	S2-S0	R2-R0	OSEL	$\overline{CC}$	INC	DRA	DRB
(Set-up)		XXX	XXX	XXX	X	0	X	XXXX	XXXX
10	BR A	110	111	000	0	X	X	0020	XXXX

Sixteen-Way Branch

To Branch 16-Way, this example uses a BR B′ instruction in Table 8. $\overline{CC}$ 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





*no register decrement

Figure 8. Branch Example 1

Figure 9. Branch Example 2

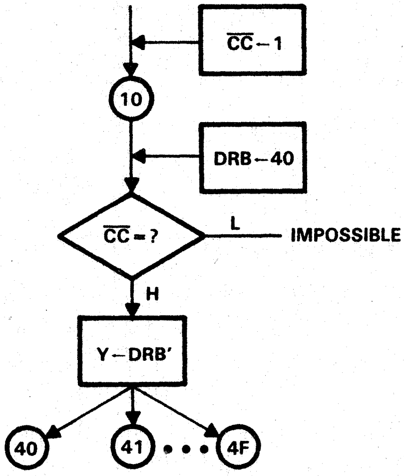


Figure 10. Sixteen-Way Branch

Conditional Branch

To Branch to address 20 Else Continue to address 13, this example uses the first instruction from Table 9 with BR A in the $\overline{CC}$ = L column and CONT/RPT in the $\overline{CC}$ = H column. INC is set high in the preceding instruction 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	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 10 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.

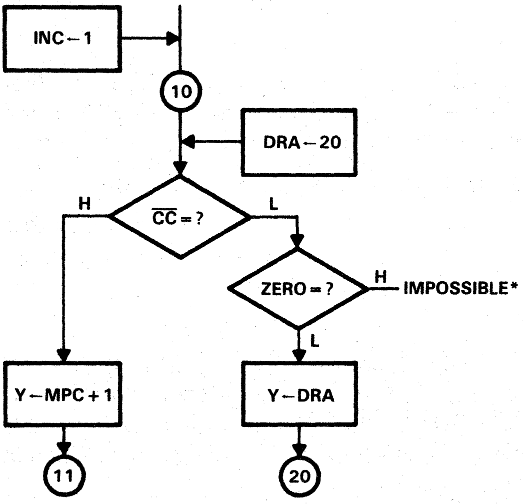
Address	Instruction	MUX2-MUX0	S2-S0	R2-R0	OSEL	$\overline{CC}$	INC	DRA	DRB
(Set-up)		XXX	XXX	XXX	X	1	1	XXXX	XXXX
10	Continue and Load Reg A	110	111	010	0	†	1	XXXX	XXXX
11	Decrement Reg A; Branch 3-Way	100	111	001	0	X	X	0020	0030

†Selected from external status

Thirty-Two-Way Branch

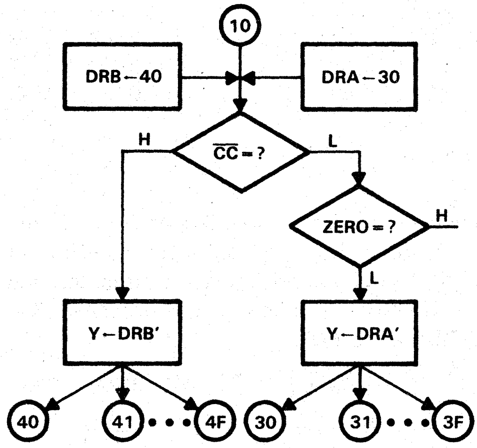
To Branch 32-Way, this example uses an instruction from Table 9 with BR A′ in the $\overline{CC}$ = L column and BR B′ in the $\overline{CC}$ = H column. The four least significant bits of the DRA′ and DRB′ addresses must be input at the B3-B0 port; these are concatenated with the 12 most significant 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	1	1	XXXX	XXXX
10	32-way Branch	101	111	000	0	X	X	0040	0030



*no register decrement

Figure 11. Conditional Branch



*no register decrement

Figure 13. Thirty-Two-Way Branch

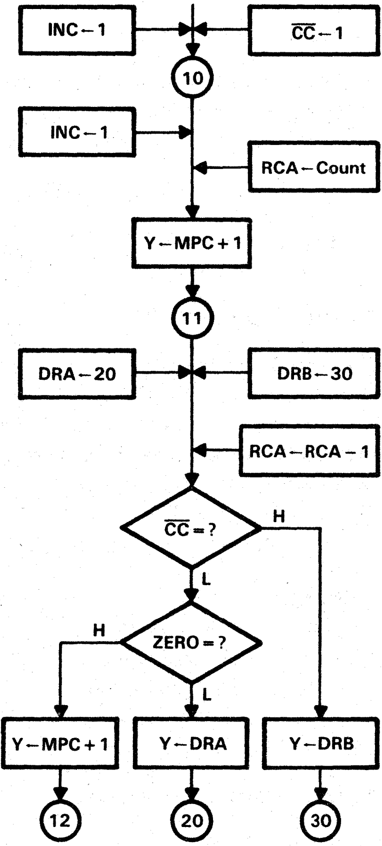


Figure 12. Three-Way Branch

Repeat

To Repeat (Instruction 10), this example uses an instruction in Table 7 with CONT/RPT in the instruction column. 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	1	XXXX	XXXX

Repeat on Stack

To Continue and push the microprogram counter onto the stack (Push), this example uses an instruction in Table 7 with CONT/RPT: PUSH in the instruction column. INC and $\overline{CC}$ must be forced high one cycle ahead for pipelining.

To Repeat (Instruction 12), an BR S instruction from the ZERO = L column of Table 8 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	X	XXXX	XXXX
12	BR Stack	010	111	000	0	1	X	XXXX	XXXX

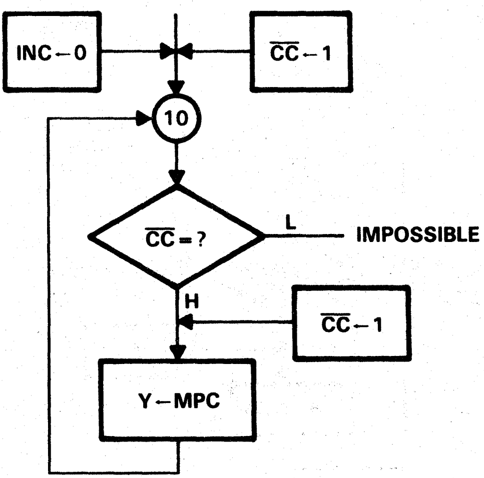
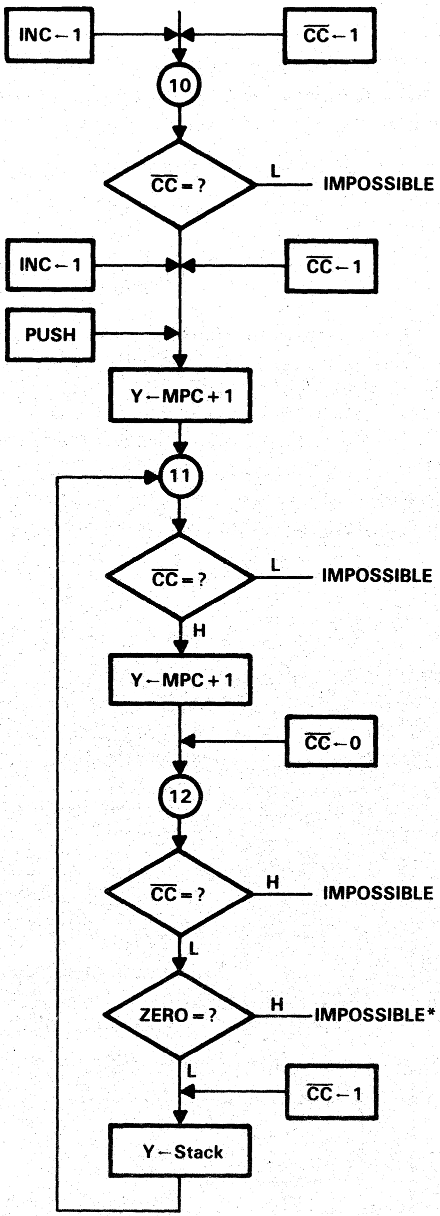


Figure 14. Repeat



*no register decrement

Figure 15. Repeat on Stack

Repeat Until $\overline{CC}$ = H

To Continue and push the microprogram counter onto the stack (Push), this example uses an instruction in Table 7 with CONT/RPT: PUSH in the instruction column. INC and $\overline{CC}$ must be forced high one cycle ahead for pipelining.

To Repeat Until $\overline{CC}$ = H (Instruction 12), an instruction from Table 9 with BR S in the $\overline{CC}$ = L column and CONT/RPT: POP in the $\overline{CC}$ = H column is used. 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	111	000	0	1	1	XXXX	XXXX

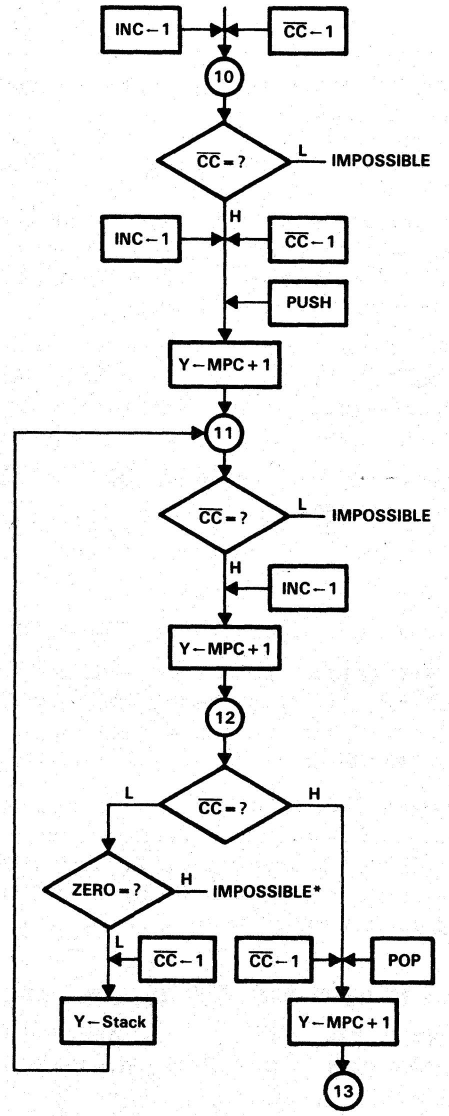
†Selected from external status

Loop Until Zero

To Continue and push the microprogram counter onto the stack (Push), this example uses an instruction in Table 7 with CONT/RPT: PUSH in the instruction column. 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 9 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



*no register decrement

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

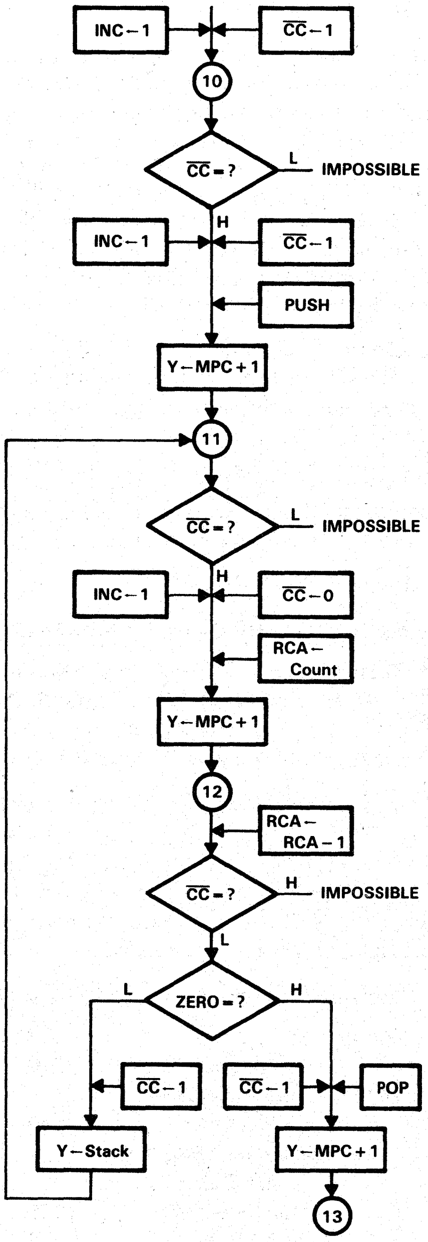


Figure 17. Loop Until Zero

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), this example uses an instruction in Table 7 with CONT/RPT: PUSH in the instruction column. INC must be high. $\overline{CC}$ is forced high in the preceding instruction for pipelining.

To Continue (Instruction 11), this example uses an instruction in Table 7 with CONT/RPT in the instruction column. 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 10 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	111	010	0	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	X	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	111	010	0	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	X	X	0025	XXXX

†Selected from external status

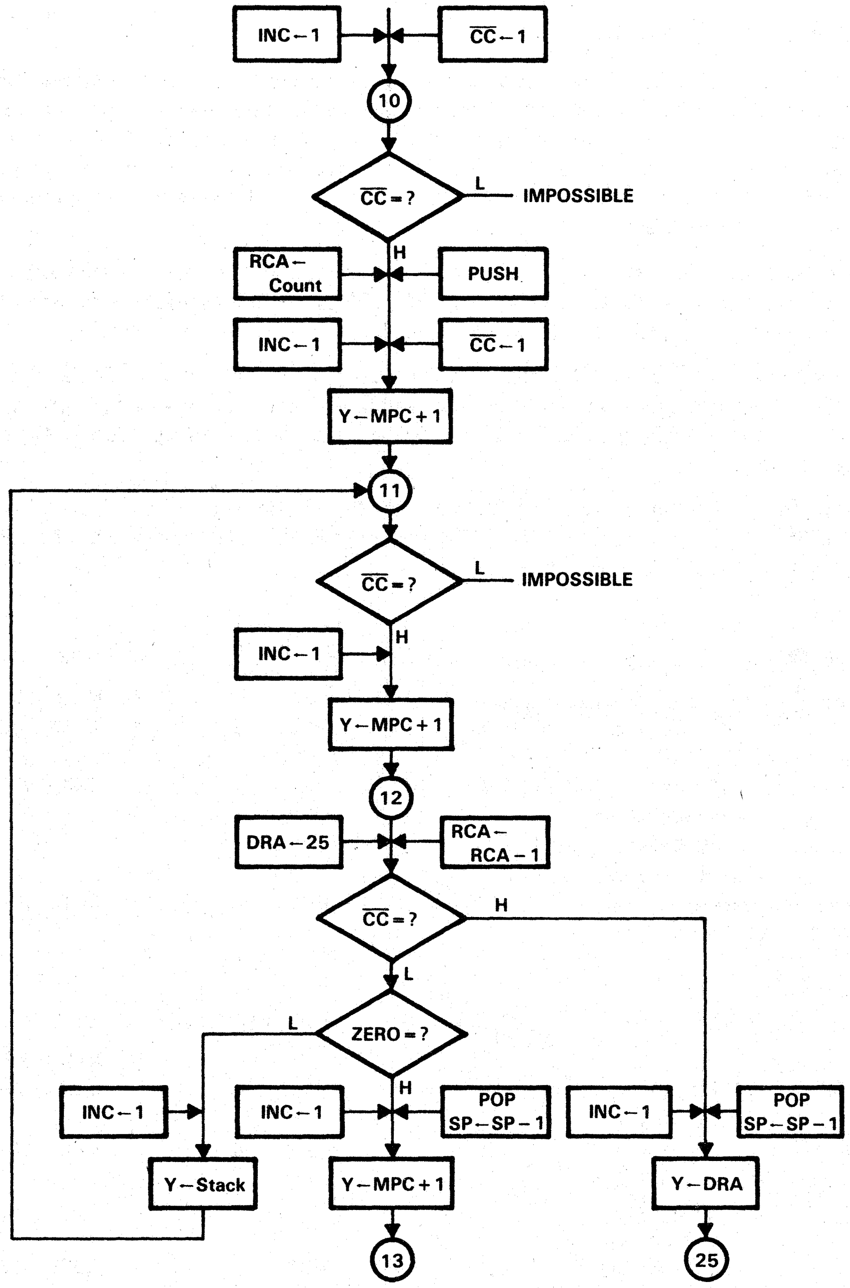


Figure 18. Conditional Loop Until Zero (Example 2)

Jump to Subroutine

To Call a Subroutine at address 30, this example uses the instruction from Table 11 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-R0	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 11 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-R0	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	1	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 11 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-R0	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

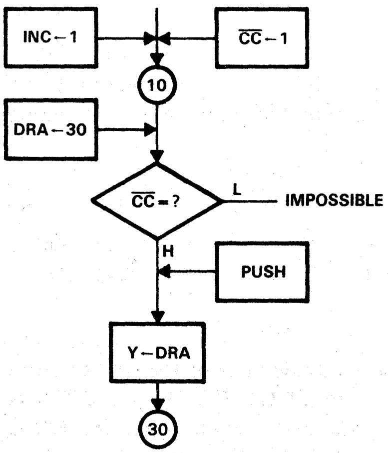
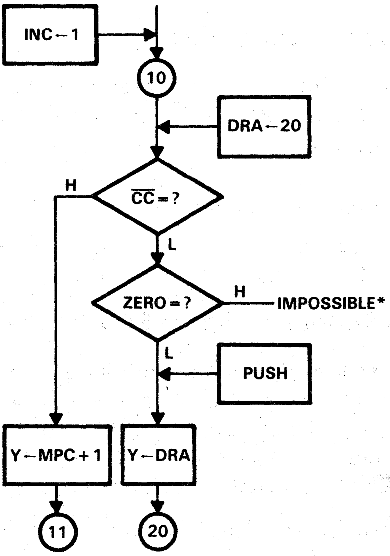
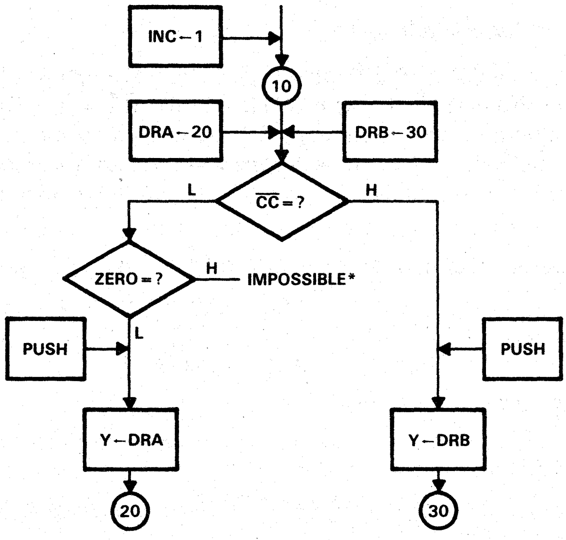


Figure 19. Jump to Subroutine



*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 13 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	0	X	XXXX	XXXX

Conditional Return from Subroutine

To conditionally Return from a Subroutine, this example uses an instruction from Table 13 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	1	X	XXXX	XXXX

†Selected from external status

Clear Pointers

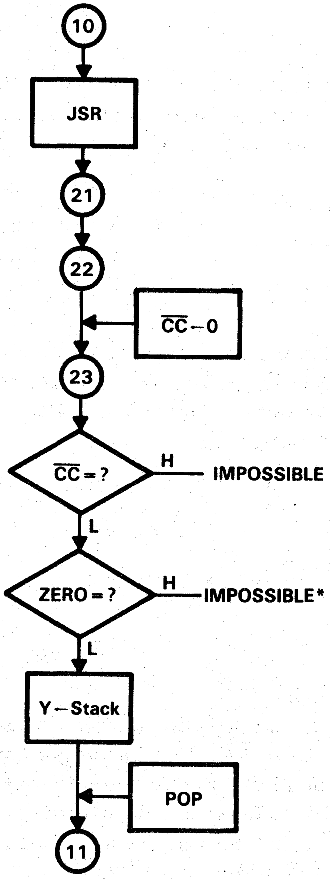
To Continue (Instruction 10), this example uses an instruction in Table 7 with CONT/RPT in the instruction column. INC must be high; $\overline{CC}$ must be programmed high in the previous instruction. To Clear the Stack and Read Pointers and Branch to address 40 (instruction 11), this example uses a BR A: Clear SP, RP instruction in Table 8. $\overline{CC}$ 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	000	0	0	X	0020	XXXX
11	BR A and Clear SP/RP	110	001	000	X	X	X	XXXX	XXXX

Reset

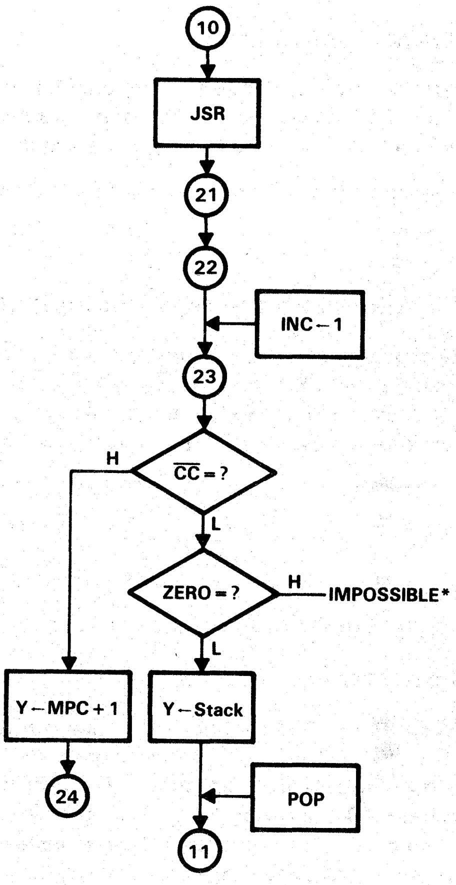
To Reset the 'ACT8818, pull the S2-S0 pins low. This clears the stack and read pointers and places the Y bus into a low state.

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



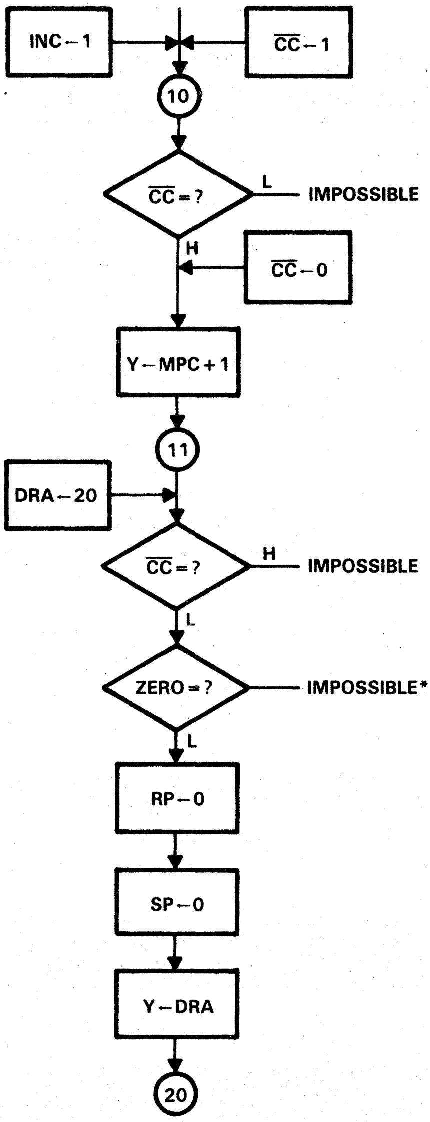
*no register decrement

Figure 22. Return from Subroutine



*no register decrement

Figure 23. Conditional Return from Subroutine



*no register decrement

Figure 24. Clear Pointers

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 'ACT8836 to create an instruction and store the previous result in a single cycle.

EPIC is a trademark of Texas Instruments Incorporated.

Contents

Contents (Continued)

Contents (Concluded)

List of Illustrations

List of Tables

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.

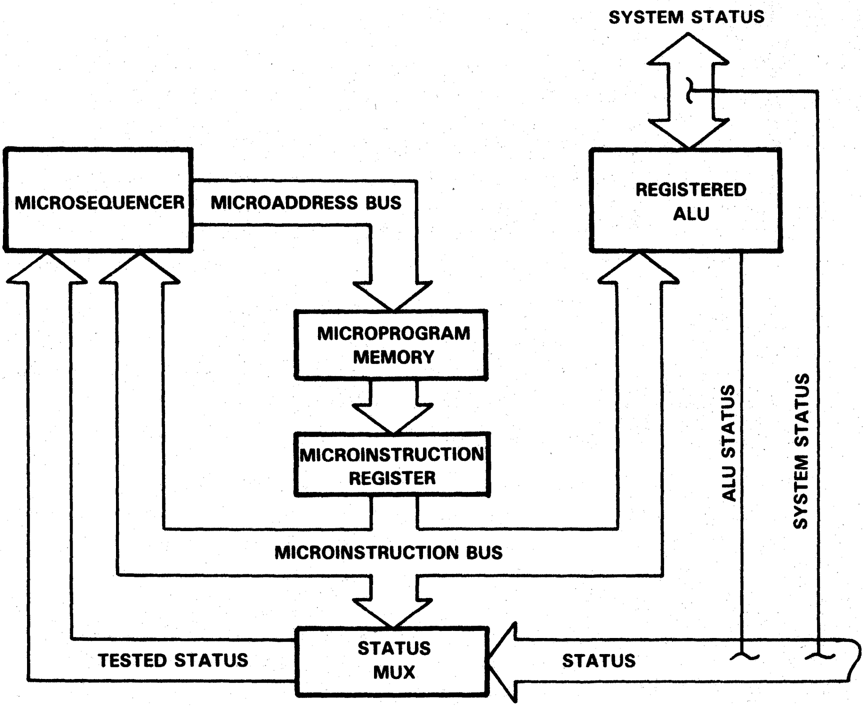


Figure 1. Microprogrammed System Block Diagram

The configuration of this processor enchances 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 'AS8800 family are also planned for future development.

Design Support

Texas Instruments Regional Technology Centers, staffed with systems-oriented engineers, offer a training course to assist users of TI's 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 LSI 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 St. 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.

'ACT8832 Pin Descriptions

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

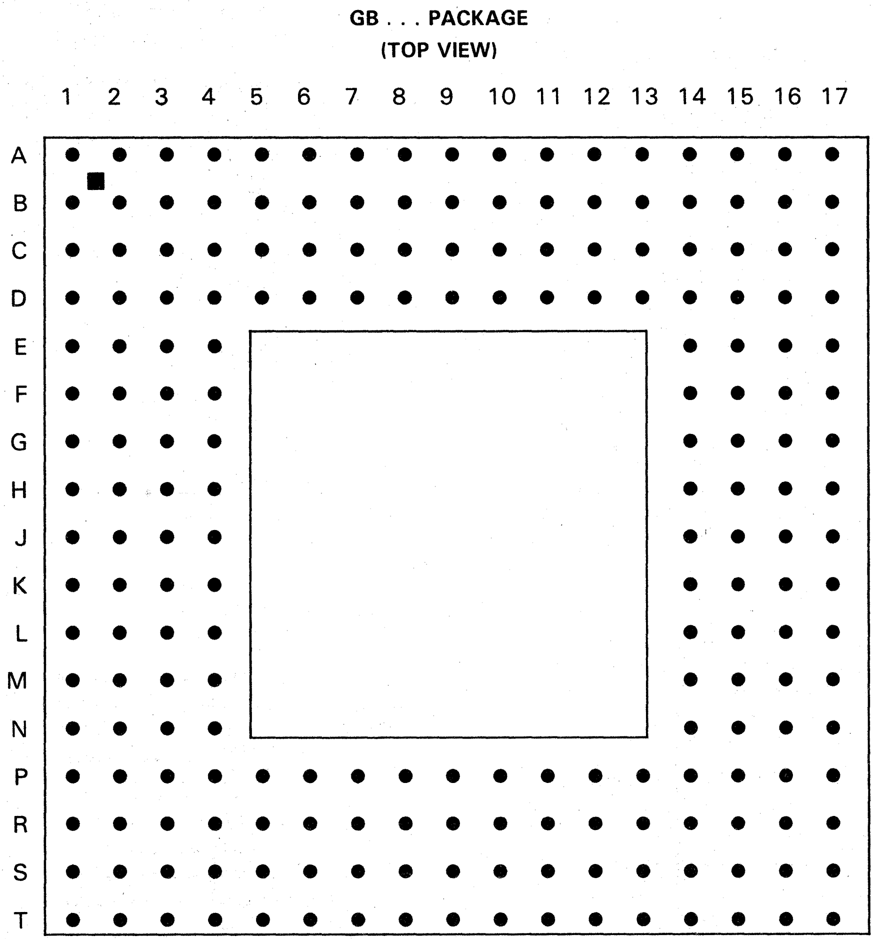


Figure 2. SN74ACT8832 . . . GB Package

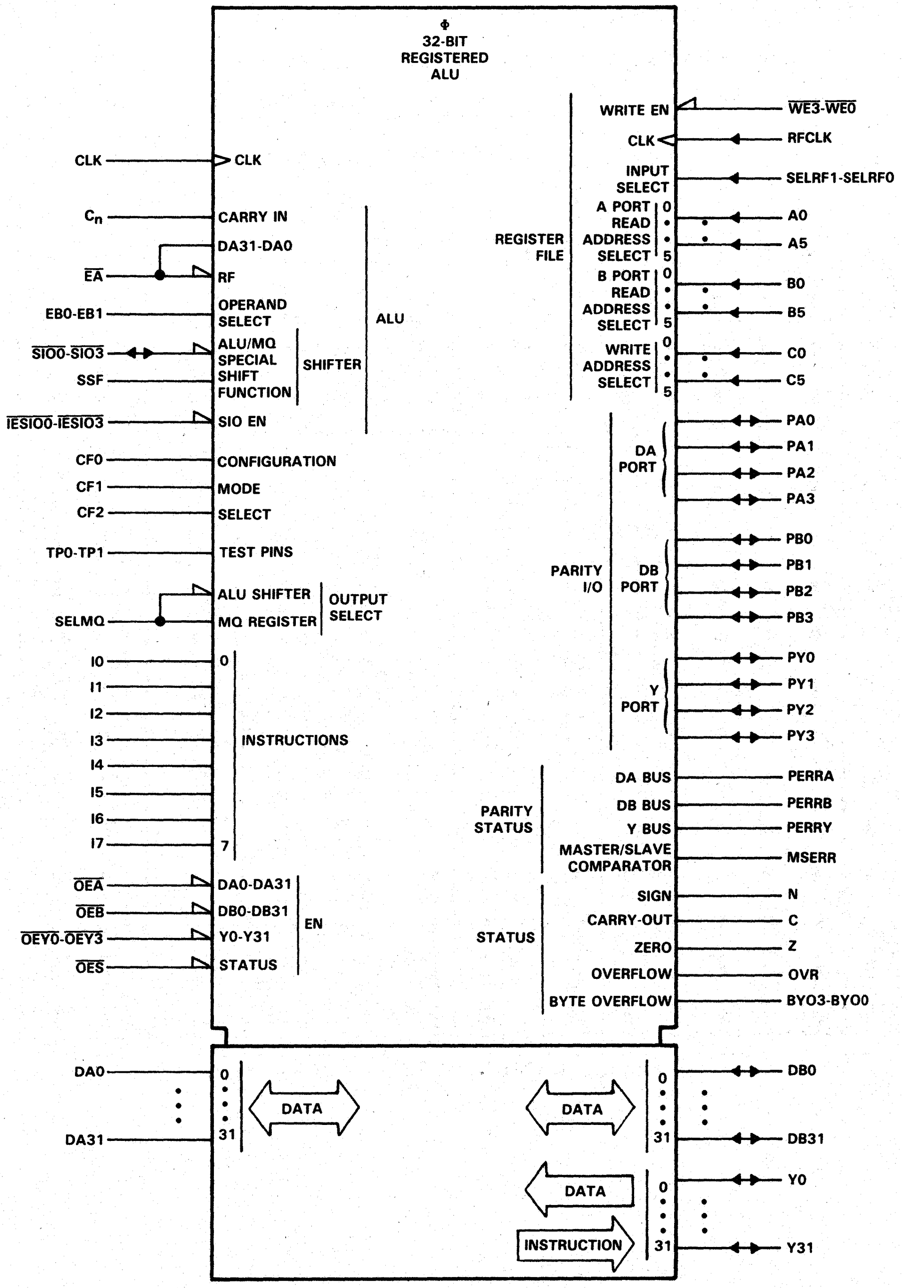


Figure 3. SN74ACT8832 . . . Logic Symbol

Table 1. SN74ACT8832 Pin Grid Allocation

PIN		PIN		PIN		PIN		PIN		PIN	
NO.	**NAME**	**NO.**	**NAME**	**NO.**	**NAME**	**NO.**	**NAME**	**NO.**	**NAME**	**NO.**	**NAME**
A1	Y7	C2	Y5	E3	Y0	J15	DA28	P1	DA5	S1	DB10
A2	Y13	C3	$\overline{\text{OEY0}}$	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	$\overline{\text{SIO3}}$	C6	Y14	E16	I2	K2	DB7	P5	DA15	S5	PERRA
A6	$\overline{\text{SIO2}}$	C7	$\overline{\text{OEY1}}$	E17	I3	K3	DA0	P6	A5	S6	A3
A7	$\overline{\text{IESIO1}}$	C8	GND	F1	EB1	K4	GND	P7	A1	S7	$\overline{\text{WE0}}$
A8	$\overline{\text{IESIO0}}$	C9	V_{CC}	F2	Cn	K14	GND	P8	V_{CC}	S8	$\overline{\text{WE3}}$
A9	$\overline{\text{SIO0}}$	C10	C	F3	CLK	K15	DA24	P9	GND	S9	RFCLK
A10	N	C11	PERRY	F4	CF2	K16	DA25	P10	C4	S10	B4
A11	$\overline{\text{OES}}$	C12	Y17	F14	$\overline{\text{OEY3}}$	K17	DA26	P11	PERRB	S11	B2
A12	SSF	C13	Y22	F15	I1	L1	PB0	P12	GND	S12	C3
A13	Y18	C14	$\overline{\text{OEY2}}$	F16	I4	L2	DA2	P13	DB22	S13	C0
A14	Y20	C15	Y28	F17	I6	L3	V_{CC}	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	EB0	L15	V_{CC}	P17	DB27	S17	DA21
B1	Y6	D2	Y1	G4	GND	L16	DB30	R1	PA0	T1	DB14
B2	BYOF0	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	$\overline{\text{OEA}}$
B6	$\overline{\text{IESIO3}}$	D7	GND	H1	DB2	M4	GND	R6	A4	T6	A2
B7	$\overline{\text{IESIO2}}$	D8	GND	H2	DB1	M14	PA2	R7	A0	T7	$\overline{\text{WE1}}$
B8	$\overline{\text{SIO1}}$	D9	V_{CC}	H3	V_{CC}	M15	DB26	R8	$\overline{\text{WE2}}$	T8	SELRF1
B9	Z	D10	GND	H4	GND	M16	DB28	R9	V_{CC}	T9	SELRF0
B10	OVR	D11	GND	H14	GND	M17	DB31	R10	B1	T10	B5
B11	MSERR	D12	GND	H15	V_{CC}	N1	DA3	R11	C2	T11	B3
B12	Y16	D13	BYOF2	H16	DA31	N2	DA6	R12	$\overline{\text{OEB}}$	T12	B0
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	I0	J3	DB5	N15	DA23	R16	DA20	T16	DB19
B17	Y29	E1	SELMQ	J4	V_{CC}	N16	DB25	R17	DB24	T17	DA17
C1	Y2	E2	CF0	J14	V_{CC}	N17	DB29				

Table 2. SN74ACT8832 Pin Description

PIN		I/O	DESCRIPTION
NAME	NO.		
A0	R7	I	Register file A port read address select
A1	P7		
A2	T6		
A3	S6		
A4	R6		
A5	P6		
B0	T12	I	Register file B port read address select
B1	R10		
B2	S11		
B3	T11		
B4	S10		
B5	T10		
BYOF0	B2	O	Status signals indicate overflow conditions in certain data bytes
BYOF1	A4		
BYOF2	D13		
BYOF3	C17		
C	C10	O	Status signal representing carry out condition
C0	S13	I	Register file write address select
C1	T14		
C2	R11		
C3	S12		
C4	P10		
C5	T13		
CF0	E2	I	Configuration mode select, single 32-bit, two 16-bit, or four 8-bit ALU's
CF1	D1		
CF2	F4		
Cn	F2	I	ALU carry input
CLK	F3	I	Clocks synchronous registers on positive edge
DA0	K3	I/O	A port data bus. Outputs register data ($\overline{\text{OEA}} = 0$) or inputs external data ($\overline{\text{OEA}} = 1$).
DA1	M1		
DA2	L2		
DA3	N1		
DA4	M2		
DA5	P1		
DA6	N2		
DA7	M3		
DA8	T2		
DA9	P4		

Table 2. SN74ACT8832 Pin Description (Continued)

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

Table 2. SN74ACT8832 Pin Description (Continued)

PIN NAME	PIN NO.	I/O	DESCRIPTION
DB16	T15	I/O	B port data bus. Outputs register data ($\overline{\text{OEB}}=0$) or used to input external data ($\overline{\text{OEB}}=1$)
DB17	S14		
DB18	R13		
DB19	T16		
DB20	S15		
DB21	R14		
DB22	P13		
DB23	S16		
DB24	R17		
DB25	N16		
DB26	M15		
DB27	P17		
DB28	M16		
DB29	N17		
DB30	L16		
DB31	M17		
$\overline{\text{EA}}$	G2	I	ALU input operand select. High state selects external DA bus and low state selects register file
EB0	G3	I	ALU input operand select. Selects between register file, external DB port and MQ register
EB1	F1		
GND	C8		Ground pins. All ground pins must be used.
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		

Table 2. SN74ACT8832 Pin Description (Continued)

PIN		I/O	DESCRIPTION
NAME	NO.		
I0	D17	I	Instruction input
I1	F15		
I2	E16		
I3	E17		
I4	F16		
I5	G15		
I6	F17		
I7	G16		
$\overline{\mathrm{IESIO0}}$	A8	I	Shift pin enables, increases system speed and reduces bus conflict, active low
$\overline{\mathrm{IESIO1}}$	A7		
$\overline{\mathrm{IESIO2}}$	B7		
IESIO3	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
$\overline{\mathrm{OEA}}$	T5	I	DA bus enable, active low
$\overline{\mathrm{OEB}}$	R12	I	DB bus enable, active low
$\overline{\mathrm{OES}}$	A11	I	Status enable, active low
$\overline{\mathrm{OEY0}}$	C3	I	Y bus output enable, active low
$\overline{\mathrm{OEY1}}$	C7		
$\overline{\mathrm{OEY2}}$	C14		
$\overline{\mathrm{OEY3}}$	F14		
OVR	B10	O	Output status signal represents overflow condition
PA0	R1	I/O	Parity bits port for DA data
PA1	R5		
PA2	M14		
PA3	G17		
PB0	L1	I/O	Parity bits port for DB data
PB1	R3		
PB2	R15		
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		I/O	DESCRIPTION
NAME	NO.		
PY0	D4	I/O	Y port parity data, input and output
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		
$\overline{\text{SIO0}}$	A9	I/O	Bidirectional shift pin, active low
$\overline{\text{SIO1}}$	B8		
$\overline{\text{SIO2}}$	A6		
$\overline{\text{SIO3}}$	A5		
SSF	A12	I	Special Shift Function, implements conditional shift algorithms
TP0	E15	I/O	Test pins, supports system testing
TP1	D16		
V_{CC}	C9		Supply voltrage (5 V)
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		
$\overline{\text{WE0}}$	S7	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. Active low.
$\overline{\text{WE1}}$	T7		
$\overline{\text{WE2}}$	R8		
$\overline{\text{WE3}}$	S8		

Table 2. SN74ACT8832 Pin Description (Concluded)

PIN		I/O	DESCRIPTION
NAME	NO.		
Y0	E3	I/O	Y port data bus
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		
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	MIN	MAX	
V_{OH}	I_{OH} = −20 μA	4.5 V		4.49				V
		5.5 V		5.49				
	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				V
		5.5 V		0.01				
	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						μA
C_i	V_I = V_{CC} or 0	5 V						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		
	$\overline{OEY3}$-$\overline{OEY0}$	7		
	Y31-Y0	4		
	$\overline{WE3}$-$\overline{WE0}$	4		
	SELRF(DA,DB,PA,PB)	5		
	SELRF(Y)	9		
	SIO	10		
	SELMQ	9		
	$\overline{IESIO3}$-$\overline{IESIO0}$	10		
t_h	ALL	0		

---

† This is the increase in supply current for each input that is at one of the specified TTL voltge levels rather then 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	MSERR	
t_{pd}	A5-A0,B5-B0	36	30	37	28		30	37	16	37			ns
	DA31-DA0,PA3-PA0 DB31-DB0,PB3-PB0	36	25	37	25	20	28	37		37			
	C_n	30	22	31	24		28	28		32			
	$\overline{EA}$	37	28	37	25		31	37		37			
	EB1-EB0	37	28	37	25		31	37		37			
	I7-I0	37	30	37	28		32	37		37			
	CF2-CF0	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	

'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-CF0	See Table 11	See Table 11
$\overline{EA}$	Selects external DA bus	Selects register file
EB1-EB0	See Table 9	See Table 9
$\overline{IESIO3}$-$\overline{IESIO0}$	Normal operation	Force corresponding SIO inputs to high impedance
I7-I0	See Table 15	See Table 15
MQSEL	Selects MQ register	Selects ALU
$\overline{OEA}$	Inhibits DA and PA output	Enables DA and PA output
$\overline{OEB}$	Inhibits DB and PB output	Enables DB and PB output
$\overline{OEY3}$-$\overline{OEY0}$	Inhibits Y and PY outputs	Enables Y and PY outputs
RFSEL1-RFSEL0	See Table 8	See Table 8
SSF	Selects shifted ALU output	Selects ALU (unshifted) output
TP1-TP0	See Table 14	See Table 14
$\overline{WE3}$-$\overline{WE0}$	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.

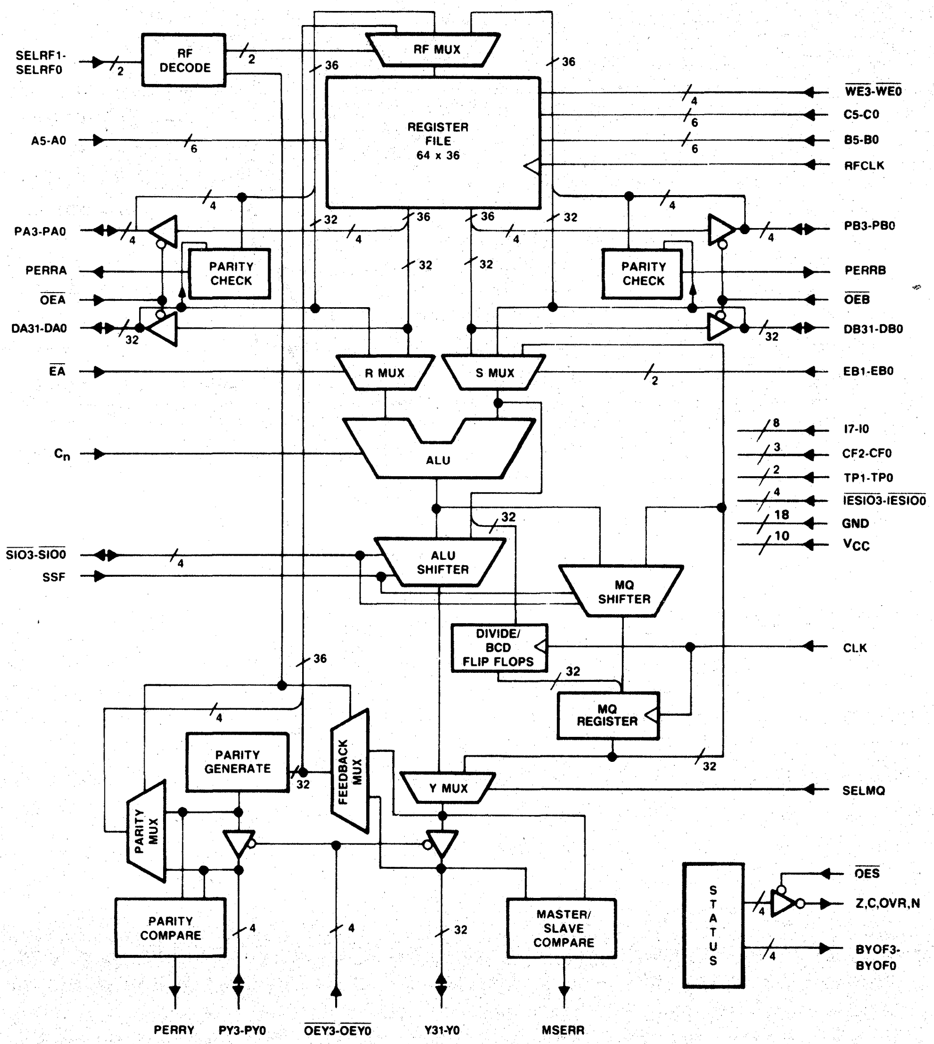


Figure 4. 'ACT8832 32-Bit Registered ALU

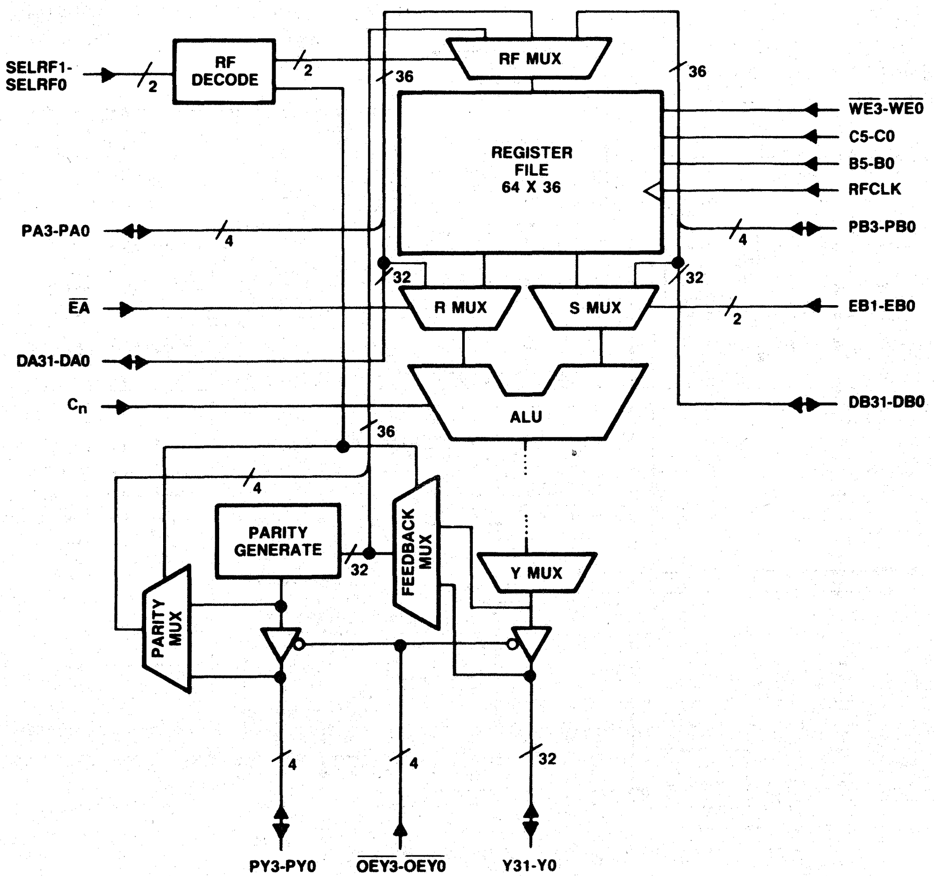


Figure 5. Data I/O

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 multiprocessor 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. $\overline{\text{WE3}}$ 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, RFSEL1 and RFSEL0, shown in Table 8 (see also Table 10).

Table 8. RF MUX Select Inputs

RFSEL1	RFSEL0	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 $\overline{\text{EA}}$	S-BUS OPERAND SELECT EB1-EB0	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 $\overline{WE}$	Y BUS OUTPUT ENABLE $\overline{OEY}$	Y MUS SELECT MQSEL	REGISTER FILE SELECT RFSEL1-RFSEL0		DA PORT OUTPUT ENABLE $\overline{OEA}$	DB PORT OUTPUT ENABLE $\overline{OEB}$	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	X	RF	← External DA/PA
0	X	X	0	1	X	1	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.

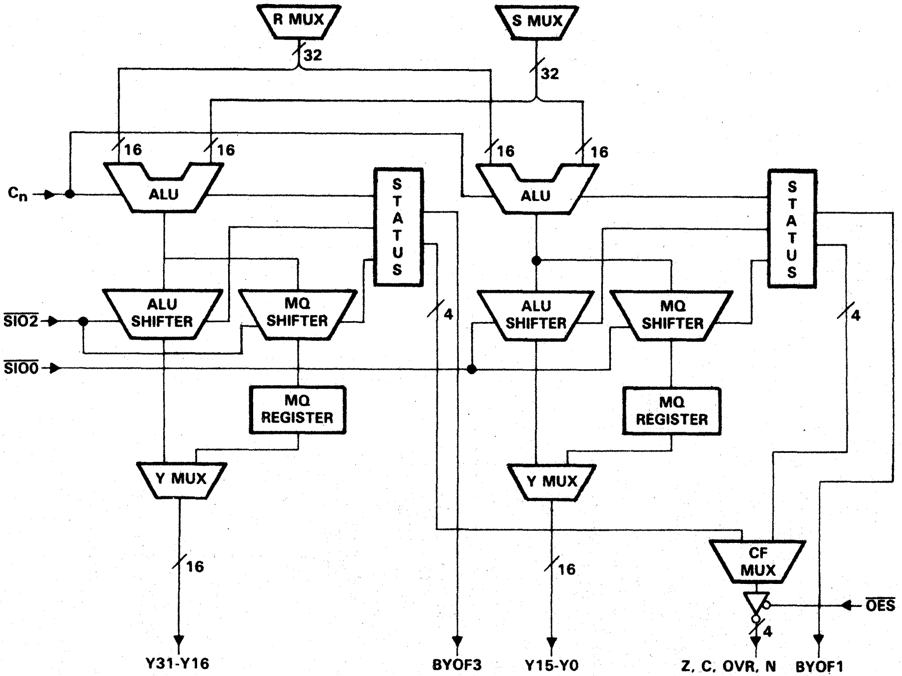


Figure 6. 16-Bit Configuration

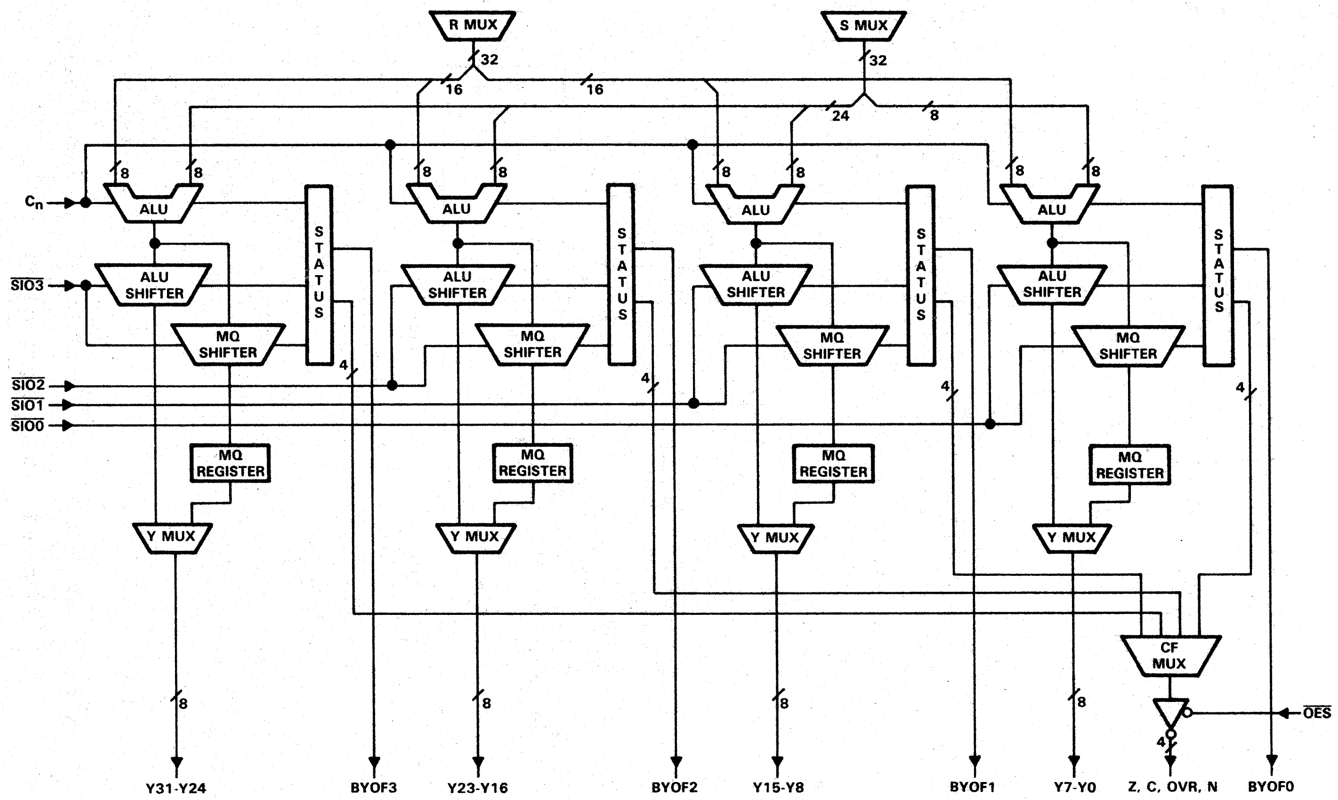


Figure 7. 8-Bit Configuration

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 $\overline{\text{SIO}}$ 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 $\overline{\text{SIO}}$ 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 $\overline{\text{SIO}}$ Input

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

To increase system speed and reduce bus conflict, four $\overline{\text{SIO}}$ input enables ($\overline{\text{IESIO3}}$-$\overline{\text{IESIO0}}$) are provided. A low on these enables will override internal pull-up resistor logic and force the corresponding $\overline{\text{SIO}}$ pins to the high impedance state required before an input signal can appear on the signal line. If the $\overline{\text{SIO}}$ 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 $\overline{\text{IESIO}}$s are defaulted to a high because of internal pull-up resistors. When an $\overline{\text{SIO}}$ pin is used as an output, a low on its corresponding $\overline{\text{IESIO}}$ pin would force $\overline{\text{SIO}}$ 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 MQSEL is low and the output of the MQ register if MQSEL is high. If $\overline{\text{OEY}}$ is low, ALU or MQ shifter output will be passed to the Y port; if $\overline{\text{OEY}}$ 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 $\overline{\text{OEY}}$ 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.

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.

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 I3-I0 (HEX)	MNEMONIC	FUNCTION
0		Used to access Group 4 instructions
1	ADD	R + S + Cn
2	SUBR	$\overline{R}$ + S + Cn
3	SUBS	R + $\overline{S}$ + Cn
4	INCS	S + Cn
5	INCNS	$\overline{S}$ + Cn
6	INCR	R + Cn
7	INCNR	$\overline{R}$ + 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	$\overline{R}$ AND S
F		Used to access Group 5 instructions

Table 15. 'ACT8832 Instruction Set (Continued)

GROUP 2 INSTRUCTIONS		
INSTRUCTION BITS I7-I0 (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	SRLD	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	SET0	Set bit 0
28	TB1	Test bit (one)
38	TB0	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

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
0F	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

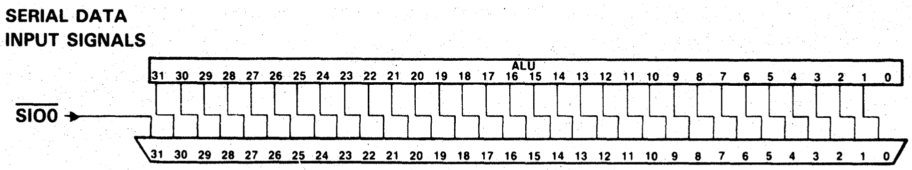
Group 1, a set of ALU arithmetic and logic operations, can be combined with the user-selected shift operations in Group 2 in one instruction cycle. The other groups contain instructions for bit and byte operations, division and multiplication, data conversion, and other functions such as sorting, normalization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

Arithmetic/Logic Instructions with Shifts

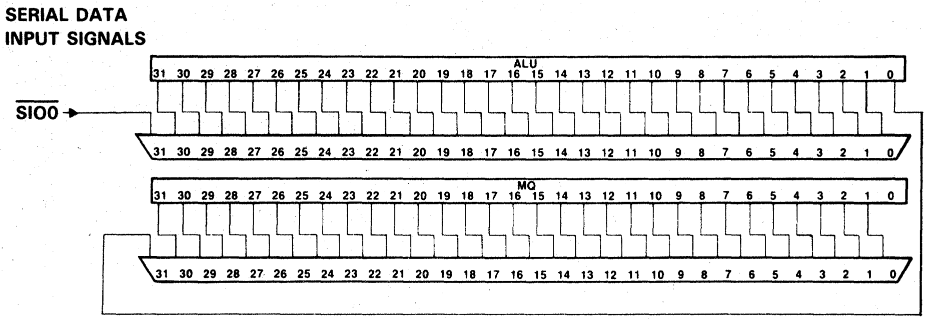
The seven Group 1 arithmetic instructions operate on data from the R and/or S multiplexers and the carry-in. Carry-out is evaluated after ALU operation; other status pins are evaluated after the accompanying shift operation, when applicable. Group 1 logic instructions do not use carry-in; carry-out is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



Single Precision Logical Right Single Shift, 32-Bit Configuration



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 $\overline{\text{SIO}}$ 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 $\overline{\text{SIO}}$ 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)	$\overline{\text{SIO}}$		
	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	ALU0
70	SMULT	O	ALU0
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 TB0 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 concatentation 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)[†]	—	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)[†]	—	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-CF0). 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.

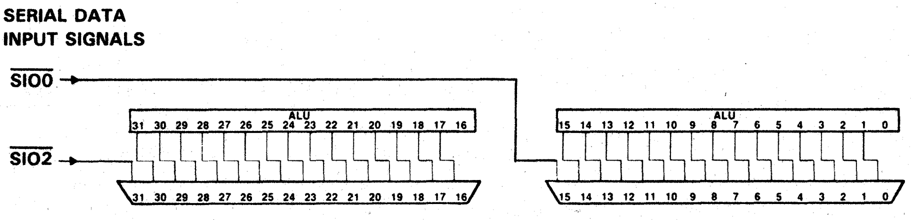
Masked 32-Bit Operation

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.

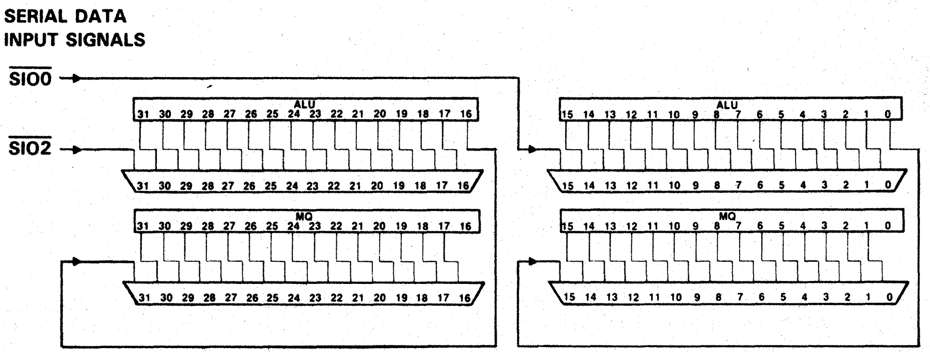
Shift Instructions

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 $\overline{\text{SIO}}$ signals associated with each byte or word in the different modes, and Table 17 indicates the specific function performed by the $\overline{\text{SIO}}$ pins during shift, multiply, and divide operations.

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



Single Precision Logical Right Single Shift, 16-Bit Configuration



Double Precision Logical Right Single Shift, 16-Bit Configuration

Figure 9. Shift Examples, 16-Bit Configuration

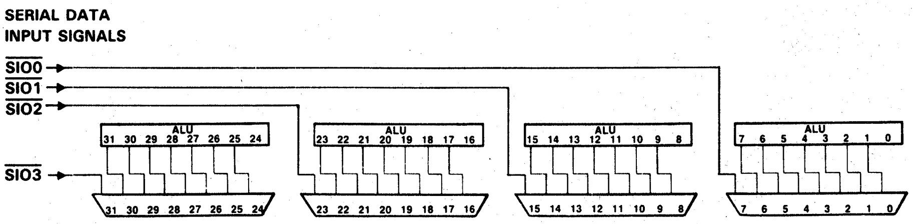
Bit and Byte Instructions

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 $\overline{\text{SIO}}$' inputs low. Instructions which set, reset, or test bits are explained later

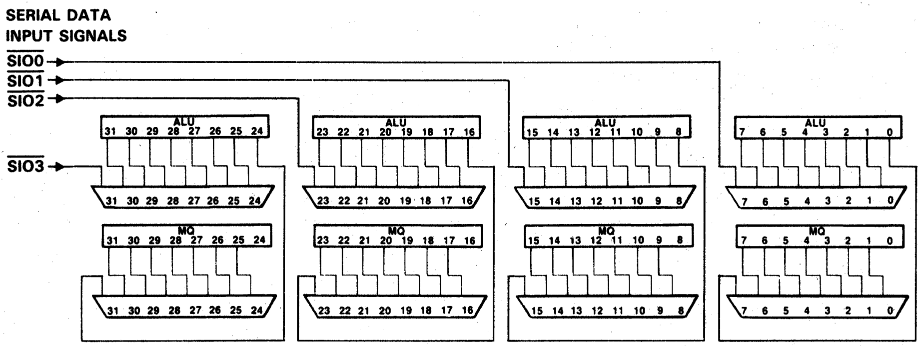
Byte operations should be performed in 32-bit mode to get the necessary status outputs. While byte overflow signals are provided for all four bytes (BYOF3-BYOF0), the other status signals (C, N, Z) are output only for the word selected with the configuration control signals (CF2-CF0).

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′ + Cn to be computed; the state of the sign bit determines whether S or S′ + Cn 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

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
Yes	No	Yes

Shift Operations

ALU	MQ
None	None

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inactive
$\overline{\text{SIO0}}$	No	Inactive
$\overline{\text{SIO1}}$	No	Inactive
$\overline{\text{SIO2}}$	No	Inactive
$\overline{\text{SIO3}}$	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)

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

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel $\overline{\text{EA}}$ EB1-EB0	Dest Addr C5-C0	Destination Selects							Cn	CF2-CF0
					SELMQ	$\overline{\text{WE3}}$-$\overline{\text{WE0}}$	SELRF1-SELRF0	$\overline{\text{OEA}}$	$\overline{\text{OEB}}$	$\overline{\text{OEY3}}$-$\overline{\text{OEY0}}$	$\overline{\text{OES}}$		
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 ← RF(1)
Destination	0000 1001 0010 0111 1110 1100 1100 0000	RF(4) ← S + Cn

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

Source	0000 1001 1101 0101 0010 0111 1100 0000	S ← RF(1)
Destination	0000 1001 1101 0101 0010 0111 1100 0000	RF(4) ← 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.

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
Yes	No	Yes

Shift Operations

ALU	MQ
Yes	Yes

* | 1 Add with Carry (R + S + Cn) ADD

Control/Data Signals

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

† 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		Dest Addr C5-C0	Destination Selects							Cn	CF2-CF0
			$\overline{EA}$	EB1-EB0		SELMQ	WE3-WE0	SELRF1-SELRF0	$\overline{OEA}$	$\overline{OEB}$	$\overline{OEY3}$-OEY0	$\overline{OES}$		
1110 0001	00 0001	XX XXXX	0	10	XX XXXX	0	1111	10	X	X	XXXX	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

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
$\overline{\text{SIO0}}$	No	Inactive
$\overline{\text{SIO1}}$	No	Inactive
$\overline{\text{SIO2}}$	No	Inactive
$\overline{\text{SIO3}}$	No	Inactive
Cn	Yes	Increments sum if set to one.

6 8 ADD Immediate ADDI

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

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel $\overline{EA}$ EB1-EB0	Dest Addr C5-C0	Destination Selects							Cn	CF2-CF0
					SELMQ	$\overline{WE3}$-$\overline{WE0}$	SELRF1-SELRF0	$\overline{OEA}$	$\overline{OEB}$	$\overline{OEY3}$-$\overline{OEY0}$	$\overline{OES}$		
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 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

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

Shift Operations

ALU	MQ
Yes	Yes

Control/Data Signals

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

Logical AND (R AND S)

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)

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

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel $\overline{\text{EA}}$ EB1-EB0	Dest Addr C5-C0	Destination Selects							Cn	CF2-CF0
					SELMQ	$\overline{\text{WE3}}$-$\overline{\text{WE0}}$	SELRF1-SELRF0	$\overline{\text{OEA}}$	$\overline{\text{OEB}}$	$\overline{\text{OEY3}}$-$\overline{\text{OEY0}}$	$\overline{\text{OES}}$		
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

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
Yes	No	Yes

Shift Operations

ALU	MQ
Yes	Yes

Control/Data Signals

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

Logic AND Negative R (R' AND S) ANDNR

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)

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

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel		Dest Addr C5-C0	Destination Selects							Cn	CF2-CF0
			$\overline{EA}$	EB1-EB0		SELMQ	$\overline{WE3}$-$\overline{WE0}$	SELRF1-SELRF0	$\overline{OEA}$	$\overline{OEB}$	$\overline{OEY3}$-$\overline{OEY0}$	$\overline{OES}$		
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 ← RF(3)
Source	1111 0110 0001 0111 1101 1000 0100 0010	S ← RF(5)
Destination	1110 0010 0000 0001 0000 0000 0000 0010	RF(10) ← $\overline{R}$ AND S

BADD Byte Add R to S with Carry | 8 | 8 |

FUNCTION

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

DESCRIPTION

$\overline{\text{SIO3}}$-$\overline{\text{SIO0}}$ are used to select bytes of R to be added to the corresponding bytes of S. A byte of R with $\overline{\text{SIO}}$ programmed low is selected for the computation of R + S + Cn. If the $\overline{\text{SIO}}$ 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

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
$\overline{\text{SIO0}}$	Yes	Byte select
$\overline{\text{SIO1}}$	Yes	Byte select
$\overline{\text{SIO2}}$	Yes	Byte select
$\overline{\text{SIO3}}$	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 3 with carry to the contents of register 1 and store the result in register 11.

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel		Dest Addr C5-C0	Destination Selects							Cn	CF2-CF0	$\overline{\text{SIO3}}$-$\overline{\text{SIO0}}$	$\overline{\text{IESIO3}}$-$\overline{\text{IESIO0}}$
			$\overline{\text{EA}}$	EB1-EB0		SELMQ	$\overline{\text{WE3}}$-$\overline{\text{WE0}}$	SELRF1-SELRF0	$\overline{\text{OEA}}$	$\overline{\text{OEB}}$	$\overline{\text{OEY3}}$-$\overline{\text{OEY0}}$	$\overline{\text{OES}}$				
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 ← RF(3)n
Source	0111 1010 1000 1111 1011 1110 0011 1110	Sn ← RF(1)n
ALU	1010 0110 1001 0001 0100 0000 1100 0000	Fn ← Rn + Sn + Cn
Destination	0111 1010 1001 0001 0100 1111 0011 1110	RF(11)n ← 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 $\overline{\text{SIO}}$ signals programmed low compute R AND S. Bytes with $\overline{\text{SIO}}$ 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

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
Yes	No	Yes

Shift Operations

ALU	MQ
None	None

Control/Data Signals

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

E 8 Byte AND R AND S (Byte Logical AND R AND S) BAND

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.

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel		Dest Addr C5-C0	Destination Selects										
			$\overline{EA}$	EB1-EB0		SELMQ	$\overline{WE3}$-$\overline{WE0}$	SELRF1-SELRF0	$\overline{OEA}$	$\overline{OEB}$	$\overline{OEY3}$-$\overline{OEY0}$	$\overline{OES}$	Cn	CF2-CF0	$\overline{SIO3}$-$\overline{SIO0}$	$\overline{IESIO3}$-$\overline{IESIO0}$
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 ← RF(3)n
Source	0100 0010 1001 0000 1011 1111 1011 1111	Sn ← DBn
Destination	0100 0010 1000 0000 1011 1110 1011 1111	RF(3)n ← 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 $\overline{\text{SIO0}}$ 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.

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

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

Shift Operations

ALU	MQ
Left	Left

Control/Data Signals

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

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 BCD number to a 4N-bit binary number in 4(N-1) plus 8 clocks. 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.

```
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 × (ACC + 4 × digit)];
                                      Circular left shift MQ.
        BCDBIN ACC, R1, ACC           Perform BCDBIN instruction, and store
                                      result in accumulator
                                      [10 × (ACC + 10 × 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.
```

Byte Increment Negative S with Carry

BINCNS

FUNCTION

S′ + Cn for selected bytes of S.

DESCRIPTION

Bytes with $\overline{\text{SIO0}}$ programmed low compute S′ + Cn. Bytes with $\overline{\text{SIO0}}$ 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

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
Yes	No	Yes

Shift Operations

ALU	MQ
None	None

Control/Data Signals

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

BINCNS Byte Increment Negative S with Carry C 8

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-$\overline{\mathrm{EA}}$ EB0	Dest Addr C5-C0	Destination Selects							Cn	CF2-CFO	$\overline{\mathrm{SIO3}}$-$\overline{\mathrm{SIO0}}$	$\overline{\mathrm{IESIO3}}$-$\overline{\mathrm{IESIO0}}$
					SELMQ	$\overline{\mathrm{WE3}}$-$\overline{\mathrm{WE0}}$	SELRF1-SELRF0	$\overline{\mathrm{OEA}}$	$\overline{\mathrm{OEB}}$	$\overline{\mathrm{OEY3}}$-$\overline{\mathrm{OEY0}}$	$\overline{\mathrm{OES}}$				
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 ← RF(3)n
ALU	0101 1100 1111 1110 0111 1110 0111 1111	Fn ← S'n + Cn
Destination	1010 0011 0000 0001 0111 1110 0111 1111	RF(3)n ← 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 SIO' inputs programmed low compute S + Cn. Bytes with $\overline{\text{SIO}}$ 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 nonsselected.

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
Yes	No	Yes

Shift Operations

ALU	MQ
None	None

Control/Data Signals

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

BINCS Byte Increment S with Carry B 8

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- $\overline{\text{EA}}$ EB0	Dest Addr C5-C0	Destination Selects							Cn	CF2- CF0	$\overline{\text{SIO3-}}$ $\overline{\text{SIO0}}$	$\overline{\text{IESIO3-}}$ $\overline{\text{IESIO0}}$
					SELMQ	$\overline{\text{WE3-}}$ $\overline{\text{WE0}}$	SELRF1- SELRF0	$\overline{\text{OEA}}$	$\overline{\text{OEB}}$	$\overline{\text{OEY3-}}$ $\overline{\text{OEY0}}$	$\overline{\text{OES}}$				
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 ← RF(7)n
ALU	0100 0000 1000 1111 1011 1111 1011 1110	Fn ← Sn + Cn
Destination	0100 0000 1000 1111 1011 1111 1011 1110	RF(2)n ← 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

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

Shift Operations

ALU	MQ
None	Left

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inactive
$\overline{\text{SIO0}}$	No	Inactive
$\overline{\text{SIO1}}$	No	Inactive
$\overline{\text{SIO2}}$	No	Inactive
$\overline{\text{SIO3}}$	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:

$$a_n 2^n + a_{n-1} 2^{n-1} + a_{n-2} 2^{n-2} + \dots + a_o =$$

$$\{[(2a_n + a_{n-1}) \times 2 + a_{n-1}] \times 2 + \dots + a_o\} \times 2 + a^o.$$

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)

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

FUNCTION

Evaluates R OR S of selected bytes.

DESCRIPTION

Bytes with $\overline{\mathrm{SIO}}$ inputs programmed low evaluate R OR S. Bytes with $\overline{\mathrm{SIO}}$ 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

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
$\overline{\mathrm{SIO0}}$	Yes	Byte select
$\overline{\mathrm{SIO1}}$	Yes	Byte select
$\overline{\mathrm{SIO2}}$	Yes	Byte select
$\overline{\mathrm{SIO3}}$	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.

Instr	Oprd	Oprd	Oprd Sel	Dest	Destination Selects										
Code	Addr	Addr	EB1-	Addr		$\overline{\text{WE3}}$-	SELRF1-			$\overline{\text{OEY3}}$-			CF2-	$\overline{\text{SIO3}}$-	$\overline{\text{IESIO3}}$-
I7-I0	A5-A0	B5-B0	$\overline{\text{EA}}$ EB0	C5-C0	SELMQ	$\overline{\text{WE0}}$	SELRF0	$\overline{\text{OEA}}$	$\overline{\text{OEB}}$	$\overline{\text{OEY0}}$	$\overline{\text{OES}}$	Cn	CF0	$\overline{\text{SIO0}}$	$\overline{\text{IESIO0}}$
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 ← RF(12)n
Source	0001 1100 1001 0000 1011 1110 1011 1100	Sn ← DBn
Destination	0001 1100 1001 1111 1011 1110 1011 1110	RF(12)n ← Fn or Sn†

†F = ALU result
n = nth package
Register file 12 gets F if byte selected, S if byte not selected.

A 8 Byte Subtract R from S with Carry BSUBR

FUNCTION

Subtracts R from S in selected bytes.

DESCRIPTION

Bytes with $\overline{\text{SIO}}$ inputs programmed low compute R′ + S + Cn. Bytes with $\overline{\text{SIO}}$ 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

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
$\overline{\text{SIO0}}$	Yes	Byte select
$\overline{\text{SIO1}}$	Yes	Byte select
$\overline{\text{SIO2}}$	Yes	Byte select
$\overline{\text{SIO3}}$	Yes	Byte select
Cn	Yes	Propagates through nonselected bytes; should be set high for two's complement subtraction.

BSUBR Byte Subtract R from S with Carry A 8

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 I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel		Dest Addr C5-C0	Destination Selects							Cn	CF2-CF0	$\overline{\text{SIO3}}$-$\overline{\text{SIO0}}$	$\overline{\text{IESIO3}}$-$\overline{\text{IESIO0}}$
			$\overline{\text{EA}}$	EB1-EB0		SELMQ	$\overline{\text{WE3}}$-$\overline{\text{WE0}}$	SELRF1-SELRF0	$\overline{\text{OEA}}$	$\overline{\text{OEB}}$	$\overline{\text{OEY3}}$-$\overline{\text{OEY0}}$	$\overline{\text{OES}}$				
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 ← RF(1)n
Source	0111 0000 0011 1010 1001 1000 1001 1000	Sn ← RF(3)n
ALU	0110 0111 0001 1111 0100 0000 0100 0000	Fn ← R'n + Sn + Cn
Destination	0111 0000 0001 1111 0100 0000 1001 1000	RF(11)n ← 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' + Cn. 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

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
$\overline{SIO0}$	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.

BSUBS Byte Subtract S from R with Carry 9 8

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 $\overline{\text{EA}}$ EB1-EB0	Dest Addr C5-C0	Destination Selects							Cn	CF2-CF0	$\overline{\text{SIO3}}$-$\overline{\text{SIO0}}$	$\overline{\text{IESIO3}}$-$\overline{\text{IESIO0}}$
					SELMQ	$\overline{\text{WE3}}$-$\overline{\text{WE0}}$	SELRF1-SELRF0	$\overline{\text{OEA}}$	$\overline{\text{OEB}}$	$\overline{\text{OEY3}}$-$\overline{\text{OEY0}}$	$\overline{\text{OES}}$				
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 ← RF(1)n
Source	0001 0100 0011 1010 1001 1000 1001 1000	Sn ← RF(3)n
ALU	0011 1110 0100 1110 0010 0000 0010 0000	Fn ← Rn + S'n + Cn
Destination	0101 0010 0100 1110 0010 0000 1011 1000	RF(11)n ← Fn or Sn[†]

[†] F = ALU result
n = nth byte
Register file 11 gets F if byte selected, S if byte not selected.

D 8

Byte XOR R and S (Byte Exclusive OR R and S)

BXOR

FUNCTION

Evaluates R exclusive OR S in selected bytes.

DESCRIPTION

Bytes with $\overline{\text{SIO}}$ inputs programmed low evaluate R exclusive OR S. Bytes with $\overline{\text{SIO}}$ 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

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
$\overline{\text{SIO0}}$	Yes	Byte select
$\overline{\text{SIO1}}$	Yes	Byte select
$\overline{\text{SIO2}}$	Yes	Byte select
$\overline{\text{SIO3}}$	Yes	Byte select
Cn	No	Inactive

BXOR Byte XOR R and S (Byte Exclusive OR R and S)

D 8

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- $\overline{\mathrm{EA}}$ EB0	Dest Addr C5-C0	Destination Selects							Cn	CF2- CF0	$\overline{\mathrm{SIO3}}$- $\overline{\mathrm{SIO0}}$	$\overline{\mathrm{IESIO3}}$- $\overline{\mathrm{IESIO0}}$
					SELMQ	$\overline{\mathrm{WE3}}$- $\overline{\mathrm{WE0}}$	SELRF1- SELRF0	$\overline{\mathrm{OEA}}$	$\overline{\mathrm{OEB}}$	$\overline{\mathrm{OEY3}}$- $\overline{\mathrm{OEY0}}$	$\overline{\mathrm{OES}}$				
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 ← RF(6)n
Source	0100 0001 1001 0000 1011 1110 1011 1110	Sn ← DBn
Destination	0100 0001 0001 1111 0000 0000 1011 1110	RF(10)n ← Fn or Sn†

†F = ALU result
n = nth package
Register file 10 gets F if byte selected, S if byte not selected.

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

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
Yes	No	Yes

Shift Operations

ALU	MQ
None	None

Status Signals

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

CRC Cyclic Redundancy Character Accumulation 0 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

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

Shift Operations

ALU	MQ
Right	Right

Control/Data Signals

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

Status Signals

ZERO = 1 if result = 0
N = 0
OVR = 0
Cn = 0

CYCLIC REDUNDANCY CHARACTER CHECK

DESCRIPTION

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

$$g(x) = (g_{k-1})(x^{k-1}) + (g_{k-2})(x^{k-2}) + \,.\,.(g_o)(x^o)$$

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:

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

$Y \leftarrow S + R + 0$ 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

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

Shift Operations

ALU	MQ
None	None

DIVRF Divide Remainder Fix 4 0

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inactive
$\overline{\mathrm{SIO0}}$	No	Inactive
$\overline{\mathrm{SIO1}}$	No	Inactive
$\overline{\mathrm{SIO2}}$	No	Inactive
$\overline{\mathrm{SIO3}}$	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

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 $\overline{\text{SIO0}}$ is low. The S bus holds the most significant half; the MQ register holds the least significant half.

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

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

DNORM Double-Length Normalize 3 0

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inactive
$\overline{SIO0}$	Yes	When low, selects a one end-fill bit in LSB
$\overline{SIO1}$	No	Passes internally generated end-fill bits
$\overline{SIO2}$	No	
$\overline{SIO3}$	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- $\overline{EA}$ EB0	Dest Addr C5-C0	Destination Selects								CF2-CF0
					SELMQ	$\overline{WE3}$- $\overline{WE0}$	SELRF1-SELRF0	$\overline{OEA}$	$\overline{OEB}$	$\overline{OEY3}$- $\overline{OEY0}$	$\overline{OES}$	Cn	
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 ← RF(3)
Source	0011 0111 1111 0110 1101 1000 0100 0011	MQ shifter ← MQ register
Destination	1111 0100 1110 1011 1011 0000 1001 1101	8RF(3) ← Result (MSH)
Destination	0110 1111 1110 1101 1011 0000 1000 0110	MQ register ← Result (LSH)
	0	OVR ← 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 lmultiplexer.

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
No	No	No

Available Destination Operands

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

Shift Operations

ALU	MQ
None	None

Status Signals

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

DUMPFF Output Divide/BCD Flip-Flops 5 F

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		Dest Addr C5-C0	Destination Selects							Cn	CF2-CF0
			$\overline{EA}$	EB1-EB0		SELMQ	$\overline{WE3-}$ $\overline{WE0}$	SELRF1-SELRF0	$\overline{OEA}$	$\overline{OEB}$	$\overline{OEY3-}$ $\overline{OEY0}$	$\overline{OES}$		
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 ← Divide/BCD flip-flops
Destination	0010 1010 0000 0101 0101 0100 0111 0000	Y output ← MQ register

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 $\overline{\text{SIO}}$ inputs programmed low evaluate the correct excess-3 representation. Bytes with $\overline{\text{SIO}}$ inputs programmed high or floating, pass S unaltered.

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

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

Shift Operations

ALU	MQ
No	No

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inactive
$\overline{\text{SIO0}}$	Yes	Byte select
$\overline{\text{SIO1}}$	Yes	Byte select
$\overline{\text{SIO2}}$	Yes	Byte select
$\overline{\text{SIO3}}$	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		Dest Addr C5-C0	Destination Selects							Cn	CF2-CF0	$\overline{\text{SIO3}}$-$\overline{\text{SIO0}}$	$\overline{\text{IESIO3}}$-$\overline{\text{IESIO0}}$
			$\overline{\text{EA}}$	EB1-EB0		SELMQ	$\overline{\text{WE3}}$-$\overline{\text{WE0}}$	SELRF1-SELRF0	$\overline{\text{OEA}}$	$\overline{\text{OEB}}$	$\overline{\text{OEY3}}$-$\overline{\text{OEY0}}$	$\overline{\text{OES}}$				
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

9 F Excess-3 Word Correction EX3C

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.

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

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

Shift Operations

ALU	MQ
No	No

Control/Data Signals

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

EX3C Excess-3 Word Correction 9 F

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.

Instr Code	Oprd Addr	Oprd Addr	Oprd Sel		Dest Addr	Destination Selects								
I7-I0	A5-A0	B5-B0	$\overline{EA}$	EB1-EB0	C5-C0	SELMQ	$\overline{WE3}$-$\overline{WE0}$	SELRF1-SELRF0	$\overline{OEA}$	$\overline{OEB}$	$\overline{OEY3}$-$\overline{OEY0}$	$\overline{OES}$	Cn	CF2-CF0
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

9 | F Excess-3 Word Correction EX3C

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

Results of Instruction Cycles:

1	0000 0000 0000 0000 0000 0000 0000 0000	RF(2) ← 0
2	0011 0011 0011 0011 0011 0011 0011 0011	RF(2) ← 33333333 (Hex)
3	1000 0100 0110 0110 1001 1100 0100 0101	RF(1) ← RF(2) +DB
4	0110 0111 1011 1001 1010 0100 1001 0101	RF(3) ← RF(2) + DB
5	1110 1100 0010 0000 0100 0000 1101 1010	RF(3) ← RF(1) + RF(3)
6	1011 1001 0101 0011 0111 0011 1010 0111	RF(3) ← Corrected RF(3) result
7	1000 0110 0010 0000 0100 0000 0111 0100	RF(3) ← RF(3)−RF(2)

FUNCTION

Evaluates R' + Cn.

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.
$\overline{\text{SIO0}}$	No	
$\overline{\text{SIO1}}$	No	
$\overline{\text{SIO2}}$	No	
$\overline{\text{SIO3}}$	No	
Cn	Yes	Increments if programmed high.

Increment Negative R using Carry (R′ + Cn) INCNR

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)

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

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel		Dest Addr C5-C0	Destination Selects							Cn	CF2-CF0
			$\overline{EA}$	EB1-EB0		SELMQ	$\overline{WE3}$-$\overline{WE0}$	SELRF1-SELRF0	$\overline{OEA}$	$\overline{OEB}$	$\overline{OEY3}$-$\overline{OEY0}$	$\overline{OES}$		
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 ← DA
Destination	1100 1000 0110 1110 0000 0001 0000 1010	RF(4) ← R′ + Cn

INCNS Increment Negative S using Carry (S′ + Cn) | * | 5 |

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.
$\overline{SIO0}$	No	
$\overline{SIO1}$	No	
$\overline{SIO2}$	No	
$\overline{SIO3}$	No	
Cn	Yes	Increments if programmed high.

* | 5 Increment Negative S using Carry (S′ + Cn) INCNS

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)

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

Instr Code	Oprd Addr	Oprd Addr	Oprd Sel		Dest Addr	Destination Selects								
I7-I0	A5-A0	B5-B0	$\overline{EA}$	EB1-EB0	C5-C0	SELMQ	$\overline{WE3}$-$\overline{WE0}$	SELRF1-SELRF0	$\overline{OEA}$	$\overline{OEB}$	$\overline{OEY3}$-$\overline{OEY0}$	$\overline{OES}$	Cn	CF2-CF0
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

INCR Increment R using Carry (R + Cn) | * | 6 |

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.
$\overline{\text{SIO0}}$	No	
$\overline{\text{SIO1}}$	No	
$\overline{\text{SIO2}}$	No	
$\overline{\text{SIO3}}$	No	
Cn	Yes	Increments R if programmed high.

* 6 Increment R using Carry (R + Cn) INCR

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.

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel		Dest Addr C5-C0	Destination Selects							Cn	CF2-CF0
			$\overline{EA}$	EB1-EB0		SELMQ	$\overline{WE3}$-$\overline{WE0}$	SELRF1-SELRF0	$\overline{OEA}$	$\overline{OEB}$	$\overline{OEY3}$-$\overline{OEY0}$	$\overline{OES}$		
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 ← DA

Destination 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

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.
$\overline{\text{SIO0}}$	No	
$\overline{\text{SIO1}}$	No	
$\overline{\text{SIO2}}$	No	
$\overline{\text{SIO3}}$	No	
Cn	Yes	Increments S if programmed high.

* | 4 Increment S using Carry (S + Cn) INCS

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)

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

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel		Dest Addr C5-C0	Destination Selects							Cn	CF2-CF0
			$\overline{EA}$	EB1-EB0		SELMQ	$\overline{WE3}$-$\overline{WE0}$	SELRF1-SELRF0	$\overline{OEA}$	$\overline{OEB}$	$\overline{OEY3}$-$\overline{OEY0}$	$\overline{OES}$		
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 register
Destination	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

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
$\overline{\text{SIO0}}$	No	Inactive
$\overline{\text{SIO1}}$	No	Inactive
$\overline{\text{SIO2}}$	No	Inactive
$\overline{\text{SIO3}}$	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.

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel		Dest Addr C5-C0	Destination Selects							Cn	CF2-CF0
			$\overline{EA}$	EB1-EB0		SELMQ	$\overline{WE3}$-$\overline{WE0}$	SELRF1-SELRF0	$\overline{OEA}$	$\overline{OEB}$	$\overline{OEY3}$-$\overline{OEY0}$	$\overline{OES}$		
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 bus
Destination	0010 1010 0000 1000 1100 0110 0001 1000	Divide/BCD flip-flops ← S

LOADMQ Pass (Y ← F) and Load MQ with F

E	*

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

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)
$\overline{\text{SIO0}}'$	No	Inactive
$\overline{\text{SIO1}}$	No	Inactive
$\overline{\text{SIO2}}$	No	Inactive
$\overline{\text{SIO3}}$	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.

Pass (Y ← F) and Load MQ with F

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 I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel		Dest Addr C5-C0	Destination Selects							Cn	CF2-CF0
			$\overline{\text{EA}}$	EB1-EB0		SELMQ	$\overline{\text{WE3}}$-$\overline{\text{WE0}}$	SELRF1-SELRF0	$\overline{\text{OEA}}$	$\overline{\text{OEB}}$	$\overline{\text{OEY3}}$-$\overline{\text{OEY0}}$	$\overline{\text{OES}}$		
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):

Source	0010 1010 0000 1000 1100 0110 0001 1000	R ← RF(1)
Destination	0010 1010 0000 1000 1100 0110 0001 1000	MQ register ← R + Cn

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.
$\overline{\text{SIO0}}$	No	Inactive
$\overline{\text{SIO1}}$	No	Inactive
$\overline{\text{SIO2}}$	No	Inactive
$\overline{\text{SIO3}}$	No	Inactive
Cn	No	Affects arithmetic operation programmed in bits I3-I0 of instruction field.

Pass (Y ← F) with Circular Left MQ Shift

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.

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.

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel EB1- $\overline{\text{EA}}$ EB0	Dest Addr C5-C0	Destination Selects							Cn	CF2-CF0
					SELMQ	$\overline{\text{WE3}}$-$\overline{\text{WE0}}$	SELRF1-SELRF0	$\overline{\text{OEA}}$	$\overline{\text{OEB}}$	$\overline{\text{OEY3}}$-$\overline{\text{OEY0}}$	$\overline{\text{OES}}$		
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 4DA99A0E (Hex).

Source	0010 0101 0000 1000 1100 0110 0001 1000	R ← RF(1)
Source	0001 0001 0000 0000 0111 0101 0011 0000	S ← DB bus
Destination	0011 0110 0000 1001 0011 1011 0100 1001	RF(1) ← R + S + Cn
Source	0100 1101 1010 1001 1001 1010 0000 1110	MQ shifter ← MQ register
Destination	1001 1011 0101 0011 0011 0100 0001 1100	MQ register ← 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 $\overline{\text{SIO}}$ 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.

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 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.
$\overline{\text{SIO0}}$	Yes	Fills a zero in LSB of MQ shifter if high or floating; sets LSB to one if low.
$\overline{\text{SIO1}}$	No	Inactive in 32-bit configuration; used in configurations to select end-fill in LSBs.
$\overline{\text{SIO2}}$	No	
$\overline{\text{SIO3}}$	No	
Cn	No	Affects arithmetic operation programmed in bits I3-I0 of instruction field.

Pass (Y ← F) with Logical Left MQ Shift

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.

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		Dest Addr C5-C0	Destination Selects							Cn	CF2-CF0	$\overline{\text{SIO3}}$-$\overline{\text{SIO0}}$	$\overline{\text{IESIO3}}$-$\overline{\text{IESIO0}}$
			$\overline{\text{EA}}$	EB1-EB0		SELMQ	$\overline{\text{WE3}}$-$\overline{\text{WE0}}$	SELRF1-SELRF0	$\overline{\text{OEA}}$	$\overline{\text{OEB}}$	$\overline{\text{OEY3}}$-$\overline{\text{OEY0}}$	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 ← RF(7)
Source	0101 0100 0000 0000 0111 0101 0011 0000	S ← DB bus
Destination	1100 0111 0000 1001 0011 1011 0100 1001	RF(7) ← R + S + Cn
Source	0110 0001 1010 1001 1001 1010 0000 1100	MQ shifter ← MQ register
Destination	1100 0011 0101 0011 0011 0100 0001 1000	MQ register ← MQ shifter

MQSRA Pass (Y ← F) with Arithmetic Right MQ Shift | A | * |

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.
$\overline{\text{SIO0}}$	No	Outputs LSB of MQ shifter (inverted).
$\overline{\text{SIO1}}$	No	Inactive in 32-bit configurations; used in other configurations to output LSBs from MQ shifter (inverted).
$\overline{\text{SIO2}}$	No	
$\overline{\text{SIO3}}$	No	
Cn	No	Affects arithmetic operation programmed in bits I3-I0 of instruction field.

Pass (Y ← F) with Arithmetic Right MQ Shift MQSRA

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.

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	Oprd Sel		Dest Addr C5-C0	Destination Selects							Cn	CF2-CF0
			$\overline{EA}$	EB1-EB0		SELMQ	$\overline{WE3}$-$\overline{WE0}$	SELRF1-SELRF0	$\overline{OEA}$	$\overline{OEB}$	$\overline{OEY3}$-$\overline{OEY0}$	$\overline{OES}$		
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 98A99A0E (Hex).

Source	0101 0110 0000 1000 1100 0110 0001 1000	R ← RF(1)
Source	0001 0100 0000 0000 0111 0101 0011 0000	S ← RF(10)
Destination	0110 1010 0000 1001 0011 1011 0100 1001	RF(1) ← R + S + Cn
Source	1001 1000 1010 1001 1001 1010 0000 1110	MQ shifter ← MQ register
Destination	1100 1100 0101 0100 1100 1101 0000 0111	MQ register ← 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 $\overline{\text{SIO}}$ 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.
$\overline{\text{SIO0}}$	Yes	Fills a zero in LSB of MQ shifter if high or floating; sets LSB to one if low.
$\overline{\text{SIO1}}$	No	Inactive in 32-bit configuration; used in other configurations to select end-fill in LSBs.
$\overline{\text{SIO2}}$	No	
$\overline{\text{SIO3}}$	No	
Cn	No	Affects arithmetic operation programmed in bits I3-I0 of instruction field.

B * Pass (Y ← F) with Logical Right MQ Shift MQSRL

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.

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		Dest Addr C5-C0	Destination Selects							Cn	CF2-CF0
			$\overline{EA}$	EB1-EB0		SELMQ	$\overline{WE3}$-$\overline{WE0}$	SELRF1-SELRF0	$\overline{OEA}$	$\overline{OEB}$	$\overline{OEY3}$-$\overline{OEY0}$	$\overline{OES}$		
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 98A99A0E (Hex).

Source	0101 0110 0000 1000 1100 0110 0001 1000	R ← RF(1)
Source	0001 0100 0000 0000 0111 0101 0011 0000	S ← DB bus
Destination	0110 1010 0000 1001 0011 1011 0100 1001	RF(1) ← R + S + Cn
Source	1001 1000 1010 1001 1001 1010 0000 1110	MQ shifter ← MQ register
Destination	0100 1100 0101 0100 1100 1101 0000 0111	MQ register ← 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.
$\overline{\text{SIO0}}$	No	
$\overline{\text{SIO1}}$	No	
$\overline{\text{SIO2}}$	No	
$\overline{\text{SIO3}}$	No	
Cn		Inactive

* | C Logical NAND (R NAND S) NAND

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)

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

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel		Dest Addr C5-C0	Destination Selects								
			$\overline{EA}$	EB1-EB0		SELMQ	$\overline{WE3}$-$\overline{WE0}$	SELRF1-SELRF0	$\overline{OEA}$	$\overline{OEB}$	$\overline{OEY3}$-$\overline{OEY0}$	$\overline{OES}$	Cn	CF2-CF0
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

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

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
Yes	No	Yes

Shift Operations

ALU	MQ
None	None

Status Signals

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

EXAMPLE (assumes a 32-bit configuration)

Clear register 12.

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel $\overline{EA}$	Oprd Sel EB1-EB0	Dest Addr C5-C0	Destination Selects SELMQ	$\overline{WE3}$-$\overline{WE0}$	SELRF1-SELRF0	$\overline{OEA}$	$\overline{OEB}$	$\overline{OEY3}$-$\overline{OEY0}$	$\overline{OES}$	Cn	CF2-CF0
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

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.
$\overline{\text{SIO0}}$	No	
$\overline{\text{SIO1}}$	No	
$\overline{\text{SIO2}}$	No	
$\overline{\text{SIO3}}$	No	
Cn	No	Inactive

Logical NOR (R NOR S)

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)

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

Instr Code	Oprd Addr	Oprd Addr	Oprd Sel		Dest Addr	Destination Selects								
I7-I0	A5-A0	B5-B0	$\overline{EA}$	EB1-EB0	C5-C0	SELMQ	$\overline{WE3}$-$\overline{WE0}$	SELRF1-SELRF0	$\overline{OEA}$	$\overline{OEB}$	$\overline{OEY3}$-$\overline{OEY0}$	$\overline{OES}$	Cn	CF2-CF0
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.
$\overline{\text{SIO0}}$	No	
$\overline{\text{SIO1}}$	No	
$\overline{\text{SIO2}}$	No	
$\overline{\text{SIO3}}$	No	
Cn	No	Inactive

Logical OR (R OR S)

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)

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

Instr Code	Oprd Addr	Oprd Addr	Oprd Sel		Dest Addr	Destination Selects								
I7-I0	A5-A0	B5-B0	$\overline{\text{EA}}$	EB1-EB0	C5-C0	SELMQ	$\overline{\text{WE3}}$-$\overline{\text{WE0}}$	SELRF1-SELRF0	$\overline{\text{OEA}}$	$\overline{\text{OEB}}$	$\overline{\text{OEY3}}$-$\overline{\text{OEY0}}$	$\overline{\text{OES}}$	Cn	CF2-CF0
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

PASS — Pass (Y ← F) — F | *

FUNCTION

Passes the result of the ALU instruction specified in the upper 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
$\overline{\text{SIO0}}$	No	Inactive
$\overline{\text{SIO1}}$	No	Inactive
$\overline{\text{SIO2}}$	No	Inactive
$\overline{\text{SIO3}}$	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.

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		Dest Addr C5-C0	Destination Selects							Cn	CF2-CF0
			$\overline{\text{EA}}$	EB1-EB0		SELMQ	$\overline{\text{WE3}}$-$\overline{\text{WE0}}$	SELRF1-SELRF0	$\overline{\text{OEA}}$	$\overline{\text{OEB}}$	$\overline{\text{OEY3}}$-$\overline{\text{OEY0}}$	$\overline{\text{OES}}$		
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).

Source	1001 0011 0000 1000 1100 0110 0001 1000	R ← RF(1)
Source	0010 0100 0000 0000 0111 0101 0011 0000	S ← DB bus
Destination	1011 0111 0000 1001 0011 1011 0100 1001	RF(10) ← R + S + Cn

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

$F \leftarrow R + S$	if the test fails
$F \leftarrow R' + S + Cn$	if the test passes

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
$\overline{\text{SIO0}}$	No	Pass internally generated end-fill bits.
$\overline{\text{SIO1}}$	No	
$\overline{\text{SIO2}}$	No	
$\overline{\text{SIO3}}$	No	
Cn	Yes	Should be programmed high

Status Signals

ZERO = 1 if intermediate result = 0
N = 0
OVR = 0
C = 1 if carry-out

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 reqister 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
$\overline{\text{SIO0}}$	No	Pass internally generated end-fill bits.
$\overline{\text{SIO1}}$	No	
$\overline{\text{SIO2}}$	No	
$\overline{\text{SIO3}}$	No	
Cn	No	Inactive

Status Signals

ZERO = 1 if divisor = 0
N = 0
OVR = 0
Cn = 0

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).

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

9 0 Signed Divide Start SDIVIS

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inactive
$\overline{\text{SIO0}}$	No	Pass internally generated end-fill bits.
$\overline{\text{SIO1}}$	No	
$\overline{\text{SIO2}}$	No	
$\overline{\text{SIO3}}$	No	
Cn	Yes	Should be programmed high.

Status Signals

ZERO = 1 if intermediate result = 0
N = 0
OVR = 0
C = 1 if carry-out

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;

$Y \leftarrow R + S$ if the test fails
$Y \leftarrow R' + S + Cn$ if the test passes

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
$\overline{\text{SIO0}}$	No	Pass internally generated end-fill bits.
$\overline{\text{SIO1}}$	No	
$\overline{\text{SIO2}}$	No	
$\overline{\text{SIO3}}$	No	
Cn	Yes	Should be programmed high

Status Signals

ZERO = 1 if intermediate result = 0
N = 0
OVR = 0
C = 1 if carry-out

SDIVO Signed Divide Overflow Test A F

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

Signed Divide Overflow Test

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inactive
$\overline{\text{SIO0}}$	No	Inactive
$\overline{\text{SIO1}}$	No	Inactive
$\overline{\text{SIO2}}$	No	Inactive
$\overline{\text{SIO3}}$	No	Inactive
Cn	Yes	Should be programmed high

Status Signals

ZERO = 1 if divisor = 0
N = 0
OVR = 0
C = 1 if carry-out

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:

$Y \leftarrow S + 1$ if a fix is required
$Y \leftarrow S + 0$ if no fix is required

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.

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
$\overline{\text{SIO0}}$	No	Inactive
$\overline{\text{SIO1}}$	No	Inactive
$\overline{\text{SIO2}}$	No	Inactive
$\overline{\text{SIO3}}$	No	Inactive
Cn	Yes	Should be programmed high

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

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.
$\overline{\text{SIO0}}$	No	Inactive
$\overline{\text{SIO1}}$	No	Inactive
$\overline{\text{SIO2}}$	No	Inactive
$\overline{\text{SIO3}}$	No	Inactive
Cn	No	Inactive

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.

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.

[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 $\overline{EA}$ EB1-EB0	Dest Addr C5-C0	Destination Selects SELMQ	$\overline{WE3}$-$\overline{WE0}$	SELRF1-SELRF0	$\overline{OEA}$	$\overline{OEB}$	$\overline{OEY3}$-$\overline{OEY0}$	$\overline{OES}$	Cn	CF2-CF0
1111 0011	00 0001	00 0011	0 00	XX XXXX	0	XXXX	XX	X	X	0000	0	1	110
0001 0000	00 0001	00 0011	0 00	00 0101	0	0000	10	X	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

SET0 Reset Bit 1 8

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 $\overline{\text{SIO}}$ inputs programmed low perform the Reset Bit instruction. Bytes with their $\overline{\text{SIO}}$ inputs programmed high or floating pass S unaltered.

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
$\overline{\text{SIO0}}$	No	Byte-select
$\overline{\text{SIO1}}$	No	Byte-select
$\overline{\text{SIO2}}$	No	Byte-select
$\overline{\text{SIO3}}$	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.

Instr Code I7-I0	Mask (LSH) A3-A0	Oprd Addr B5-B0	Oprd Sel $\overline{EA}$	Oprd Sel EB1-EB0	Mask (MSH) C3-C0	Destination Selects SELMQ	$\overline{WE3}$-$\overline{WE0}$	SELRF1-SELRF0	$\overline{OEA}$	$\overline{OEB}$	$\overline{OEY3}$-$\overline{OEY0}$	$\overline{OES}$	Cn	CF2-CF0	$\overline{SIO3}$-$\overline{SIO0}$	$\overline{IESIO3}$-$\overline{IESIO0}$
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 ← C3-C0::A3-A0
Source	1010 0000 1000 0011 1011 1110 1011 1110	Sn ← RF(3)n
ALU	1010 0000 1000 0000 1011 0000 1011 1110	Fn ← Sn AND Rn
Destination	1010 0000 1000 0000 1011 0000 1011 1110	RF(8)n ← Fn or Sn[†]

[†] F = ALU result
n = nth byte
Register file 8 gets F if byte selected, S if byte not selected.

SET1 Set Bit 0 8

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 $\overline{\text{SIO}}$ inputs programmed low perform the Set Bit instruction. Bytes with their $\overline{\text{SIO}}$ inputs programmed high or floating pass S unaltered.

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
$\overline{\text{SIO0}}$	Yes	Byte-select
$\overline{\text{SIO1}}$	No	Byte-select
$\overline{\text{SIO2}}$	No	Byte-select
$\overline{\text{SIO3}}$	No	Byte-select
Cn	No	Inactive

0 8 Set Bit SET1

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		Mask (MSH) C3-C0	Destination Selects							Cn	CF2-CF0	$\overline{\text{SIO3}}$-$\overline{\text{SIO0}}$	$\overline{\text{IESIO3}}$-$\overline{\text{IESIO0}}$
			$\overline{\text{EA}}$	EB1-EB0		SELMQ	$\overline{\text{WE3}}$-$\overline{\text{WE0}}$	SELRF1-SELRF0	$\overline{\text{OEA}}$	$\overline{\text{OEB}}$	$\overline{\text{OEY3}}$-$\overline{\text{OEY0}}$	$\overline{\text{OES}}$				
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 ← C3-C0::A3-A0
Source	1010 0000 1000 0011 1011 1110 1011 1110	Sn ← RF(1)n
ALU	1010 0000 1000 0011 1011 1111 1011 1110	Fn ← Sn OR Rn
Destination	1010 0000 1000 0011 1011 1111 1011 1110	RF(1)n ← 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 $\overline{\text{SIO}}$ 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.
$\overline{\text{SIO0}}$	Yes	Fills a zero in LSB of each word if high; fills a one in LSB if low.
$\overline{\text{SIO1}}$	Yes	
$\overline{\text{SIO2}}$	Yes	
$\overline{\text{SIO3}}$	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.

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.

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel		Dest Addr C5-C0	Destination Selects							Cn	CF2-CF0	$\overline{\text{SIO3}}$-$\overline{\text{SIO0}}$	$\overline{\text{IESIO3}}$-$\overline{\text{IESIO0}}$	SSF
			$\overline{\text{EA}}$	EB1-EB0		SELMQ	$\overline{\text{WE3}}$-$\overline{\text{WE0}}$	SELRF1-SELRF0	$\overline{\text{OEA}}$	OEB	$\overline{\text{OEY3}}$-$\overline{\text{OEY0}}$	$\overline{\text{OES}}$					
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 ← RF(1)
Source	0100 0100 0000 0000 0111 0101 0011 0000	S ← DB bus
Intermediate Result	0101 0111 0000 1001 0011 1011 0100 1000	ALU Shifter ← R + S + Cn
Destination	1010 1110 0001 0010 0111 0110 1001 0001	RF(1) ← ALU shift result

SLAD Arithmetic Left Double Precision Shift

5	*

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 $\overline{\text{SIO}}$ 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.
$\overline{\text{SIO0}}$	Yes	Fills a zero in LSB of each word if high; fills a one in LSB if low.
$\overline{\text{SIO1}}$	Yes	
$\overline{\text{SIO2}}$	Yes	
$\overline{\text{SIO3}}$	Yes	
Cn	No	Affects arithmetic operation specified in bits I3-I0 of instruction field.

Arithmetic Left Double Precision Shift SLAD

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.

EXAMPLE (assumes a 32-bit configuration)

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 I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel EB1- $\overline{EA}$ EB0	Dest Addr C5-C0	Destination Selects							Cn	CF2-CF0	$\overline{SIO3}$-$\overline{SIO0}$	$\overline{IESIO3}$-$\overline{IESIO0}$	SSF
					SELMQ	$\overline{WE3}$-$\overline{WE0}$	SELRF1-SELRF0	$\overline{OEA}$	OEB	$\overline{OEY3}$-$\overline{OEY0}$	$\overline{OES}$					
0101 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 2408C618 (Hex), DB bus holds 26007530 (Hex), and MQ register holds 50A99A0E (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.

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.
$\overline{\text{SIO0}}$	No	Bit 7 of ALU result
$\overline{\text{SIO1}}$	No	Bit 15 of ALU result
$\overline{\text{SIO2}}$	No	Bit 23 of ALU result
$\overline{\text{SIO3}}$	No	Bit 31 of ALU result
Cn	No	Affects arithmetic operation specified in bits I3-I0 of instruction field.

6 * Circular Left Single Precision Shift SLC

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.

EXAMPLE (assumes a 32-bit configuration)

Perform a circular left shift of register 6 and store the result in register 1.

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel		Dest Addr C5-C0	Destination Selects							Cn	CF2-CF0	SSF
			$\overline{EA}$	EB1-EB0		SELMQ	$\overline{WE3}$-$\overline{WE0}$	SELRF1-SELRF0	$\overline{OEA}$	$\overline{OEB}$	$\overline{OEY3}$-$\overline{OEY0}$	$\overline{OES}$			
0110 0110	00 0110	XX XXXX	0	00	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	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.
$\overline{\text{SIO0}}$	No	Bit 7 of ALU result
$\overline{\text{SIO1}}$	No	Bit 15 of ALU result
$\overline{\text{SIO2}}$	No	Bit 23 of ALU result
$\overline{\text{SIO3}}$	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.

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.

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel		Dest Addr C5-C0	Destination Selects							Cn	CF2-CF0	SSF
			$\overline{EA}$	EB1-EB0		SELMQ	$\overline{WE3}$-$\overline{WE0}$	SELRF1-SELRF0	$\overline{OEA}$	$\overline{OEB}$	$\overline{OEY3}$-$\overline{OEY0}$	$\overline{OES}$			
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 50A99A0E (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, SMTC 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.

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
Yes	No	Yes

Shift Operations

ALU	MQ
None	None

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inactive
$\overline{\text{SIO0}}$	No	Inactive
$\overline{\text{SIO1}}$	No	Inactive
$\overline{\text{SIO2}}$	No	Inactive
$\overline{\text{SIO3}}$	No	Inactive
Cn	Yes	Should be programmed high for proper conversion

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	Oprd Addr	Oprd Addr	Oprd Sel	Dest Addr	Destination Selects								
I7-I0	A5-A0	B5-B0	EB1-EB0 $\overline{\text{EA}}$	C5-C0	SELMQ	$\overline{\text{WE3}}$-$\overline{\text{WE0}}$	SELRF1-SELRF0	$\overline{\text{OEA}}$	$\overline{\text{OEB}}$	$\overline{\text{OEY3}}$-$\overline{\text{OEY0}}$	$\overline{\text{OES}}$	Cn	CF2-CF0
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 ← RF(1)

Destination 1011 1100 0000 1001 0010 0111 1100 0000 RF(4) ← S' + Cn

Example 2: Assume register file 1 holds 550927C0 (Hex).

Source 0101 0101 0000 1001 0010 0111 1100 0000 S ← RF(1)

Destination 0101 0101 0000 1001 0010 0111 1100 0000 RF(4) ← 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:

$F \leftarrow R + S + Cn$	if the addition is required
$F \leftarrow S$	if no addition is required

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.

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	No

Shift Operations

ALU	MQ
Right	Right

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inactive
$\overline{\text{SIO0}}$	No	Passes LSB from ALU shifter to MSB of MQ shifter.
$\overline{\text{SIO1}}$	No	
$\overline{\text{SIO2}}$	No	
$\overline{\text{SIO3}}$	No	
Cn	Yes	Should be programmed low

Status Signals

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

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:

F ← R' + S + Cn	if the addition is required
F ← S	if no addition is required

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.

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

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

Shift Operations

ALU	MQ
Right	Right

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inactive
$\overline{\text{SIO0}}$	No	Passes LSB from ALU shifter to MSB of MQ shifter.
$\overline{\text{SIO1}}$	No	
$\overline{\text{SIO2}}$	No	
$\overline{\text{SIO3}}$	No	
Cn	Yes	Should be programmed low

Status Signals

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

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 $\overline{\text{SIO}}$ 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.

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)

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

Shift Operations (Conditional)

ALU	MQ
No	Left

2 0 Single-Length Normalize SNORM

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.

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

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 $\overline{EA}$ EB1-EB0	Dest Addr C5-C0	Destination Selects SELMQ							Cn	CF2-CF0
					SELMQ	$\overline{WE3}$-$\overline{WE0}$	SELRF1-SELRF0	$\overline{OEA}$	$\overline{OEB}$	$\overline{OEY3}$-$\overline{OEY0}$	$\overline{OES}$		
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).

Operand

Source	0011 0110 1001 1001 1101 1000 0100 1110	MQ shifter ← MQ register
Destination	0110 1101 0011 0011 1011 0000 1001 1100	MQ register ← MQ shifter

Count

Source	0000 0000 0000 0000 0000 0000 0000 0011	S ← RF(3)
Destination	0000 0000 0000 0000 0000 0000 0000 0100	RF(3) ← S + Cn

SRA Arithmetic Right Single Precision Shift

0	*

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.
$\overline{\text{SIO0}}$	No	LSB is shifted out from each word, which may be 1, 2, or 4 bytes long depending on selected configuration
$\overline{\text{SIO1}}$	No	
$\overline{\text{SIO2}}$	No	
$\overline{\text{SIO3}}$	No	
Cn	No	Affects arithmetic operation specified in bits I3-I0 of instruction field.

0 * Arithmetic Right Single Precision Shift SRA

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.

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.

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel		Dest Addr C5-C0	Destination Selects							Cn	CF2-CF0	SSF
			$\overline{EA}$	EB1-EB0		SELMQ	$\overline{WE3}$-$\overline{WE0}$	SELRF1-SELRF0	$\overline{OEA}$	$\overline{OEB}$	$\overline{OEY3}$-$\overline{OEY0}$	$\overline{OES}$			
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).

SRAD Arithmetic Right Double Precision Shift 1 *

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.
$\overline{\text{SIO0}}$	No	LSB of ALU shifter is passed to MSB of MQ shifter,
$\overline{\text{SIO1}}$	No	and LSB of MQ shifter is dropped.
$\overline{\text{SIO2}}$	No	
$\overline{\text{SIO3}}$	No	
Cn	No	Affects arithmetic operation specified in bits I3-I0 of instruction field.

Arithmetic Right Double Precision Shift SRAD

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.

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.

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel		Dest Addr C5-C0	Destination Selects							Cn	CF2-CF0	SSF
			$\overline{EA}$	EB1-EB0		SELMQ	$\overline{WE3}$-$\overline{WE0}$	SELRF1-SELRF0	$\overline{OEA}$	$\overline{OEB}$	$\overline{OEY3}$-$\overline{OEY0}$	$\overline{OES}$			
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 ← RF(1)
Source	0101 0001 0000 0000 0111 0101 0011 0000	S ← DB bus
Intermediate‡ Result	1001 1011 0000 1001 0011 1011 0100 1000	ALU Shifter ← R + S + Cn
Destination	0100 1101 1000 0100 1001 1101 1010 0100	RF(1) ← ALU shift result

LSH

Source	0001 0111 0010 1001 1001 1010 0000 1111	MQ shifter ← MQ register
Destination	0000 1011 1001 0100 1100 1101 0000 0111	MQ register ← 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.
$\overline{\text{SIO0}}$	No	Rotates LSB to MSB of the same word, which may be 1, 2, or 4 bytes long depending on configuration
$\overline{\text{SIO1}}$	No	
$\overline{\text{SIO2}}$	No	
$\overline{\text{SIO3}}$	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.

EXAMPLE (assumes a 32-bit configuration)

Perform a circular right shift of register 6 and store the result in register 1.

Instr	Oprd	Oprd	Oprd Sel		Dest	Destination Selects									
Code	Addr	Addr		EB1-	Addr		$\overline{WE3}$-	SELRF1-			$\overline{OEY3}$-			CF2-	
I7-I0	A5-A0	B5-B0	$\overline{EA}$	EB0	C5-C0	SELMQ	$\overline{WE0}$	SELRF0	$\overline{OEA}$	$\overline{OEB}$	$\overline{OEY0}$	$\overline{OES}$	Cn	CF0	SSF
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

SRCD Circular Right Double Precision Shift 9 *

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.
$\overline{\text{SIO0}}$	No	Rotates LSB of ALU shifter to MSB of MQ shifter,
$\overline{\text{SIO1}}$	No	and LSB of MQ shifter to MSB of ALU shifter
$\overline{\text{SIO2}}$	No	
$\overline{\text{SIO3}}$	No	
Cn	No	Affects arithmetic operation specified in bits I3-I0 of instruction field.

Circular Right Double Precision Shift

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.

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.

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel $\overline{EA}$	Oprd Sel EB1-EB0	Dest Addr C5-C0	Destination Selects SELMQ	Destination Selects $\overline{WE3}$-$\overline{WE0}$	Destination Selects SELRF1-SELRF0	Destination Selects $\overline{OEA}$	Destination Selects $\overline{OEB}$	Destination Selects $\overline{OEY3}$-$\overline{OEY0}$	Destination Selects $\overline{OES}$	Cn	CF2-CF0
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 ← RF(6)
Intermediate Result	0011 0111 0000 1000 1100 0110 0001 1000	ALU shifter ← R + Cn
Destination	1001 1011 1000 0100 0110 0011 0000 1100	RF(6) ← ALU shift result

LSH

Source	0101 0000 1010 1001 1001 1010 0000 1111	MQ shifter ← MQ register
Destination	0010 1000 0101 0100 1100 1101 0000 0111	MQ register ← MQ shift result

Logical Right Single Precision Shift

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.

Logical Right Single Precision Shift SRL

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		Dest Addr C5-C0	Destination Selects							Cn	CF2-CF0	$\overline{\text{SIO3-}}$ $\overline{\text{SIO0}}$	$\overline{\text{IESIO3-}}$ $\overline{\text{IESIO0}}$	SSF
			$\overline{\text{EA}}$	EB1-EB0		SELMQ	$\overline{\text{WE3-}}$ $\overline{\text{WE0}}$	SELRF1-SELRF0	$\overline{\text{OEA}}$	OEB	$\overline{\text{OEY3-}}$ $\overline{\text{OEY0}}$	$\overline{\text{OES}}$					
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.

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 $\overline{\mathrm{SIO}}$ 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
$\overline{\mathrm{SIO0}}$	Yes	Fills a zero in MSB if high or floating;
$\overline{\mathrm{SIO1}}$	Yes	fills a one MSB if low.
$\overline{\mathrm{SIO2}}$	Yes	
$\overline{\mathrm{SIO3}}$	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.

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.

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel $\overline{\text{EA}}$ EB1-EB0	Dest Addr C5-C0	Destination Selects							Cn	CF2-CF0	$\overline{\text{SIO3}}$-$\overline{\text{SIO0}}$	$\overline{\text{IESIO3}}$-$\overline{\text{IESIO0}}$
					SELMQ	$\overline{\text{WE3}}$-$\overline{\text{WE0}}$	SELRF1-SELRF0	$\overline{\text{OEA}}$	$\overline{\text{OEB}}$	$\overline{\text{OEY3}}$-$\overline{\text{OEY0}}$	$\overline{\text{OES}}$				
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 ← RF(1)
Intermediate Result	0010 1101 1010 1000 1100 0110 0001 0101	ALU Shifter ← S + Cn
Destination	1001 0110 1101 0100 0110 0011 0000 1010	RF(1) ← ALU shift result

LSH

Source	0101 0000 1010 1001 1001 1010 0000 1110	MQ shifter ← MQ register
Destination	1010 1000 0101 0100 1100 1101 0000 0111	MQ register ← 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

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
Yes	No	Yes

Shift Operations

ALU	MQ
None	None

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inactive
$\overline{\text{SIO0}}$	No	Inactive
$\overline{\text{SIO1}}$	No	Inactive
$\overline{\text{SIO2}}$	No	Inactive
$\overline{\text{SIO3}}$	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.

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel $\overline{EA}$ EB1-EB0	Dest Addr C5-C0	Destination Selects							Cn	CF2-CF0
					SELMQ	$\overline{WE3}$-$\overline{WE0}$	SELRF1-SELRF0	$\overline{OEA}$	$\overline{OEB}$	$\overline{OEY3}$-$\overline{OEY0}$	$\overline{OES}$		
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

SUBR Subtract R with Carry (R' + S + Cn)

* 2

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.
$\overline{\text{SIO0}}$	No	
$\overline{\text{SIO1}}$	No	
$\overline{\text{SIO2}}$	No	
$\overline{\text{SIO3}}$	No	
Cn	Yes	Two's complement subtraction if programmed high.

* 2 Subtract R with Carry (R′ + S + Cn) SUBR

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)

Subtract data in register 1 from data on the DB bus, and store the result in the MQ register.

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel EB1- $\overline{EA}$ EB0	Dest Addr C5-C0	Destination Selects							Cn	CF2-CF0
					SELMQ	$\overline{WE3}$-$\overline{WE0}$	SELRF1-SELRF0	$\overline{OEA}$	$\overline{OEB}$	$\overline{OEY3}$-$\overline{OEY0}$	$\overline{OES}$		
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

SUBS Subtract S with Carry (R + S' + Cn) | * | 3 |

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.
$\overline{SIO0}$	No	
$\overline{SIO1}$	No	
$\overline{SIO2}$	No	
$\overline{SIO3}$	No	
Cn	Yes	Two's complement subtraction if programmed high.

* 3 Subtract S with Carry (R + S′ + Cn) SUBS

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)

Subtract data on the DB bus from data in register 1, and store the result in the MQ register.

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel EB1- $\overline{EA}$ EB0	Dest Addr C5-C0	Destination Selects SELMQ	$\overline{WE3}$- $\overline{WE0}$	SELRF1- SELRF0	$\overline{OEA}$	$\overline{OEB}$	$\overline{OEY3}$- $\overline{OEY0}$	$\overline{OES}$	Cn	CF2- CF0
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 $\overline{\text{SIO}}$ 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.

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
$\overline{\text{SIO0}}$	Yes	Byte Select
$\overline{\text{SIO1}}$	Yes	Byte Select
$\overline{\text{SIO2}}$	Yes	Byte Select
$\overline{\text{SIO3}}$	Yes	Byte Select
Cn	No	Inactive

3 8 Test Bit (Zero) TBO

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 I7-I0	Mask (LSH) A3-A0	Oprd Addr B5-B0	Oprd Sel		Mask (MSH) C3-C0	Destination Selects							Cn	CF2-CF0	$\overline{\text{SIO3}}$-$\overline{\text{SIO0}}$	$\overline{\text{IESIO3}}$-$\overline{\text{IESIO0}}$
			$\overline{\text{EA}}$	EB1-EB0		SELMQ	$\overline{\text{WE3}}$-$\overline{\text{WE0}}$	SELRF1-SELRF0	$\overline{\text{OEA}}$	$\overline{\text{OEB}}$	$\overline{\text{OEY3}}$-$\overline{\text{OEY0}}$	$\overline{\text{OES}}$				
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	1110 0000 1110 0000 1110 0000 1110 0000	R ← Mask (C3-C0::A3-A0)
Source	1000 1000 0001 1100 1101 0000 0000 0011	SN ← RF(3)n[†]
Output	1	ZERO ← 1

[†] n = nth byte

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 $\overline{\text{SIO}}$ 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
$\overline{\text{SIO0}}$	Yes	Byte Select
$\overline{\text{SIO1}}$	Yes	Byte Select
$\overline{\text{SIO2}}$	Yes	Byte Select
$\overline{\text{SIO3}}$	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.

Instr Code I7-I0	Mask (LSH) A3-A0	Oprd Addr B5-B0	Oprd Sel		Mask (MSH) C3-C0	Destination Selects							Cn	CF2-CF0	$\overline{SIO3}$-$\overline{SIO0}$	$\overline{IESIO3}$-$\overline{IESIO0}$
			$\overline{EA}$	EB1-EB0		SELMQ	$\overline{WE3}$-$\overline{WE0}$	SELRF1-SELRF0	$\overline{OEA}$	$\overline{OEB}$	$\overline{OEY3}$-$\overline{OEY0}$	$\overline{OES}$				
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` Rn ← Mask (C3-C0::A3-A0)

Source `1000 1000 0001 1100 1101 0000 0000 0011` Sn ← RF(3)n[†]

Output `0` ZERO ← 0

[†] 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:

$F \leftarrow R + S$	if the test is failed
$F \leftarrow R' + S + Cn$	if the test is passed

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

C 0 Unsigned Divide Iterate UDIVI

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
$\overline{\text{SIO0}}$	No	Passes internally generated end-fill bit.
$\overline{\text{SIO1}}$	No	
$\overline{\text{SIO2}}$	No	
$\overline{\text{SIO3}}$	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 Instructjions" 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:

$$F \leftarrow R' + S + Cn$$

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

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

Shift Operations

ALU	MQ
Left	Left

B 0 Unsigned Divide Start UDIVIS

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inactive
$\overline{\text{SIO0}}$	No	Passes internally generated end-fill bit.
$\overline{\text{SIO1}}$	No	
$\overline{\text{SIO2}}$	No	
$\overline{\text{SIO3}}$	No	
Cn	Yes	Should be programmed high.

Status Signals

ZERO = 1 if intermediate result = 0
N = 0
OVR = 1 if divide overflow
C = 1 if carry-out

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:

$Y \leftarrow R + S$ if the test is failed
$Y \leftarrow R' + S + Cn$ if the test is passed

The contents of the MQ register are shifted one bit to the left; the unfixed quotient bit is circulated into the least significant bit.

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	Left

F 0 Unsigned Divide Terminate UDIVIT

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inactive
$\overline{\text{SIO0}}$	No	Passes internally generated end-fill bit.
$\overline{\text{SIO1}}$	No	
$\overline{\text{SIO2}}$	No	
SIO3	No	
Cn	Yes	Should be programmed high.

Status Signals

ZERO = 1 if intermediate result = 0
N = 0
OVR = 0
C = 1 if carry-out

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:

$F \leftarrow R + S + Cn$	if the addition is required
$F \leftarrow S$	if no addition is required

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.

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
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 high.

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

XOR Exclusive OR (R XOR S) | * | 9 |

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.
$\overline{\text{SIO0}}$	No	
$\overline{\text{SIO1}}$	No	
$\overline{\text{SIO2}}$	No	
$\overline{\text{SIO3}}$	No	
Cn	No	Inactive

* | 9 Exclusive OR (R XOR S) XOR

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)

Exclusive OR the contents of register 3 and register 5, and store the result in register 5.

Instr Code I7-I0	Oprd Addr A5-A0	Oprd Addr B5-B0	Oprd Sel		Dest Addr C5-C0	Destination Selects							Cn	CF2-CF0
			$\overline{EA}$	EB1-EB0		SELMQ	$\overline{WE3}$-$\overline{WE0}$	SELRF1-SELRF0	$\overline{OEA}$	$\overline{OEB}$	$\overline{OEY3}$-$\overline{OEY0}$	$\overline{OES}$		
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

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.

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-μm 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.

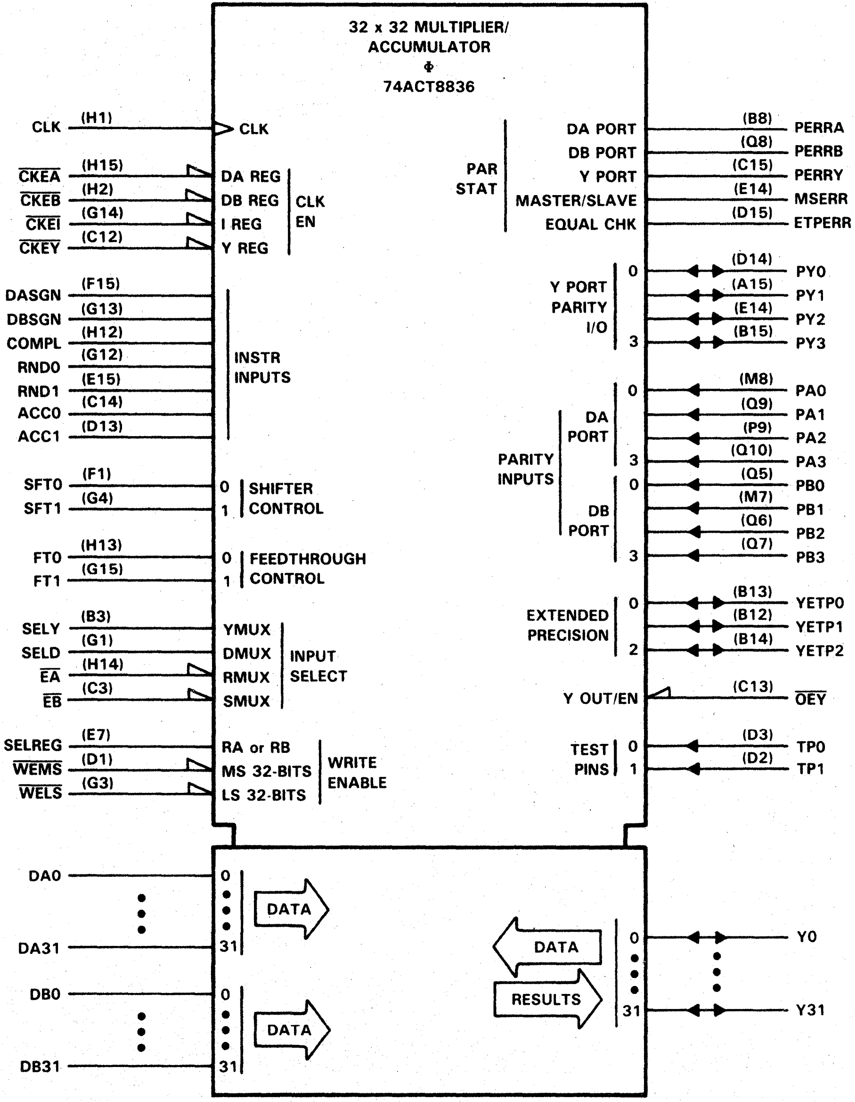
A multiply/accumulate mode is provided to add or subtract the accumulator from the product or the complement of the product. The accumulator is 67 bits wide to accommodate possible overflow. A warning flag (ETPERR) indicates whether overflow has occurred.

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.

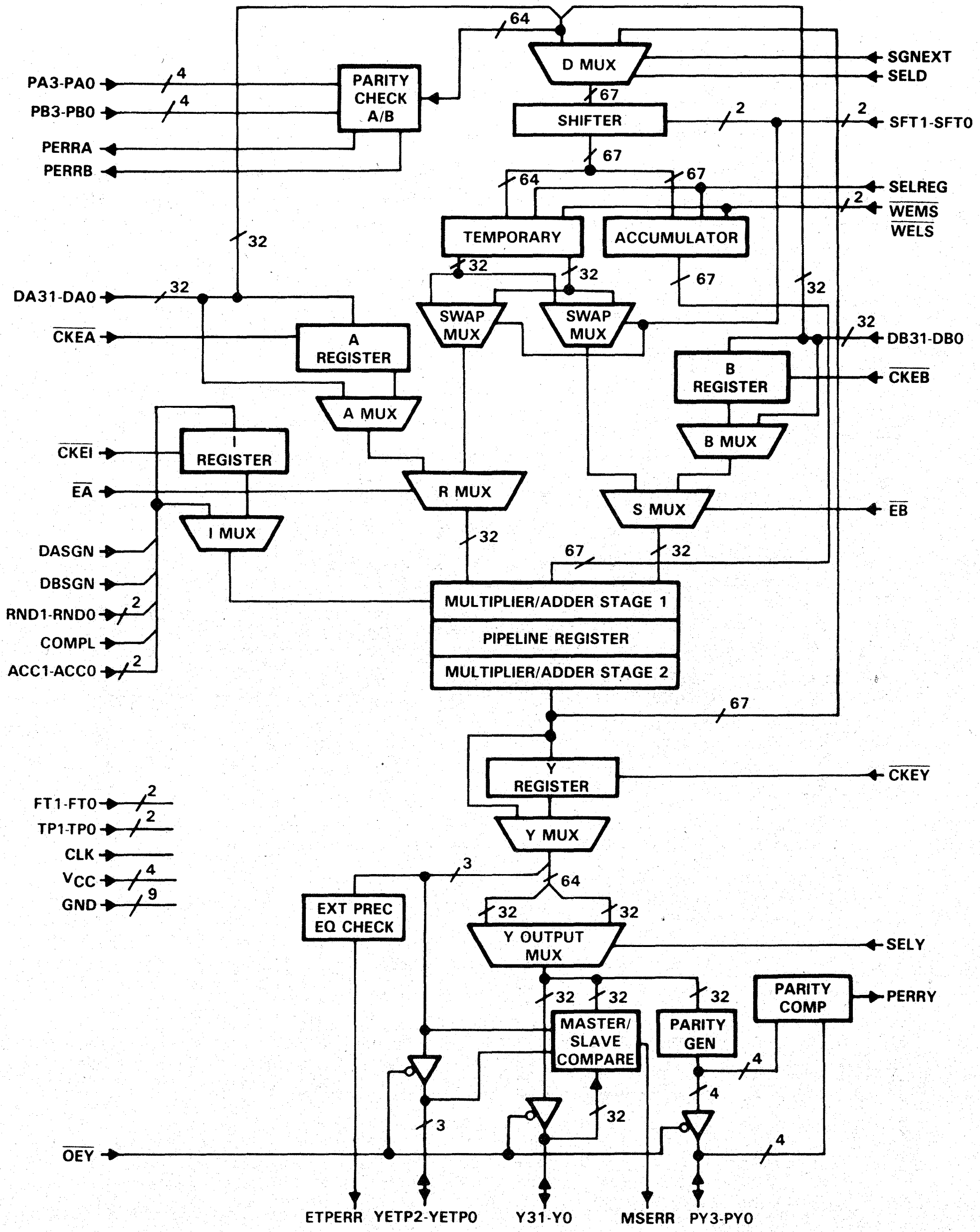
The SN74ACT8836 is characterized for operation from 0°C to 70°C.

EPIC is a trademark of Texas Instruments Incorporated

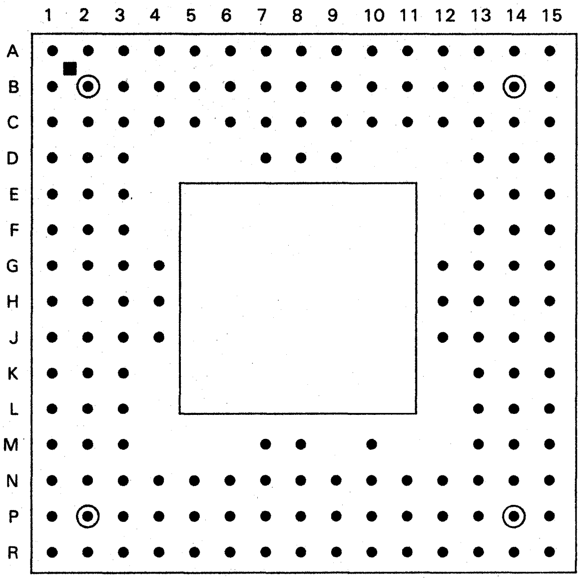
logic symbol



functional block diagram (positive logic)



GB PIN-GRID-ARRAY PACKAGE
(TOP VIEW)



GB PACKAGE PIN ASSIGNMENTS

PIN		PIN		PIN		PIN		PIN		PIN	
NO.	NAME	NO.	NAME	NO.	NAME	NO.	NAME	NO.	NAME	NO.	NAME
A1	Y8	B12	YETP1	D14	PY0	H12	COMPL	M3	DB18	P5	DB25
A2	Y10	B13	YETP0	D15	ETPERR	H13	FT0	M7	PB1	P6	DB29
A3	Y11	B14	YETP2	E1	SELREG	H14	EA	M8	PA0	P7	DB31
A4	Y13	B15	PY3	E2	Y3	H15	$\overline{CKEA}$	M10	DA6	P8	PERRA
A5	Y14	C1	Y0	E3	GND	J1	DB2	M13	DA16	P9	PA2
A6	Y16	C2	Y4	E13	GND	J2	DB3	M14	DA17	P10	DA2
A7	Y18	C3	EB	E14	PY2	J3	DB5	M15	DA25	P11	DA8
A8	Y19	C4	Y5	E15	RND1	J4	DB7	N1	DB10	P12	DA12
A9	Y21	C5	V_{CC}	F1	SFT0	J12	DA26	N2	DB19	P13	DA14
A10	Y23	C6	GND	F2	Y1	J13	DA24	N3	DB20	P14	DA11
A11	Y25	C7	Y15	F3	GND	J14	DA30	N4	DB21	P15	DA21
A12	Y27	C8	GND	F13	GND	J15	DA31	N5	DB23	R1	DB14
A13	Y28	C9	Y22	F14	MSERR	K1	DB4	N6	DB27	R2	DB26
A14	Y30	C10	GND	F15	DASGN	K2	DB9	N7	V_{CC}	R3	DB28
A15	PY1	C11	V_{CC}	G1	SELD	K3	DB11	N8	GND	R4	DB30
B1	Y2	C12	$\overline{CKEY}$	G2	SGNEXT	K13	DA22	N9	DA0	R5	PB0
B2	Y6	C13	$\overline{OEY}$	G3	$\overline{WELS}$	K14	DA28	N10	DA4	R6	PB2
B3	SELY	C14	ACC0	G4	SFT1	K15	DA29	N11	DA10	R7	PB3
B4	Y7	C15	PERRY	G12	RND0	L1	DB6	N12	DA13	R8	PERRB
B5	Y9	D1	$\overline{WEMS}$	G13	DBSGN	L2	DB15	N13	DA15	R9	PA1
B6	Y12	D2	TP1	G14	$\overline{CKEI}$	L3	DB13	N14	DA19	R10	PA3
B7	Y17	D3	TP0	G15	FT1	L13	DA18	N15	DA23	R11	DA1
B8	Y20	D7	GND	H1	CLK	L14	DA20	P1	DB12	R12	DA3
B9	Y26	D8	V_{CC}	H2	$\overline{CKEB}$	L15	DA27	P2	DB16	R13	DA5
B10	Y29	D9	Y24	H3	DB0	M1	DB8	P3	DB24	R14	DA7
B11	Y31	D13	ACC1	H4	DB1	M2	DB17	P4	DB22	R15	DA9

PIN		I/O	DESCRIPTION
NAME	NO.		
ACC0	C14	I	Accumulate mode opcode (see Table 2)
ACC1	D13		
CLK	H1	I	System clock
$\overline{\text{CKEA}}$	H15	I	Clock enable for A register, active low
$\overline{\text{CKEB}}$	H2	I	Clock enable for B register, active low
$\overline{\text{CKEI}}$	G14	I	Clock enable for I register, active low
$\overline{\text{CKEY}}$	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	I	DA port input data bits 0 through 31
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		
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

PIN NAME	PIN NO.	I/O	DESCRIPTION
DB0	H3	I	DB port input data bits 0 through 31
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		
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.
$\overline{\text{EA}}$	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.
$\overline{\text{EB}}$	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 FT1	H13 G15	I	Feedthrough control signals for A, B, I, Pipeline and Y registers (see Table 4).

PIN		I/O	DESCRIPTION
NAME	NO.		
GND	C6		Ground pins. All ground pins should be used and connected.
GND	C8		
GND	C10		
GND	D7		
GND	E3		
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.
$\overline{\text{OEY}}$	C13	I	Y, YETP2-YETP0, and PY3-PY0 output enable, active low.
PA0	M8	I	Parity input data bus for DA input data
PA1	R9		
PA2	P9		
PA3	R10		
PB0	R5	I	Parity input data bus for DB input data
PB1	M7		
PB2	R6		
PB3	R7		
PY0	D14	I/O	Y output parity data bus. Outputs data from parity generator ($\overline{\text{OEY}}$ = L) or inputs external parity data ($\overline{\text{OEY}}$ = H).
PY1	A15		
PY2	E14		
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.
RND0	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		
TP0	D3	I	Test pins (see Table 5)
TP1	D2		
V_{CC}	C5		Supply voltage (5 V)
V_{CC}	C11		
V_{CC}	D8		
V_{CC}	N7		
$\overline{\text{WEMS}}$	D1	I	Write enable for most significant 32 bits of temporary register and accumulator active low.
$\overline{\text{WELS}}$	G3	I	Write enable for least significant 32 bits of temporary register and accumulator active low.

PIN NAME	PIN NO.	I/O	DESCRIPTION
Y0	C1	I/O	Y port data bus. Outputs data from Y register ($\overline{\text{OEY}}$ = L); inputs data to master/slave comparator ($\overline{\text{OEY}}$ = H).
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		
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		
YETP0	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
RNDO	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
ACC0 ACC1	See Table 2	See Table 2

TEXAS INSTRUMENTS
POST OFFICE BOX 655012 • DALLAS, TEXAS 75265

TABLE 2. MULTIPLIER/ADDER CONTROL INPUTS

ACC1	ACC0	$\overline{EA}$	$\overline{EB}$	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		Registers Bypassed				
FT1	FT0	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.

architectual 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

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, $\overline{\text{CKEA}}$ and $\overline{\text{CKEB}}$, 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

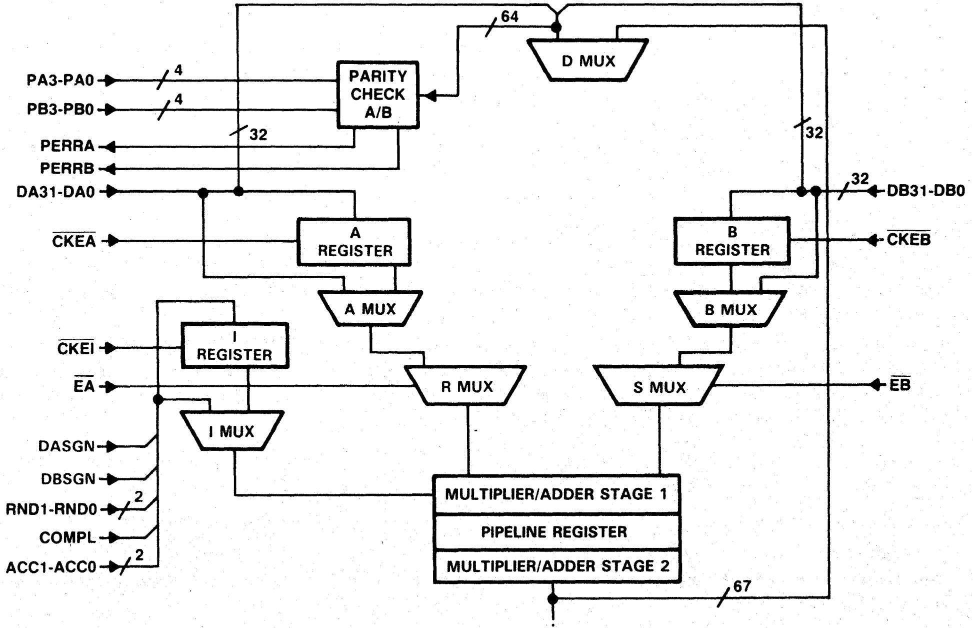
Instruction inputs to the device are shown in Table 1. These signals control signed, unsigned, and mixed multiplication modes, fractional and integer rounding, accumulator operations and complementing of products. They can be latched into instruction register I when clock enable $\overline{\text{CKEI}}$ 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 muliplier 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-ACC0 control the operation of the multiplier/adder. Possible operations are shown in Table 2.



INPUT REGISTERS AND PARITY CHECK

l, 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.

ıultiplier/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.

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, $\overline{\mathrm{WELS}}$ and $\overline{\mathrm{WEMS}}$, 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 $\overline{\mathrm{WELS}}$ is low; the most significant bits when $\overline{\mathrm{WEMS}}$ is low. When $\overline{\mathrm{WELS}}$ and $\overline{\mathrm{WEMS}}$ are both low, the entire word from the shifter is loaded into the selected register.

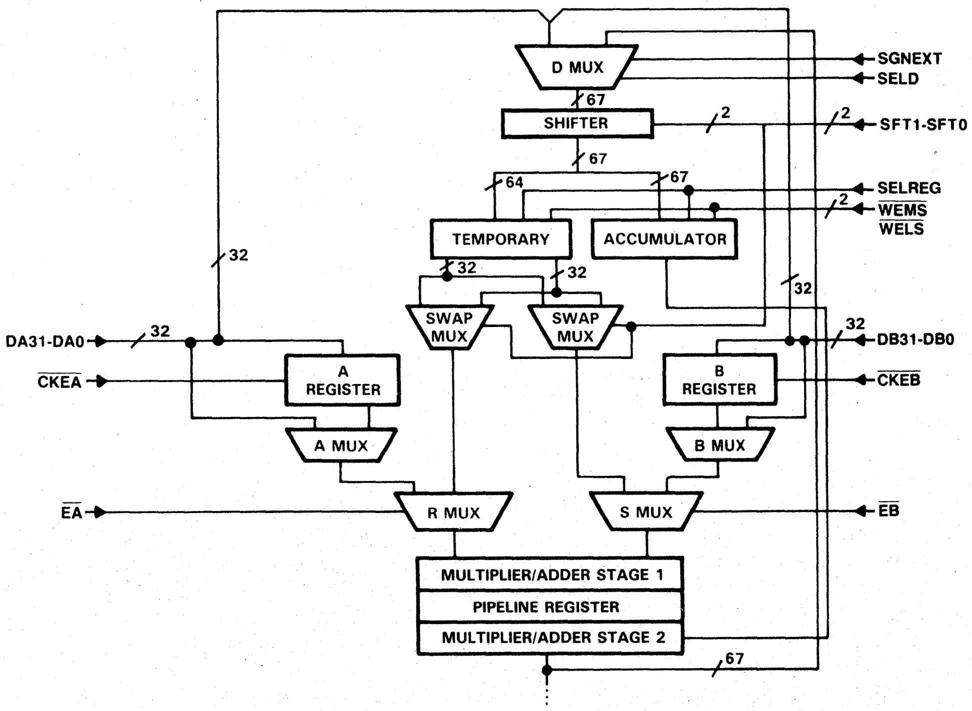


FIGURE 1. TEMPORARY REGISTER AND ACCUMULATOR

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 $\overline{\text{CKEY}}$ 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.

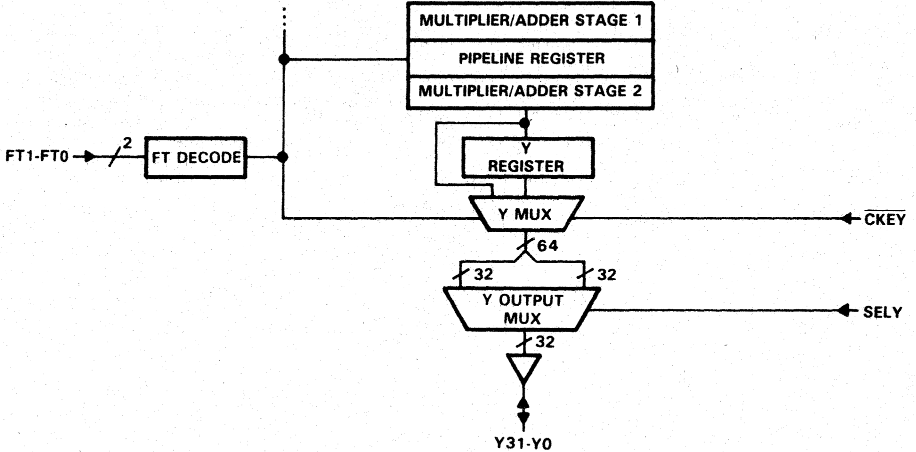


FIGURE 2. Y OUTPUT

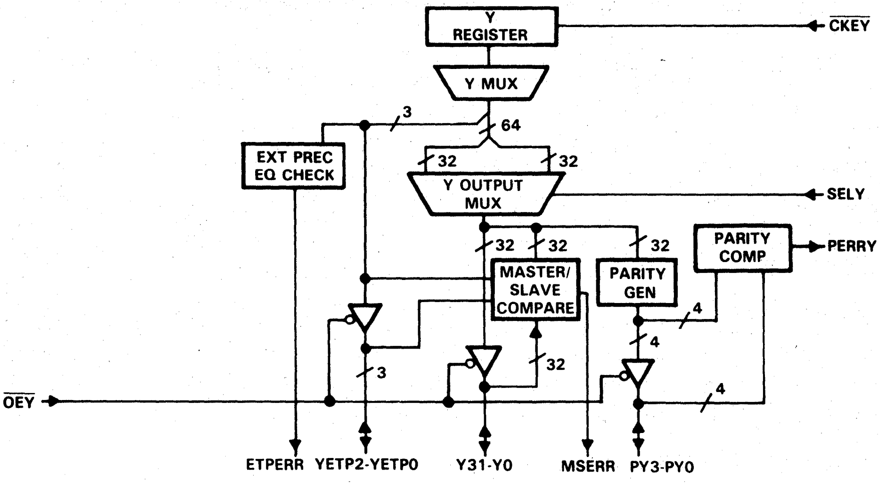


FIGURE 3. OUTPUT ERROR CONTROL

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 $\overline{OEY}$ 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:

PRDT = XTP : : MSP : : LSP

or

PRDT = YETP2 - YETP0 : : Y63 - Y0

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

31	30	29	....	2	1	0
-2^{31} Sign)	2^{30}	2^{29}	....	2^{2}	2^{1}	2^{0}

Input Operand B

31	30	29	....	2	1	0
-2^{31} (Sign)	2^{30}	2^{29}	....	2^{2}	2^{1}	2^{0}

Unsigned Integer Inputs

Input Operand A

31	30	29	....	2	1	0
2^{31}	2^{30}	2^{29}	....	2^{2}	2^{1}	2^{0}

Input Operand B

31	30	29	....	2	1	0
2^{31}	2^{30}	2^{29}	....	2^{2}	2^{1}	2^{0}

Two's Complement Fractional Inputs

Input Operand A

31	30	29	....	2	1	0
-2^{0} ign)	2^{-1}	2^{-2}	....	2^{-29}	2^{-30}	2^{-31}

Input Operand B

31	30	29	....	2	1	0
-2^{0} (Sign)	2^{-1}	2^{-2}	....	2^{-29}	2^{-30}	2^{-31}

Unsigned Fractional Inputs

Input Operand A

31	30	29	...	2	1	0
2^{-1}	2^{-2}	2^{-3}	...	2^{-30}	2^{-31}	2^{-32}

Input Operand B

31	30	29	...	2	1	0
2^{-1}	2^{-2}	2^{-3}	...	2^{-30}	2^{-31}	2^{-32}

Two's Complement Integer Outputs

Extended Product (YETP2-YETP0)

66	65	64
-2^{66}	2^{65}	2^{64}

(Sign)

Most Significant Product (Y63-Y32)

63	62	61	...	30	31	32
2^{63}	2^{62}	2^{61}	...	2^{34}	2^{33}	2^{32}

Least Significant Product (Y31-Y0)

31	30	29	...	2	1	0
2^{31}	2^{30}	2^{29}	...	2^{2}	2^{1}	2^{0}

Unsigned Integer Outputs

Extended Product (YETP2-YETP0)

66	65	64
2^{66}	2^{65}	2^{64}

Most Significant Product (Y63-Y32)

63	62	61	...	30	31	32
2^{63}	2^{62}	2^{61}	...	2^{34}	2^{33}	2^{32}

Least Significant Product (Y31-Y0)

31	30	29	...	2	1	0
2^{31}	2^{30}	2^{29}	...	2^{2}	2^{1}	2^{0}

Two's Complement Fractional Outputs

Extended Product (YETP2-YETP0)

66	65	64
-2^{4}	2^{3}	2^{2}

(Sign)

Most Significant Product (Y63-Y32)

63	62	61	...	30	31	32
2^{1}	2^{0}	2^{-1}		2^{-28}	2^{-29}	2^{-30}

Least Significant Product (Y31-Y0)

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)

66	65	64
2^{2}	2^{1}	2^{0}

Most Significant Product (Y63-Y32)

63	62	61	...	30	31	32
2^{-1}	2^{-2}	2^{-3}		2^{-30}	2^{-31}	2^{-32}

Least Significant Product (Y31-Y0)

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
$A_m = a_s, a_{62}, a_{61}, \ldots, a_{32}$
and
$A_l = a_{31}, a_{30}, a_{29}, \ldots, a_0$ (a_0 = LSB)
Therefore:
$A = (A_m \times 2^{32}) + A_l$
Likewise:
$B = (B_m \times 2^{32}) + B_l$
Thus:

$$A \times B = [(A_m \times 2^{32}) + A_l] \times [(B_m \times 2^{32}) + B_l]$$
$$= (A_m \times B_m)\,2^{64} + (A_m \times B_l + A_l] \times B_m)2^{32} + A_l \times B_l$$

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 (Am x Bm), 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.

Example 1. Single Precision Multiply, 32-Bit Result

Operand Select R bus S bus		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 Comple-ment	Multiplier/ Adder Mode											I	A	B	Y		
$\overline{EA}$	$\overline{EB}$	DASGN	DBSGN	RND1	RND0	COMPL	ACC1	ACC0	SELD	SGNEXT	SFT1	SFT0	SELREG	$\overline{WEH}$	$\overline{WEL}$	FT1	FT0	$\overline{CKEI}$	$\overline{CKEA}$	$\overline{CKEB}$	$\overline{CKEY}$	SELY	$\overline{OEY}$
1	1	1	1	0	1	0	0	0	0	X	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		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 Comple-ment	Multiplier/ Adder Mode											I	A	B	Y		
	$\overline{EA}$	$\overline{EB}$	DASGN	DBSGN	RND1	RND0	COMPL	ACC1	ACC0	SELD	SGNEXT	SFT1	SFT0	SELREG	$\overline{WEH}$	$\overline{WEL}$	FT1	FT0	$\overline{CKEI}$	$\overline{CKEA}$	$\overline{CKEB}$	$\overline{CKEY}$	SELY	$\overline{OEY}$
(1)	1	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	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	1	1	1	1	X	X
(3)	1	1	1	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	1	1	1	1	0	0
(4)	1	1	1	1	0	0	0	0	1	X	0	0	0	X	X	X	0	0	1	1	1	1	0	0
(5)	1	1	1	1	0	0	0	0	1	X	0	0	0	X	X	X	0	0	1	1	1	1	1	0

Example 3. Newton-Raphson Division

Instruction Number	Operand Select R bus S bus		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 Comple-ment	Multiplier/ Adder Mode											I	A	B	Y		
	$\overline{EA}$	$\overline{EB}$	DASGN	DBSGN	RND1	RND0	COMPL	ACC1	ACC0	SELD	SGNEXT	SFT1	SFT0	SELREG	$\overline{WEH}$	$\overline{WEL}$	FT1	FT0	$\overline{CKEI}$	$\overline{CKEA}$	$\overline{CKEB}$	$\overline{CKEY}$	SELY	$\overline{OEY}$
Repeat N Times*																								
(1)	0	1	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	0	0	0	0	0	1	0
(2)	0	0	0	0	0	1	0	0	0	0	0	0	1	1	0	1	1	0	0	0	0	0	1	0
End Repeat																								
(3)	0	1	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	0	0	0	0	0	1	0
(4)	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	1	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, $Xi+1 = Xi(2\text{-}BXi)$ 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.

Step 1 $X1 = X0(2\text{-}BX0)$
Step 2 $X2 = X1(2\text{-}BX1)$
Step 3 $X3 = X2(2\text{-}BX2)$

Step n $Xn = Xn\text{-}1(2\text{-}BXn\text{-}1)$

The successive approximation of Xi, for all i, approaches the reciprocal 1/B as the number of iterations increases; that is

$$\lim_{i \to n} Xi = 1/B$$

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 xi from its corresponding $xi + 1$. If the difference $|Xi+1 - Xi|$ 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, X0, is called the seed approximation. The seed must be supplied to the Newton-Raphson process externally and must fall within the range of $0 < X0 < 2/B$ if B is greater than 0 or $2/B < X0 < 0$ 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 $1/2 \le B < 1$.

Since Xi from Newton-Raphson must lie between $0 < Xi < 2/B$ and since the range of the positive fraction B is $1/2 \le B < 1$, then the limits of Xi become $1 \le Xi < 2$.

The range of $-BXi$ will therefore be $-2 \le -BXi \le -1/2$.

The limits of $-BXi$ 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			63	62	61	......	2	1	0
	66	65	64							
−2	1	1	1	0	0	0	......	0	0	0
−½	1	1	1	1	1	0	......	0	0	0

The diagram indicates that $-BXi$ is always of the form:

1 1 1 d0 . d1 d2...........dn − 2 dn − 1

The next step in Newton-Raphson is to complete the 2 − BXi equation. The fractional representation of 2 is:

0 0 1 0 . 0 0 ....0 0

Completion of the 2 − BXi 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	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 Xi is stored in the temporary register:

Step 1

Accumulator ← −(DB × temporary register)
= 2 − BXi

Step 2

Temporary Register ← Left shift one bit of
(accumulator times temporary register)
= Xi + 1

= Xi (2 − BXi)

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 Xi 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
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			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}.

setup and hold times

PARAMETER		MIN	MAX	UNIT
t_{su1}	Instruction before CLK↑	14		ns
t_{su2}	Data before CLK↑	12		
t_{su3}	$\overline{\text{CKEA}}$ before CLK↑	14		
t_{su4}	$\overline{\text{CKEB}}$ before CLK↑	14		
t_{su5}	$\overline{\text{CKEI}}$ before CLK↑	10		
t_{su6}	$\overline{\text{CKEY}}$ before CLK↑	19		
t_{su7}	SELREG before CLK↑	12		
t_{su8}	$\overline{\text{WEMS}}$ before CLK↑	11		
t_{su9}	$\overline{\text{WELS}}$ before CLK↑	11		
t_{h1}	Instruction after CLK↑	0		
t_{h2}	Data after CLK↑	0		
t_{h3}	$\overline{\text{CKEA}}$ after CLK↑	0		
t_{h4}	$\overline{\text{CKEB}}$ after CLK↑	0		
t_{h5}	$\overline{\text{CKEI}}$ after CLK↑	0		
t_{h6}	$\overline{\text{CKEY}}$ after CLK↑	0		
t_{h7}	SELREG after CLK↑	0		
t_{h8}	$\overline{\text{WEMS}}$ after CLK↑	0		
t_{h9}	$\overline{\text{WELS}}$ after CLK↑	0		

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}[†]	CLK	PIPE	11			36	ns
t_{pd2}[†]	PIPE	Y REG	11			36	
t_{pd3}[†]	PIPE	ACCUM	11			36	
t_{pd4}[†]	Y REG	Y	All modes			18	
t_{pd5}	SELY	Y	All modes			18	
t_{pd6}[†]	CLK	Y REG	01			54	
t_{pd7}[†]	CLK	ACCUM	10 or 01			67	
t_{pd8}	CLK	Y	10			67	
t_{pd9}	DATA	Y	00			60	
t_{pd10}[†]	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}	$\overline{\text{OEY}}$	YETP	All modes			20	
t_{en1}	$\overline{\text{OEY}}$	Y	All modes			20	
t_{dis1}	$\overline{\text{OEY}}$	YETP	All modes			15	
t_{dis2}	$\overline{\text{OEY}}$	Y	All modes			15	

clock requirements

PARAMETER		SN74ACT8836		UNIT
		MIN	MAX	
t_{w1}	CLK high	5		ns
t_{w2}	CLK low	20		

[†] These parameters cannot be measured but can be inferred from device operation and other measurable parameters.

PARAMETER MEASUREMENT INFORMATION

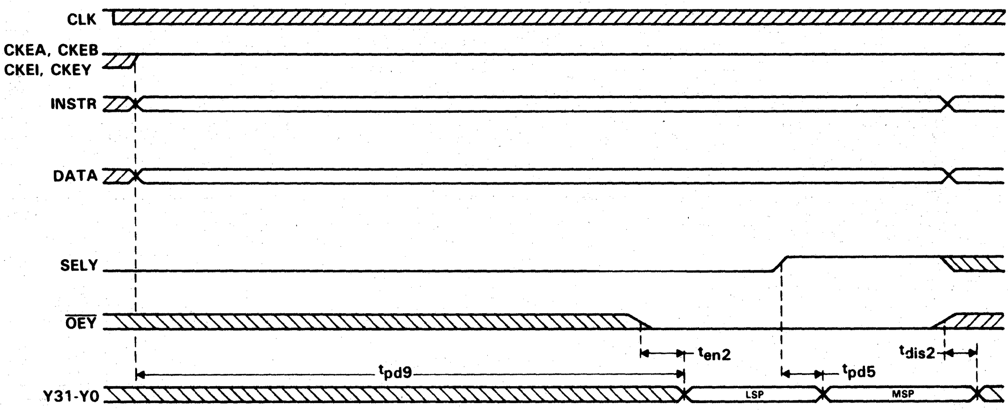


FIGURE 5. FULL FLOWTHROUGH MODE (FT = 00)

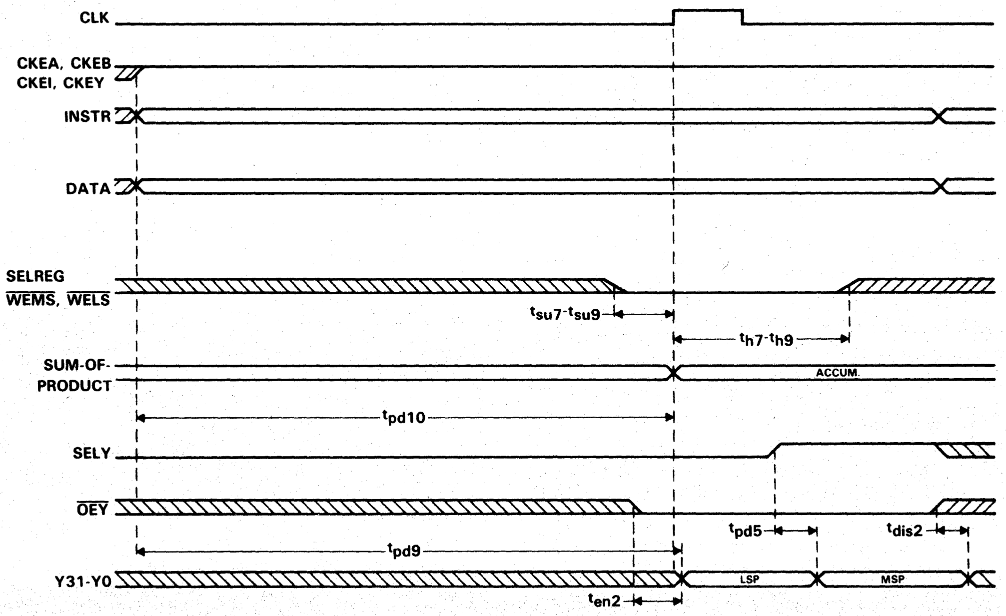


FIGURE 6. FULL FLOWTHROUGH MODE, ACCUMULATOR MODE (FT = 00)

PARAMETER MEASUREMENT INFORMATION

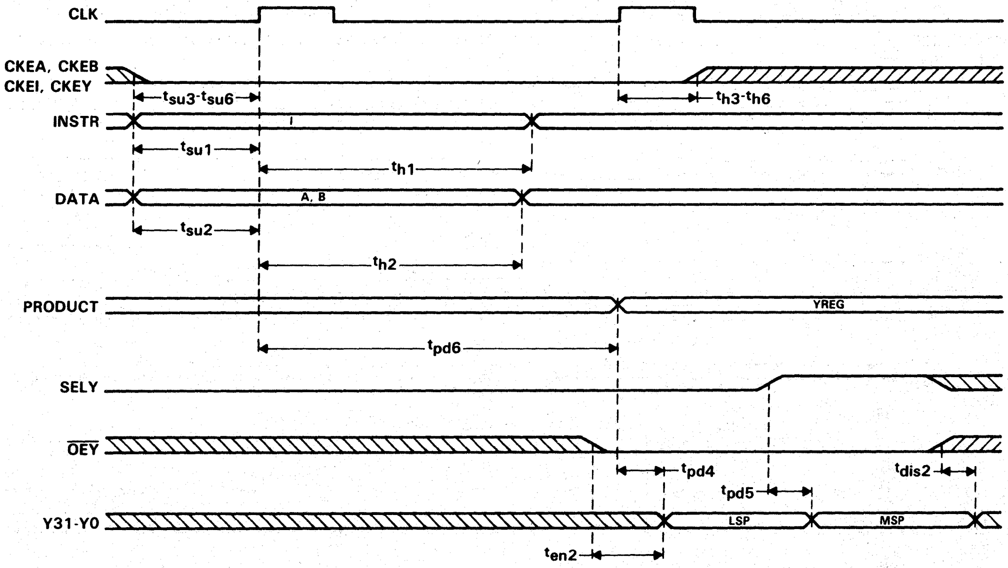


FIGURE 7. FLOWTHROUGH PIPE ONLY VOLTAGE WAVEFORMS (FT = 01)

PARAMETER MEASUREMENT INFORMATION

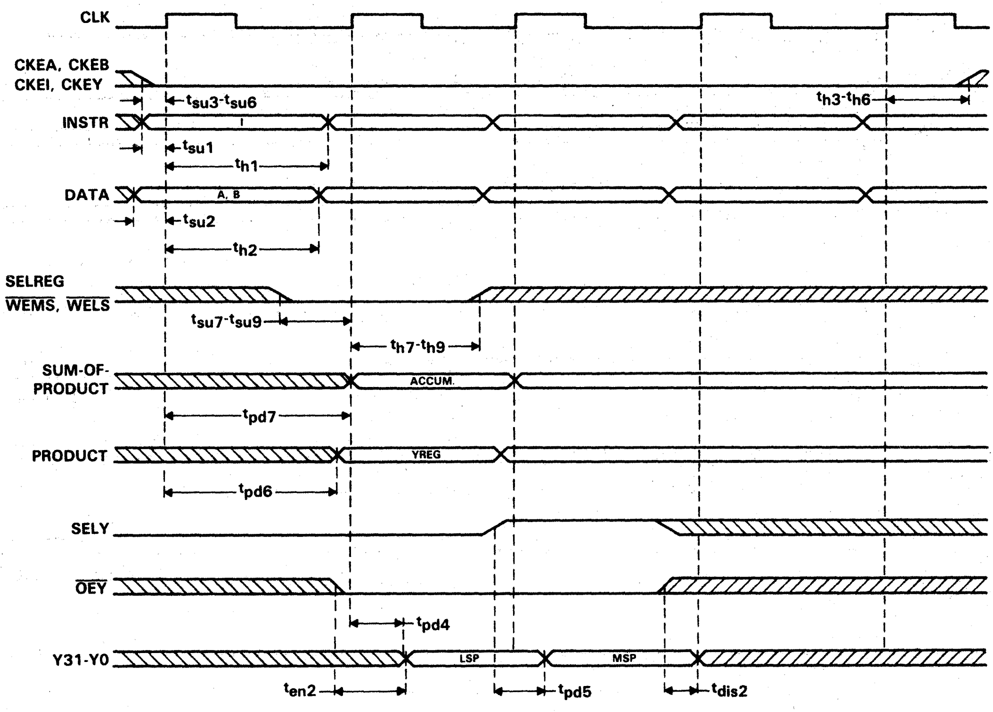


FIGURE 8. FLOWTHROUGH PIPE ONLY, ACCUMULATOR MODE (FT = 01)

PARAMETER MEASUREMENT INFORMATION

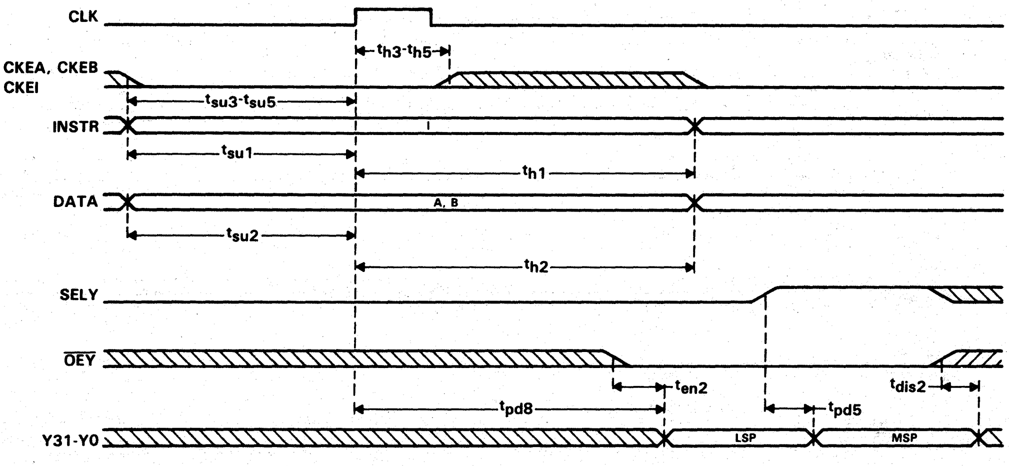


FIGURE 9. FLOWTHROUGH PIPE AND Y ONLY (FT = 10)

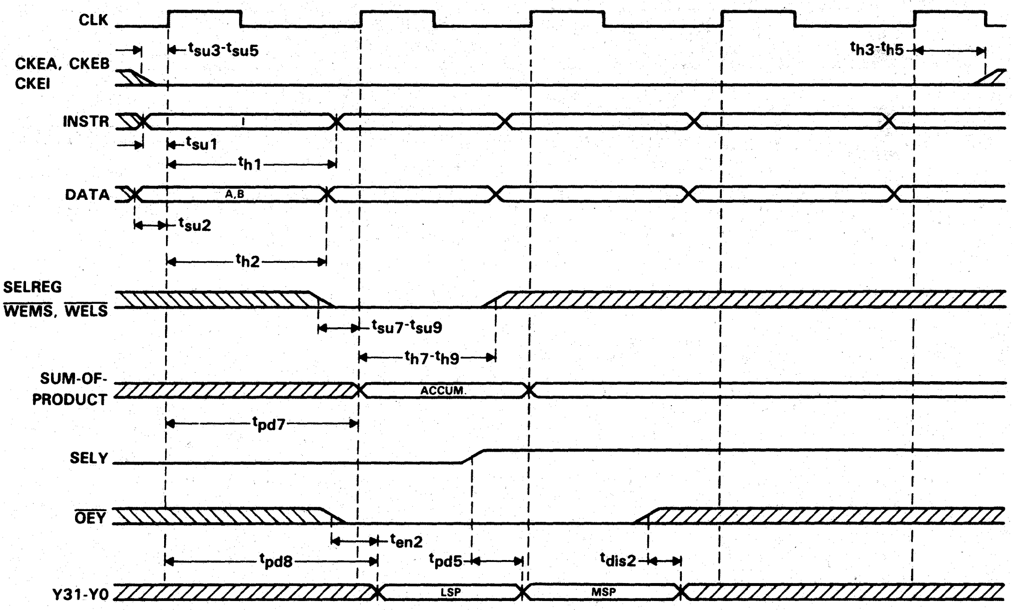


FIGURE 10. FLOWTHROUGH PIPE AND Y ONLY, ACCUMULATOR MODE (FT = 10)

PARAMETER MEASUREMENT INFORMATION

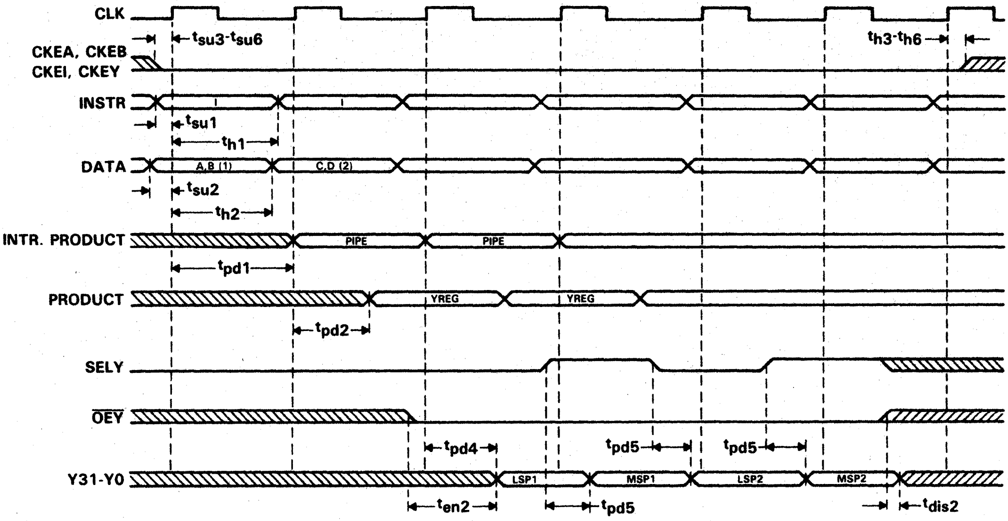


FIGURE 11. ALL REGISTERS ENABLED (FT = 11)

PARAMETER MEASUREMENT INFORMATION

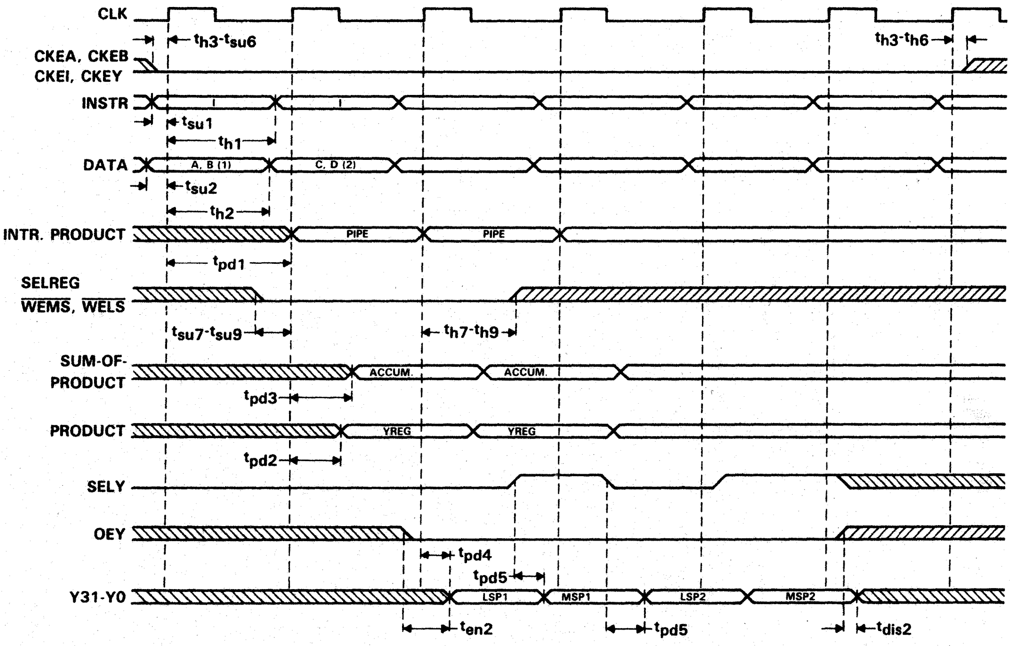
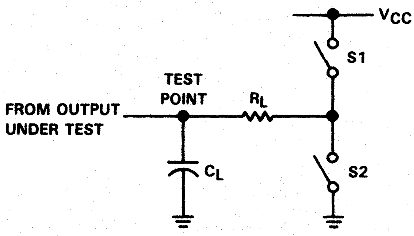


FIGURE 12. ALL REGISTERS ENABLED, ACCUMULATOR MODE (FT = 11)



PARAMETER		R_L	C_L†	S_1z	S_2
t_{en}	t_{PZH}	1 kΩ	50 pF	OPEN	CLOSED
	t_{PZL}			CLOSED	OPEN
t_{dis}	t_{PHZ}	1 kΩ	50 pF	OPEN	CLOSED
	t_{PLZ}			CLOSED	OPEN
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 ≤ 1 MHz, Z_{out} = 50 Ω, t_r = 50 Ω, t_f = 6 ns.

FIGURE 13. LOAD CIRCUIT

SN74ACT8837
64-Bit Floating Point Unit

- Multiplier and ALU in **One Chip**
- 60-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

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.

EPIC is a trademark of Texas Instruments Incorporated.

Contents

Contents (Continued)

Contents (Concluded)

List of Illustrations

List of Illustrations (Concluded)

List of Tables

List of Tables (Concluded)

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

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

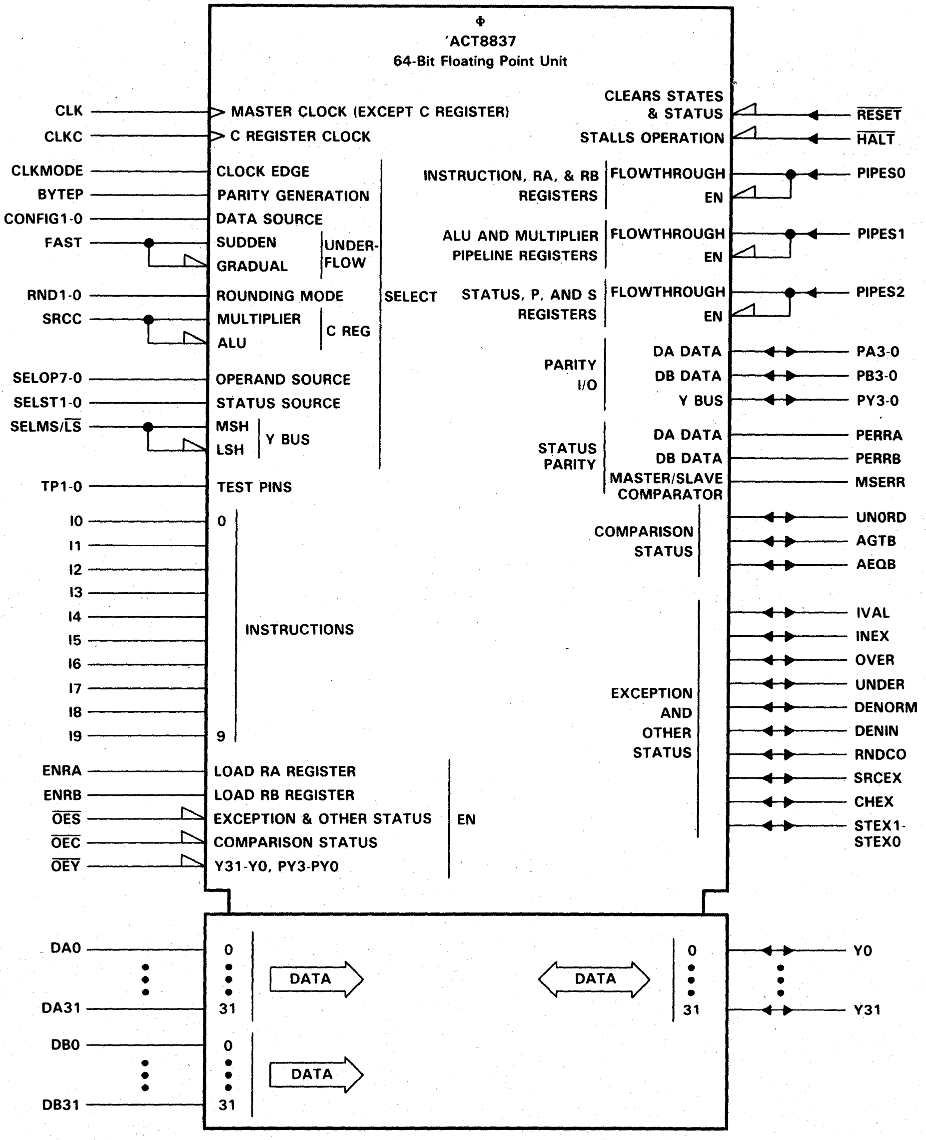
Dallas
Texas Instruments Incorporated
10001 E. Campbell Road
Richardson, TX 75081
214/680-5066

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.

'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)

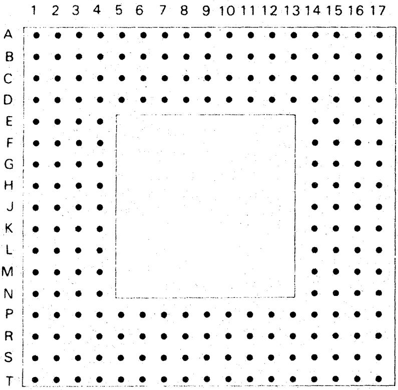


Table 1. 'ACT8837 Pin Grid Allocations

PIN		PIN		PIN		PIN		PIN		PIN	
NO.	NAME	NO.	NAME	NO.	NAME	NO.	NAME	NO.	NAME	NO.	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	$\overline{RESET}$	S3	DB0
A4	Y8	C5	Y9	E15	AGTB	K1	SELOP3	P4	PB1	S4	DB4
A5	Y11	C6	Y12	E16	AEQB	K2	SELOP4	P5	DB1	S5	DB11
A6	Y14	C7	Y15	E17	MSERR	K3	SELOP5	P6	DB5	S6	DB12
A7	Y17	C8	Y18	F1	I5	K4	GND	P7	DB9	S7	DB15
A8	Y20	C9	Y23	F2	I3	K14	GND	P8	DB16	S8	DB19
A9	Y21	C10	Y26	F3	RND0	K15	PA1	P9	DB21	S9	DB23
A10	Y24	C11	Y30	F4	GND	K16	PA2	P10	DB28	S10	DB26
A11	Y27	C12	PY1	F14	GND	K17	PA3	P11	DA0	S11	DB30
A12	Y29	C13	UNDER	F15	PERRA	L1	SELOP6	P12	DA4	S12	DA2
A13	PY0	C14	INEX	F16	$\overline{OEY}$	L2	SELOP7	P13	DA8	S13	DA6
A14	PY3	C15	DENIN	F17	$\overline{OES}$	L3	CLK	P14	DA12	S14	DA10
A15	IVAL	C16	SRCEX	G1	I7	L4	V_{CC}	P15	DA19	S15	DA14
A16	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	V_{CC}	L16	DA31	R1	PIPES1	T1	NC
B2	Y2	D3	Y1	G14	V_{CC}	L17	PA0	R2	$\overline{HALT}$	T2	PB3
B3	Y4	D4	GND	G15	$\overline{OEC}$	M1	ENRB	R3	PB2	T3	DB3
B4	Y7	D5	V_{CC}	G16	SELMS/$\overline{LS}$	M2	ENRA	R4	DB2	T4	DB7
B5	Y10	D6	GND	G17	TP1	M3	CLKC	R5	DB6	T5	DB8
B6	Y13	D7	GND	H1	I9	M4	GND	R6	DB10	T6	DB13
B7	Y16	D8	V_{CC}	H2	NC	M14	V_{CC}	R7	DB14	T7	DB17
B8	Y19	D9	GND	H3	I8	M15	DA27	R8	DB18	T8	DB20
B9	Y22	D10	GND	H4	GND	M16	DA28	R9	DB22	T9	DB24
B10	Y25	D11	V_{CC}	H14	GND	M17	DA29	R10	DB27	T10	DB25
B11	Y28	D12	GND	H15	TP0	N1	CONFIG0	R11	DB31	T11	DB29
B12	Y31	D13	GND	H16	SELST1	N2	CONFIG1	R12	DA3	T12	DA1
B13	PY2	D14	V_{CC}	H17	SELST0	N3	CLKMODE	R13	DA7	T13	DA5
B14	OVER	D15	STEX1	J1	SELOP2	N4	PIPES2	R14	DA11	T14	DA9
B15	RNDC0	D16	STEX0	J2	SELOP1	N14	DA18	R15	DA16	T15	DA13
B16	DENORM	D17	UNORD	J3	SELOP0	N15	DA24	R16	DA20	T16	NC
B17	NC	E1	I2	J4	V_{CC}	N16	DA25	R17	DA21	T17	NC
C1	PERRB	E2	I0	J14	V_{CC}	N17	DA26				

Table 2. 'ACT8837 Pin Functional Description

PIN		I/O	DESCRIPTION
NAME	NO.		
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.
CONFIG1	N2		
DA0	P11	I	DA 32-bit input data bus. Data can be latched in a 64-bit temporary register or loaded directly into an input register.
DA1	T12		
DA2	S12		
DA3	R12		
DA4	P12		
DA5	T13		
DA6	S13		
DA7	R13		
DA8	P13		
DA9	T14		
DA10	S14		
DA11	R14		
DA12	P14		
DA13	T15		
DA14	S15		
DA15	S16		
DA16	R15		
DA17	S17		
DA18	N14		
DA19	P15		
DA20	R16		
DA21	R17		
DA22	P16		
DA23	P17		

Table 2. 'ACT8837 Pin Functional Description (Continued)

PIN NAME	NO.	I/O	DESCRIPTION
DA24	N15		DA 32-bit input data bus. Data can be latched in a 64-bit temporary register or loaded directly into an input register
DA25	N16		
DA26	N17		
DA27	M15		
DA28	M16		
DA29	M17		
DA30	L15		
DA31	L16		
DB0	S3	I	DB 32-bit input data bus. Data can be latched in a 64-bit temporary register or loaded directly into an input register
DB1	P5		
DB2	R4		
DB3	T3		
DB4	S4		
DB5	P6		
DB6	R5		
DB7	T4		
DB8	T5		
DB9	P7		
DB10	R6		
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		I/O	DESCRIPTION
NAME	NO.		
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 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).
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		
$\overline{\text{HALT}}$	R2	I	Stalls operation without altering contents of instruction or data registers. Active low.
I0	E2	I	Instruction inputs
I1	D1		
I2	E1		
I3	F2		
I4	G3		
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 J15 P1 S1 T1 T16 T17		No internal connection. Pins should be left floating.
$\overline{\text{OEC}}$	G15	I	Comparison status output enable. Active low.
$\overline{\text{OES}}$	F17	I	Exception status and other status output enable. Active low.
$\overline{\text{OEY}}$	F16	I	Y bus output enable. Active low.
OVER	B14	I/O	Status pin indicating that the result is greater the largest allowable value for specified format (exponent overflow).
PA0 PA1 PA2 PA3	L17 K15 K16 K17	I	Parity inputs for DA data
PB0 PB1 PB2 PB3	S2 P4 R3 T2	I	Parity inputs for DB data
PERRA	F15	O	DA data parity error output. When high, signals a byte or word has failed an even parity check.
PERRB	C1	O	DB data parity error output. When high, signals a byte or word has failed an even parity check.
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	PIN NO.	I/O	DESCRIPTION
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.
PY0 PY1 PY2 PY3	A13 C12 B13 A14	I/O	Y port parity data
$\overline{\text{RESET}}$	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/$\overline{\text{LS}}$	G16	I	When low, selects LSH of 64-bit result to be output on the Y bus. When high, selects MSH of 64-bit result.
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		I/O	DESCRIPTION
NAME	NO.		
V_{CC}	D5		5-V power supply
V_{CC}	D8		
V_{CC}	D11		
V_{CC}	D14		
V_{CC}	G4		
V_{CC}	G14		
V_{CC}	J4		
V_{CC}	J14		
V_{CC}	L4		
V_{CC}	M14		
Y0	C2	I/O	32-bit Y output data bus
Y1	D3		
Y2	B2		
Y3	C3		
Y4	B3		
Y5	A3		
Y6	C4		
Y7	B4		
Y8	A4		
Y9	C5		
Y10	B5		
Y11	A5		
Y12	C6		
Y13	B6		
Y14	A6		
Y15	C7		
Y16	B7		
Y17	A7		
Y18	C8		
Y19	B8		
Y20	A8		
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

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/$\overline{\text{LS}}$ 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

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 CLK↑	18		ns
t_{su2}	Setup time, data operand before CLK↑	18		ns
t_{su3}	Setup time, data operand before second CLK↑ for double-precision operation (input register not enabled)	65		ns
t_{h1}	Hold time, Instruction input after CLK↑	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. PA0 and PB0 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).

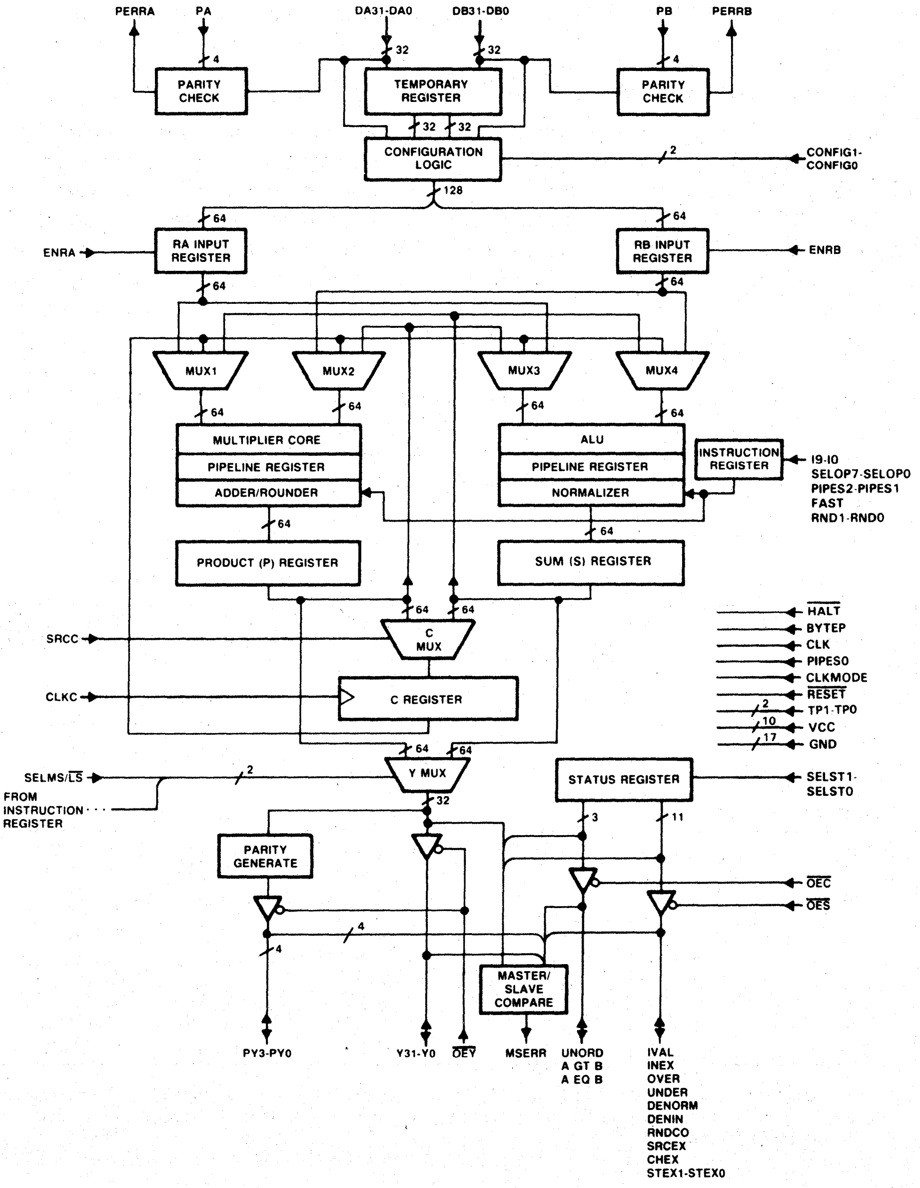


Figure 1. 'ACT8837 Floating Point Unit

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

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

[†] On the first active clock edge (see CLKMODE, 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

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

Table 5. Double-Precision Input Data Register Sources

		RA SOURCE		RB SOURCE	
CONFIG1	CONFIG0	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

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-SELOP0 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 = Multiplier 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 1 = |A|	0 = B 1 = |B|	0 = Y 1 = −Y	000	A * B
			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 $\overline{\text{OEY}}$, $\overline{\text{OES}}$ and $\overline{\text{OEC}}$ 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, $\overline{\text{OEC}}$ for comparison status and $\overline{\text{OES}}$ 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 MULTIPLIER 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 = Multiplier result	0 = Normal operation 1 = Forces B2 input of ALU to zero	0 = Normal operation 1 = Forces B1 input of multiplier 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.)
STEX0	A NaN or a denorm has been input on the B port.
STEX1	A NaN or a denorm has been input on the A port.
UNDER	The result is inexact and less than the minimum allowable value for the specified format. In the FAST mode, this condition causes the result to go to zero.

In chained mode, 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, CHEX 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-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 19).

Table 19. 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

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
$\overline{\text{HALT}}$	No effect	Stalls device operation but does not affect registers, internal states, or status
$\overline{\text{OEC}}$	Disables compare pins	Enables compare pins
$\overline{\text{OES}}$	Disables status outputs	Enables status outputs
$\overline{\text{OEY}}$	Disables Y bus	Enables Y bus
PIPES2-PIPES0	See Table 17 (pipeline mode control)	See Table 17 (pipeline mode control)
$\overline{\text{RESET}}$	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)
SELOP7-SELOP0	See Tables 6 and 7 (multiplier/ALU operand selection)	See Tables 6 and 7 (multiplier/ALU operand selection)
SELMS/$\overline{\text{LS}}$	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-TP0	See Table 19 (test pin control inputs)	See Table 19 (test pin control inputs)

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)

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.

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)

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 (infinity)	$(-1)^s$ (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 be 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 be 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/$\overline{\text{LS}}$, $\overline{\text{OEY}}$)

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/$\overline{\text{LS}}$ signal in the middle of the clock period.

When a single-precision result is output, the SELMS/$\overline{\text{LS}}$ signal has no effect. The SELMS/$\overline{\text{LS}}$ signal is set low only to read out the LSH of a double-precision result. Whenever this signal is selecting a valid result for output on the Y bus, the $\overline{\text{OEY}}$ enable must be pulled low at the beginning of that clock cycle.

Status Output Controls (SELST1-SELST0, $\overline{\text{OES}}$, $\overline{\text{OEC}}$)

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 ($\overline{\text{OES}}$, $\overline{\text{OEC}}$, or both).

Stalling the Device ($\overline{\text{HALT}}$)

Operation of the 'ACT8837 can be stalled nondestructively by means of the $\overline{\text{HALT}}$ signal. Pulling the $\overline{\text{HALT}}$ 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 $\overline{\text{HALT}}$ signal is set high. Using $\overline{\text{HALT}}$ 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 $\overline{\text{HALT}}$ input low may interrupt loading of the RA, RB, and instruction registers, as well as stalling operation. In clock mode 1, the temporary register loads on the falling edge of the clock, but the $\overline{\text{HALT}}$ signal going low would prevent the RA, RB, and instruction registers from loading on the next rising clock edge. It is therefore necessary to have the instruction and data inputs on the pins when the $\overline{\text{HALT}}$ signal is set high again and normal operation resumes.

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	x = Don't care I8 selects precision of A operand: 0 = A (SP) 1 = A (DP) I4 selects absolute value of A operand: 0 = A 1 = |A| During integer to floating point conversion, |A| is not allowed as a result.
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	
Undefined	0x 001x 0101	
Convert from floating point to floating point (adjusts precision of input: SP→DP, DP→SP)	0x 001x 0110	
Undefined	0x 001x 0111	
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	x = Don't Care
Subtract A − B	0x x000 0x01	I8 selects precision of A operand:
Subtract |A| − B	0x x001 0x01	0 = A (SP)
Subtract A − |B|	0x x000 1x01	1 = A (DP)
Subtract |A| − |B	0x x001 1x01	I7 selects precision of B operand:
Compare A, B	0x x000 0x10	0 = B (SP)
Compare |A|, B	0x x001 0x10	1 = B (DP)
Compare A, |B|	0x x000 1x10	I2 selects either Y or its absolute value:
Compare |A|, |B|	0x x001 1x10	0 = Y
Subtract B − A	0x x000 0x11	1 = |Y|
Subtract B − |A|	0x x001 0x11	
Subtract |B| − A	0x x000 1x11	
Subtract |B| − |A|	0x x001 1x11	

Table 25. Independent Multiplier Operations

MULTIPLIER OPRATION AND OPERANDS	INSTRUCTION INPUTS I9-I0	NOTES
Multiply A * B	0x x100 00xx	x = Don't Care I8 selects A operand precision (0 = SP, 1 = DP) I7 selects B operand precision (0 = SP, 1 = DP) I1 selects A operand format (0 = Normal, 1 = Wrapped) I0 selects B operand format (0 = Normal, 1 = Wrapped)
Multiply −(A * B)	0x x100 01xx	
Multiply A * |B|	0x x100 10xx	
Multiply −(A * |B|)	0x x100 11xx	
Multiply |A| * B	0x x101 00xx	
Multiply −(|A| * B)	0x x101 01xx	
Multiply |A| * |B|	0x x101 10xx	
Multiply −(|A| * |B|)	0x x101 11xx	

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	x = Don't Care I8 selects precision of RA inputs: 0 = RA (SP) 1 = RA (DP I7 selects precision of RB inputs: 0 = RB (SP) 1 = RB (DP) I3 negates ALU result: 0 = Normal 1 = Negated I2 negates multiplier result: 0 = Normal 1 = Negated
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	
A * B	B − A	ALU	1x x000 xx11	
A * B	B − A	Multiplier	1x x100 xx11	
A * B	A + 0	ALU	1x x010 xx00	
A * B	A + 0	Multiplier	1x x110 xx00	
A * B	0 − A	ALU	1x x010 xx11	
A * B	0 − A	Multiplier	1x x110 xx11	
A * 1	A + B	ALU	1x x001 xx00	
A * 1	A + B	Multiplier	1x x101 xx00	
A * 1	A − B	ALU	1x x001 xx01	
A * 1	A − B	Multiplier	1x x101 xx01	
A * 1	2 − A	ALU	1x x001 xx10	
A * 1	2 − A	Multiplier	1x x101 xx10	
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.

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

I 9-0	CLKMODE	CONFIG 1-0	PIPES 2-0	SELOP 7-0	RND 1-0	FAST	ENRA	ENRB	SRCC	SELMS/$\overline{\text{L}}$S	$\overline{\text{O}}$EY	$\overline{\text{O}}$EC	$\overline{\text{O}}$ES	BYTEP	SELST 1-0	$\overline{\text{R}}$ESET	$\overline{\text{H}}$ALT	TP 1-0
00 0010 1100	0	01	111	xxxx 11xx	00	0	1	1	0	1	0	0	0	x	11	1	1	11

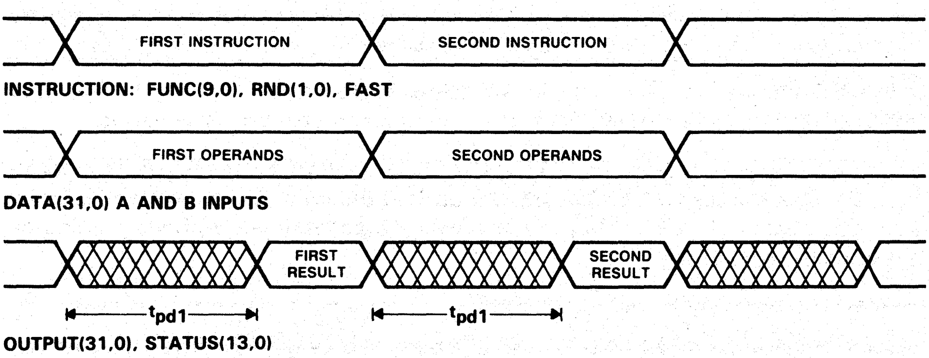


Figure 2. Single-Precision Operation, All Registers Disabled (PIPES = 111, CLKMODE = 0)

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|

I9-0	CLKMODE	CONFIG1-0	PIPES2-0	SELOP7-0	RND1-0	FAST	ENRA	ENRB	SRCC	SELMS/$\overline{\text{LS}}$	$\overline{\text{OEY}}$	$\overline{\text{OEC}}$	$\overline{\text{OES}}$	BYTEP	SELST1-0	$\overline{\text{RESET}}$	$\overline{\text{HALT}}$	TP1-0
00 0001 1010	0	01	110	xxxx 1111	00	0	1	1	0	1	0	0	0	x	11	1	1	11

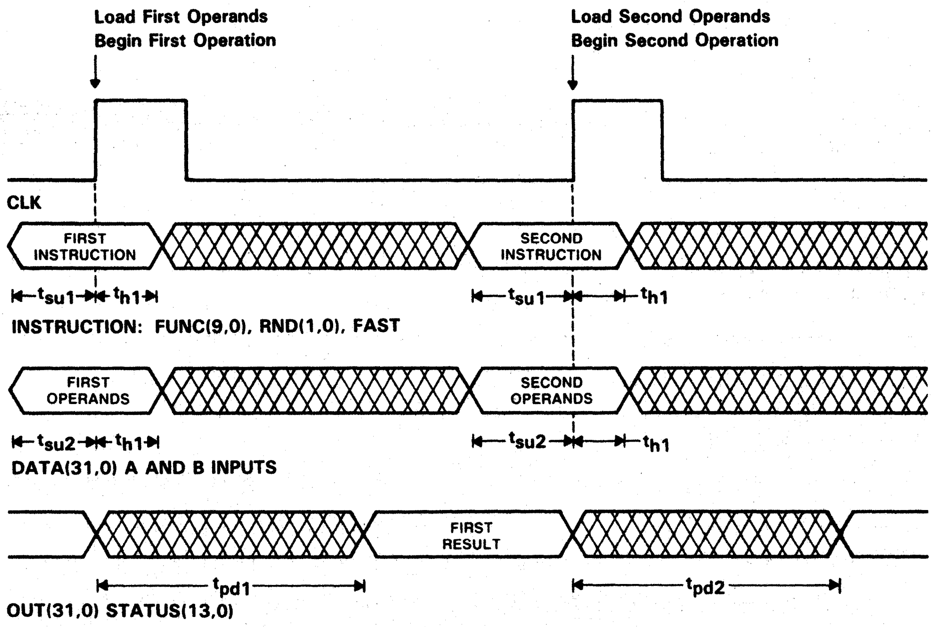


Figure 3. Single-Precision Operation, Input Registers Enabled (PIPES = 110, CLKMODE = 0)

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

I9–0	CLKMODE	CONFIG1–0	PIPES2–0	SELOP7–0	RND1–0	FAST	ENRA	ENRB	SRCC	SELMS/$\overline{\text{LS}}$	$\overline{\text{OEY}}$	$\overline{\text{OEC}}$	$\overline{\text{OES}}$	BYTEP	SELST1–0	$\overline{\text{R}}$ESET	$\overline{\text{H}}$ALT	TP1–0
00 0000 0011	0	01	010	xxxx 1111	00	0	0	0	0	1	0	0	0	x	11	1	1	11

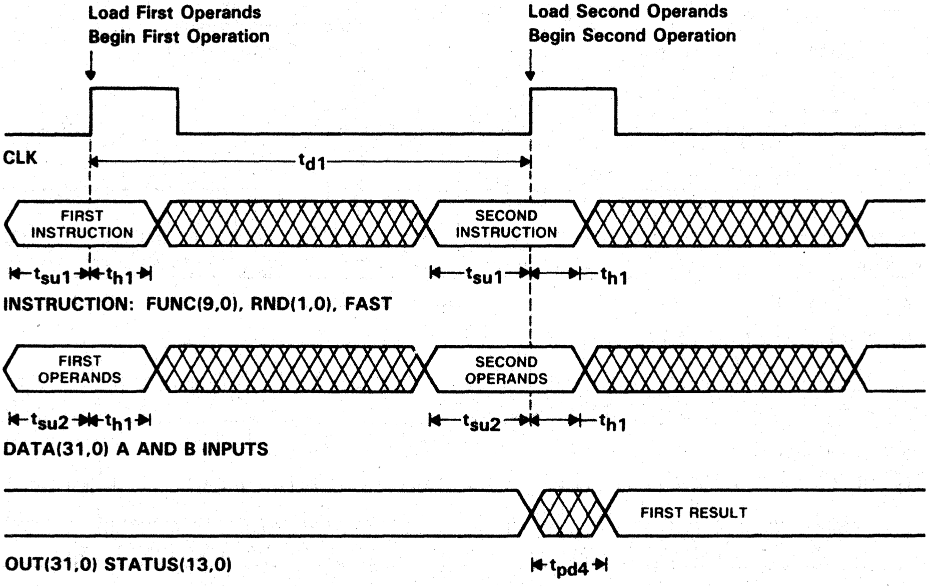


Figure 4. Single-Precision Operation, Input and Output Registers Enabled (PIPES = 010, CLKMODE = 0)

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

I 9–0	CLKMODE	CONFIG 1–0	PIPES 2–0	SELOP 7–0	RND 1–0	FAST	ENRA	ENRB	SRCC	SELMS/$\overline{LS}$	$\overline{OEY}$	$\overline{OEC}$	$\overline{OES}$	BYTEP	SELST 1–0	$\overline{RESET}$	$\overline{HALT}$	TP 1–0
00 0100 0000	0	01	000	1111 xxxx	00	0	1	1	1	1	0	0	0	x	11	1	1	11

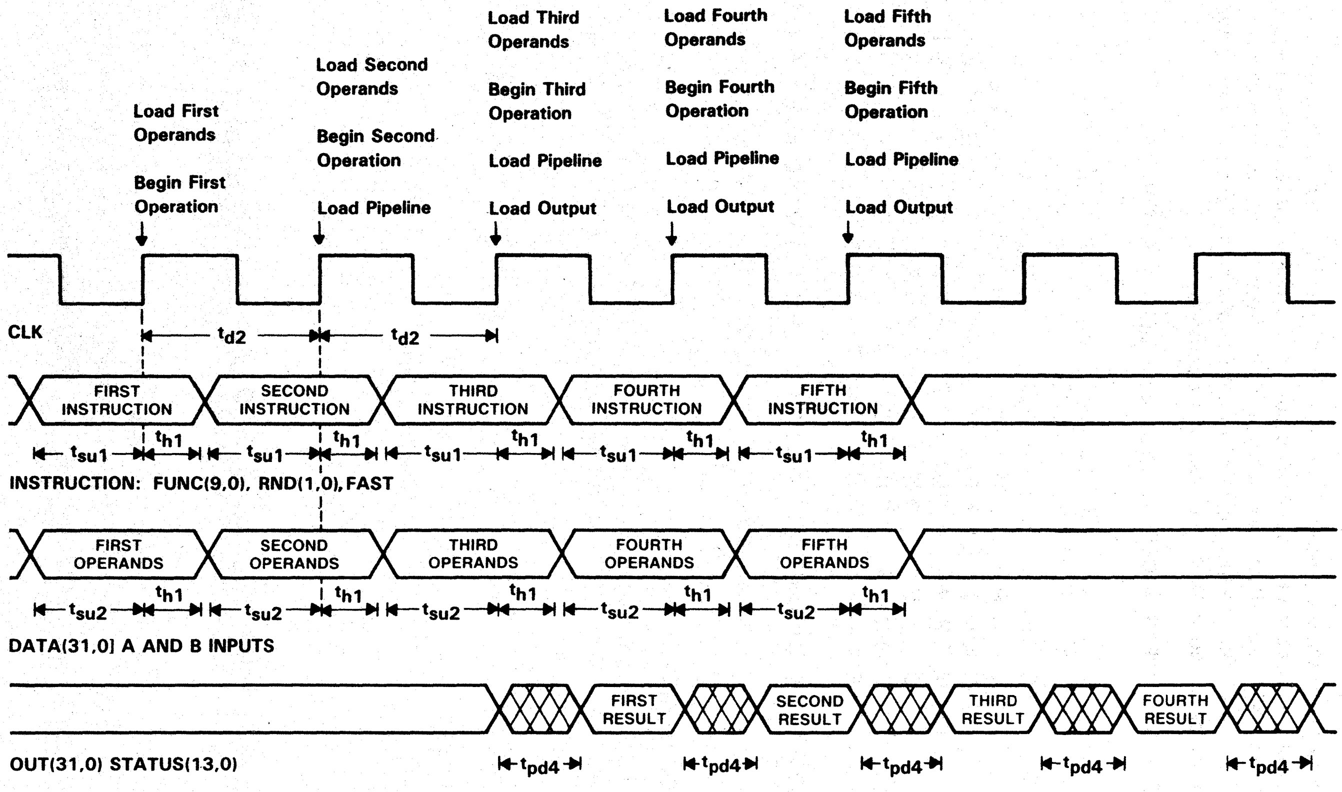


Figure 5. Single-Precision Operation, All Registers Enabled (PIPES = 000, CLKMODE = 0)

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).

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|

I9–0	CLKMODE	CONFIG1–0	PIPES2–0	SELOP7–0	RND1–0	FAST	ENA	ENB	SRCC	SELMS/$\overline{LS}$	$\overline{OEY}$	$\overline{OEC}$	$\overline{OES}$	BYTEP	SELST1–0	$\overline{RESET}$	$\overline{HALT}$	TP1–0
01 1000 1000	0	11	111	xxxx 1111	00	0	1	1	0	x	0	0	0	x	11	1	1	11

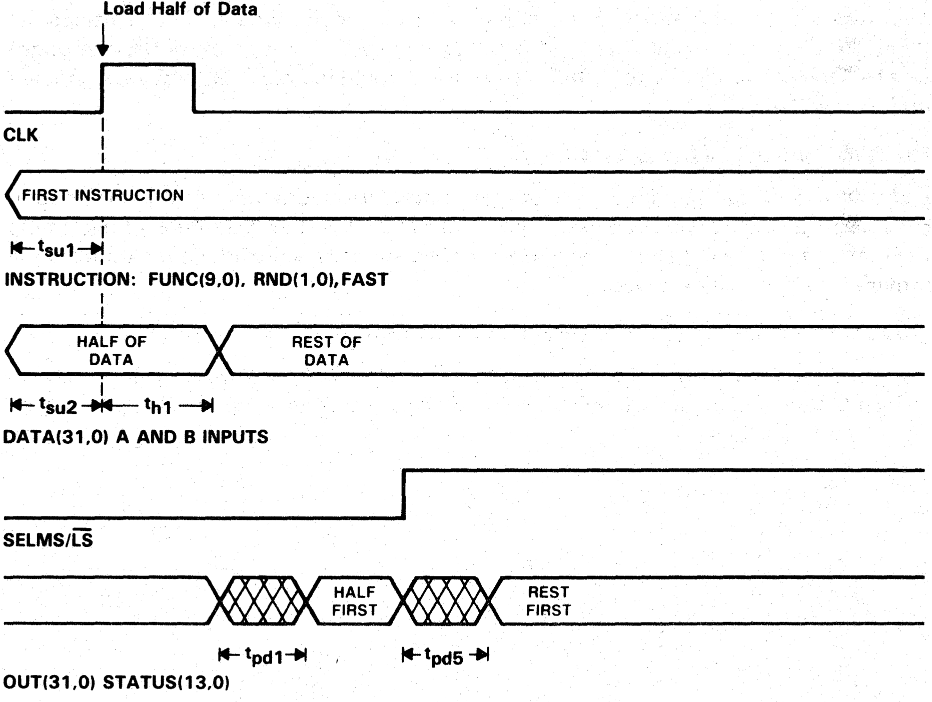


Figure 6. Double-Precision ALU Operation, All Registers Disabled (PIPES = 111, CLKMODE = 0)

In the second example the input register is enabled (PIPES2-PIPES0 = 110). Operands A and B for the instruction, |B| − |A|, are loaded using 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).

CLKMODE = 0 PIPES = 110 Operation: |B| − |A|

```
                                         S
                                         E
          C C C                          L
          L O O P P   S S                M         S S
          K N N I I   E E                S       B E E R̅
          M F F P P   L L   R R  F E E S /       Y L L E H̅
          O I I E E   O O   N N  A N N R   O̅ O̅ O̅ T S S S A T T
   I I    D G G S S   P P   D D  S R R C L̅ 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
```

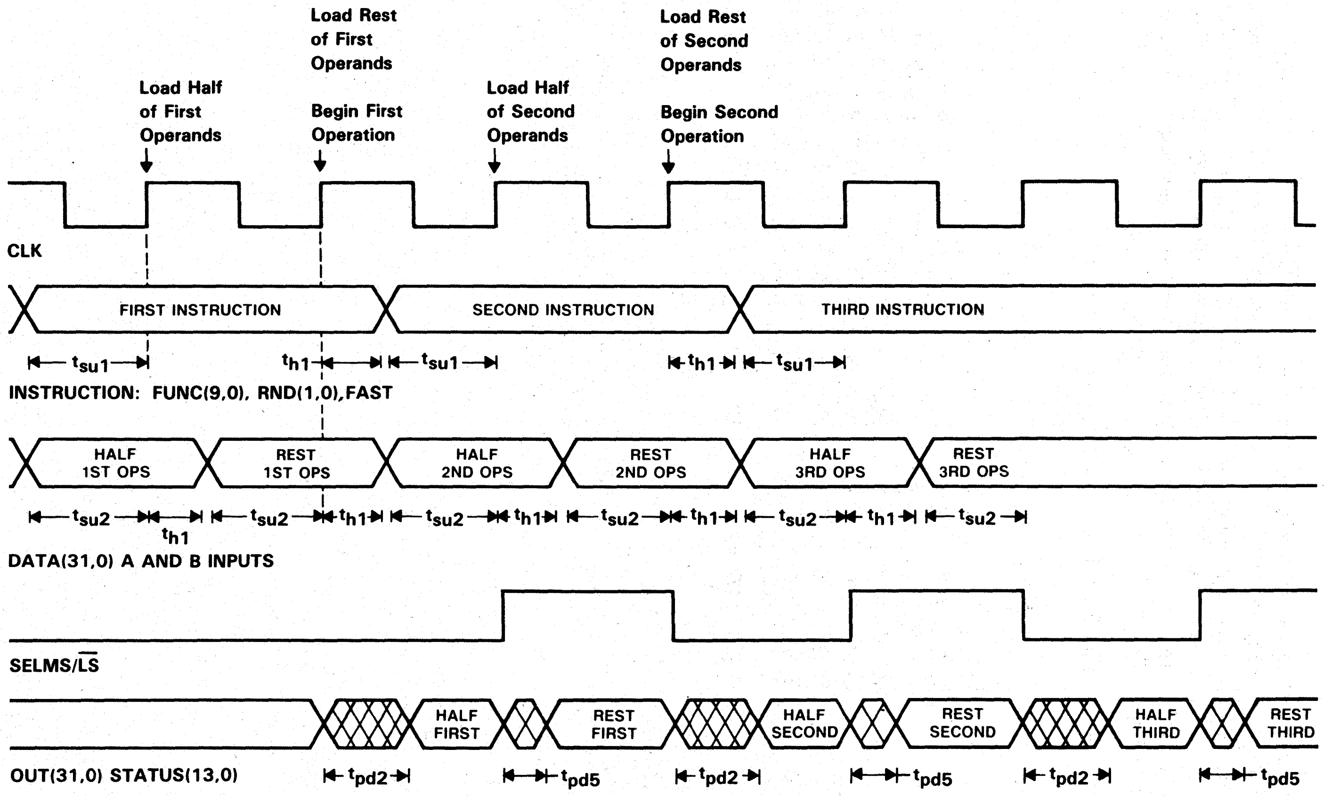


Figure 7. Double-Precision ALU Operation, Input Registers Enabled (PIPES = 110, CLKMODE = 0)

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

II 9–0	CLKMODE	CONFIG 1–0	PIPES 2–0	SELOP 7–0	RND 1–0	FAST	ENRA	ENRB	SRCC	SELMS/$\overline{LS}$	$\overline{OEY}$	$\overline{OEC}$	$\overline{OES}$	BYTEP	SELST 1–0	$\overline{RESET}$	$\overline{HALT}$	TP 1–0
01 1010 1000	0	01	010	xxxx 11xx	00	0	1	1	0	x	0	0	0	x	11	1	1	11

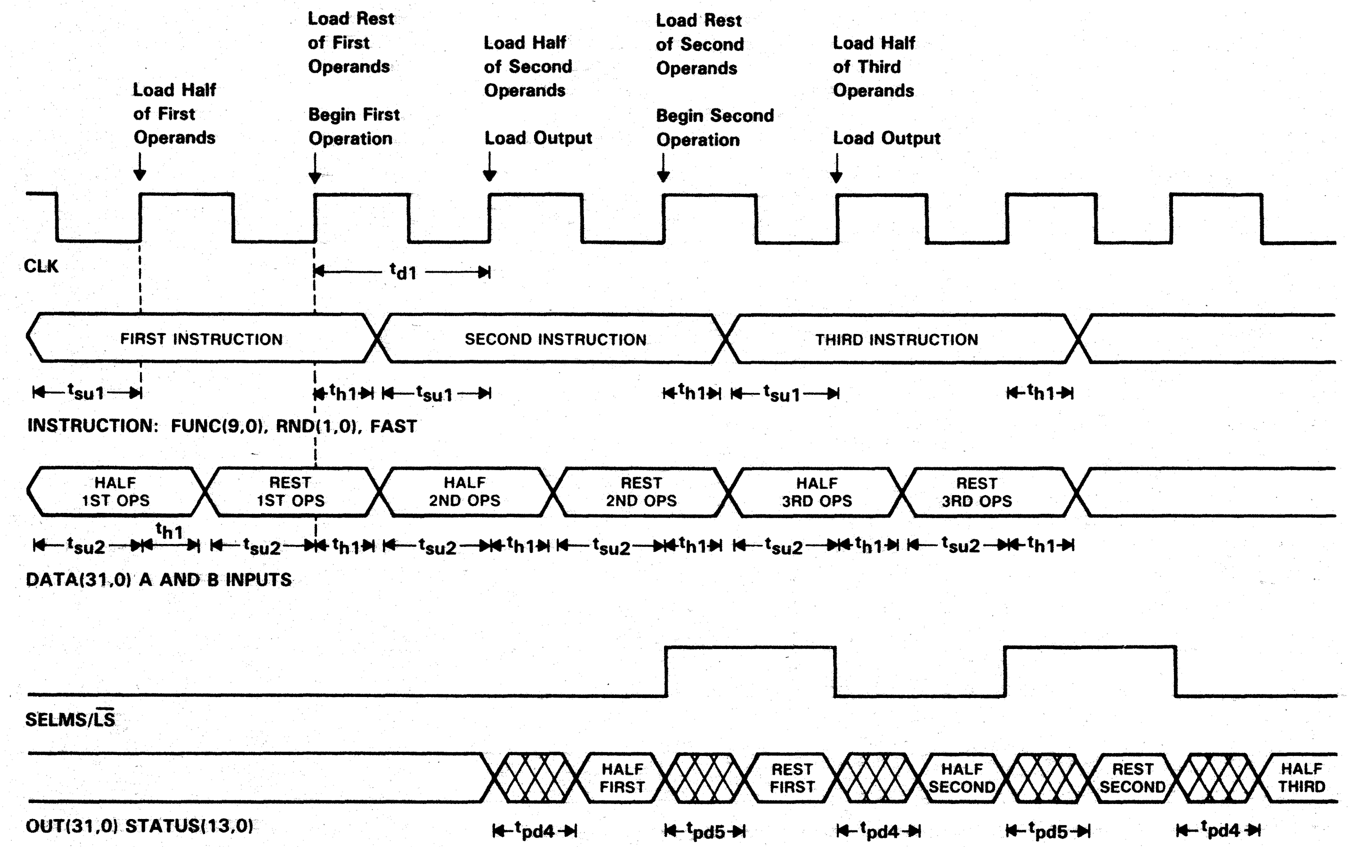


Figure 8. Double-Precision ALU Operation, Input and Output Registers Enabled (PIPES = 010, CLKMODE = 0)

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

I9-0	CLKMODE	CONFIG1-0	PIPES2-0	SELOP7-0	RND1-0	FAST	ENRA	ENRB	SRCC	SELMS/$\overline{L}$S	$\overline{OEY}$	$\overline{OEC}$	$\overline{OES}$	BYTEP	SELST1-0	$\overline{R}$ESET	$\overline{H}$ALT	TP1-0
01 1010 0010	0	11	000	xxxx 1100	00	0	1	1	0	x	0	0	0	x	11	1	1	11

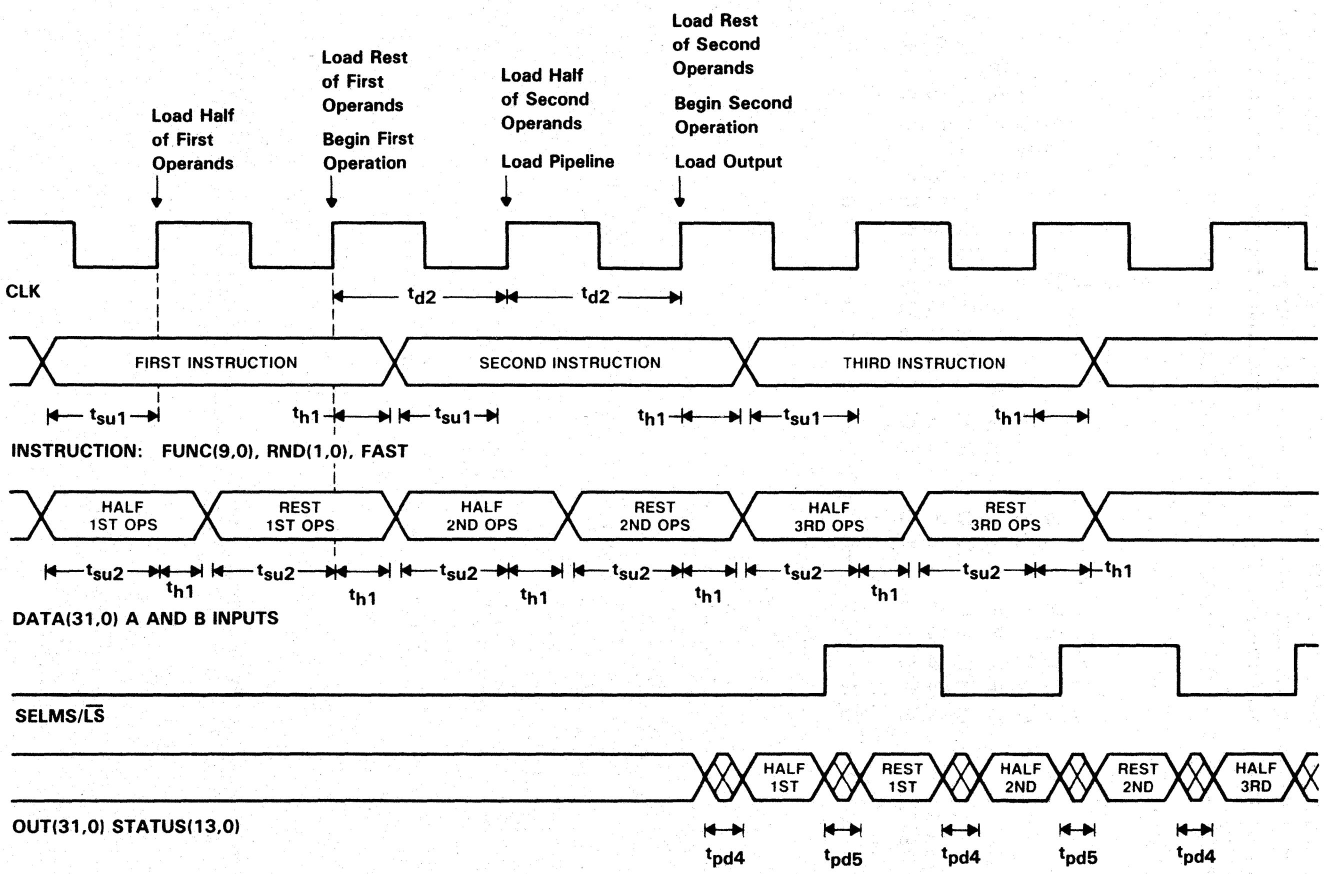


Figure 9. Double-Precision ALU Operation, All Registers Enabled (PIPES = 000, CLKMODE = 0)

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|

I9–I0	CLKMODE	CONFIG1–0	PIPES2–0	SELOP7–0	RND1–0	FTSA	ENRA	ENRB	SRCC	SELMS/$\overline{\text{LS}}$	$\overline{\text{OEY}}$	$\overline{\text{OEC}}$	$\overline{\text{OES}}$	BYTEP	SELST1–0	$\overline{\text{R}}$ESET	$\overline{\text{H}}$ALT	TP1–0
01 1000 1000	1	11	111	xxxx 1111	00	0	1	1	0	x	0	0	0	x	xx	1	1	11

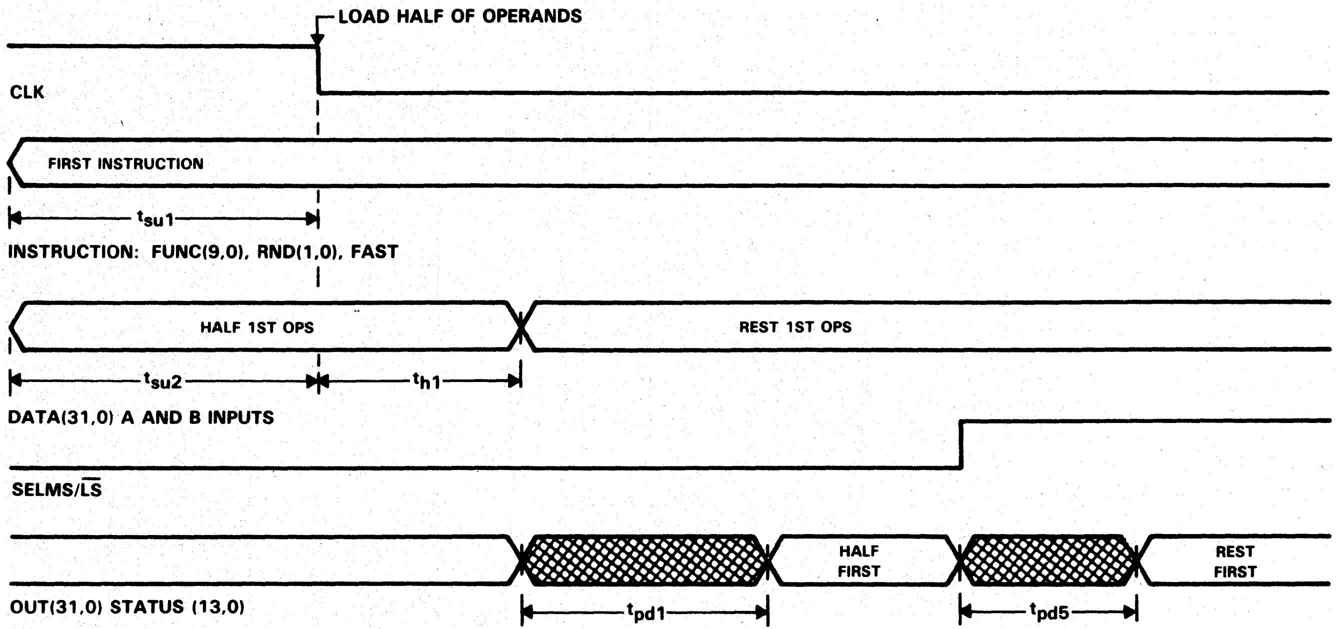


Figure 10. Double-Precision ALU Operation, All Registers Disabled (PIPES = 111, CLKMODE = 1)

The second example executes subtraction of absolute values for both operands. Only the RA and RB registers are enabled (PIPES2-PIPES0 = 110). Timing is shown in Figure 11.

CLKMODE = 1 PIPES = 110 Operation: Subtract |B| − |A|

I9–0	CLKMODE	CONFIG1–0	PIPES2–0	SELOP7–0	RND1–0	FAST	ENRA	ENRB	SRCC	SELMS/$\overline{LS}$	$\overline{OEY}$	$\overline{OEC}$	$\overline{OES}$	BYTEP	SELST1–0	$\overline{RESET}$	$\overline{HALT}$	TP1–0
01 1001 1011	1	11	110	xxxx 1111	00	0	1	1	0	x	0	0	0	x	xx	1	1	11

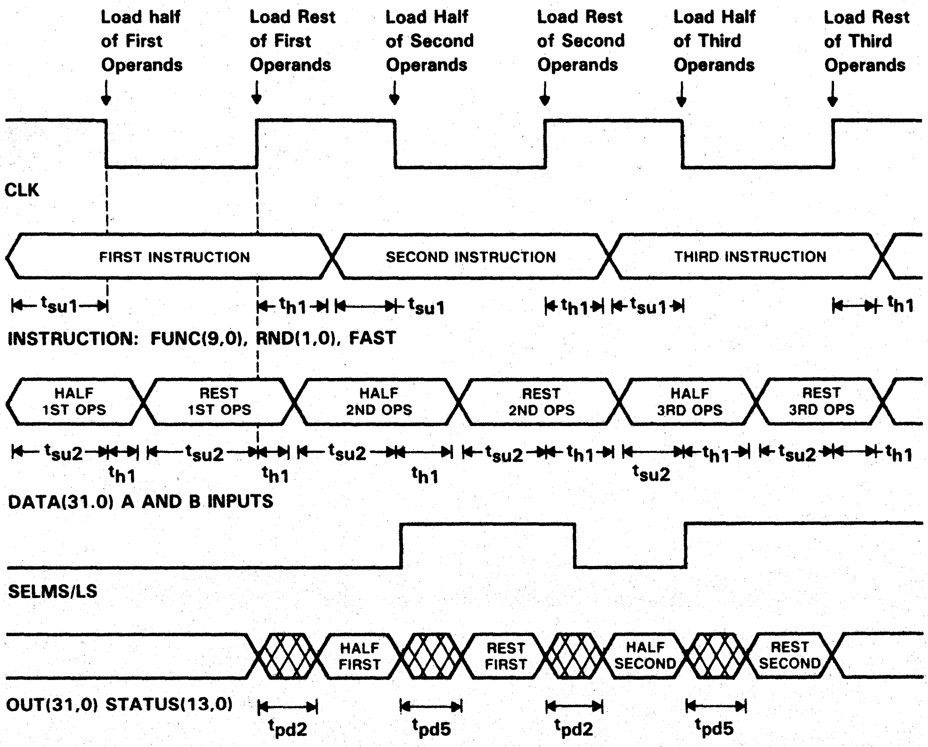


Figure 11. Double-Precision ALU Operation, Input Registers Enabled (PIPES = 110, CLKMODE = 1)

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

I9–0	CLKMODE	CONFIG1–0	PIPES2–0	SELOP7–0	RND1–0	FAST	ENRA	ENRB	SRCC	SELMS/$\overline{\text{LS}}$	$\overline{\text{OE}}$Y	$\overline{\text{OE}}$C	$\overline{\text{OE}}$S	BYTEP	SELST1–0	$\overline{\text{R}}$ESET	$\overline{\text{H}}$ALT	TP1–0
01 1010 1000	1	11	010	xxxx 11xx	00	0	1	0	0	x	0	0	0	x	xx	1	1	11

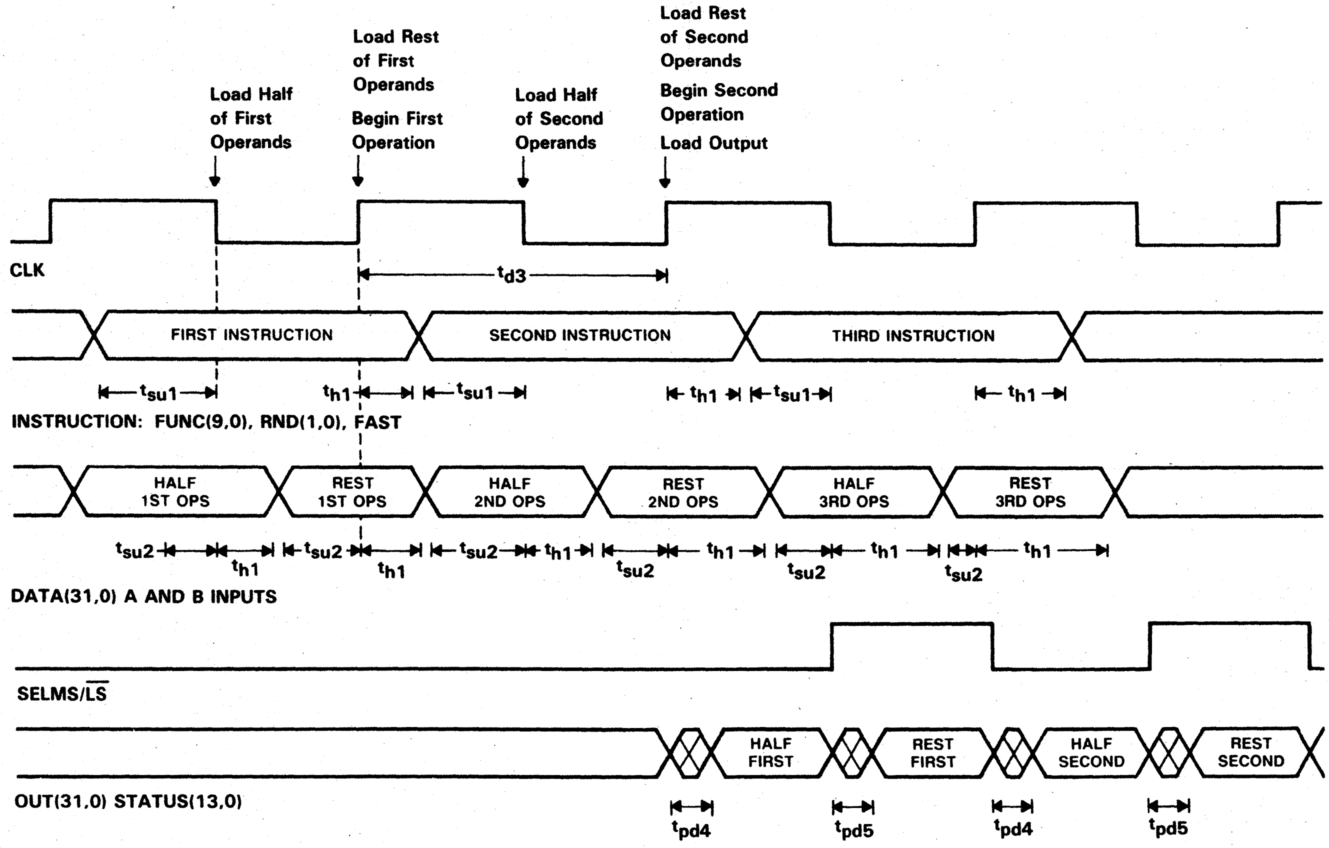


Figure 12. Double-Precision ALU Operation, Input and Output Registers Enabled (PIPES = 010, CLKMODE = 1)

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

I 9–0	CLKMODE	CONFIG 1–0	PIPES 2–0	SELOP 7–0	RND 1–0	FAST	ENRA	ENRB	SRCC	SELMS/$\overline{\text{L}}$S	$\overline{\text{OE}}$Y	$\overline{\text{OE}}$C	$\overline{\text{OE}}$S	BYTEP	SELST 1–0	$\overline{\text{R}}$ESET	$\overline{\text{H}}$ALT	TP 1–0
01 1010 0010	0	11	000	xxxx 1100	00	0	1	1	x	x	0	0	0	x	xx	1	1	11

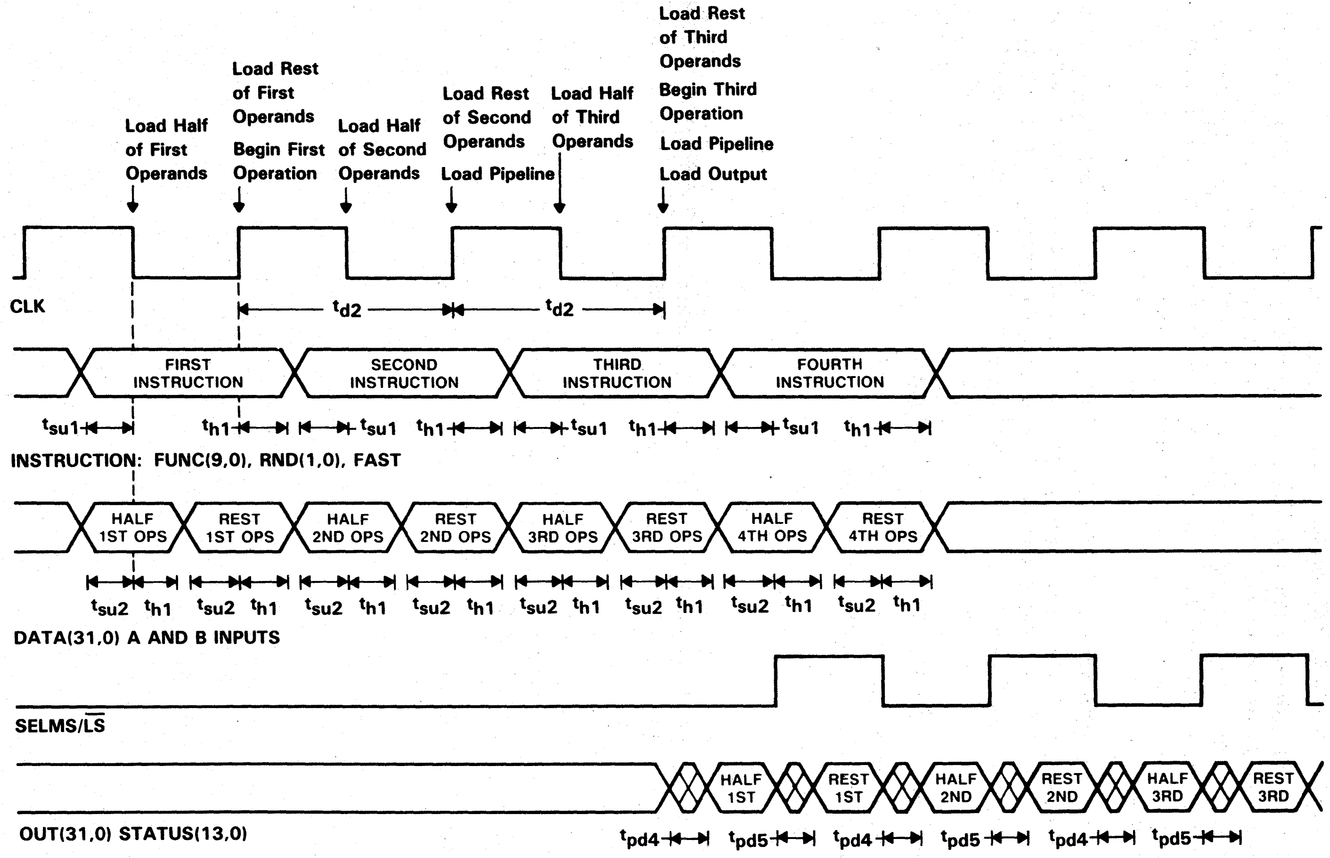


Figure 13. Double-Precision ALU Operation, All Registers Enabled (PIPES = 000, CLKMODE = 1)

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|

II 9-0	CLKMODE	CONFIG 1-0	PIPES 2-0	SELOP 7-0	RND 1-0	FAST	ENRA	ENRB	SRCC	SELMS/$\overline{\text{L}}$S	$\overline{\text{O}}$EY	$\overline{\text{O}}$EC	$\overline{\text{O}}$ES	BYTEP	SELST 1-0	$\overline{\text{R}}$ESET	$\overline{\text{H}}$ALT	TP 1-0
01 1100 1000	0	11	111	1111 xxxx	00	0	x	x	x	x	0	0	0	x	xx	1	1	11

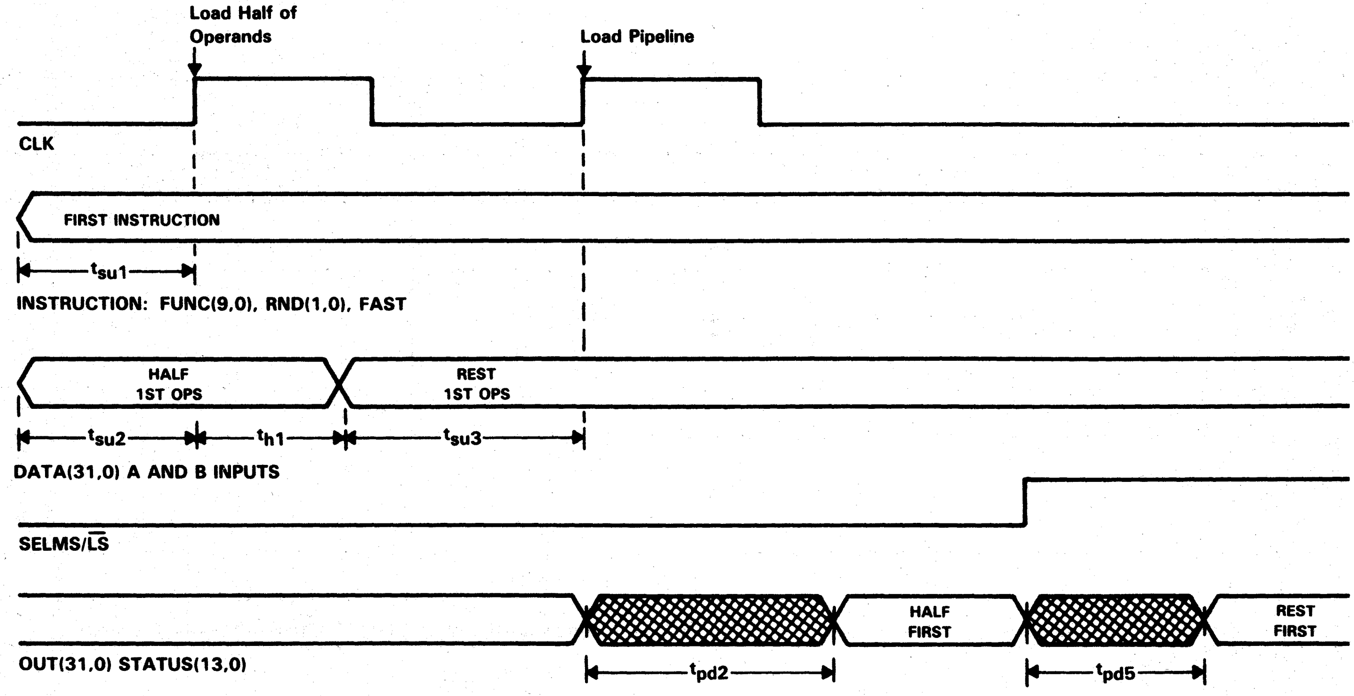


Figure 14. Double-Precision Multiplier Operation, All Registers Disabled (PIPES = 111, CLKMODE = 0)

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 $-(|A| * B)$

I9-0	CLKMODE	CONFIG1-0	PIPES2-0	SELOP7-0	RND1-0	FAST	ENRA	ENRB	SRCC	SELMS/$\overline{LS}$	$\overline{OEY}$	$\overline{OEC}$	$\overline{OES}$	BYTEP	SELST1-0	$\overline{RESET}$	$\overline{HALT}$	TP1-0
01 1101 0100	0	11	110	1111 xxxx	00	0	1	1	x	x	0	0	0	x	xx	1	1	11

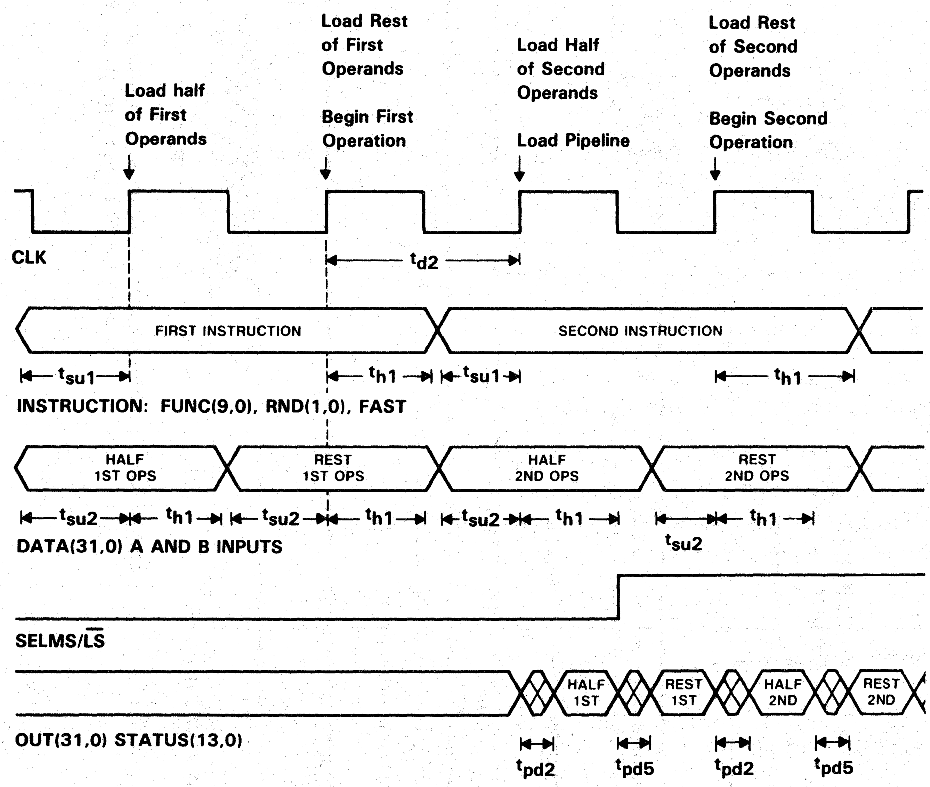


Figure 15. Double-Precision Multiplier Operation, Input Registers Enabled (PIPES = 110, CLKMODE = 0)

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|

I9–0	CLKMODE	CONFIG1–0	PIPES2–0	SELOP7–0	RND1–0	FAST	ENRA	ENRB	SRCC	SELMS/$\overline{LS}$	$\overline{OEY}$	$\overline{OEC}$	$\overline{OES}$	BYTEP	SELST1–0	$\overline{RESET}$	$\overline{HALT}$	TP1–0
01 1101 1000	0	10	010	1111 xxxx	00	0	1	1	x	x	0	0	0	x	xx	1	1	11

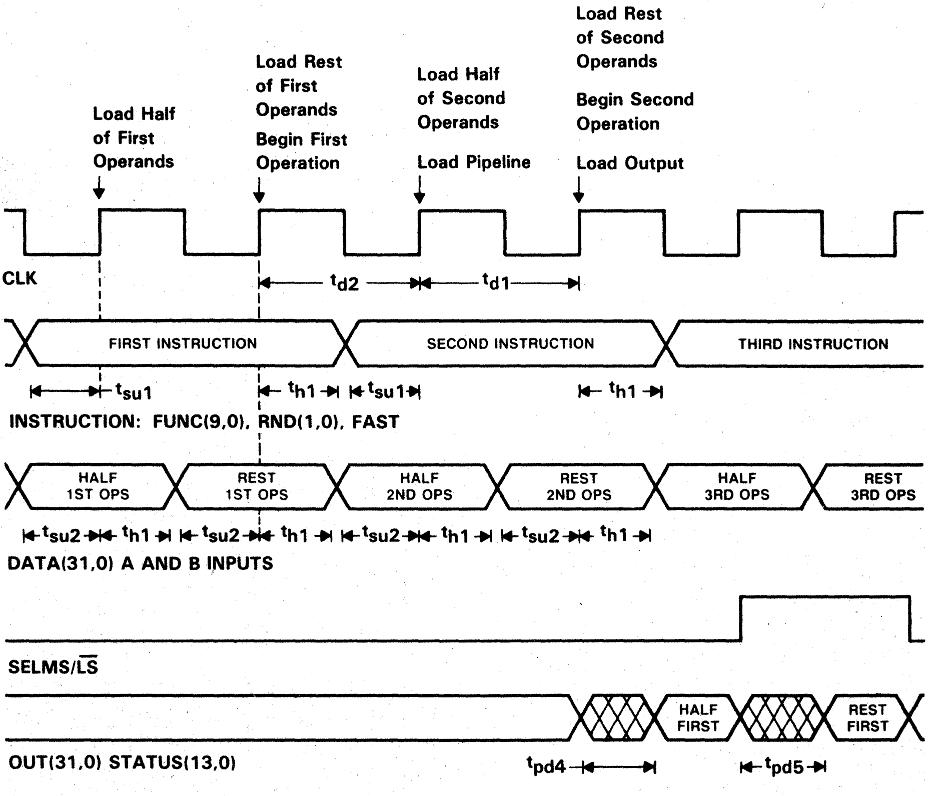


Figure 16. Double-Precision Multiplier Operation, Input and Output Registers Enabled (PIPES = 010, CLKMODE = 0)

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)

I9-0	CLKMODE	CONFIG1-0	PIPES2-0	SELOP7-0	RND1-0	FAST	ENRA	ENRB	SRCC	SELMS/$\overline{\text{LS}}$	$\overline{\text{OEY}}$	$\overline{\text{OEC}}$	$\overline{\text{OES}}$	BYTEP	SELST1-0	$\overline{\text{RESET}}$	$\overline{\text{HALT}}$	TP1-0
01 1100 0100	0	01	000	1111 xxxx	00	0	1	1	x	x	0	0	0	x	xx	1	1	11

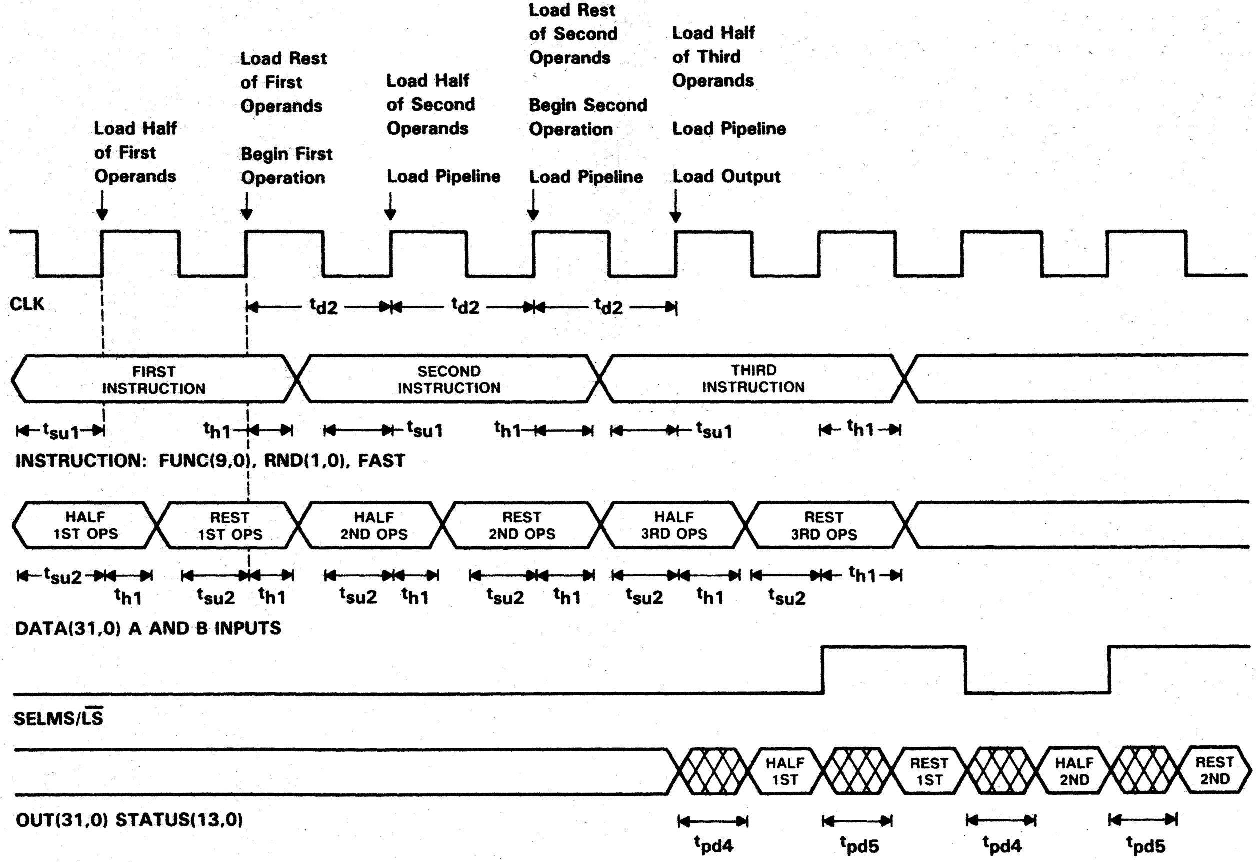


Figure 17. Double-Precision Multiplier Operation, All Registers Enabled (PIPES = 000, CLKMODE = 0)

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|

```
                                           S
                                           E
              C C C                        L
              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 _ 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 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 1100 1000  1  11  111  1111 xxxx 00   0 x x x x 0 0 0 x  xx  1 1 11
```

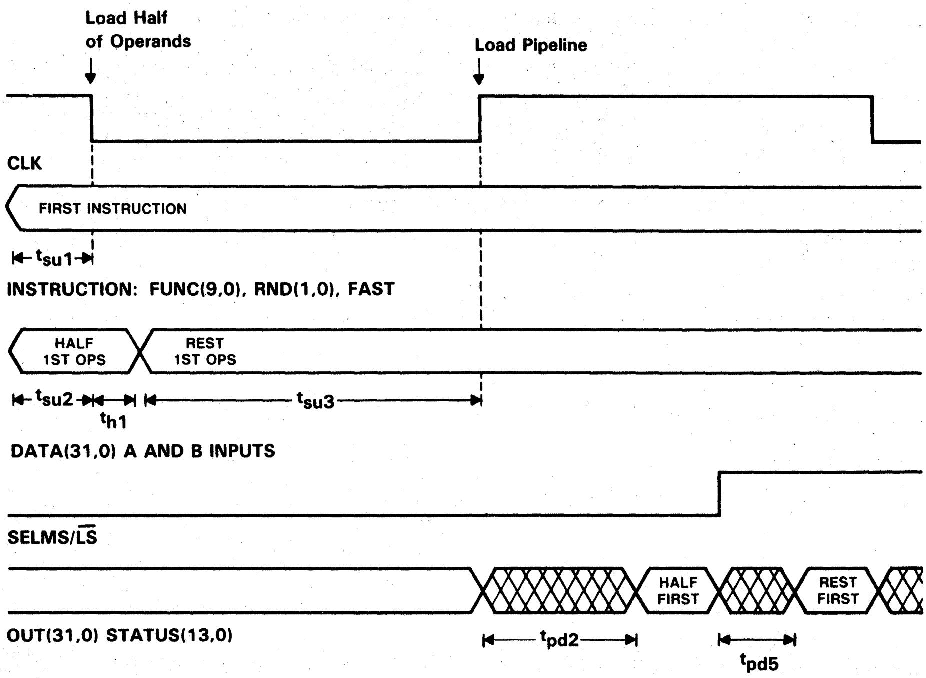


Figure 18. Double-Precision Multiplier Operation, All Registers Disabled (PIPES = 111, CLKMODE = 1)

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)

```
                                             S
                                             E
            C C C                            L
            L O O P P   S S                  M         S S
            K N N I I   E E                  S       B E E R̄
            M F F P P   L L    R R  F E E S  /       Y L L E H̄
            O I I E E   O O    N N  A N N R   Ō Ō Ō  T S S S A T T
      I I   D G G S S   P P    D D  S R R C  L̄ 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 0100  1  11  110  1111 xxxx  00  0 1 1 x  x 0 0 0  x  xx  1 1 11
```

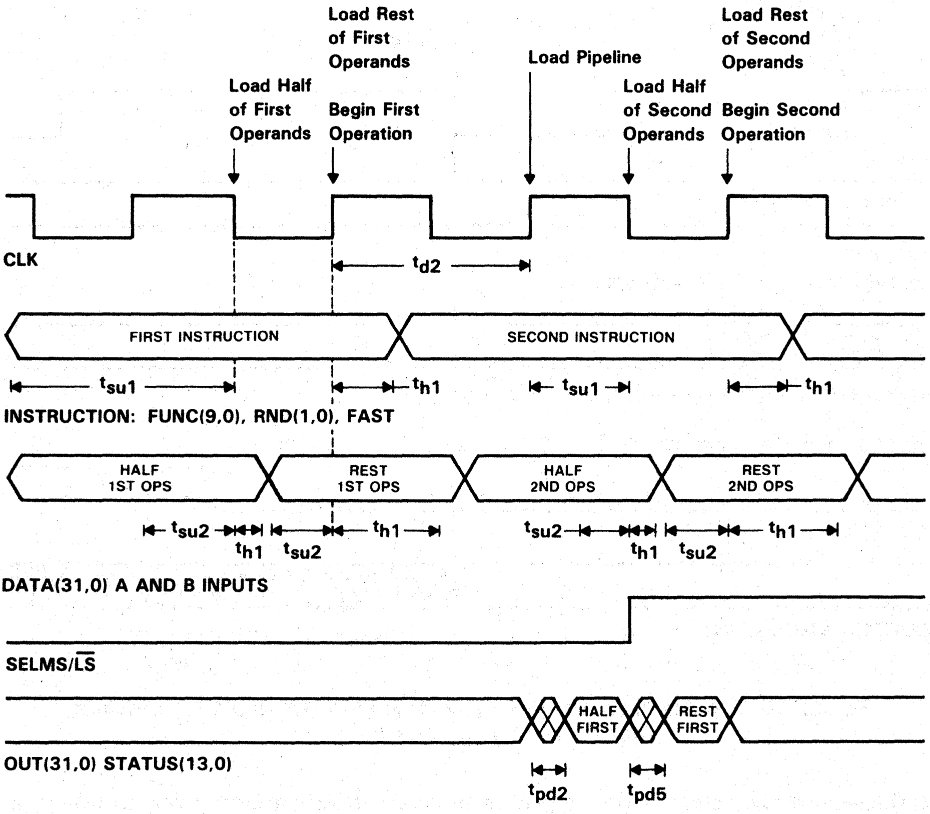


Figure 19. Double-Precision Multiplier Operation, Input Registers Enabled (PIPES = 110, CLKMODE = 1)

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|

I 9–0	CLKMODE	CONFIG 1–0	PIPES 2–0	SELOP 7–0	RND 1–0	FAST	ENRA	ENRB	SRCC	SELMS/$\overline{\text{L}}$S	$\overline{\text{O}}$EY	$\overline{\text{O}}$EC	$\overline{\text{O}}$ES	BYTEP	SELST 1–0	SELST	$\overline{\text{R}}$ESET	$\overline{\text{H}}$ALT	TP 1–0
01 1101 1000	1	11	010	1111 xxxx	00	0	1	1	x	x	0	0	0	x	xx		1	1	11

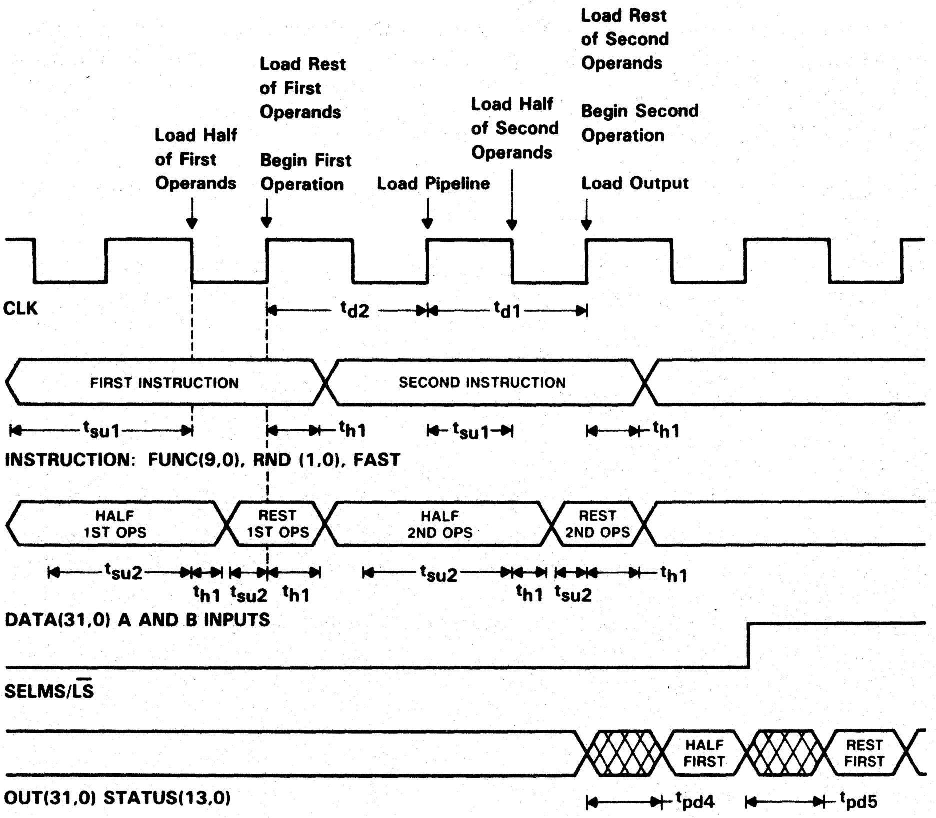


Figure 20. Double-Precision Multiplier Operation, Input and Output Registers Enabled (PIPES = 010, CLKMODE = 1)

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 section on mixed operations and operands.

CLKMODE = 1 PIPES = 000 Operation: Multiply −(A ∗ B)

II 9–0	CLKMODE	CONFIG 1–0	PIPES 2–0	SELOP 7–0	RND 1–0	FAST	ENRA	ENRB	SRCC	SELMS/$\overline{\text{LS}}$	$\overline{\text{OEY}}$	$\overline{\text{OEC}}$	$\overline{\text{OES}}$	BYTEP	SELST 1–0	$\overline{\text{RESET}}$	$\overline{\text{HALT}}$	TP 1–0
01 1100 0100	1	11	000	1111 xxxx	00	0	1	1	x	x	0	0	0	x	xx	1	1	11

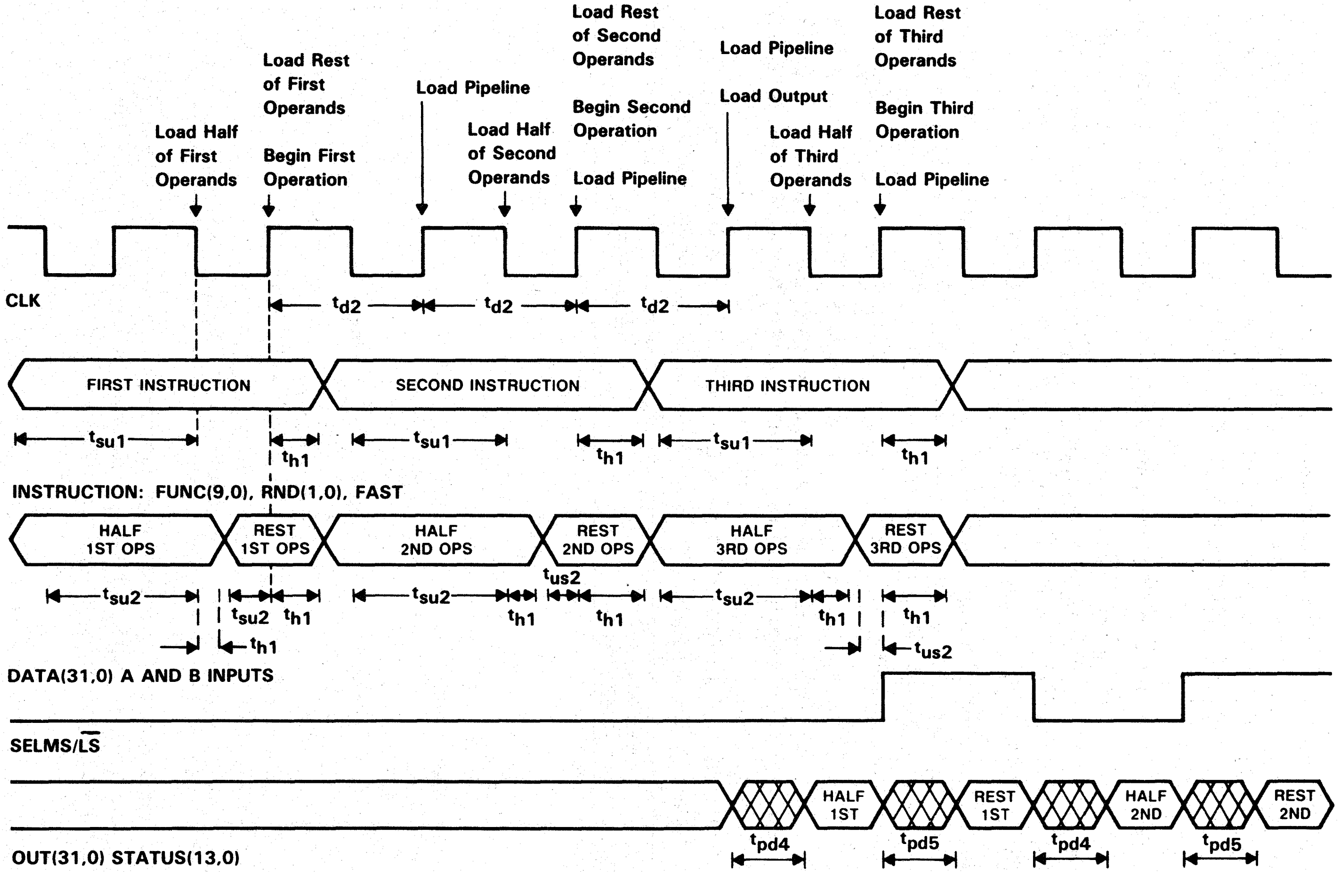


Figure 21. Double-Precision Multiplier Operation, All Registers Enabled (PIPES = 000, CLKMODE = 1)

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

I 9-0	CLKMODE	CONFIG 1-0	PIPES 2-0	SELOP 7-0	RND 1-0	FAST	ENRA	ENRB	SRCC	SELMS/$\overline{LS}$	$\overline{OEY}$	$\overline{OEC}$	$\overline{OES}$	BYTEP	SELST 1-0	$\overline{RESET}$	$\overline{HALT}$	TP 1-0
00 0100 0000	0	01	010	1111 xxxx	00	0	1	1	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	xx	1	1	11
10 0000 0000	0	01	010	1111 1010	00	0	1	1	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	xx	1	1	11
00 0000 0000	0	01	010	xxxx 1010	00	0	x	x	x	x	x	x	x	x	xx	1	1	11
xx xxxx xxxx	x	xx	xxx	xxxx xxxx	xx	x	x	x	x	x	0	0	0	x	xx	1	1	11

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.

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.

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↔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 MSH	B1 MSH	A1,B1MSH	A1 * B1									
↑ 2	A1 LSH	B1 LSH	A1,B1MSH	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,B2MSH	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,B3MSH	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,B4MSH	PR + SR A4 * B4	PR + SR, A4 * B4	A4	B4	A3 * B3	P2	P1	P1 + P1		
↑ 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,B5MSH	PR + SR A5 * B5	PR + SR, A5 * B5	A5	B5	A4 * B4	P3	P3	S1 + P3	S1	
↑ 11	A6 MSH	B6 MSH	A6,B6(M)	PR + SR A6 * B6	PR + SR, A5 * B5	A5	B5	A5 * B5	P4	P3	XXXXX	S2	
↑ 12													

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.

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.

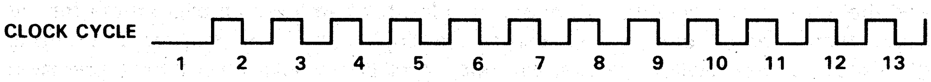
Calculations involving mixed-precision operands must be performed as double-precision operations (see Table 12). The instruction settings (I8-I7) should be set to indicate the precision of each operand from the RA and RB input registers. (Feedback operands from internal registers are also double-precision.) Mixed-precision operations should not be performed in chained mode.

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:



FUNCTION AND DATA	A,B	A,B	C,D	C,D		E,F	G,H	G,H	I,J	I,J	K,L	M,N	
RESULTS AND STATUS			XXXX	A,B	XXXX	C,D	E,F	E,F	G,H	G,H	I,J	K,L	M,N

A,B,C,D — double precision multiply; E,F — single precision operation; G,H,I,J — double precision add; K,L — single precision opration. 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:

CLOCK CYCLE	1	2	3	4	5	6	7	8	9	10	11	12	13
FUNCTION AND DATA	A,B		C,D		E,F	G,H	I,J	K,L	M,N	O,P		Q,R	
RESULTS AND STATUS					A,B	A,B	C,D	E,F	G,H	I,J	K,L	M,N	M,N

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.

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:

$$\frac{dx1}{dt} = x\ 2, \ldots, \quad \frac{dx(n-1)}{dt} = xn$$

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

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

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

Expanding the matrix equation for one state variable, dx1/dt, results in the following expression:

$$\dot{X}1 = (a11 * x1 + ... + a1n * xn) + (b11 * u1 + ... + b1n * un)$$

where $\dot{X}1 = dx1/dt$.

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 × n matrix A multiplied by an n × n matrix X yields an n × n matrix C whose elements cij are given by this equation:

$$cij = \sum_{k=1}^{n} aik * xkj \quad \text{for } i=1, ... ,n \quad j=1, ... ,n \qquad (1)$$

For the cij elements to be calculated by the 'ACT8837, the corresponding elements aik and xkj 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:

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

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

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

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

The rotation of an object about an arbitrary axis in a three-dimensional space can be carried out by first translating the object such that the desired axis of rotation passes through the origin of the coordinate system, then rotating the object about the axis through the origin, and finally translating the rotated object such that the axis of rotation resumes its initial position. If the axis of rotation passes through the point P = [a b c 1], then the transformation matrix is representable in this form:

$$[x\ y\ z\ h] = [x\ y\ z\ 1] \underset{\substack{\downarrow \\ \text{translation} \\ \text{to origin}}}{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -a & -b & -c & 1 \end{bmatrix}} \underset{\substack{\downarrow \\ \text{rotation} \\ \text{about} \\ \text{origin}}}{\begin{bmatrix} R \end{bmatrix}} \underset{\substack{\downarrow \\ \text{translation} \\ \text{back to initial} \\ \text{position}}}{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ a & b & c & 1 \end{bmatrix}} \qquad (2)$$

where R may be expressed as:

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

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

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

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

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

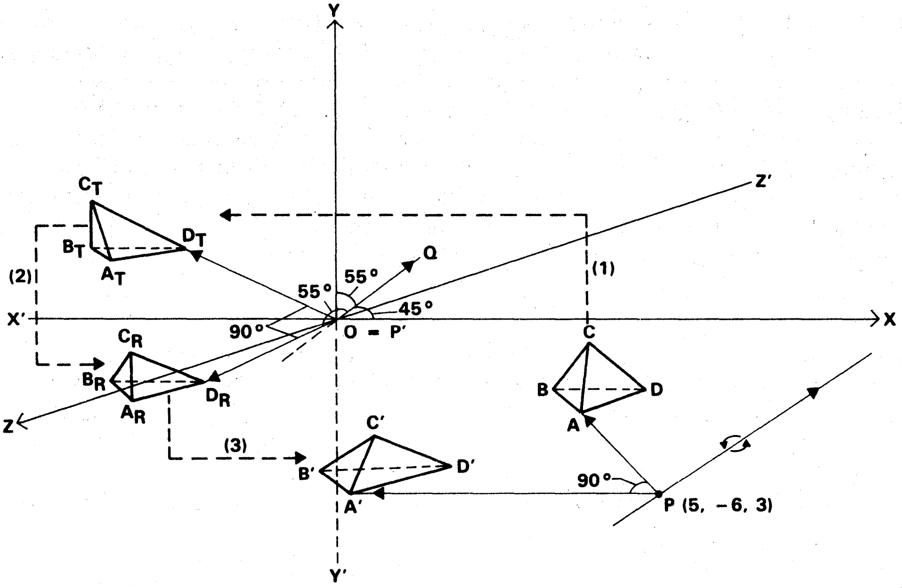
$\bar{Q}$ = vector defining axis of rotation = $[q1\ q2\ q3]$

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 [n1, n2, n3], and the angle ϕ of rotation about the axis, all of which are needed to define matrix [R]. Suppose, for example, that a tetrahedron ABCD, represented by the coordinate matrix below is to be rotated about an axis of rotation RX which passes through a point P = [5 −6 3 1] and whose direction cosines are given by unit vector [n1 = 0.866, n2 = 0.5, n3 = 0.707]. The angle of rotation 0 is 90 degrees (see Figure 24). The rotation matrix [R] becomes

$$\begin{array}{|cccc|l} \hline 2 & -3 & 3 & 1 & \rightarrow A \\ 1 & -2 & 2 & 1 & \rightarrow B \\ 2 & -1 & 2 & 1 & \rightarrow C \\ 2 & -2 & 2 & 1 & \rightarrow D \\ \hline \end{array}$$

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



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

Figure 24. Sequence of Matrix Operations

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

$$
\begin{vmatrix} xa & ya & za & h1 \\ xb & yb & zb & h2 \\ xc & yc & zc & h3 \\ xd & yd & zd & h4 \end{vmatrix}
=
\begin{vmatrix} 2 & -3 & 3 & 1 \\ 1 & -2 & 2 & 1 \\ 2 & -1 & 2 & 1 \\ 2 & -2 & 2 & 1 \end{vmatrix}
\underset{\substack{\downarrow \\ \text{translation} \\ \text{to origin}}}{\begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -5 & 6 & -3 & 1 \end{vmatrix}}
\underset{\substack{\downarrow \\ \text{rotation about origin}}}{\begin{vmatrix} 0.750 & 1.140 & 0.112 & 0 \\ -0.274 & 0.250 & 1.22 & 0 \\ 1.112 & -0.513 & 0.500 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}}
\underset{\substack{\downarrow \\ \text{translation} \\ \text{back to} \\ \text{initial} \\ \text{position}}}{\begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 5 & -6 & 3 & 1 \end{vmatrix}}
\tag{3}
$$

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:

$$\boxed{\begin{matrix} 2 & -3 & 3 & 1 \\ 1 & -2 & 2 & 1 \\ 2 & -1 & 2 & 1 \\ 2 & -2 & 2 & 1 \end{matrix}} \quad \underset{\substack{\downarrow \\ \text{translation} \\ \text{to origin}}}{\boxed{\begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -5 & 6 & -3 & 1 \end{matrix}}} = \underset{\substack{\downarrow \\ \text{vertices of translated} \\ \text{tetrahedron}}}{\boxed{\begin{matrix} (2-5) & (-3+6) & (3-3) & 1 \\ (1-5) & (-2+6) & (2-3) & 1 \\ (2-5) & (-1+6) & (2-3) & 1 \\ (2-5) & (-2+6) & (2-3) & 1 \end{matrix}}}$$

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

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:

$$\boxed{\begin{matrix} -3 & 3 & 0 & 1 \\ -4 & 4 & -1 & 1 \\ -3 & 5 & -1 & 1 \\ -3 & 4 & -1 & 1 \end{matrix}} \underset{\substack{\downarrow \\ \text{rotation about origin}}}{\boxed{\begin{matrix} 0.750 & 1.140 & 0.112 & 0 \\ -0.274 & 0.250 & 1.22 & 0 \\ 1.112 & -0.513 & 0.500 & 0 \\ 0 & 0 & 0 & 1 \end{matrix}}} = \underset{\substack{\downarrow \\ \text{rotated coordinates}}}{\boxed{\begin{matrix} -3.072 & -2.670 & 3.324 & 1 \\ -5.208 & -3.047 & 3.932 & 1 \\ -4.732 & -1.657 & 5.264 & 1 \\ -4.458 & -1.907 & 4.044 & 1 \end{matrix}}}$$

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:

$$\boxed{\begin{matrix} -3.072 & -2.670 & 3.324 & 1 \\ -5.208 & -3.047 & 3.932 & 1 \\ -4.732 & -1.657 & 5.264 & 1 \\ -4.458 & -1.907 & 4.044 & 1 \end{matrix}} \underset{\substack{\downarrow \\ \text{translate back}}}{\boxed{\begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 5 & -6 & 3 & 1 \end{matrix}}} = \underset{\substack{\downarrow \\ \text{final rotated coordinates}}}{\boxed{\begin{matrix} 1.928 & -8.670 & 6.324 & 1 \\ -0.208 & -9.047 & 6.932 & 1 \\ 0.268 & -7.657 & 8.264 & 1 \\ 0.542 & -7.907 & 7.044 & 1 \end{matrix}}}$$

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:

$$\boxed{\begin{array}{cccc} xa & ya & za & h1 \\ xb & yb & zb & h2 \\ xc & yc & zc & h3 \\ xd & yd & zd & h4 \end{array}} = \boxed{\begin{array}{cccc} 2 & -3 & 3 & 1 \\ 1 & -2 & 2 & 1 \\ 2 & -1 & 2 & 1 \\ 2 & -2 & 2 & 1 \end{array}} \quad \underset{\substack{\downarrow \\ \text{transformation matrix}}}{\boxed{\begin{array}{cccc} 0.750 & 1.140 & 0.112 & 0 \\ -0.274 & 0.250 & 1.220 & 0 \\ 1.112 & -0.513 & 0.500 & 0 \\ -3.730 & -8.661 & 8.260 & 1 \end{array}}}$$

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:

$$cij = \sum_{k=1}^{n} aik * xkj \quad \text{for } i=1,\ldots,n \quad j=1,\ldots,n \qquad (1)$$

For example, the coordinates may be computed as follows:

$$\begin{aligned} xa = c11 &= a11 * x11 + a12 * x21 + a13 * x31 + a14 * x41 \\ &= 2 * 0.750 + (-3) * (-0.274) + 3 * 1.112 + 1 * (-3.73) \\ &= 1.5 + 0.822 + 3.336 - 3.73 \\ &= 1.928 \end{aligned}$$

$$\begin{aligned} ya = c12 &= a11 * x12 + a12 * x22 + a13 * x32 + a14 * x42 \\ &= 2 * 1.140 + (-3) * 0.250 + 3 * (-0.513) + 1x(-8.661) \\ &= 2.28 - 0.75 - 1.539 - 8.661 \\ &= -8.67 \end{aligned}$$

$$\begin{aligned} za = c13 &= a11 * x13 + a12 * x23 + a13 * x33 + a14 * x43 \\ &= 2 * 0.112 + (-3) * 1.220 + 3 * 0.500 + 1 * 8.260 \\ &= 0.224 - 3.66 + 1.5 + 8.260 \\ &= 6.324 \end{aligned}$$

$$\begin{aligned} h1 = c14 &= a11 * x14 + a12 * x24 + a13 * x34 + a14 * x44 \\ &= 2 * 0 + (-3) * 0 + 3 * 0 + 1 * 1 \\ &= 0 + 0 + 0 + 1 \\ &= 1 \end{aligned}$$

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

The other rotated vertices are computed in a similar manner:

$$\begin{aligned} B' &= [-5.208 \; -3.047 \; 3.932 \; 1] \\ C' &= [-4.732 \; -1.657 \; 5.264 \; 1) \\ D' &= [-4.458 \; -1.907 \; 4.044 \; 1] \end{aligned}$$

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 h1, h2, h3 and h4 may be made unity due to the availability of large dynamic range. In the example presented below, some of the calculations pertaining to vertex A' are shown but the same approach can be applied to any number of points and any vector space.

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

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

I 9–0	CLKMODE	CONFIG 1–0	PIPES 2–0	SELOP 7–0	RND 1–0	FAST	ENRA	ENRB	ENRC	SELMS/$\overline{\text{LS}}$	$\overline{\text{OEY}}$	$\overline{\text{OEC}}$	$\overline{\text{OES}}$	BYTEP	SELST 1–0	$\overline{\text{RESET}}$	$\overline{\text{HALT}}$	TP 1–0
00 0100 0000	0	01	010	1111 xxxx	00	0	1	1	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	xx	1	1	11
10 0000 0000	0	01	010	1111 1010	00	0	1	1	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	xx	1	1	11
10 0000 0000	0	01	010	1111 1010	00	0	1	1	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	xx	1	1	11
10 0000 0000	0	01	010	1111 1010	00	0	1	1	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	xx	1	1	11
10 0000 0000	0	01	010	1111 1010	00	0	1	1	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	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:

$$\boxed{\begin{matrix} 0 \\ 0 \\ 0 \\ 1 \end{matrix}}$$

Table 32. Single-Precision Matrix Multiplication (PIPES2-PIPES0 = 010)

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

The h-scalars h1, h2, h3, and h4 are equal to 1. The number of clock cycles to generate each 4-tuple can then be decreased from 16 to 13 cycles. Total number of clock cycles to calculate all four vertices is reduced from 66 to 54 clocks. Figure 25 summarizes the overall matrix transformation.

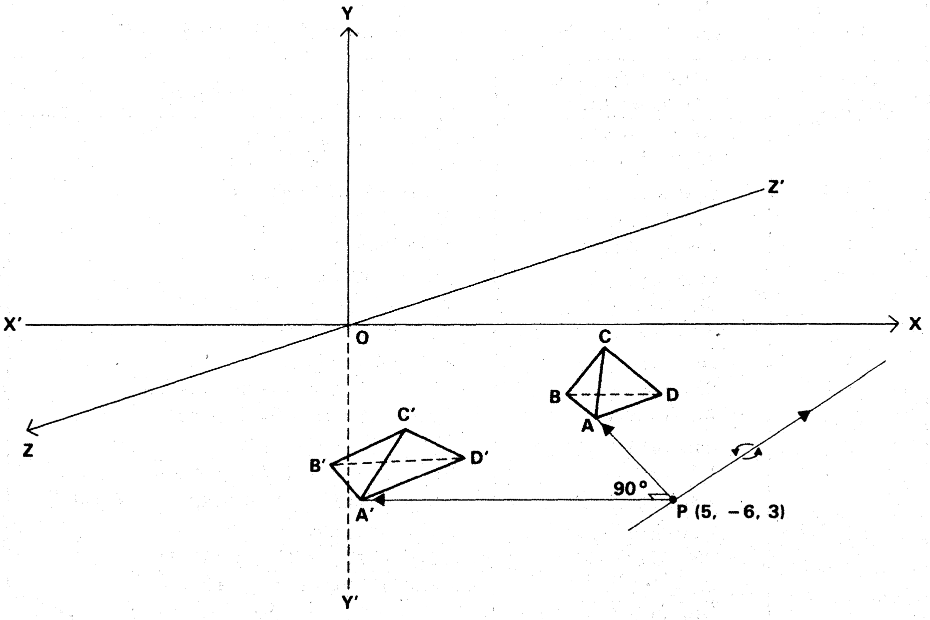


Figure 25. Resultant Matrix Transformation

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 33 presents the sequence of calculations for the first two coordinates, xa and ya.

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		p1			
4	Chn	x12	a11	Mul	x41	a14	p3	s1	p2			
5	Chn	x22	a12	Chn	x12	a11	p4	†	p3	s1	p2	
6	Chn	x32	a13	Chn	x22	a12	p5	s2	p4	†	p2	
7	Chn	x42	a14	Chn	x32	a13	p6	s3	p5	s2	p2	
8	Chn	x13	a11	Chn	x42	a14	p7	s4	p6	s3	s2	
9	Chn	x23	a12	Chn	x13	a11	p8	xa	p7	s4	p6	
10	Chn	x33	a13	Chn	x23	a12	p9	s5	p8	xa	p6	xa
11	Chn	x43	a14	Chn	x33	a13	p10	s6	p9	s5	p6	
12	Chn	x14	a11	Chn	x43	a14	p11	s7	p10	s6	s5	
13	Chn	x24	a12	Chn	x14	a11	p12	ya	p11	s7	p10	
14	Chn	x34	a13	Chn	x24	a12	p13	s8	p12	ya	p10	ya
15	Chn	x44	a14	Chn	x34	a13	p14	s9	p13	s8	p10	

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

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:

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

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:

$$Q = A/B = A * Xn, \text{ where } Xn = \text{the value of X after the nth iteration}$$
$$n = \text{the number of iterations to achieve the desired precision}$$

Intermediate values of X are calculated using the following expression:

$$Xi + 1 = Xi * (2 - B * Xi), \text{ where } X0 = \text{approximates } 1/B \text{ for the range } 0 < X0 < 2/B$$

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 X0 (approximately equal to 1/B) is assumed to be given, and the following operations are performed to evaluate Q from double-precision inputs:

$$X1 = X0(2 - B * X0)$$

$$X2 = X1(2 - B * X1) = X0(2 - B * X0) * (2 - B * X0(2 - B * X0))$$

$$X3 = X2(2 - B * X2)$$

$$X3 = X0(2-B * X0) * (2-B * X0(2-B * X0)) * (2-B * X0 * (2-B * X0) * (2-B * X0 * (2-B * X0)))$$

$$Q = A * 1/B = A * X3$$

$$A/B = A * X0(2-B * X0) * (2-B * X0(2-B * X0)) * (2-B * X0 * (2-B * X0) * (2-B * X0 * (2-B * X0)))$$

X1 X1 X1 X1

X2 X2

X3

Table 36 presents decimal and hexadecimal values for A, B, and X0, 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
X0	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 * X0	B → RA.2, X0 → RB.2 RA.2 * RB.2 → P.4
3, 4	2 – B * X0	2 – P.4 → S.6
5, 6	X1 = X0(2 – B * X0)	RB.2 * S.6 → P.8
7, 8	B * X1	RA.2 * P.8 → P.10
9, 10	2 – B * X1	P.8 → C.9, 2 – P.10 → S.12
11, 12	X2 = X1(2 – B * X1)	C.9 * S.12 P.14
13, 14	B * X2	RA.2 * P.14 → P.16
15, 16	2 – B * X2	P.14 → C.15, 2 – P.16 → S.18
17, 18	X3 = X2(2 – B * X2)	A → RA.18, C.15 * S.18 → P.20
19, 20	A * X3	RA.18 * P.20 → P.22
21, 22	Output MSH	P.22.MSH → 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:

```
                          S
      I                   E
      N C C               L
      S L C               M
L     T K O P S           S       B S R̅   T
I   C R M N I E   F E E S /       Y E E H̅ E D       D
N C L U O F P L R A N N R   O̅ O O T L S A S A     D B     D
E L K C D I E O N S R R C L̅ 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 hhhhhhhh hhhhhhhh 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 X0 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 X0, 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 $n = 2$:

$$Q = A * 1/B = A * X2$$

$$\text{where } X2 = X1 * (2 - B * X1) \text{ and } X1 = X0 * (2 - B * X0)$$

$$A * 1/B = A * X0 * (2 - B * X0) * [2 - B * X0 * (2 - B * X0)]$$

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 3 1 1 3 00000000 00000000 0 0
14 1 0 040 0 0 2 EF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 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 3 1 1 3 00000000 00000000 0 0
16 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 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 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 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 3 1 1 3 00000000 00000000 0 0
20 1 0 040 0 0 2 EF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 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 3 1 1 3 00000000 00000000 0 0
22 1 0 020 0 0 2 EF 0 0 0 0 1 1 0 0 0 0 3 1 1 3 00000000 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:

$$A/B = A * X0 * (2 - B * X0) * [2 - B * X0 * (2 - B * X0)]$$
$$* (2 - B * X0 .(2 - B * X0) * [2 - B * X0 .(2 - B * X0)])$$

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
```

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 * Xn

where Xn = the value of X after the nth iteration given

Xi + 1 = 0.5 * Xi * [3 − B * (Xi ZZ 2)]

X0 = a guess at 1/sqrt(B) where 0 < X0 < 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 X0 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 * X2

where X2 = 0.5 * X1 * [3 − B * (X1 2)]

and X1 = 0.5 * X0 * [3 − B * (X0 2)]

A = B * 0.5 * 0.5 * X0 * [3 − B * (X0 2)]
* [3 − B * (0.5 * X0 * [3 − B * (X0 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 * X0  2
;               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 * X0  2)
;               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
;
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
```

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 sqrt(B) is to be evaluated:

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)]
    * [3 − B * (0.5 * X0 * [3 − B * (X0 2)]) 2]) 2]

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
;
;
;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 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 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
```

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 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 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 00000000 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 00000000 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 00000000 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 00000000 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 00000000 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 00000000 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 00000000 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 00000000 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 00000000 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 00000000 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 00000000 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 00000000 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
```

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 00000000 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 00000000 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 00000000 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 00000000 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 00000000 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 00000000 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 00000000 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 00000000 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 00000000 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 00000000 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 00000000 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 00000000 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 00000000 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 00000000 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 * ∞ 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.

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 $X0 = m' * (2^{e'})$, 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: $1 \leq m < 2$.

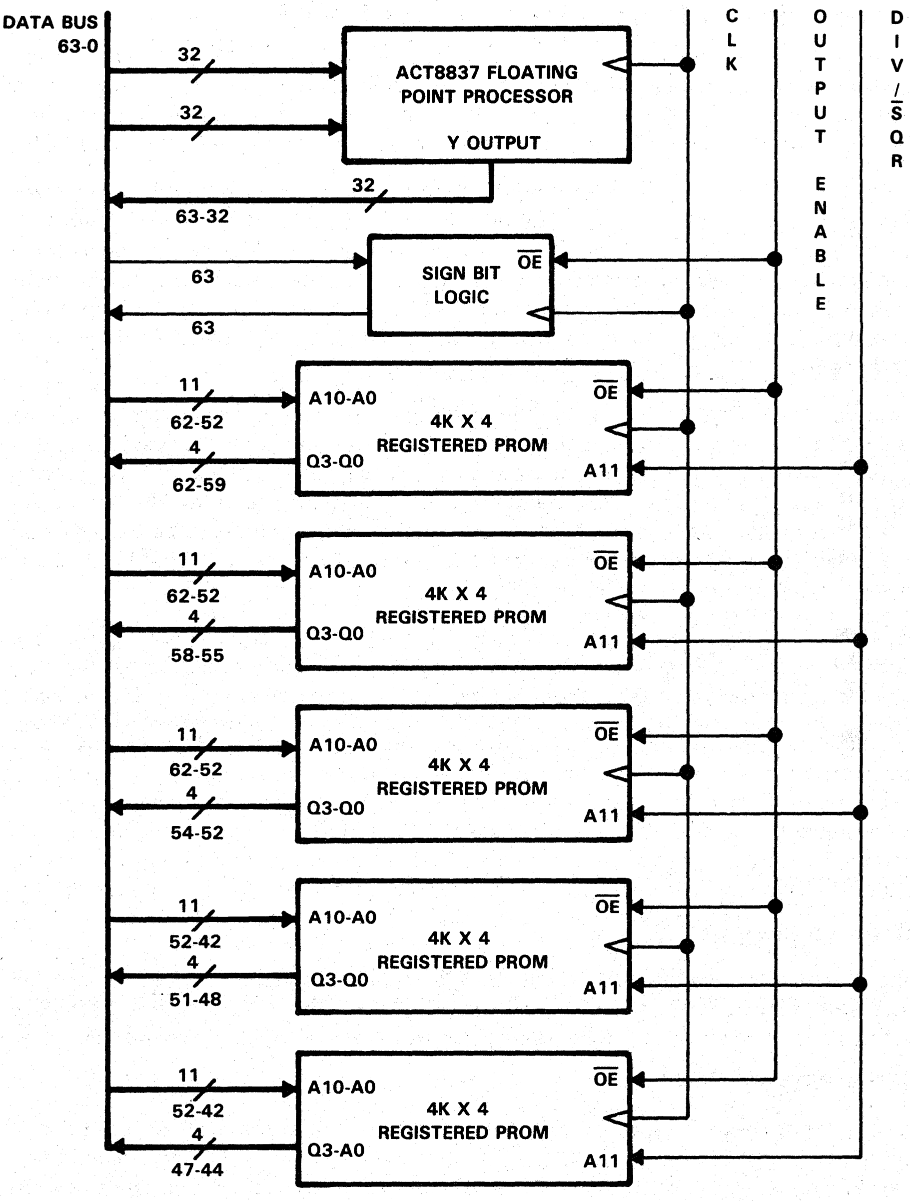


Figure 26. IEEE Double-Precision Seed ROM for Newton-Raphson Division and Square Root

This range of values of m can be subdivided into two cases:

$m = 1$, 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

$$m'' = m' * (2^1) = 2/m \qquad (1)$$

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 $X0 = m' * (2^{e'})$, to use m″ in the PROMs, we must have an e″ in the exponent such that $X0 = m'' * (2^{e''})$. This is true for $e'' = e' - 1$. Since, $X0 = m'' * (2^{e''})$, the following substitution can be made:

$$\begin{aligned} X0 &= (m' * (2^1)) * (2^{(e'-1)}) \\ &= m' * (2^1) * (2^{e'}) * (2^{(-1)}) \\ &= m' * (2^{e'}) * (2^{(1-1)}) \\ &= m' * (2^{e'}) * (2^0) \\ &= m' * (2^{e'}) \end{aligned}$$

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 $m = 1$,

$$m'' = 2 / m - \text{delta}$$

Where $\text{delta} = 2^{(-8)}$

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

$$X'' = 2045 - X \qquad (2)$$

Given that the mantissa seed ROM uses 10 bits of the mantissa to determine the seed, each seed Xm will be used for some range of mantissas, Bm to (Bm + 2 * delta). The formula for Xm is from formula (1).

$$\begin{aligned} &2/Bm && \rightarrow Xm \\ &2/(Bm + 2 * delta) && \rightarrow Xm \end{aligned}$$

Where delta $= 2^{(-11)}$

This value is used since the actual Xm should be generated by the mantissa in the center of the given range:

$$Xm = 2/(Bm + delta)$$

This would result in a more accurate seed on the average. Therefore, the formula used to generate the mantissa part of the seed is

$$Xm = 2/(Bm + (2^{(-11)})) \tag{3}$$

Square Root PROMs

The seed for the square root, X0, is actually the reciprocal of the square root of the data, B:

$$X0 = 1 / (B^{(1/2)})$$

Given B $= m * (2^e)$ and X0 $= m' * (2^{e'})$, the expression for X0 can be evaluated by substitution and reduction:

$$\begin{aligned} X0 &= 1 / ((m * (2^e))^{(1/2)}) \\ &= 1 / (m^{(1/2)} * (2^{(e/2)})) \\ &= m^{(-1/2)} * (2^{(-e/2)}) \end{aligned}$$

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″:

$$\begin{aligned} e' &= -e/2 && \text{for e even} \\ e' &= e'' + 1/2 && \text{for e odd} \end{aligned}$$

Rewriting e″ in terms of e produces this expression:

$$e'' = e' - 1/2 = (-e/2) - 1/2 \qquad \text{for e odd}$$

Then a valid IEEE exponent, e″, can be written for all e as

$$\begin{aligned} e'' &= -e/2 && \text{for e even} \\ e'' &= (-e/2) - 1/2 && \text{for e odd} \end{aligned}$$

This is equivalent to $e'' = \mathrm{int}(-e/2)$ for all e. However, the 1/2 affects the mantissa:

$X0 = m' * (2^{e'})$
$X0 = m' * (2^{(e'' + 1/2)})$ for odd e
$X0 = m' * (2^{1/2}) * (2^{e''})$ for odd e

Since $X0 = m'' * (2^{e''})$ m'' can be rewritten as

$m'' = m'$ for even e
$m'' = m' * (2^{1/2})$ for odd e

In terms of m, $m'' = m^{-1/2}$ for even e
$m'' = (m^{-1/2}) * (2^{1/2})$ for odd e

Simplifying m'' for odd e,

$m'' = (1/m^{1/2}) * (2^{1/2})$ for odd e
$m'' = (2/m^{1/2})$ for odd e

Just as the divide exponent needed to be converted to excess 1023 notation, so the same must be done for the square root:

$X'' = e'' + 1023$
$X = e + 1023$
$X'' = \mathrm{int}(-e/2) + 1023$
$X'' = \mathrm{int}((1023\text{-}X) / 2) + 1023$

The IEEE bits for the exponent seed, X'', can be expressed in terms of the IEEE bits for the exponent of B, X:

$X'' = \mathrm{int}(\ (1023\text{-}X) / 2\ ) + 1023$

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,

$m'' = m^{-1/2}$ for odd X
$m'' = 2/m^{1/2}$ for even X

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.

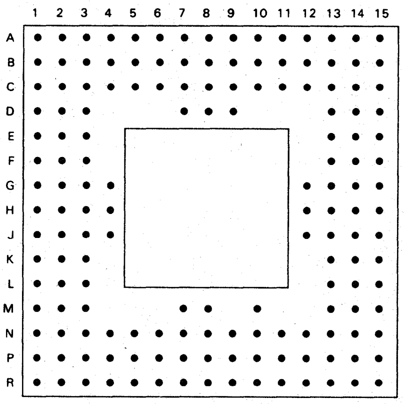
The 'ACT8841 is ideal for multiprocessor application, where memory bottlenecks tend to occur. For example, four 32-bit buses can be easily connected by two 'ACT8841 devices. System architectures based on the 16-port 'ACT8841 can include up to 16 switching nodes (i.e., processors, memories, or bus interfaces). Larger processor arrays can be built with multistage interconnect schemes.

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**

GB PACKAGE
(TOP VIEW)



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

TEXAS INSTRUMENTS
POST OFFICE BOX 655012 • DALLAS, TEXAS 75265

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 opertion from 0°C to 70°C.

Table 1. 'ACT8841 Pin Grid Allocation

PIN		PIN		PIN		PIN	
NO.	NAME	NO.	NAME	NO.	NAME	NO.	NAME
A1	GND	C10	D31	H12	V_{CC}	N7	CNTR13
A2	GND	C11	$\overline{\text{OED6}}$	H13	LSCLK	N8	CREAD0
A3	D37	C12	V_{CC}	H14	SELDLS	N9	V_{CC}
A4	D35	C13	GND	H15	CNTR3	N10	D0
A5	D33	C14	D23	J1	$\overline{\text{OEC}}$	N11	D3
A6	$\overline{\text{WE}}$	C15	D21	J2	CRWRITE0	N12	D6
A7	CRADR1	D1	D43	J3	CRWRITE1	N13	GND
A8	CNTR7	D2	D42	J4	GND	N14	D8
A9	CNTR4	D3	V_{CC}	J12	GND	N15	D9
A10	$\overline{\text{OED7}}$	D7	GND	J13	CNTR2	P1	GND
A11	D29	D8	V_{CC}	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	$\overline{\text{OED12}}$	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	$\overline{\text{OED10}}$	K15	$\overline{\text{OED3}}$	P9	TP0
B4	D36	E13	$\overline{\text{OED5}}$	L1	D49	P10	$\overline{\text{OED0}}$
B5	D34	E14	D18	L2	D50	P11	D2
B6	$\overline{\text{OED8}}$	E15	D17	L3	$\overline{\text{OED13}}$	P12	D4
B7	CRADR0	F1	$\overline{\text{OED11}}$	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	$\overline{\text{OED4}}$	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	V_{CC}	R5	D61
B15	GND	G3	CNTR10	M10	GND	R6	$\overline{\text{OED15}}$
C1	D41	G4	GND	M13	V_{CC}	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	$\overline{\text{OED9}}$	G15	CRSEL0	N2	D55	R11	D1
C6	D32	H1	CNTR11	N3	GND	R12	$\overline{\text{OED1}}$
C7	V_{CC}	H2	SELDMS	N4	V_{CC}	R13	D5
C8	CRCLK	H3	MSCLK	N5	$\overline{\text{OED14}}$	R14	GND
C9	CNTR6	H4	V_{CC}	N6	D63	R15	GND

Table 2. 'ACT8841 Pin Functional Description

PIN NAME	PIN NO.	I/O	DESCRIPTION
CNTR0 CNTR1 CNTR2 CNTR3 CNTR4 CNTR5 CNTR6 CNTR7 CNTR8 CNTR9 CNTR10 CNTR11 CNTR12 CNTR13 CNTR14 CNTR15	J15 J14 J13 H15 A9 B9 C9 A8 G1 G2 G3 H1 P7 N7 R7 P8	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.
CRADR0 CRADR1	B7 A7	I	Control register address. Selects 16-bits of control flip-flops as a source/destination for outputs/inputs on CNTR0-CNTR15. (see Table 7)
CRCLK	C8	I	Control register clock. Clocks CNTR0-CNTR15 into the control flip-flops on low-to-high transition.
CREAD0 CREAD1 CREAD2	N8 R8 R9	I	Selects one of eight banks of control flip-flops to read out on CNTR0-CNTR15 in 16-bit words addressed by CRADR1-CRADR0.
CRSEL0 CRSEL1 CRSEL2 CRSEL3	G15 G14 G13 F15	I	Selects one of ten control configurations.
CRSRCE	B8	I	Load source select. When low selects CNTR inputs, when high selects DATA inputs.

Table 2. 'ACT8841 Pin Functional Description (continued)

PIN		I/O	DESCRIPTION
NAME	NO.		
CRWRITE0	J2	I	Destination select. Selects one of eight control banks. (see Table 4)
CRWRITE1	J3		
CRWRITE2	K1		
D0	N10	I/O	I/O data bits 0 through 31 (data bits 0 through 31 are the least significant half).
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		
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	I/O	I/O data bits 32 through 35 (data bits 32 through 63 are the most significant half).
D33	A5		
D34	B5		
D35	A4		

Table 2. 'ACT8841 Pin Functional Description (continued)

PIN		I/O	DESCRIPTION
NAME	NO.		
D36	B4	I/O	I/O data bits 36 through 63 (data bits 32 through 63 are the most significant half).
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		
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		Ground (all pins must be used).
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		

Table 2. 'ACT8841 Pin Functional Description (continued)

PIN		I/O	DESCRIPTION
NAME	NO.		
GND	J4		Ground (all pins must be used).
GND	J12		
GND	M7		
GND	M10		
GND	N3		
GND	N13		
GND	P1		
GND	P2		
GND	P14		
GND	P15		
GND	R1		
GND	R2		
GND	R14		
GND	R15		
LSCLCK	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.
$\overline{\text{OEC}}$	J1	I	Output enable for control flip-flops, active low
$\overline{\text{OED}}$0	P10	I	Output enables for data nibbles, active low
$\overline{\text{OED}}$1	R12		
$\overline{\text{OED}}$2	L13		
$\overline{\text{OED}}$3	K15		
$\overline{\text{OED}}$4	F14		
$\overline{\text{OED}}$5	E13		
$\overline{\text{OED}}$6	C11		
$\overline{\text{OED}}$7	A10		
$\overline{\text{OED}}$8	B6		
$\overline{\text{OED}}$9	C5		
$\overline{\text{OED}}$10	E3		
$\overline{\text{OED}}$11	F1		
$\overline{\text{OED}}$12	K2		
$\overline{\text{OED}}$13	L3		
$\overline{\text{OED}}$14	N5		
$\overline{\text{OED}}$15	R6		

Table 2. 'ACT8841 Pin Functional Description (concluded)

PIN NAME	NO.	I/O	DESCRIPTION
SELDLS	H14	I	When low, selects the stored, least significant data input to the main internal bus. When high, real-time data is selected.
SELDMS	H2	I	When low, selects the stored, most significant data input to the main internal bus. When high, real-time data is selected.
TP0 TP1	P9 R10	I	Test pins. High during normal operation. (see Table 9)
V_{CC}	C7		5-V supply
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		
$\overline{WE}$	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.

logic symbol

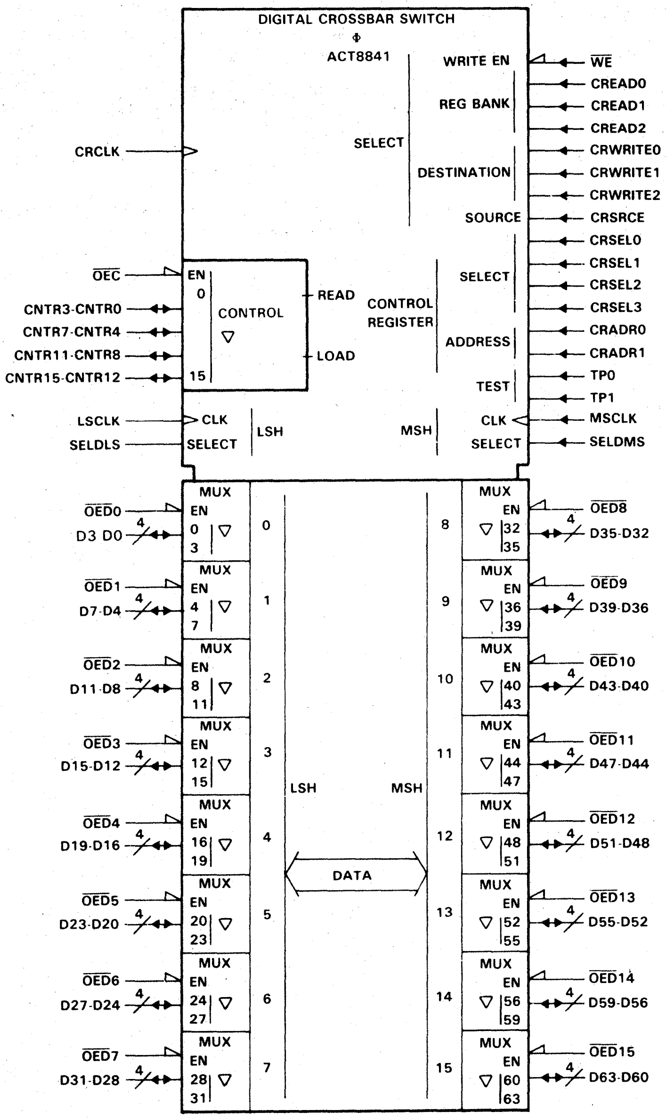


FIGURE 1

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.

functional block diagram

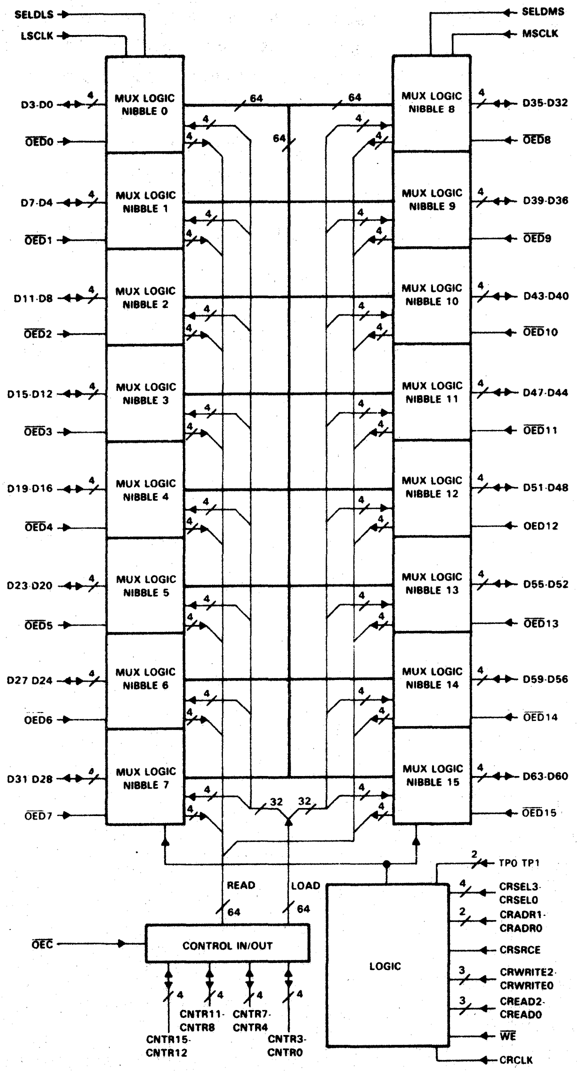


FIGURE 2

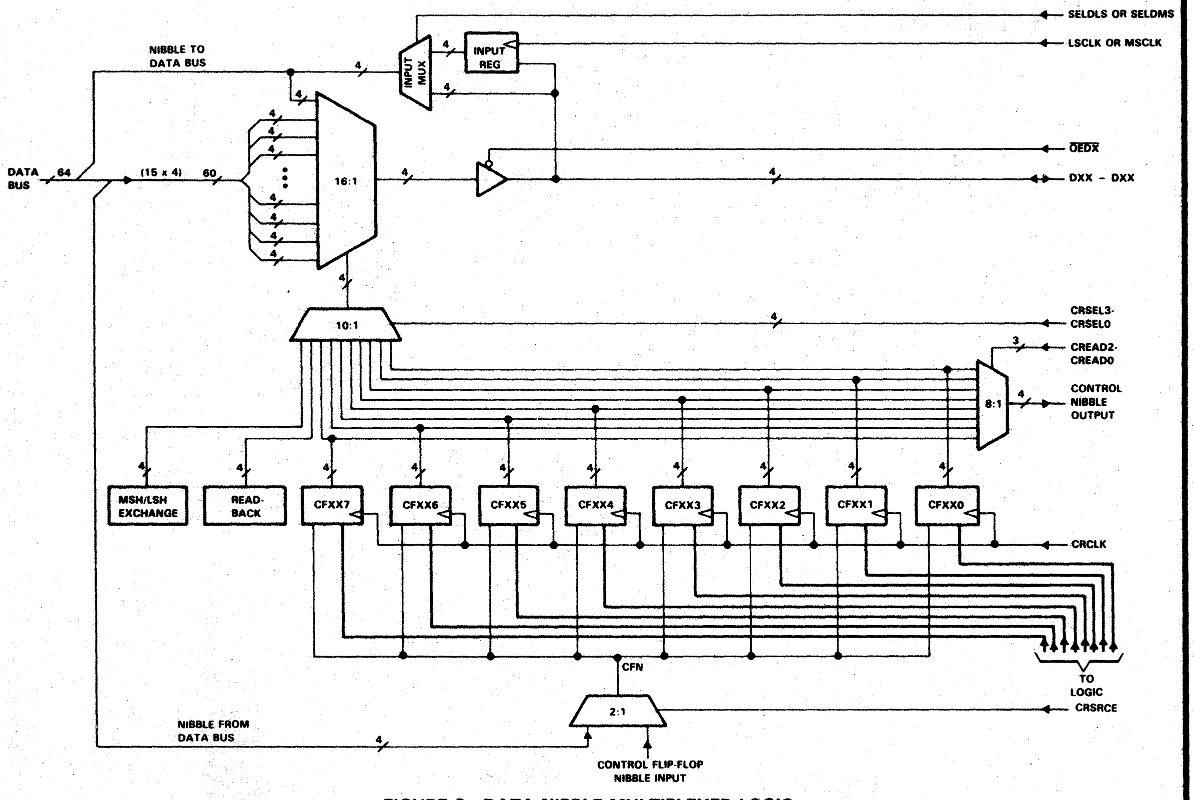


FIGURE 3. DATA NIBBLE MULTIPLEXER LOGIC

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 CFXX0 to CFXX7, where XX indicates the number of the nibble multiplexer logic group (0 < = XX < = 15). CFXX0 represents the 16 CF groups (one from each logic block) which make up flip-flop control bank 0, CFXX1 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

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
CFN3	CFN2	CFN1	CFN0	MULTIPLEXER OUTPUT
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

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, $\overline{\text{WE}}$, 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	$\overline{\text{WE}}$	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 $\overline{\text{OEC}}$ 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.

test pins

TP1-TP0 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-TP0 and all output enables ($\overline{\text{OED}}$15-$\overline{\text{OED}}$0 and $\overline{\text{OEC}}$). To force all outputs and I/Os high, TP1 and all output enables are pulled low, and TP0 is driven high. When TP0 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	TP0	$\overline{\text{OED}}$15-$\overline{\text{OED}}$0	$\overline{\text{OEC}}$	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-CFN0 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. $\overline{\text{WE}}$ 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.

Example 1. Programming a Nibble Broadcast

INST. NO.	CRSRCE	CRWRITE2	CRWRITE1	CRWRITE0	CRADR1	CRADR0	CNTR I/O NUMBERS 15-12	11-8	7-4	3-0	CRSEL3	CRSEL2	CRSEL1	CRSEL0	$\overline{WE}$	SELDMS	SELDLS	$\overline{OED15}$-$\overline{OED0}$	$\overline{OEC}$	CRCLK
1	0	0	0	0	0	0	1111	1111	1111	1111	X	X	X	X	0	X	X	XXXX XXXX XXXX XXXX	1	↑
2	0	0	0	0	0	1	1111	1111	1111	1111	X	X	X	X	0	X	X	XXXX XXXX XXXX XXXX	1	↑
3	0	0	0	0	1	0	1111	1111	1111	1111	X	X	X	X	0	X	X	XXXX XXXX XXXX XXXX	1	↑
4	0	0	0	0	1	1	1111	1111	1111	1111	X	X	X	X	0	X	X	XXXX XXXX XXXX XXXX	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, CF0 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	CRWRITE0	CRADR1	CRADR0	CNTR I/O NUMBERS 15-12	11-8	7-4	3-0	CRSEL3	CRSEL2	CRSEL1	CRSEL0	$\overline{WE}$	SELDMS	SELDLS	$\overline{OED15}$-$\overline{OED0}$	$\overline{OEC}$	CRCLK
1	0	1	1	1	0	0	0100	0000	1100	1000	X	X	X	X	0	X	X	XXXX XXXX XXXX XXXX	1	↑
2	0	1	1	1	0	1	0101	0001	1101	1001	X	X	X	X	0	X	X	XXXX XXXX XXXX XXXX	1	↑
3	0	1	1	1	1	0	0111	0011	1111	1011	X	X	X	X	0	X	X	XXXX XXXX XXXX XXXX	1	↑
4	0	1	1	1	1	1	0111	0011	1111	1011	X	X	X	X	0	X	X	XXXX XXXX XXXX XXXX	1	↑
5	X	X	X	X	X	X	XXXX	XXXX	XXXX	XXXX	0	1	1	1	1	0	0	0000 0000 0000 0000	1	None

Comments

INST. NO.	COMMENT
1	Loads CF12, CF8, CF4, CF0 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 $\overline{\text{WE}}$ 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.

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	$\overline{\text{WE}}$	SELDMS	SELDLS	$\overline{\text{OED}}$15-$\overline{\text{OED}}$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	$\overline{\text{WE}}$	SELDMS	SELDLS	$\overline{\text{OED}}$15-$\overline{\text{OED}}$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. $\overline{\text{WE}}$ is held high and $\overline{\text{OEC}}$ 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	$\overline{\text{OEC}}$	CRADR1	CRADR0	$\overline{\text{WE}}$	CNTR I/O NUMBERS 15-12	11-8	7-4	3-0	COMMENT
1	0	1	0	0	0	0	1	0100	0000	1100	1000	Read CF12, CF8, CF4, CF0
2	0	1	0	0	0	1	1	0101	0001	1101	1001	Read CF13, CF9, CF5, CF1
3	0	1	0	0	1	0	1	0110	0010	1110	1010	Read CF14, CF10, CF6, CF2
4	0	1	0	0	1	1	1	0111	0011	1111	1011	Read CF15, CF11, CF7, CF3

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	5.0	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

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}	$I_{OH} = -20\ \mu A$	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}	$I_{OL} = 20\ \mu A$	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}.

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	CNTRn		10	18	
	CRCLK			10	18	
	CRAD1, CRAD0			8	16	
	TP1, TP0	All outputs		10	19	
t_{en}	TP1, TP0	All outputs		10	15	ns
	$\overline{OED}$	Data out		7	12	
	$\overline{OEC}$	CNTRn		8	14	
t_{dis}	TP1, TP0	All outputs		10	15	ns
	$\overline{OED}$	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
t_{su} Setup time before CRCLK	Data	7		ns
	CNTRn	7		
	SELDMS, SELDLS	9		
	CRADR1,CRADR0	8		
	CRSRCE, CRWRITE2-CRWRITE0	8		
	LSCLK, MSCLK	10		
	$\overline{WE}$	8		
t_{su} Setup time, data before LSCLK or MSCLK		7		ns
t_h Hold time after CRCLK	Data	0		ns
	CNTRn	0		
	SELDMS, SELDLS	0		
	CRADR1, CRADR0	0		
	CRSRCE, CRWRITE	0		
	$\overline{WE}$	0		
t_h Hold time, data after LSCLK or MSCLK		0		ns

'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 V_{CC} 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.

SN74ACT8847
64-Bit Floating Point Unit

- Superset of TI's SN74ACT8837
- 30-ns, 40-ns and 60-ns Pipelined Performance
- Low-Power EPIC™ CMOS
- 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

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.

Contents

Contents (Continued)

Contents (Concluded)

List of Illustrations

List of Illustrations (Continued)

List of Illustrations (Concluded)

List of Tables

List of Tables (Concluded)

Introduction

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.

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 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.

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

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

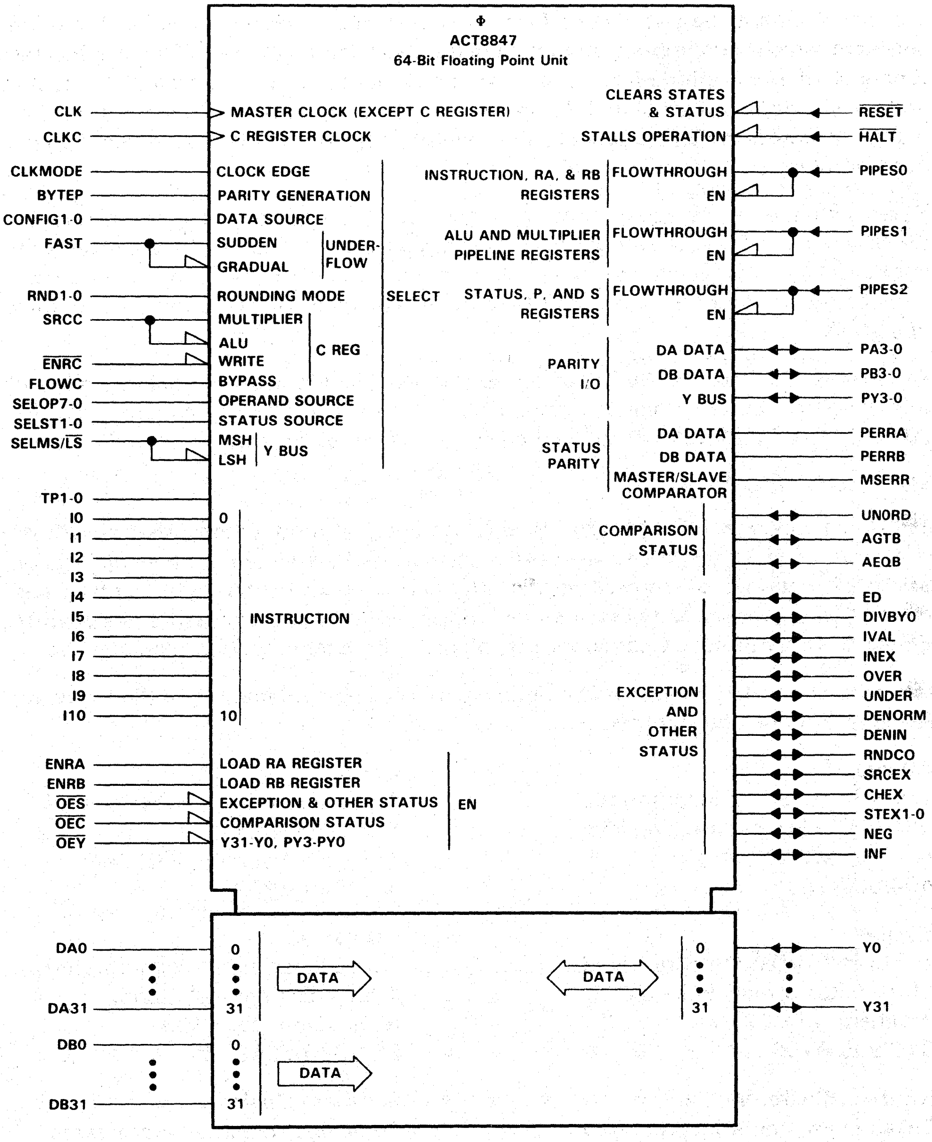
Northern California
Texas Instruments Incorporated
5353 Betsy Ross Drive
Santa Clara, CA 95054
408/748-2220

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

'ACT8847 Logic Symbol



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 Pin Descriptions

Pin descriptions and grid allocation for the 'ACT8847 are given on the following pages.

208 PIN . . . GA PACKAGE
(TOP VIEW)

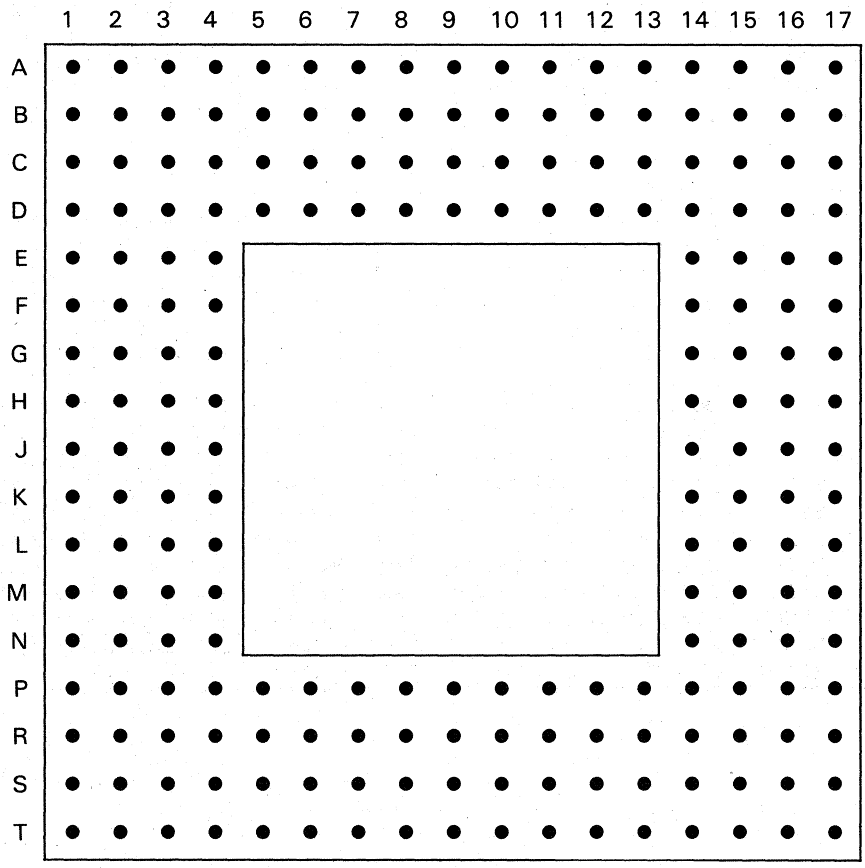


Table 1. 'ACT8847 Pin Grid Allocation

PIN		PIN		PIN		PIN		PIN		PIN	
NO.	NAME	NO.	NAME	NO.	NAME	NO.	NAME	NO.	NAME	NO.	NAME
A1	NC	C2	Y0	E3	FAST	J15	FLOWC	P1	$\overline{\text{ENRC}}$	S1	NC
A2	INF	C3	Y3	E4	GND	J16	SRCC	P2	PIPES0	S2	PB0
A3	Y5	C4	Y6	E14	GND	J17	BYTEP	P3	$\overline{\text{RESET}}$	S3	DB0
A4	Y8	C5	Y9	E15	AGTB	K1	SELOP3	P4	PB1	S4	DB4
A5	Y11	C6	Y12	E16	AEQB	K2	SELOP4	P5	DB1	S5	DB11
A6	Y14	C7	Y15	E17	MSERR	K3	SELOP5	P6	DB5	S6	DB12
A7	Y17	C8	Y18	F1	I5	K4	GND	P7	DB9	S7	DB15
A8	Y20	C9	Y23	F2	I3	K14	GND	P8	DB16	S8	DB19
A9	Y21	C10	Y26	F3	RND0	K15	PA1	P9	DB21	S9	DB23
A10	Y24	C11	Y30	F4	GND	K16	PA2	P10	DB28	S10	DB26
A11	Y27	C12	PY1	F14	GND	K17	PA3	P11	DA0	S11	DB30
A12	Y29	C13	UNDER	F15	PERRA	L1	SELOP6	P12	DA4	S12	DA2
A13	PY0	C14	INEX	F16	$\overline{\text{OEY}}$	L2	SELOP7	P13	DA8	S13	DA6
A14	PY3	C15	DENIN	F17	$\overline{\text{OES}}$	L3	CLK	P14	DA12	S14	DA10
A15	IVAL	C16	SRCEX	G1	I7	L4	V_{CC}	P15	DA19	S15	DA14
A16	NEG	C17	CHEX	G2	I6	L14	GND	P16	DA22	S16	DA15
A17	NC	D1	I1	G3	I4	L15	DA30	P17	DA23	S17	DA17
B1	ED	D2	RND1	G4	V_{CC}	L16	DA31	R1	PIPES1	T1	NC
B2	Y2	D3	Y1	G14	V_{CC}	L17	PA0	R2	$\overline{\text{HALT}}$	T2	PB3
B3	Y4	D4	GND	G15	$\overline{\text{OEC}}$	M1	ENRB	R3	PB2	T3	DB3
B4	Y7	D5	V_{CC}	G16	SELMS/$\overline{\text{LS}}$	M2	ENRA	R4	DB2	T4	DB7
B5	Y10	D6	GND	G17	TEST1	M3	CLKC	R5	DB6	T5	DB8
B6	Y13	D7	GND	H1	I10	M4	GND	R6	DB10	T6	DB13
B7	Y16	D8	V_{CC}	H2	I9	M14	V_{CC}	R7	DB14	T7	DB17
B8	Y19	D9	GND	H3	I8	M15	DA27	R8	DB18	T8	DB20
B9	Y22	D10	GND	H4	GND	M16	DA28	R9	DB22	T9	DB24
B10	Y25	D11	V_{CC}	H14	GND	M17	DA29	R10	DB27	T10	DB25
B11	Y28	D12	GND	H15	TEST0	N1	CONFIG0	R11	DB31	T11	DB29
B12	Y31	D13	GND	H16	SELST1	N2	CONFIG1	R12	DA3	T12	DA1
B13	PY2	D14	V_{CC}	H17	SELST0	N3	CLKMODE	R13	DA7	T13	DA5
B14	OVER	D15	STEX1	J1	SELOP2	N4	PIPES2	R14	DA11	T14	DA9
B15	RNDC0	D16	STEX0	J2	SELOP1	N14	DA18	R15	DA16	T15	DA13
B16	DENORM	D17	UNORD	J3	SELOP0	N15	DA24	R16	DA20	T16	NC
B17	DIVBY0	E1	I2	J4	V_{CC}	N16	DA25	R17	DA21	T17	NC
C1	PERRB	E2	I0	J14	V_{CC}	N17	DA26				

Table 2. 'ACT8847 Pin Functional Description

PIN NAME	PIN NO.	I/O	DESCRIPTION
AEQB	E16	I/O	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.
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 an exception. If I6 is high, indicates the ALU is the source of an 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 CONFIG1	N1 N2	I	Select data sources for RA and RB registers from DA bus, DB bus and temporary register
DA0	P11	I	DA 32-bit input data bus. Data can be latched in a 64-bit temporary register or loaded directly into an input register
DA1	T12		
DA2	S12		
DA3	R12		
DA4	P12		
DA5	T13		
DA6	S13		
DA7	R13		
DA8	P13		
DA9	T14		
DA10	S14		
DA11	R14		
DA12	P14		
DA13	T15		
DA14	S15		
DA15	S16		
DA16	R15		
DA17	S17		
DA18	N14		
DA19	P15		
DA20	R16		
DA21	R17		
DA22	P16		
DA23	P17		
DA24	N15		

Table 2. 'ACT8847 Pin Functional Description (Continued)

PIN		I/O	DESCRIPTION
NAME	NO.		
DA25	N16	I	DA 32-bit input data bus. Data can be latched in a 64-bit temporary register or loaded directly into an input register.
DA26	N17		
DA27	M15		
DA28	M16		
DA29	M17		
DA30	L15		
DA31	L16		
DB0	S3	I	DB 32-bit input data bus. Data can be latched in a 64-bit temporary register or loaded directly into an input register.
DB1	P5		
DB2	R4		
DB3	T3		
DB4	S4		
DB5	P6		
DB6	R5		
DB7	T4		
DB8	T5		
DB9	P7		
DB10	R6		
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.
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.

Table 2. 'ACT8847 Pin Functional Description (Continued)

PIN NAME	PIN NO.	I/O	DESCRIPTION
DIVBY0	B17	I/O	Status pin indicating an attempted operation involved dividing by zero
ED	B1	I/O	Exception detect status signal representing logical OR of all enabled exceptions in the exception disable register
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).
$\overline{\text{ENRC}}$	P1	I	When low, enables write to C register when CLKC goes high.
FAST	E3	I	When low, selects gradual underflow (IEEE model). When high, selects sudden underflow, forcing all denormalized inputs and outputs to zero.
FLOWC	J15	I	When high, causes product or sum to bypass C register, so that product or sum appears on the C register output bus. Timing is similar to P register or S register feedback operands. C register remains unchanged. Product or sum may also be simultaneously fed back in usual manner (not through C register).
GND GND GND GND GND GND GND GND GND GND GND GND GND GND GND GND GND	D4 D6 D7 D9 D10 D12 D13 E4 E14 F4 F14 H4 H14 K4 K14 L14 M4		Ground pins. NOTE: All ground pins should be used and connected.
$\overline{\text{HALT}}$	R2	I	Stalls operation without altering contents of instruction or data registers. Active low.

Table 2. 'ACT8847 Pin Functional Description (Continued)

PIN		I/O	DESCRIPTION
NAME	NO.		
I0	E2	I	Instruction inputs
I1	D1		
I2	E1		
I3	F2		
I4	G3		
I5	F1		
I6	G2		
I7	G1		
I8	H3		
I9	H2		
I10	H1		
INEX	C14	I/O	Status pin indicating an inexact output
INF	A2	I/O	When high, indicates output value is infinity.
IVAL	A15	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		No internal connection. Pins should be left floating
NC	A17		
NC	S1		
NC	T1		
NC	T16		
NC	T17		
NEG	A15	I/O	When high, indicates result has negative sign.
$\overline{\text{OEC}}$	G15	I	Comparison status output enable. Active low.
$\overline{\text{OES}}$	F17	I	Exception status and other status output enable. Active low.
$\overline{\text{OEY}}$	F16	I	Y bus output enable. Active low.
OVER	B14	I/O	Status pin indicating that the result is greater the largest allowable value for specified format (exponent overflow).
PA0	L17	I	Parity inputs for DA data
PA1	K15		
PA2	K16		
PA3	K17		
PB0	S2	I	Parity inputs for DB data
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.

Table 2. 'ACT8847 Pin Functional Description (Continued)

PIN NAME	NO.	I/O	DESCRIPTION
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.
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.
PY0 PY1 PY2 PY3	A13 C12 B13 A14	I/O	Y port parity data
$\overline{\text{RESET}}$	P3	I	Clears internal states, status, and exception disable register. Contents of internal pipeline registers are lost. Does not affect other data registers. Active low.
RND0 RND1	F3 D2	I	Rounding mode control pins. Select four IEEE rounding modes.
RNDC0	B15	I/O	When high, indicates the mantissa of a wrapped 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/LS	G16	I	When low, selects LSH of 64-bit result to be output on the Y bus. When high, selects MSH of 64-bit result. (No effect on single-precision operations.)
SRCC	J16	I	When low, selects ALU as data source for C register. When high, selects multiplier as data source for C register.
SRCEX	C16	I/O	Status pin indicating source of exception, 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).

Table 2. 'ACT8847 Pin Functional Description (Continued)

PIN		I/O	DESCRIPTION
NAME	NO.		
TEST0	H15	I	Test pins
TEST1	G17		
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).
V_{CC}	D5		5-V power supply
V_{CC}	D8		
V_{CC}	D11		
V_{CC}	D14		
V_{CC}	G4		
V_{CC}	G14		
V_{CC}	J4		
V_{CC}	J14		
V_{CC}	L4		
V_{CC}	M14		
Y0	C2	I/O	32-bit Y output data bus
Y1	D3		
Y2	B2		
Y3	C3		
Y4	B3		
Y5	A3		
Y6	C4		
Y7	B4		
Y8	A4		
Y9	C5		
Y10	B5		
Y11	A5		
Y12	C6		
Y13	B6		
Y14	A6		
Y15	C7		
Y16	B7		
Y17	A7		
Y18	C8		
Y19	B8		
Y20	A8		
Y21	A9		
Y22	B9		
Y23	C9		
Y24	A10		
Y25	B10		
Y26	C10		

Table 2. 'ACT8847 Pin Functional Description (Concluded)

PIN		I/O	DESCRIPTION
NAME	NO.		
Y27	A11	I/O	32-bit Y output data bus
Y28	B11		
Y29	A12		
Y30	C11		
Y31	B12		

'ACT8847 Specifications

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

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

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

recommended operating conditions

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

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

PARAMETER	TEST CONDITIONS	V_{CC}	T_A = 25°C			SN74ACT8847			UNIT
			MIN	TYP	MAX	MIN	TYP	MAX	
V_{OH}	I_{OH} = −20 μA	4.5 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		SN74ACT8847-30		UNIT
		MIN	MAX	
t_{pd1}	Propagation delay from DA/DB/I input register to Y output		72	ns
t_{pd2}	Propagation delay from input register to output buffer		70	ns
t_{pd3}	Propagation delay from pipeline register to output buffer		45	ns
t_{pd4}	Propagation from output register to output buffer		18	ns
t_{pd5}	Propagation delay from SELMS/$\overline{LS}$ to Y output		18	ns
t_{d1}	Propagation delay time, input register to output register		56	ns
t_{d2}	Delay time, input register to pipeline register or pipeline register to output register	30		ns
t_{d3}	Delay time, CLKC after CLK to insure data captured in C register is data clocked into the sum or product register by that clock		8	ns
t_{d4}	Delay time, CLKC after CLK to insure data captured in C register is data clocked into the sum or product register by the previous clock		2	ns

Note: Switching data must be used with timing diagrams for different operating modes.

setup and hold times

PARAMETER			SN74ACT8847-30		UNIT
			MIN	MAX	
t_{su}	Setup time	Instruction before CLK↑	10		ns
		Data operand before CLK↑	10		
		Data operand before second CLK↑ for double-precision operation (input register not enabled)	40		
		SRCC with respect to CLKC		4	
t_h	Hold time	Instruction input after CLK↑	0		ns
		Valid Y bus output of the previous CLK cycle after rising clock edge	5.5		
		Valid status output of the previous CLK cycle after rising clock edge	3		

clock requirements

PARAMETER			SN74ACT8847-30		UNIT
			MIN	MAX	
t_w	Pulse duration	CLK high	10		ns
		CLK low	10		
		CLK low[†]	10		
	Clock period				ns

[†] Clock mode 1 cannot be used.

switching characteristics (see Note)

PARAMETER		SN74ACT8847-40		UNIT
		MIN	MAX	
t_{pd1}	Propagation delay from DA/DB/I input register to Y output		95	ns
t_{pd2}	Propagation delay from input register to output buffer		90	ns
t_{pd3}	Propagation delay from pipeline register to output buffer		60	ns
t_{pd4}	Propagation from output register to output buffer		20	ns
t_{pd5}	Propagation delay from SELMS/$\overline{\text{LS}}$ to Y output		20	ns
t_{d1}	Propagation delay time, input register to output register		75	ns
t_{d2}	Delay time, input register to pipeline register or pipeline register to output register	40		ns
t_{d3}	Delay time, CLKC after CLK to insure data captured in C register is data clocked into the sum or product register by that clock		9	ns
t_{d4}	Delay time, CLKC after CLK to insure data captured in C register is data clocked into the sum or product register by the previous clock		2	ns

Note: Switching data must be used with timing diagrams for different operating modes.

setup and hold times

PARAMETER			SN74ACT8847-40		UNIT
			MIN	MAX	
t_{su}	Setup time	Instruction before CLK↑	12		ns
		Data operand before CLK↑	12		
		Data operand before second CLK↑ for double-precision operation (input register not enabled)	52		
		SRCC with respect to CLKC		4.5	
t_h	Hold time	Instruction input after CLK↑	0		ns
		Valid Y bus output of the previous CLK cycle after rising clock edge	5.5		
		Valid status output of the previous CLK cycle after rising clock edge	4		

clock requirements

PARAMETER			SN74ACT8847-40		UNIT
			MIN	MAX	
t_w	Pulse duration	CLK high	15		ns
		CLK low	15		
		CLK low[†]	10		
	Clock period				ns

[†] Clock mode 1 cannot be used.

switching characteristics (see Note)

PARAMETER		SN74ACT8847-60		UNIT
		MIN	MAX	
t_{pd1}	Propagation delay from DA/DB/I input register to Y output		125	ns
t_{pd2}	Propagation delay from input register to output buffer		120	ns
t_{pd3}	Propagation delay from pipeline register to output buffer		75	ns
t_{pd4}	Propagation from output register to output buffer		28	ns
t_{pd5}	Propagation delay from SELMS/$\overline{\text{LS}}$ to Y output		28	ns
t_{d1}	Propagation delay time, input register to output register		100	ns
t_{d2}	Delay time, input register to pipeline register or pipeline register to output register	60		ns
t_{d3}	Delay time, CLKC after CLK to insure data captured in C register is data clocked into the sum or product register by that clock		12	ns
t_{d4}	Delay time, CLKC after CLK to insure data captured in C register is data clocked into the sum or product register by the previous clock		2	ns

Note: Switching data must be used with timing diagrams for different operating modes.

setup and hold times

PARAMETER		SN74ACT8847-60		UNIT
		MIN	MAX	
t_{su} Setup time	Instruction before CLK↑	16		ns
	Data operand before CLK↑	16		
	Data operand before second CLK↑ for double-precision operation (input register not enabled)	75		
	SRCC with respect to CLKC		6	
t_h Hold time	Instruction input after CLK↑	0		ns
	Valid Y bus output of the previous CLK cycle after rising clock edge	5.5		
	Valid status output of the previous CLK cycle after rising clock edge	4		

clock requirements

PARAMETER		SN74ACT8847-60		UNIT
		MIN	MAX	
t_w Pulse duration	CLK high	20		ns
	CLK low	20		
	CLK low[†]	11		
Clock period				ns

[†] Clock mode 1 cannot be used.

SN74ACT8847 FLOATING POINT UNIT

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

The 'ACT8847 FPU also performs integer arithmetic, logical operations, and logical shifts. Absolute value conversions, floating point to integer conversions, and integer to floating point conversions are 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 1).

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.

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.

Data Flow

Data enters the 'ACT8847 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 3). 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 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. The 64-bit C register is provided for temporary storage of a result from the ALU or multiplier.

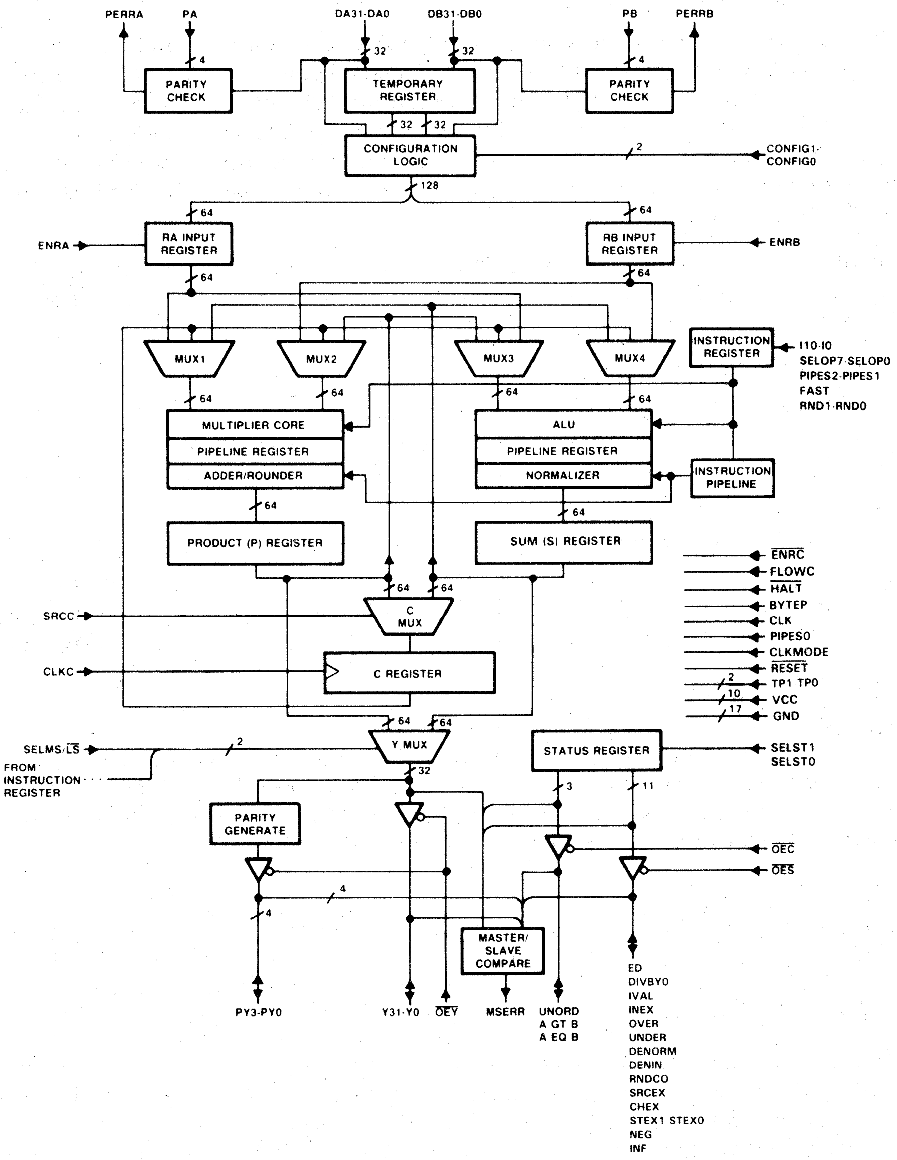


Figure 1. 'ACT8847 Floating Point Unit

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. PA0 and PB0 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).

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 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.

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

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

[†] On the first active clock edge (see CLKMODE, Table 62), 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

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

Table 5. Double Precision Input Data Register Sources

		RA SOURCE		RB SOURCE	
CONFIG1	CONFIG0	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

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-SELOP0 as shown in Tables 6 and 7. For division and square root operations, operands must be sourced from the input registers, RA and RB.

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

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

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

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

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. 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 Tables 8 through 11. Parallel ALU/multiplier functions are listed in Table 16.

Table 8. Independent ALU Operations, Single Floating-Point Operand
(I10 = 0,I9 = 0, I7 = 0, I6 =0)

CHAINED OPERATION I10	OPERAND FORMAT I9	PRECISION RA I8	PRECISON RB I7	OUTPUT SOURCE I6	OPERAND TYPE I5	ABSOLUTE VALUE A I4	ALU OPERATION	
							I3-I0	RESULT
0 = Not Chained	0 = Floating point	0 = A(SP) 1 = A(DP)	0 = B(SP)	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 status 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.

[‡] 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 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)	0 = B(SP)	0 = ALU result	0 = Two operands	0 = A	0 = B	0 = Y	00	A + B
		1 = A(DP)	1 = B(DP)			1 = |A|	1 = |B|	1 = |Y|	01	A − B
									10	Compare A, B
									11	B − A

Table 10. Independent ALU Operations, Single Integer Operand (I10 = 0, I9 = 1, I6 = 0)

CHAINED OPERATION I10	OPERAND FORMAT/PRECISION			OUTPUT SOURCE I6	OPERAND TYPE I5	ALU OPERATION	
	I9	I8	I7			I4-I0	RESULT
0 = Not Chained	1 = Integer	0	0 = SP 2's complement	0 = ALU result	1 = Single Operands	00000	Pass A operand
						00001	Pass −A operand
		0	1 = SP unsigned integer			00010	Negate A operand (1's complement)
						00101	Pass B operand
						01000	Shift A operand left logical[†]
						01001	Shift A operand right logical[†]
						01101	Shift A operand right arithmetic[†]

[†] B operand is number of bit positions A is to be shifted (See instruction description for "Independent ALU Operations".)

Table 11. Independent ALU Operations, Two Integer Operands
(I10 = 0, I9 = 1, I6 = 0)

CHAINED OPERATION	OPERAND FORMAT/PRECISION			OUTPUT SOURCE	OPERAND TYPE	ALU OPERATION	
I10	I9	I8	I7	I6	I5	I4-I0	RESULT
0 = Not Chained	1 = Integer	0	0 = SP 2's complement	0 = ALU result	0 = Two Operands	00000	A + B
						00001	A − B
		0	1 = SP unsigned integer			00010	Compare A, B
						00011	B − A
						01000	Logical AND (A, B)
						01001	Logical AND (A, NOT B)
						01010	Logical AND (NOT A, B)
						01100	Logical OR (A, B)
						01101	Logical XOR (A, B)

Table 12. Independent Multiplier Operations

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

---

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

Pipelined Multiplier

The pipelined multiplier performs a basic multiply function, division and square root. The operands can be single-precision or double-precision numbers and can be converted to absolute values before multiplication takes place. Independent multiplier operations are summarized in Tables 12 through 15.

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. Possible operations in this chained mode are listed in Table 16. Division and square root are performed as independent multiplier operations, even though both multiplier and ALU are active during divide and SQRT operations.

Table 13. 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|)

Table 14. 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).

Table 15. Formats Selected by I8-I7 (I10 = 0, 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

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 opration 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

Odd 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.

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 $\overline{\text{OEY}}$, $\overline{\text{OES}}$ and $\overline{\text{OEC}}$ 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 Generation

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

Table 16. Chained Multiplier/ALU Operations (I10 = 1)

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

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

are valid. 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 17 and 18.

Table 17. 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.

Table 18. 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.
DIVBY0	An invalid operation involving a zero divisor has been detected by the multiplier.
ED	Exception detect status signal representing logical OR of all enabled exceptions in the exception disable register.
INEX	The result of an operation is not exact.
INF	The output is the IEEE representation of infinity.
IVAL	A NaN has been input to the multiplier or the ALU, or an invalid operation ($0 \cdot \infty$ or $\pm \infty \pm \infty$) 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 wrapped number has been increased in magnitude by rounding and 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.

An exception mask register is available to mask 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. Individual status outputs remain active and can be read independently from mask register operations.

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.

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 19.

Table 19. 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)

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 20).

Table 20. 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. Denormal numbers must be wrapped before being input to the multiplier. If the multiplier result underflows, a wrapped number will be output.

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 21.

Table 21. 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-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 22).

Table 22. 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

Summary of Control Inputs

Control input signals for the 'ACT8847 are summarized in Table 23.

Table 23. 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) on rising edge	No effect
CLKC	Clocks C register on rising edge	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 42 (RA and RB register data source selects)
ENC_B	No effect	Enables C register load when CLKC goes high.
ENRA	If register is not in flow through, enables clocking of RA register	If register is not in flow through, through, holds contents of RA register
ENRB	If register is not in flow through, enables 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
FLOW_C	Causes output value to bypass C register and appear on C register output bus.	No effect
$\overline{\text{HALT}}$	No effect	Stalls device operation but does not affect registers, internal states, or status
$\overline{\text{OEC}}$	Disables compare pins	Enables compare pins
$\overline{\text{OES}}$	Disables status outputs	Enables status outputs
$\overline{\text{OEY}}$	Disables Y bus	Enables Y bus
PIPES2-PIPES0	See Table 20 (Pipeline Mode Control)	See Table 20 (Pipeline Mode Control)
$\overline{\text{RESET}}$	No effect	Clears internal states, status, internal pipeline registers, and exception disable register. Does not affect other data registers.
RND1-RND0	See Table 21 (Rounding Mode Control)	See Table 21 (Rounding Mode Control)
SELOP7-SELOP0	See Tables 6 and 7 (Multiplier/ALU operand selection)	See Tables 6 and 7 (Multiplier/ALU operand selection
SELMS/$\overline{\text{LS}}$	Selects MSH of 64-bit result for output output on the Y bus (no effect on single-precision operands)	Selects LSH of 64-bit result for output on the Y bus (no effect on single-precision operands)
SELST1-SELST0	See Table 19 (Status Output Selection)	See Table 19 (Status Output Selection)
SRCC	Selects multiplier result for input to C register	Selects ALU result for input to C register
TP1-TP0	See Table 22 (Test Pin Control Inputs)	See Table 22 (Test Pin Control Inputs)

Instruction Set

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. The ALU and the multiplier of the 'ACT8847 can operate either independently or simultaneously, depending on the setting of instruction inputs I10-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 27 through 35, 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 '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.

Configuration Controls (CONFIG1-CONFIG0)

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.

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.

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 'ACT8847 is operated in clock mode 1.

Internal Register Operations

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.

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 I10-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), input source select (SRCC) and write enable $\overline{\text{ENRC}}$ (active low). 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. When the flowthrough control FLOWC is high, the output value bypasses the C register without affecting C register contents. Timing for FLOWC feedback is similar to P or S register feedback, which is not affected by FLOWC feedback.

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 20 shows the settings of the registers controlled by PIPES2-PIPES0. Operating modes range from fully pipelined (PIPES2-PIPES0 = 000) to flowthrough (PIPES2-PIPES0 = 111).

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 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 open clock cycle must intervene between the dissimilar operations.

C Register Controls (SRCC, CLKC, FLOWC, $\overline{\text{ENRC}}$)

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 when $\overline{\text{ENRC}}$ is low.

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.

Separate controls are available to enable the C register (ENRC) or to bypass the C register when feeding an operand back on the C register output bus (FLOWC).

Operand Selection (SELOP7-SELOPO)

As shown in Tables 6 and 7, data operands can be selected 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. During division or square root operations, operands may not be selected except from the RA and RB input registers (SELOPS7-SELOPSO = 11111111).

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 'ACT8847 provides four rounding modes selected by RND1-RNDO.

Table 21 shows the four selectable rounding modes. The usual default rounding mode is round to nearest (RND1-RNDO = LL). In round-to-nearest mode, the 'ACT8847 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 flags are provided to signal both floating point and integer exceptions. Integer status is provided using AEQB for zero (Z), 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.

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 24). 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.

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, 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 = 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, INF, NEG, 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 be 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. 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

Table 24. IEEE Floating Point Representations

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

† = sign bit.

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 .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 = 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 25 describes the handling of wrapped multiplier results and the status flags that are set when wrapped numbers are output from the multiplier.

Table 25. 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	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.

Exception Disable Mask Register

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

If the FPU is reset ($\overline{\text{RESET}}$ = 0), the exception disable mask register is cleared. Table 26 describes the settings for the mask register load instruction and the status exceptions which can be enabled or disabled with the mask.

Table 26. 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

Data Output Controls (SELMS/$\overline{\mathrm{LS}}$, $\overline{\mathrm{OEY}}$)

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/$\overline{\mathrm{LS}}$ signal in the middle of the clock period.

When a single-precision result is output, the SELMS/$\overline{\mathrm{LS}}$ signal has no effect. The SELMS/$\overline{\mathrm{LS}}$ 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 $\overline{\mathrm{OEY}}$ enable must be pulled low at the beginning of that clock cycle.

Status Output Controls (SELST1-SELST0,$\overline{\mathrm{OES}}$, $\overline{\mathrm{OEC}}$)

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 19, 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, to read the status outputs, the output enables ($\overline{\mathrm{OES}}$, $\overline{\mathrm{OEC}}$, or both) must be pulled low.

Stalling the Device ($\overline{\mathrm{HALT}}$)

Operation of the 'ACT8847 can be stalled nondestructively by means of the $\overline{\mathrm{HALT}}$ signal. Pulling the $\overline{\mathrm{HALT}}$ 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 $\overline{\mathrm{HALT}}$ signal is set high. Using $\overline{\mathrm{HALT}}$ 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 $\overline{\text{HALT}}$ input low may interrupt loading of the RA, RB, and instruction registers, as well as stalling operation. In clock mode 1, the temporary register loads on the falling edge of the clock, but the $\overline{\text{HALT}}$ signal going low would prevent the RA, RB, and instruction registers from loading on the next rising clock edge. It is therefore necessary to have the instruction and data inputs on the pins when the $\overline{\text{HALT}}$ signal is set high again and normal operation resumes.

Instruction Inputs (I10-I0)

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 operating mode is treated separately in the following sections.

In addition to the ALU and multiplier instructions described below, a NOP (no operation) instruction is provided, for example, to retain a double-precision result on the Y output bus for an additional cycle:

NOP I10-I0 = 011 0000 0000

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. Tables 27 and 29 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. Wrapping and unwrapping of operands is also done in this mode.

Logical shifts can be performed on integer operands using the instructions shown in Table 68. The data operand to be shifted is input on the DA bus, and the number of bit positions the operand is to be shifted is input on 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. Neither the data operand nor the shift amount can be selected from sources other than the RA and RB registers, respectively.

Tables 28 and 29 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.

Table 27. Independent ALU Operations with One Floating Point Operand

ALU OPERATION ON A OPERAND	INSTRUCTION INPUTS I10-I0	NOTES
Pass A operand	00x 001x 0000	x = Don't care I8 selects precision of A operand 0 = A (SP) 1 = A (DP) I4 selects absolute value of a operand: 0 = A 1 = |A| During integer to floating point conversion, |A| is not allowed as a result.
Pass −A operand	00x 001x 0001	
Convert from 2's complement integer to floating point[†]	00x 0010 0010	
Convert from floating point to 2's complement integer	00x 001x 0011	
Move A operand (pass without NaN detect or status flags active)	00x 001x 0100	
Pass B operand	00x 001x 0101	
Convert from floating point to floating point (adjusts precision of input: SP → DP, DP → SP)	00x 001x 0110 00x 001x 0111	
Floating point to unsigned integer conversion		
Wrap denormal operand	00x 001x 1000	
Unsigned integer to floating point conversion	00x 001x 1010	
Unwrap exact number	00x 001x 1100	
Unwrap inexact number	00x 001x 1101	
Unwrap rounded input	00x 001x 1110	

[†] During this operation, I8 selects the precision of the result.

Table 28. Independent ALU Operations with Two Floating Point Operands

ALU OPERATIONS AND OPERANDS	INSTRUCTION INPUTS I10-I0	NOTES
Add A + B	00x x000 0x00	x = Don't Care I8 selects precision of A operand: 0 = A (SP) 1 = A (DP) I7 selects precision of B operand: 0 = B (SP) 1 = B (DP) I2 selects either Y or its absolute value: 0 = Y 1 = |Y|
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 29. Independent ALU Operations with One Integer Operand

ALU OPERATION ON A OPERAND	INSTRUCTION INPUTS I10-I0	NOTES
Pass A operand	010 x010 0000	x = Don't Care I8 selects format of A integer operand: 0 = Single-precision 2's complement 1 = Single-precision unsigned integer
Pass −A operand (2's complement)	010 x010 0001	
Negate A operand (1's complement)	010 x010 0010	
Pass B operand	010 x010 0101	
Shift left logical	010 x010 1000	
Shift right logical	010 x010 1001	
Shift right arithmetic	010 x010 1101	

Table 30. Independent ALU Operations with Two Integer Operands

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

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. Separate instruction tables are provided for floating point operations and integer operations.

Floating point operands may be normalized or wrapped numbers, as indicated by the settings for instruction inputs I1-I0. As shown in Table 31, 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.

Table 31. Independent Floating Point Multiply Operations

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

Table 32. Independent Floating-Point Divide/Square Root Operations

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

[†] I7 should be low or equal to I8 for square root operations

Table 33. Independent Integer Multiply/Divide/Square Root Operations

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

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

Chained Multiplier/ALU Operations

In chained mode the 'ACT8847 performs simultaneous operations in the multiplier and the ALU. Operations include not only addition, subtraction, and multiplication, but also several optional operations which increase the flexibility of the device. Division and square root operations are not available in chained mode.

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.

Table 34. 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	x = Don't Care I8 selects precision of RA inputs: 0 = RA (SP) 1 = RA (DP) I7 selects precision of RB inputs: 0 = RB (SP) 1 = RB (DP) I3 negates ALU result: 0 = Normal 1 = Negated I2 negates multiplier result: 0 = Normal 1 = Negated
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	
A * B	2 − A	Multiplier	10x x100 xx10	
A * B	B − A	ALU	10x x000 xx11	
A * B	B − A	Multiplier	10x x100 xx11	
A * B	A + 0	ALU	10x x010 xx00	
A * B	A + 0	Multiplier	10x x110 xx00	
A * B	0 − A	ALU	10x x010 xx11	
A * B	0 − A	Multiplier	10x x110 xx11	
A * 1	A + B	ALU	10x x001 xx00	
A * 1	A + B	Multiplier	10x x101 xx00	
A * 1	A − B	ALU	10x x001 xx01	
A * 1	A − B	Multiplier	10x x101 xx01	
A * 1	2 − A	ALU	10x x001 xx10	
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 35. Chained Multiplier/ALU Integer Operations

CHAINED OPERATIONS		OUTPUT SOURCE	INSTRUCTION INPUTS I10-I0	NOTES
MULTIPLIER	ALU			
A * B	A + B	ALU	110 x000 0000	x = Don't Care I7 selects format of A and B operands: 0 = SP 2's complement 1 = SP unsigned integer
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	

MICROPROGRAMMING THE 'ACT8847

Because the 'ACT8847 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.

Division and square root operations are presented after the discussion of ALU and multiplier operations. 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.

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

```
                                      S
                                      E
      C C C                           L
      L O O P P   S S                 M         S S
      K N N I I   E E                 S       B E E R̄
      M F F P P   L L   R R   F E E S /       Y L L E H̄
      O I I E E   O O   N N   A N N R   Ō Ō Ō T S S S A T T
I I   D G G S S   P P   D D   S R R C L̄ E E E E T T E L P P
0-0   E 1-0 2-0   7-0   1-0   T A B C S Y C S P 1-0 T T 1-0

000 0010 1100 0 01 111 xxxx 11xx 00 0 1 1 0 1 0 0 0 x 11 1 1 11
```

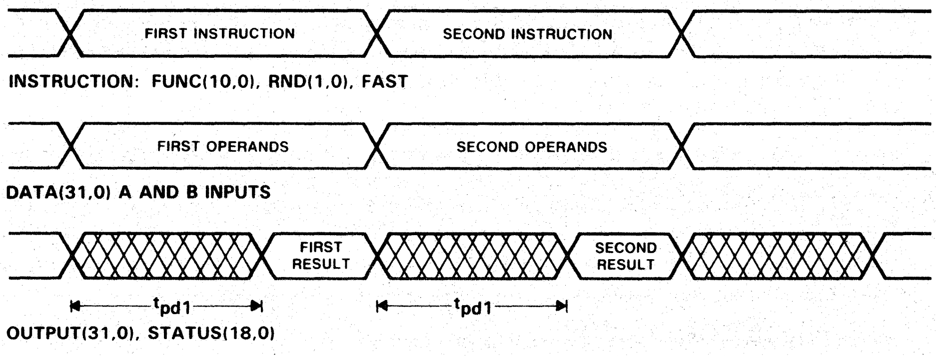


Figure 2. Single-Precision Operation, All Registers Disabled (PIPES = 111, CLKMODE = 0)

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|

I I 0-0	CLKMODE	CONFIG1-0	PIPES2-0	SELOP7-0	RND1-0	FAST	ENRA	ENRB	SRCC	SELMS/$\overline{\text{LS}}$	$\overline{\text{OE}}$Y	$\overline{\text{OE}}$C	$\overline{\text{OE}}$S	BYTEP	SELST1-0	$\overline{\text{R}}$ESET	$\overline{\text{H}}$ALT	TP1-0
000 0001 1010	0	01	110	xxxx 1111	00	0	1	1	0	1	0	0	0	x	11	1	1	11

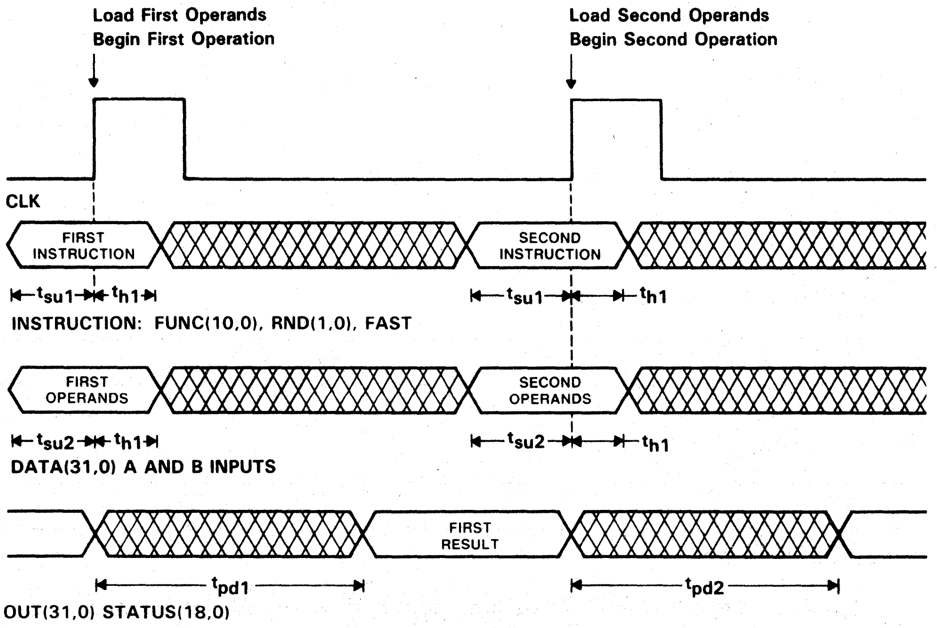


Figure 3. Single-Precision Operation, Input Registers Enabled (PIPES = 110, CLKMODE = 0)

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

I10-0	CLKMODE	CONFIG1-0	PIPES2-0	SELOP7-0	RND1-0	FAST	ENRA	ENRB	SRCC	SELMS/$\overline{\text{LS}}$	$\overline{\text{OE}}$Y	$\overline{\text{OE}}$C	$\overline{\text{OE}}$S	BYTEP	SELST1-0	$\overline{\text{R}}$ESET	$\overline{\text{H}}$ALT	TP1-0
000 0000 0011	0	01	010	xxxx 1111	00	0	0	0	0	1	0	0	0	x	11	1	1	11

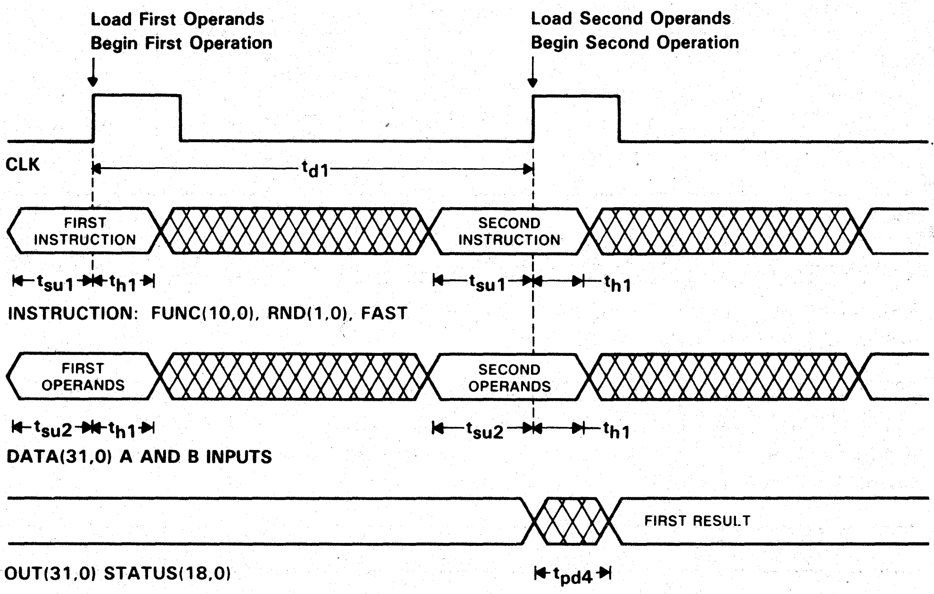


Figure 4. Single-Precision Operation, Input and Output Registers Enabled (PIPES = 010, CLKMODE = 0)

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

II 0-0	CLKMODE	CONFIG 1-0	PIPES 2-0	SELOP 7-0	RND 1-0	FAST	ENRA	ENRB	SRCC	SELMS/$\overline{LS}$	$\overline{OEY}$	$\overline{OEC}$	$\overline{OES}$	BYTEP	SELST 1-0	$\overline{RESET}$	$\overline{HALT}$	TP 1-0
000 0100 0000	0	01	000	1111 xxxx	00	0	1	1	1	1	0	0	0	x	11	1	1	11

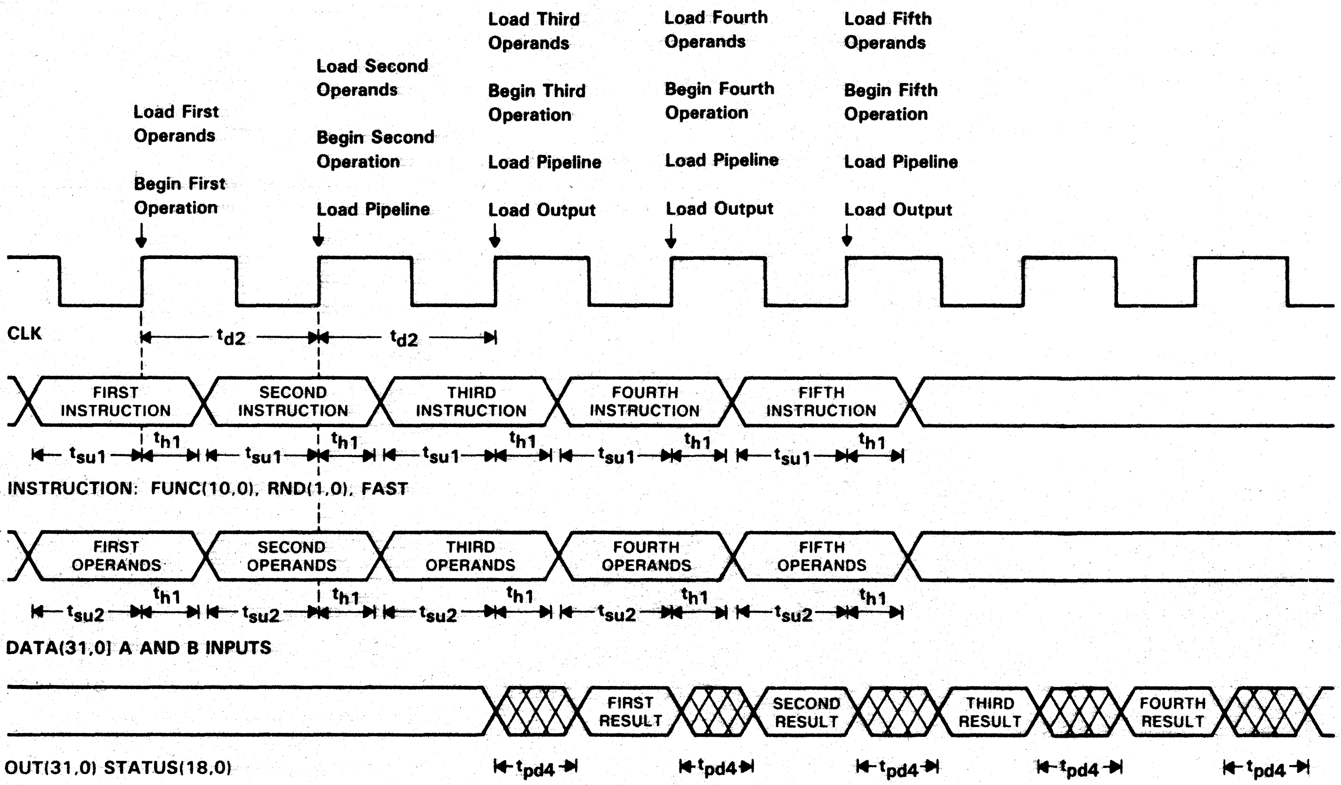


Figure 5. Single-Precision Operation, All Registers Enabled
(PIPES = 000, CLKMODE = 0)

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).

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|

I10-0	CLKMODE	CONFIG1-0	PIPES2-0	SELOP7-0	RND1-0	FAST	ENRA	ENRB	SRCC	SELMS/$\overline{\text{LS}}$	$\overline{\text{OEY}}$	$\overline{\text{OEC}}$	$\overline{\text{OES}}$	BYTEP	SELST1-0	$\overline{\text{RESET}}$	$\overline{\text{HALT}}$	TP1-0
001 1000 1000	0	11	111	xxxx 1111	00	0	1	1	0	x	0	0	0	x	11	1	1	11

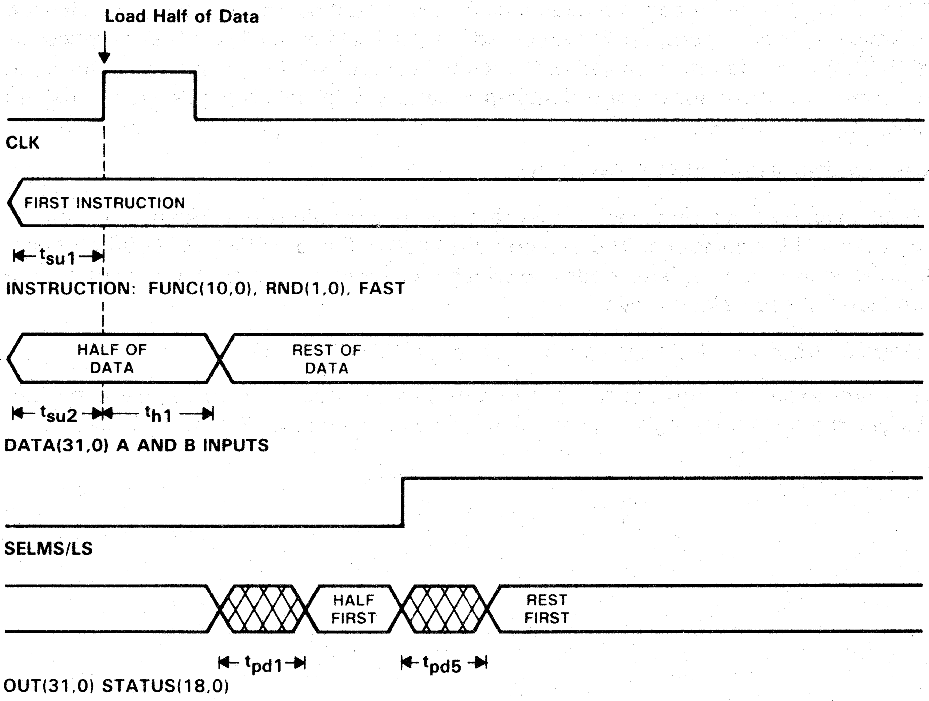


Figure 6. Double-Precision ALU Operation, All Registers Disabled (PIPES = 111, CLKMODE = 0)

In the second example the input register is enabled (PIPES2-PIPES0 = 110). Operands A and B for the instruction, |B| − |A|, are loaded using 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).

CLKMODE = 0   PIPES = 110   Operation: |B| − |A|

```
                                                S
                                                E
               C C C                            L
               L O O P P    S S                 M         S S
               K N N I I    E E                 S       B E E R̄
               M F F P P    L L     R R F E E S /       Y L L E H̄
               O I I E E    O O     N N A N N R   Ō Ō Ō T S S S A T T
      I I      D G G S S    P P     D D S R R C L̄ E E E E T T E L P P
     0-0       E 1-0 2-0    7-0     1-0 T A B C S Y C S P 1-0 T T 1-0

001 1001 1011  0 00 110 xxxx 1111  00   0 1 1 0 x 0 0 0 x 11  1 1 11
```

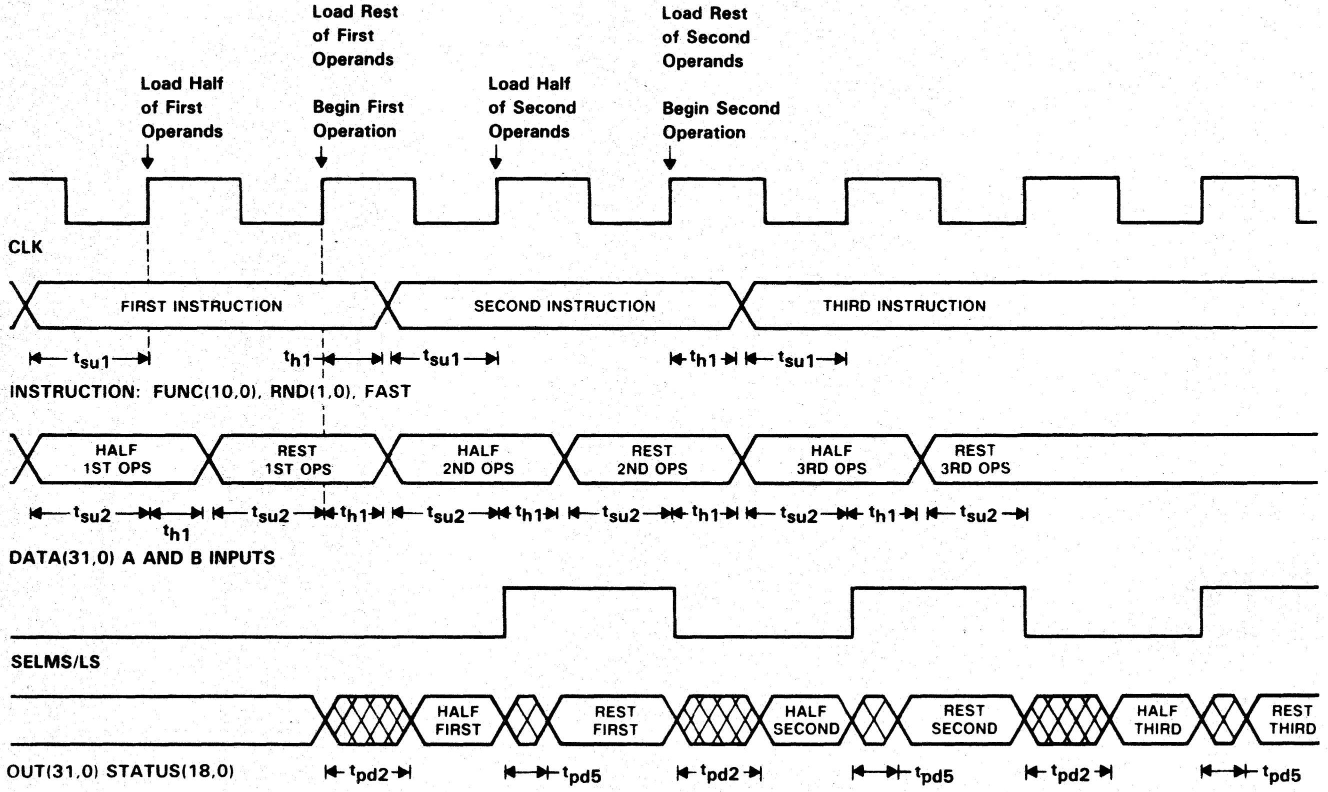


Figure 7. Double-Precision ALU Operation, Input Registers Enabled (PIPES = 110, CLKMODE = 0)

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   Operaion: Wrap Denormal Input

```
                                            S
                                            E
              C C C                         L
              L O O P P   S S               M         S S
              K N N I I   E E               S       B E E R̄
              M F F P P   L L   R R F E E S /       Y L L E H̄
              O I I E E   O O   N N A N N R   Ō Ō Ō T S S S A T T
      I I     D G G S S   P P   D D S R R C L̄ E E E E T T E L P P
      0-0     E 1-0 2-0   7-0   1-0 T A B C S Y C S P 1-0 T T 1-0

 001 1010 1000 0 01 010 xxxx 11xx 00  0 1 1 0 x 0 0 0 x 11  1 1 11
```

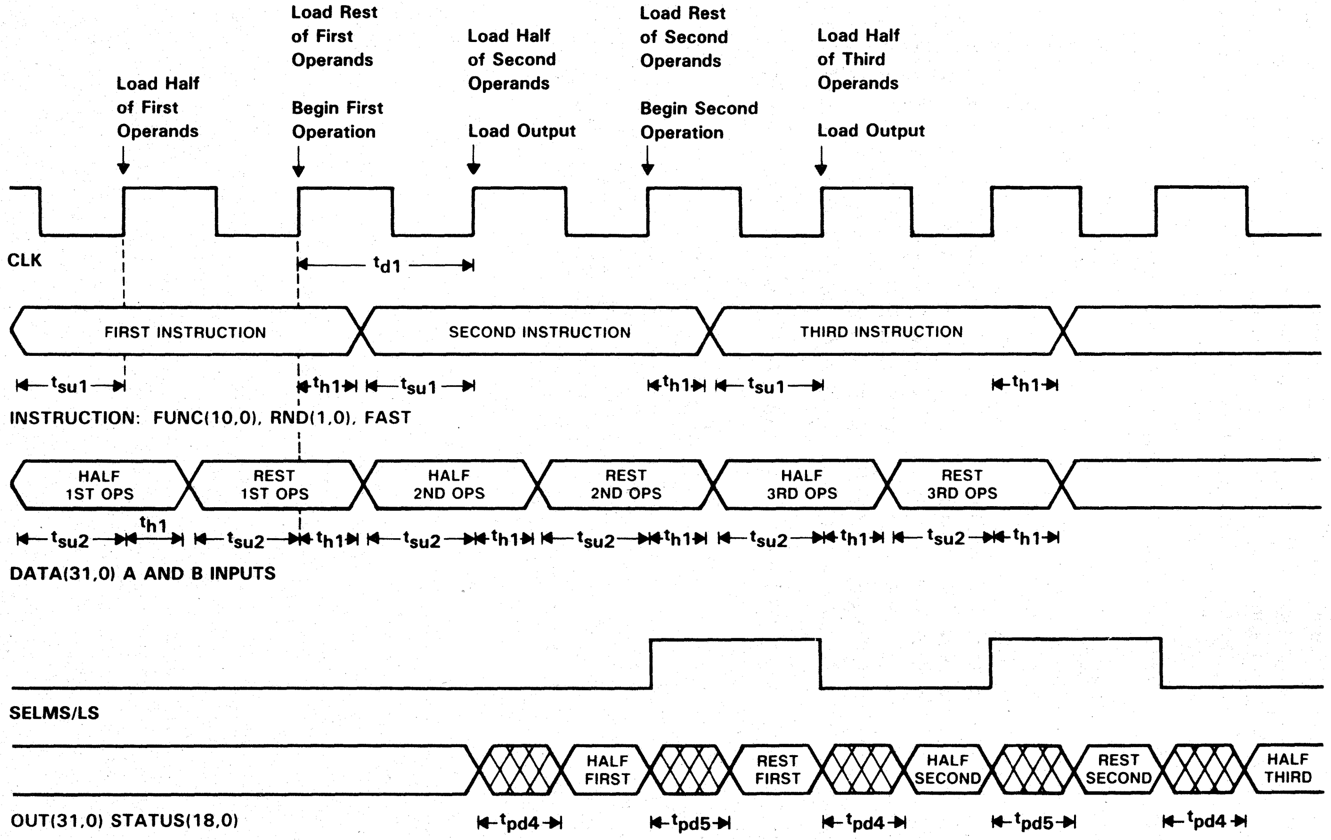


Figure 8. Double-Precision ALU Operation, Input and Output Registers Enabled (PIPES = 010, CLKMODE = 0)

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

```
                                        S
                                        E
          C C C                         L
          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 _ 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 E T T E L P P
  0-0     E 1-0 2-0   7-0   1-0 T A B C S Y C S P 1-0 T T 1-0

001 1010 0010 0 11 000 xxxx 1100 00 0 1 1 0 x 0 0 0 x 11 1 1 11
```

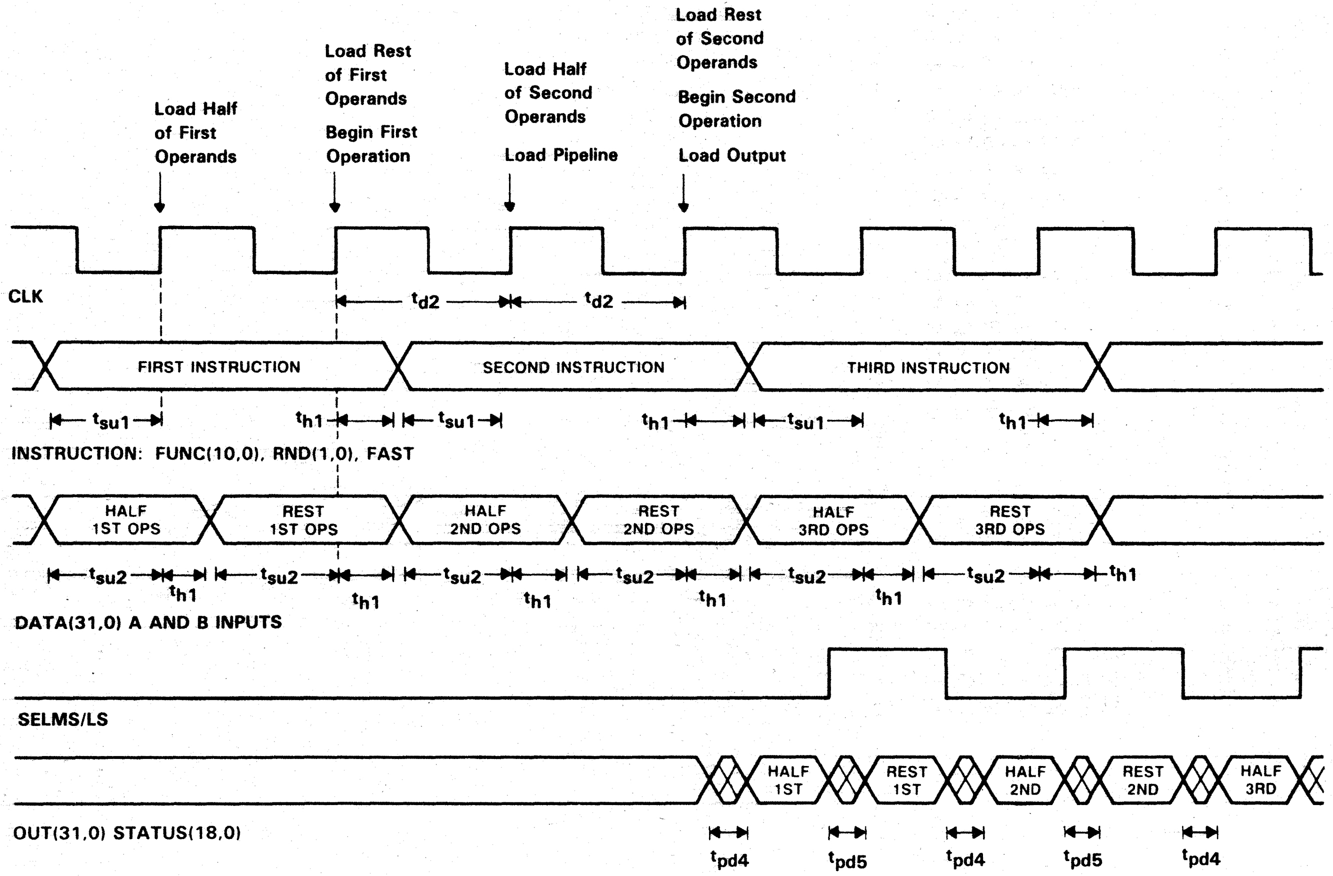


Figure 9. Double-Precision ALU Operation, All Registers Enabled
(PIPES = 000, CLKMODE = 0)

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|

I I 0–0	CLKMODE	CONFIG1–0	PIPES2–0	SELOP7–0	RND1–0	FAST	ENRA	ENRB	SRCC	SELMS/$\overline{\text{LS}}$	$\overline{\text{OEY}}$	$\overline{\text{OEC}}$	$\overline{\text{OES}}$	BYTEP	SELST1–0	$\overline{\text{RESET}}$	$\overline{\text{HALT}}$	TP1–0
001 1000 1000	1	11	111	xxxx 1111	00	0	1	1	0	x	0	0	0	x	xx	1	1	11

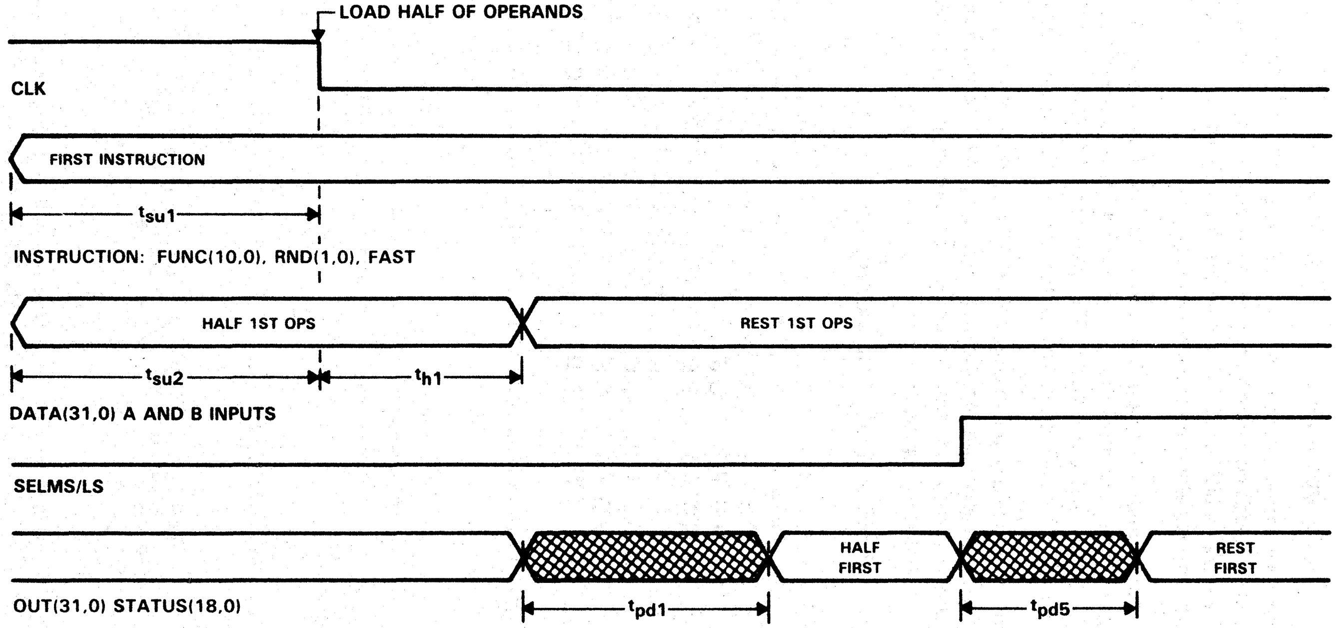


Figure 10. Double-Precision ALU Operation, All Registers Disabled
(PIPES = 111, CLKMODE = 1)

The second example executes subtraction of absolute values for both operands. Only the RA and RB registers are enabled (PIPES2-PIPES0 = 110). Timing is shown in Figure 11.

CLKMODE = 1   PIPES = 110   Operation: Subtract |B| − |A|

I10-0	CLKMODE	CONFIG1-0	PIPES2-0	SELOP7-0	RND1-0	FAST	ENRA	ENRB	SRCC	SELMS/$\overline{\text{L}}$S	$\overline{\text{O}}$EY	$\overline{\text{O}}$EC	$\overline{\text{O}}$ES	BYTEP	SELST1-0	$\overline{\text{R}}$ESET	$\overline{\text{H}}$ALT	TP1-0
001 1001 1011	1	11	110	xxxx 1111	00	0	1	1	0	x	0	0	0	x	xx	1	1	11

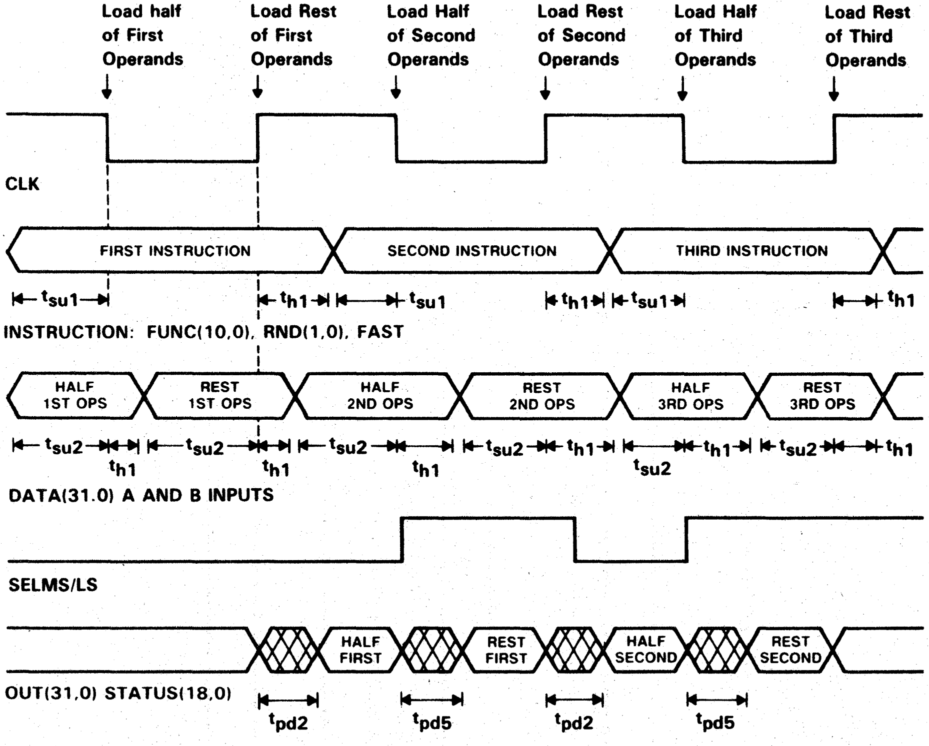


Figure 11. Double-Precision ALU Operation, Input Registers Enabled (PIPES = 110, CLKMODE = 1)

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-PIPESO = 010). Timing is shown in Figure 12.

CLKMODE = 1   PIPES = 010   Operation: Wrap Denormal Input

```
                                                  S
                                                  E
                    C C C                         L
                    L O O P P   S S               M         S S
                    K N N I I   E E               S       B E E R̅
                    M F F P P   L L   R R F E E S /       Y L L E H̅
                    O I I E E   O O   N N A N N R   O̅ O̅ O̅ T S S S A T T
          I I       D G G S S   P P   D D S R R C L̅ E E E E T T E L P P
          0-0       E 1-0 2-0   7-0   1-0 T A B C S Y C S P 1-0 T T 1-0

 001 1010 1000  1 11  010 xxxx 11xx  00  0 1 0 0 x 0 0 0 x  xx  1 1 11
```

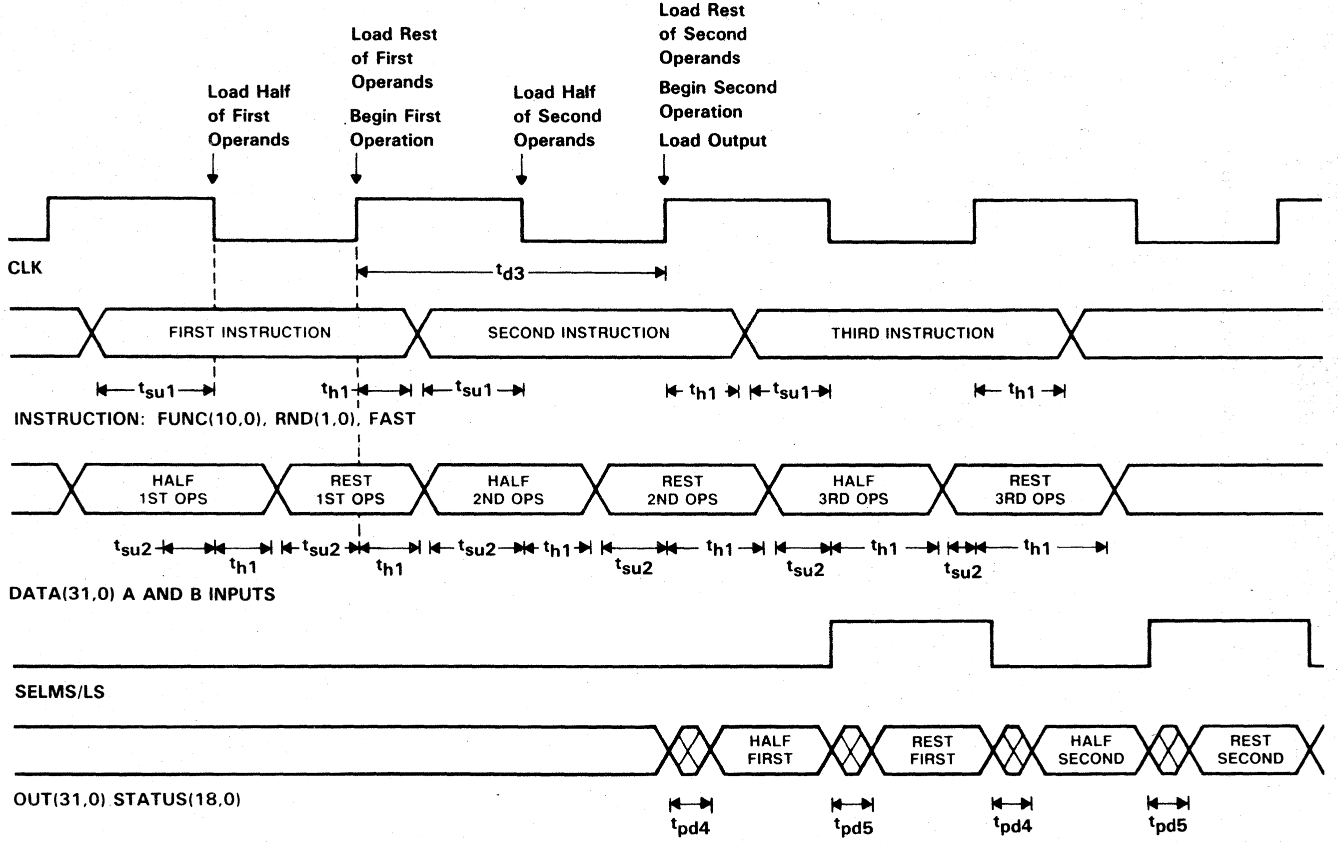


Figure 12. Double-Precision ALU Operation, Input and Output Registers Enabled (PIPES = 010, CLKMODE = 1)

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

```
                                S
                                E
    C C C                       L
    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   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 E T T E L P P
0-0 E 1-0 2-0  7-0  1-0 T A B C S Y C S P 1-0 T T 1-0

 001 1010 0010  1 11  000 xxxx 1100  00  0 1 1 x x 0 0 0 x  xx  1 1 11
```

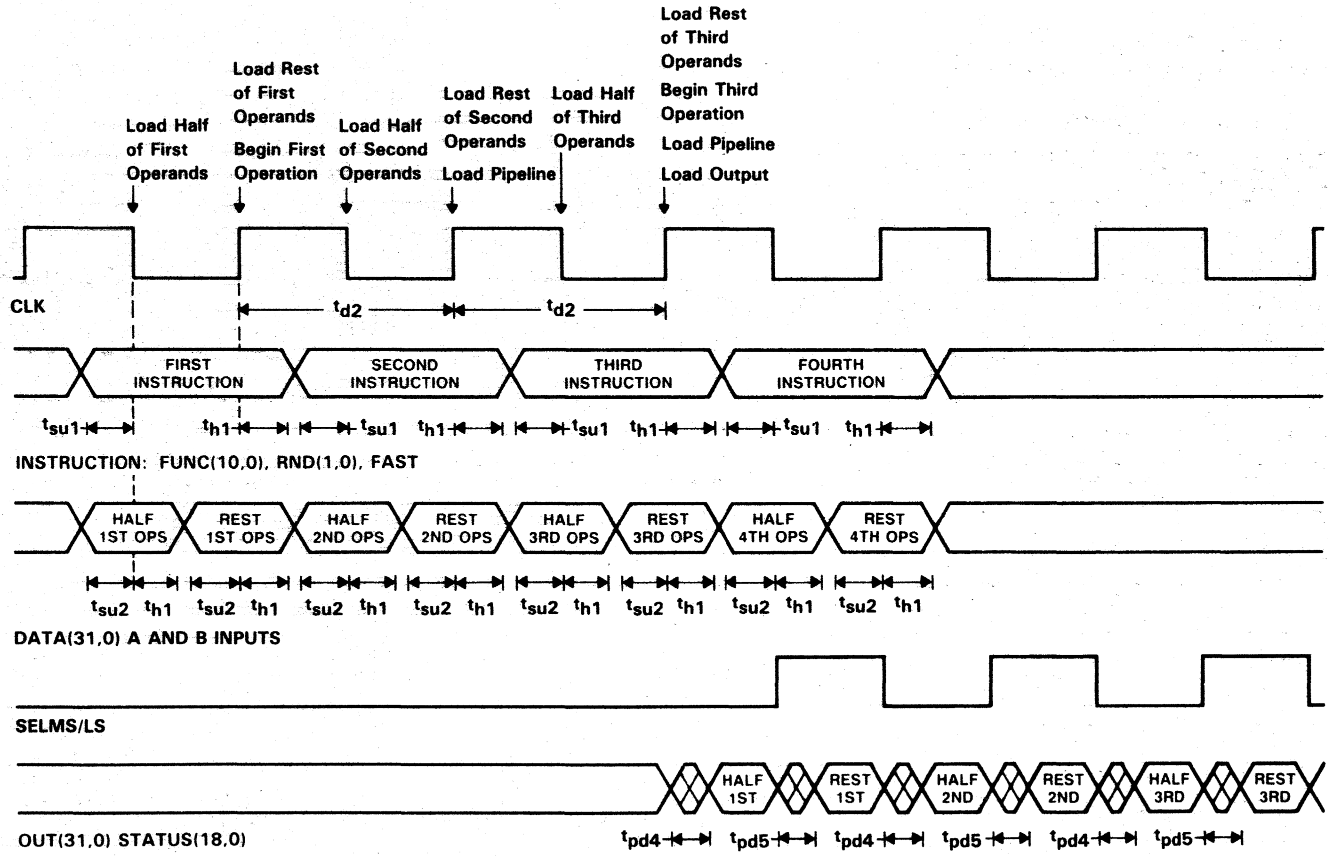


Figure 13. Double-Precision ALU Operation, All Registers Enabled
(PIPES = 000, CLKMODE = 1)

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|

```
                                            S
                                            E
            C C C                           L
            L O O P P    S S                M         S S
            K N N I I    E E                S       B E E R̅
            M F F P P    L L    R R F E E S /       Y L L E H̅
            O I I E E    O O    N N A N N R   O̅ O̅ O̅ T S S S A T T
    I I     D G G S S    P P    D D S R R C L̅ E E E E T T E L P P
    0-0     E 1-0 2-0    7-0    1-0 T A B C S Y C S P 1-0 T T 1-0

001 1100 1000  0 11  111 1111 xxxx 00  0 x x x x 0 0 0 x  xx  1 1 11
```

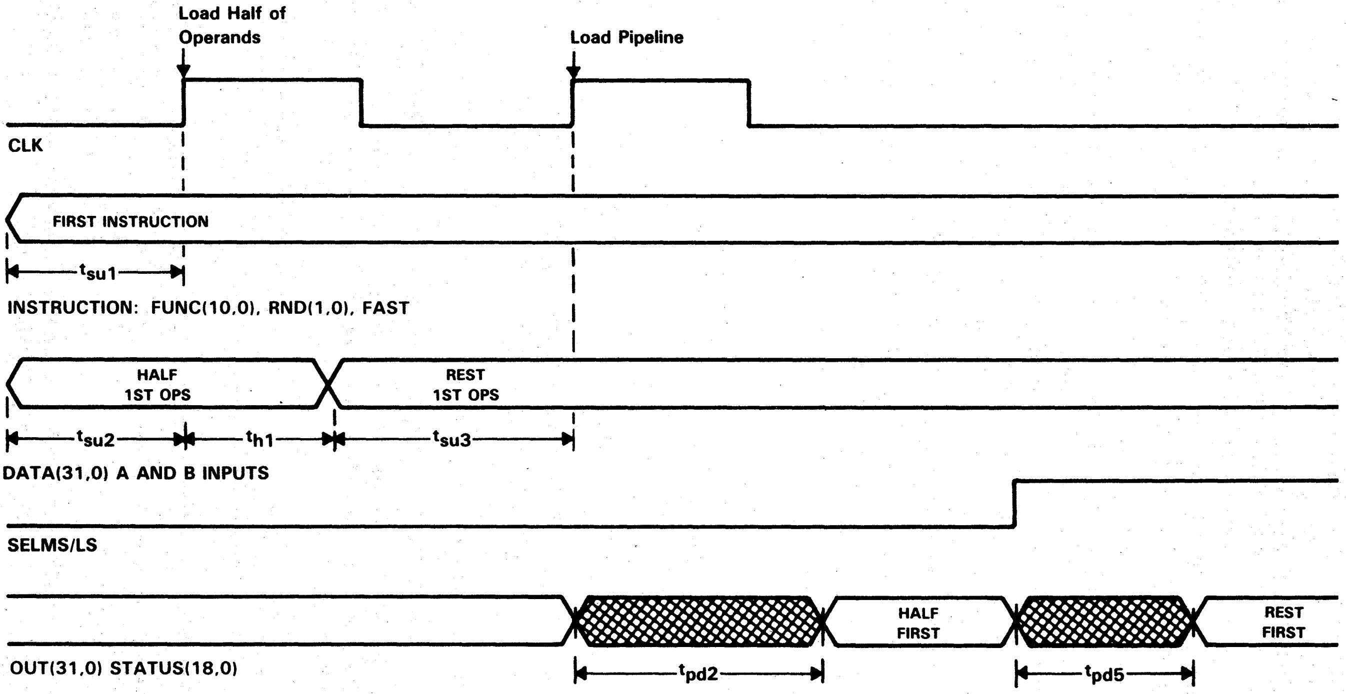


Figure 14. Double-Precision Multiplier Operation, All Registers Disabled (PIPES = 111, CLKMODE = 0)

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 $-(|A| * B)$

```
                                        S
                                        E
        C C C                           L
        L O O P P   S S                 M         S S
        K N N I I   E E                 S       B E E R̄
        M F F P P   L L   R R   F E E S /       Y L L E H̄
        O I I E E   O O   N N   A N N R   Ō Ō Ō T S S S A T T
I I     D G G S S   P P   D D   S R R C L̄ E E E E T T E L P P
0-0     E 1-0 2-0   7-0   1-0   T A B C S Y C S P 1-0 T T 1-0

001 1101 0100 0 11 110 1111 xxxx 00 0 1 1 x x 0 0 0 x xx 1 1 11
```

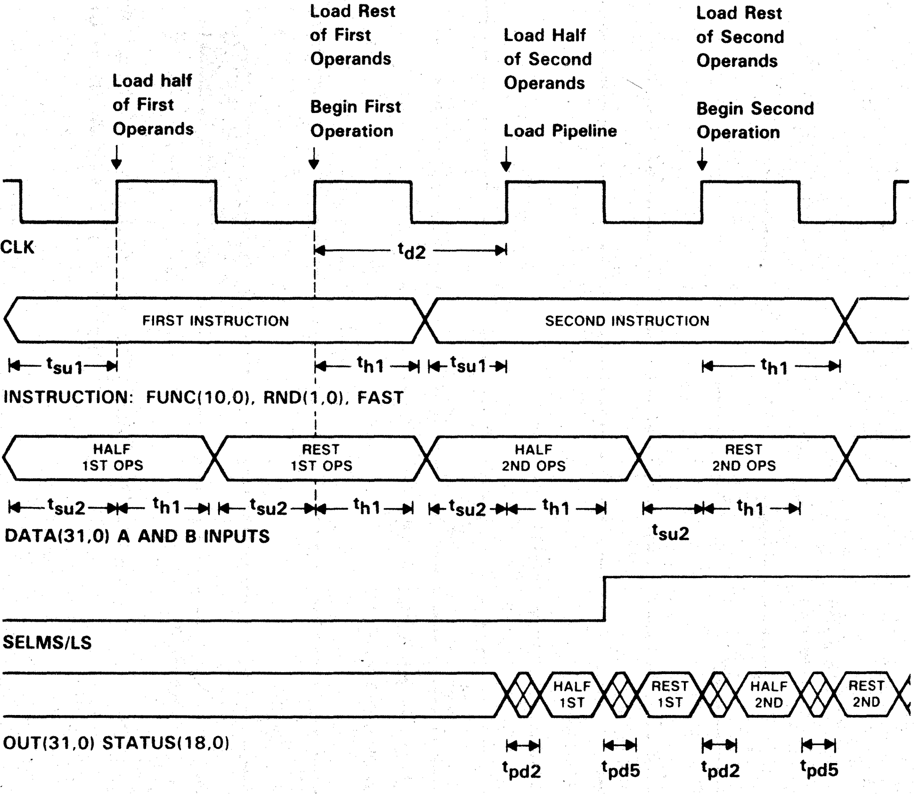


Figure 15. Double-Precision Multiplier Operation, Input Registers Enabled (PIPES = 110, CLKMODE = 0)

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|

```
                                           S
                                           E
             C C C                         L
             L O O P P   S S               M         S S
             K N N I I   E E               S       B E E R̅
             M F F P P   L L   R R F E E S /       Y L L E H̅
             O I I E E   O O   N N A N N R   O̅ O̅ O̅ T S S S A T T
   I I       D G G S S   P P   D D S R R C L̅ E E E E T T E L P P
   0-0       E 1-0 2-0   7-0   1-0 T A B C S Y C S P 1-0 T T 1-0

001 1101 1000  0 10  010 1111 xxxx 00  0 1 1 x x 0 0 0 x xx  1 1 11
```

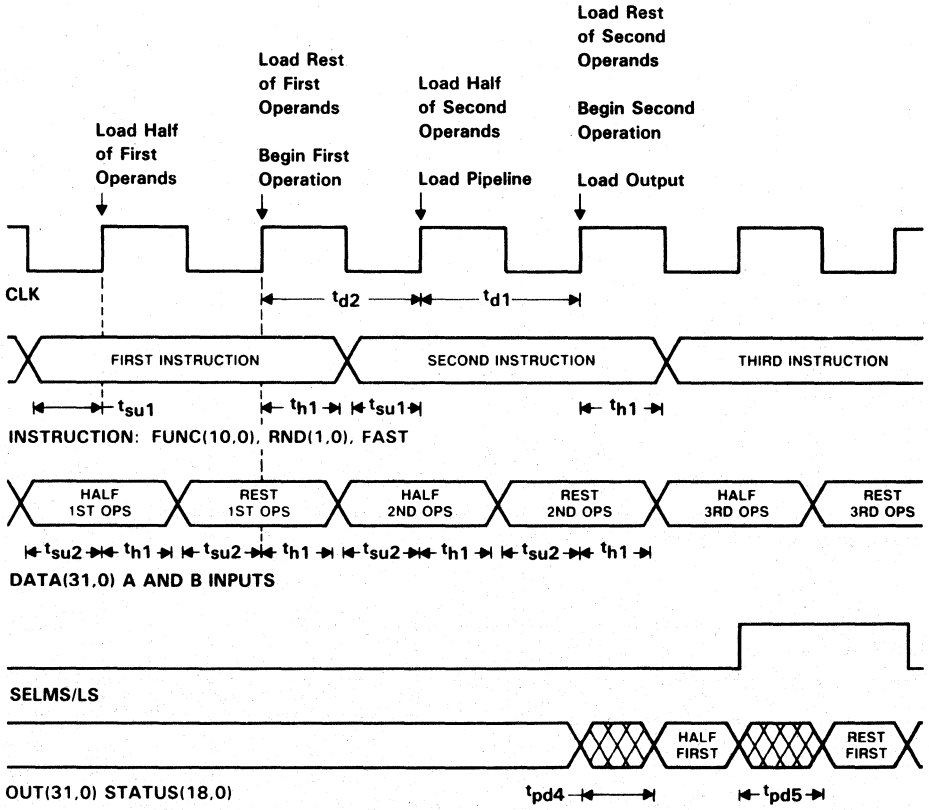


Figure 16. Double-Precision Multiplier Operation, Input and Output Registers Enabled (PIPES = 010, CLKMODE = 0)

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)

```
                                          S
                                          E
            C C C                         L
            L O O P P   S S               M         S S
            K N N I I   E E               S       B E E R̄
            M F F P P   L L   R R F E E S /       Y L L E H̄
            O I I E E   O O   N N A N N R   Ō Ō Ō T S S S A T T
    I I     D G G S S   P P   D D S R R C L̄ E E E E T T E L P P
    0-0     E 1-0 2-0   7-0   1-0 T A B C S Y C S P 1-0 T T 1-0

001 1100 0100  0 01  000 1111 xxxx 00  0 1 1 x x 0 0 0 x  xx  1 1 11
```

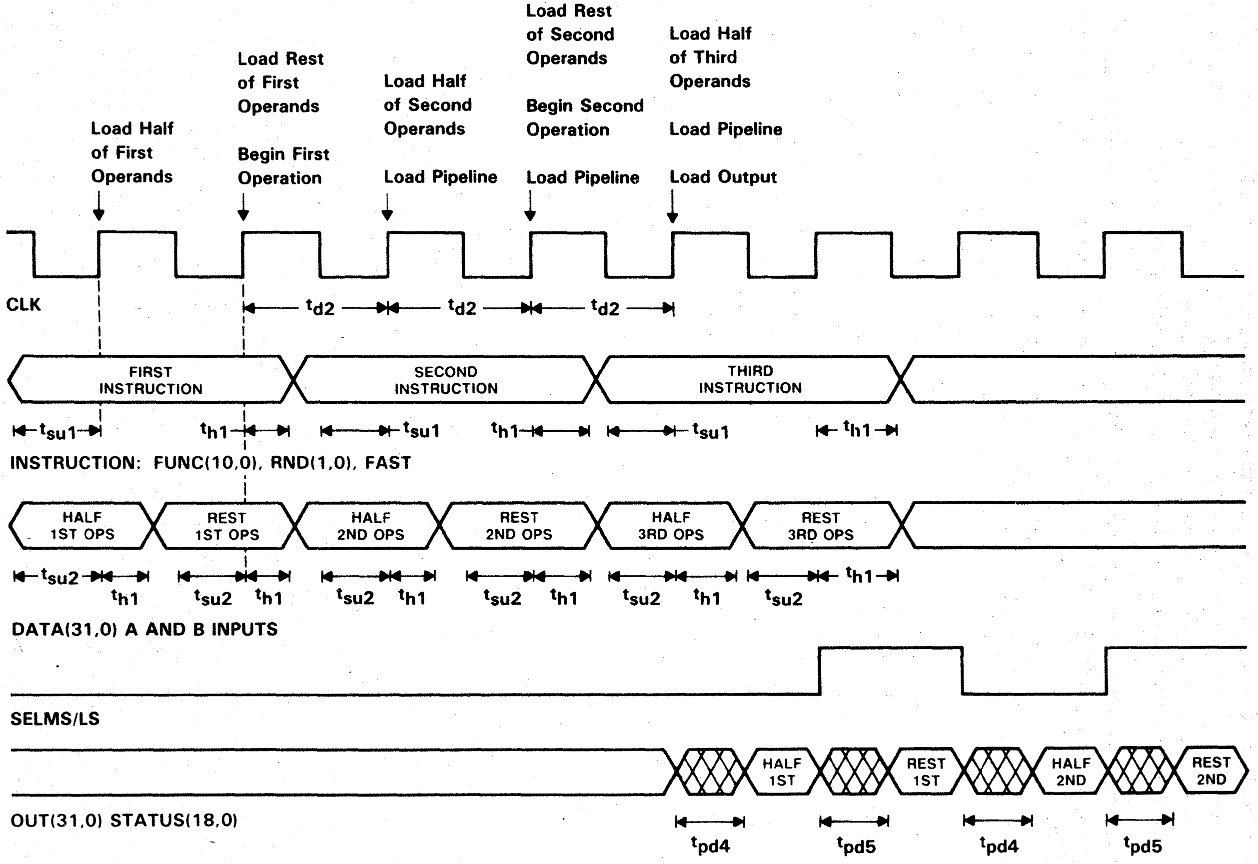


Figure 17. Double-Precision Multiplier Operation, All Registers Enabled (PIPES = 000, CLKMODE = 0)

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|

```
                                                  S
                                                  E
                    C C C                         L
                    L O O P P   S S               M         S S
                    K N N I I   E E               S       B E E R̅
                    M F F P P   L L   R R F E E S /       Y L L E H̅
                    O I I E E   O O   N N A N N R   O̅ O̅ O̅ T S S S A T T
          I I       D G G S S   P P   D D S R R C L̅ E E E E T T E L P P
          0-0       E 1-0 2-0   7-0   1-0 T A B C S Y C S P 1-0 T T 1-0

 001 1100 1000 1 11  111 1111 xxxx 00  0 x x x x 0 0 0 x  xx  1 1 11
```

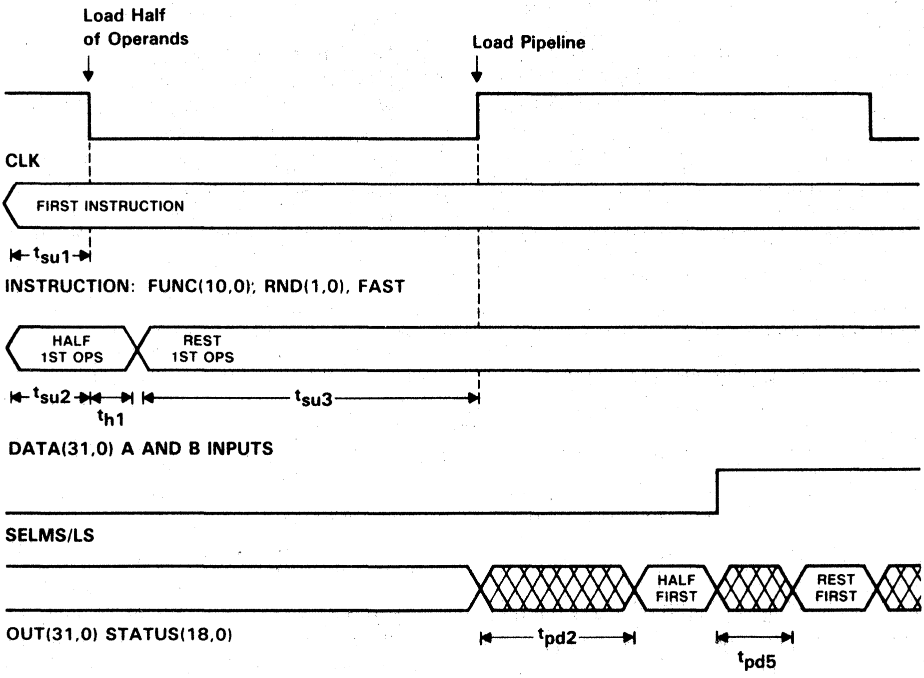


Figure 18. Double-Precision Multiplier Operation, All Registers Disabled (PIPES = 111, CLKMODE = 1)

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)

I10-0	CLKMODE	CONFIG1-0	PIPES2-0	SELOP7-0	RND1-0	FAST	ENRA	ENRB	SRCC	SELMS/$\overline{\mathrm{LS}}$	$\overline{\mathrm{OEY}}$	$\overline{\mathrm{OEC}}$	$\overline{\mathrm{OES}}$	BYTEP	SELST1-0	$\overline{\mathrm{RESET}}$	$\overline{\mathrm{HALT}}$	TP1-0
001 1101 0100	1	11	110	1111 xxxx	00	0	1	1	x	x	0	0	0	x	xx	1	1	11

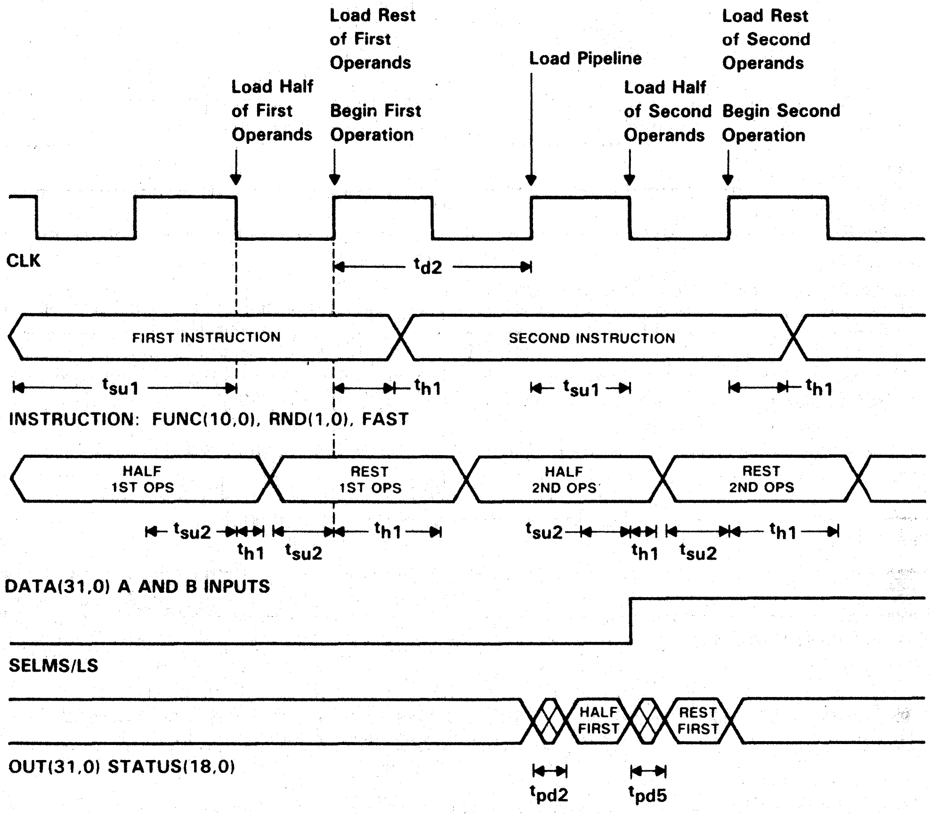


Figure 19. Double-Precision Multiplier Operation, Input Registers Enabled (PIPES = 110, CLKMODE = 1)

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|

I I 0-0	CLKMODE	CONFIG 1-0	PIPES 2-0	SELOP 7-0	RND 1-0	FAST	ENRA	ENRB	SRCC	SELMS/$\overline{\text{L}}$S	$\overline{\text{OE}}$Y	$\overline{\text{OE}}$C	$\overline{\text{OE}}$S	BYTEP	SELST 1-0	$\overline{\text{R}}$ESET	$\overline{\text{H}}$ALT	TP 1-0
001 1101 1000	1	11	010	1111 xxxx	00	0	1	1	x	x	0	0	0	x	xx	1	1	11

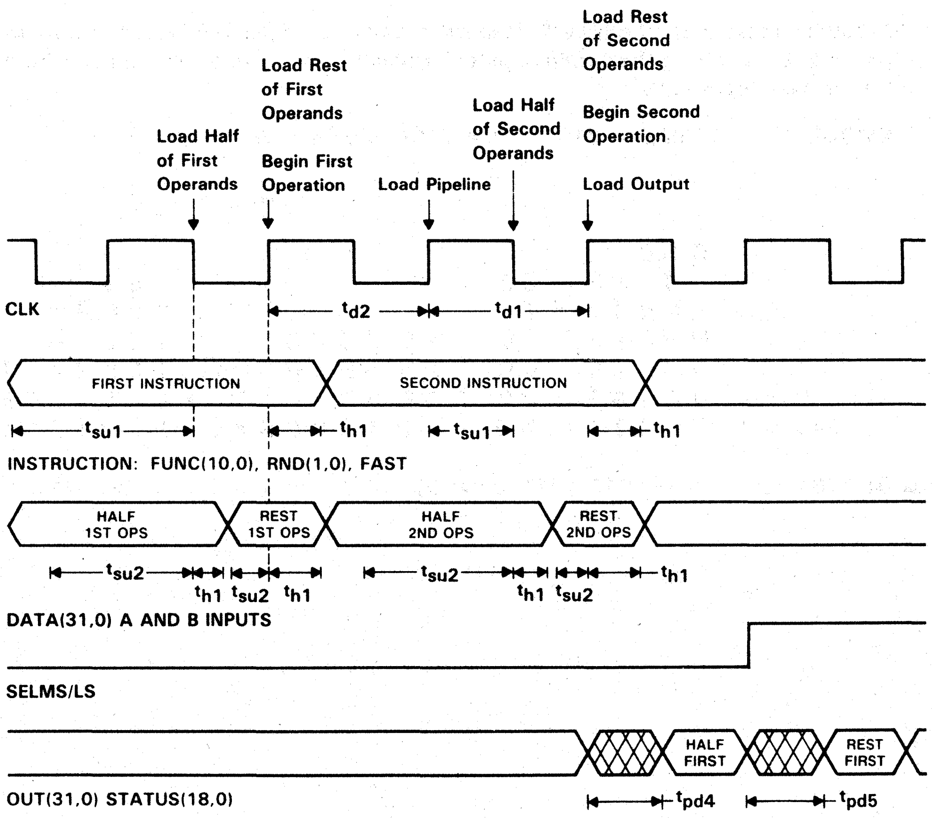


Figure 20. Double-Precision Multiplier Operation, Input and Output Registers Enabled (PIPES = 010, CLKMODE = 1)

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 section on mixed operations and operands.

CLKMODE = 1   PIPES = 000   Operation: Multiply −(A ∗ B)

```
                                      S
                                      E
      C C C                           L
      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 _ 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 E T T E L P P
0-0   E 1-0 2-0   7-0   1-0   T A B C S Y C S P 1-0 T T 1-0

001 1100 0100  1 11  000 1111 xxxx 00  0 1 1 x x 0 0 0 x  xx  1 1 11
```

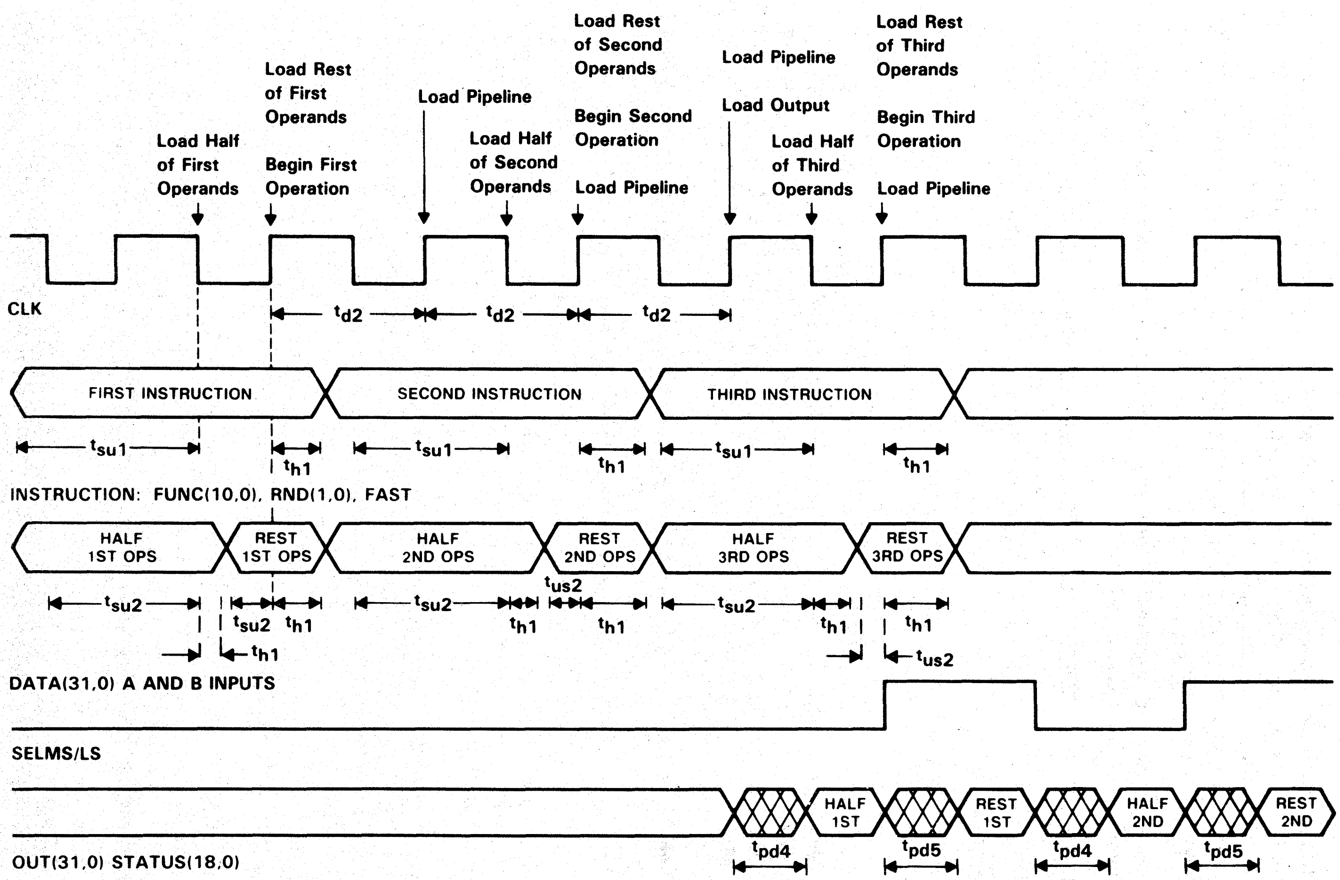


Figure 21. Double-Precision Multiplier Operation, All Registers Enabled (PIPES = 000, CLKMODE = 1)

Division and Square Root Operations

The examples presented below contain sample microinstructions and timing diagrams for all modes of division and square root. The permissible settings of internal registers (PIPES2-PIPES0) are presented separately. To perform a valid divide or square root calculation, at least the RA and RB input registers must be enabled (PIPES0 = 0).

Sample timing waveforms are provided both for division and square root, along with sample microinstructions for permissible register control settings. Results for these operations are valid for one cycle and may be written over by results of the next instruction. This may happen when a double-precision operation is followed by a single-precision operation, notably when only the input registers are enabled (PIPES2-PIPES0 = 110).

To retain a double-precision result on the Y output bus for an extra cycle, a NOP (no operation) may be coded using the following instruction:

NOP I10-I0 = 011 0000 0000

Input enables ENRA and ENRB may also be set low for NOP this instruction to prevent loading of new data. The $\overline{\text{OEY}}$ signal is set low and SELMS/$\overline{\text{LS}}$ is set to get out the second half of the double-precision result.

Division Microinstructions

Division calculations are executed as independent multiplier operations selected by instruction bits I10, I5 and I3. Independent multiplier operation is selected when I10 is set low. A divide operation requires that I5 is high and I3 is low.

In all division operations the A operand is the dividend and the B operand is the divisor. Operands may be single-precision integers, single-precision floating point numbers, or double-precision floating point numbers. The operands must be either both integer or both floating point.

Mixed-precision floating point operations are executed as double-precision operations, with the single-precision operand converted to double-precision format automatically. However, a single-precision wrapped input cannot be converted to double precision, so mixed-precision division involving a single-precision wrapped operand is not permitted.

Single-Precision Floating Point Division

The following four sample microinstructions select division operations on single-precision floating point operands. Each example includes a timing diagram for a different setting of the internal register controls (PIPES2-PIPES0). Operands and data inputs are presented at the same time in these instructions.

The first example shows an instruction to perform |A| / B with only the RA and RB input registers enabled (PIPES2-PIPES0 = 110). The output is available after the seventh rising clock edge and may remain stable for that cycle only (see Figure 22).

CLKMODE = 0 PIPES = 110 Operation: |A| / B

I10–I0	CLKMODE	CONFIG1–0	PIPES2–0	SELOP7–0	RND1–0	FAST	ENRA	ENRB	SRCC	SELMS/$\overline{\text{LS}}$	$\overline{\text{OEY}}$	$\overline{\text{OEC}}$	$\overline{\text{OES}}$	BYTEP	SELST1–0	$\overline{\text{RESET}}$	$\overline{\text{HALT}}$	TP1–0
000 0111 0000	0	01	110	1111 1111	00	0	1	1	x	x	0	0	0	x	11	1	1	11

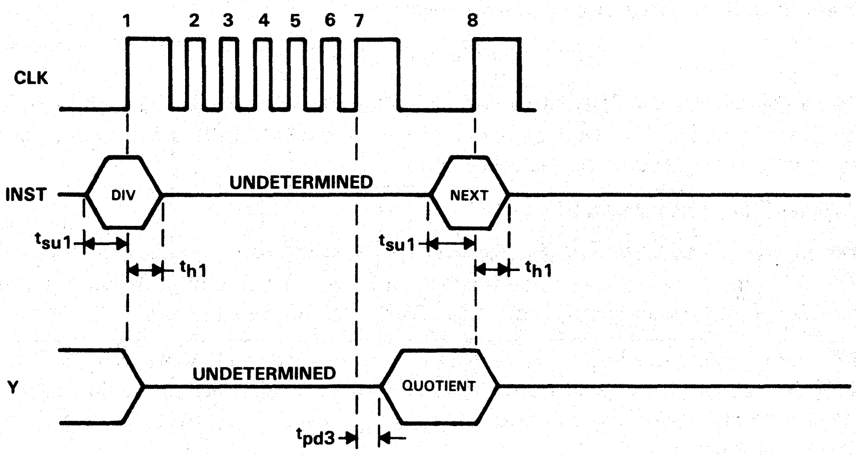


Figure 22. Single-Precision Floating Point Division (PIPES2-PIPES0 = 110)

In the second example, a wrapped number is divided by a normalized number. The result of this operation may be an inexact zero, setting the DENORM, INEX, and UNDER status flags. The multiplier pipeline is enabled, allowing the next instruction to load one cycle earlier, on the seventh rising clock.

CLKMODE = 0 PIPES = 100 Operation: A(wr) / B

I 11 0–0	CLKMODE	CONFIG 1–0	PIPES 2–0	SELOP 7–0	RND 1–0	FAST	ENRA	ENRB	SRCC	SELMS/$\overline{\text{LS}}$	$\overline{\text{OEY}}$	$\overline{\text{OEC}}$	$\overline{\text{OES}}$	BYTEP	SELST 1–0	$\overline{\text{R}}$ESET	$\overline{\text{H}}$ALT	TP 1–0
000 0110 0010	0	01	100	1111 1111	00	0	1	1	x	x	0	0	0	x	11	1	1	11

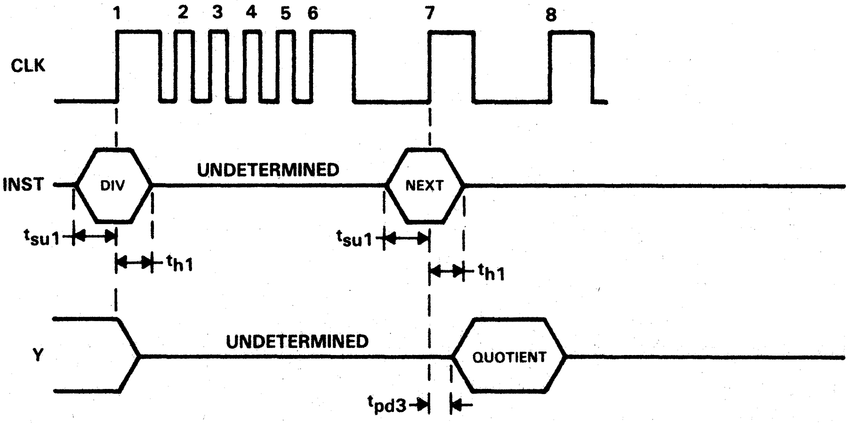


Figure 23. Single-Precision Floating Point Division (PIPES2-PIPES0 = 100)

Input and output registers are enabled (PIPES2 – PIPES0 = 010) in the next example, which shows division of two wrapped numbers. The result, which will probably be a normalized number, is negated by setting I2 high. Enabling the output register adds a one cycle delay to the output.

CLKMODE = 0 PIPES = 010 Operation: – A(wr) / B(wr)

I10–I0	CLKMODE	CONFIG1–0	PIPES2–0	SELOP7–0	RND1–0	FAST	ENRA	ENRB	SRCC	SELMS/$\overline{\text{LS}}$	$\overline{\text{OEY}}$	$\overline{\text{OEC}}$	$\overline{\text{OES}}$	BYTEP	SELST1–0	$\overline{\text{RESET}}$	$\overline{\text{HALT}}$	TP1–0
000 0110 0111	0	01	010	1111 1111	00	0	1	1	x	x	0	0	0	x	11	1	1	11

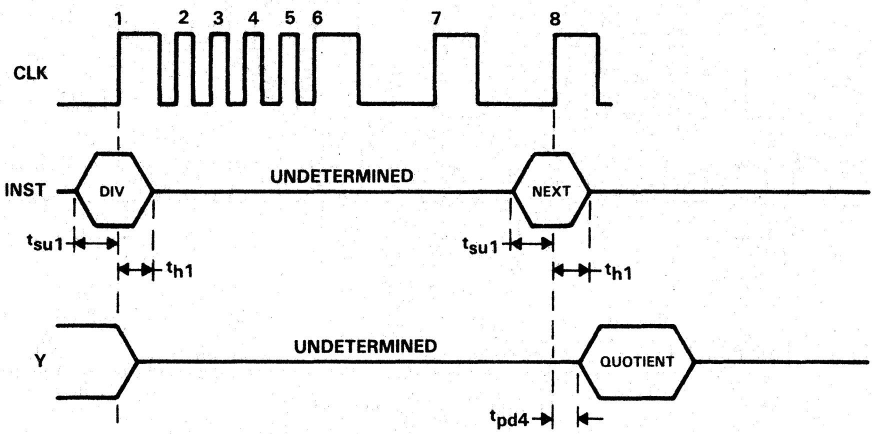


Figure 24. Single-Precision Floating Point Division (PIPES2-PIPES0 = 010)

The fourth microinstruction is A/B with all registers enabled. The output is available after the eighth rising clock edge, and the next instruction may be loaded on the seventh rising clock.

CLKMODE = 0 PIPES = 000 Operation: A / B

I10-I0	CLKMODE	CONFIG1-0	PIPES2-0	SELOP7-0	RND1-0	FAST	ENRA	ENRB	SRCC	SELMS/$\overline{LS}$	$\overline{OEY}$	$\overline{OEC}$	$\overline{OES}$	BYTEP	SELST1-0	$\overline{RESET}$	$\overline{HALT}$	TP1-0
000 0110 0000	0	01	000	1111 1111	00	0	1	1	x	x	0	0	0	x	11	1	1	11

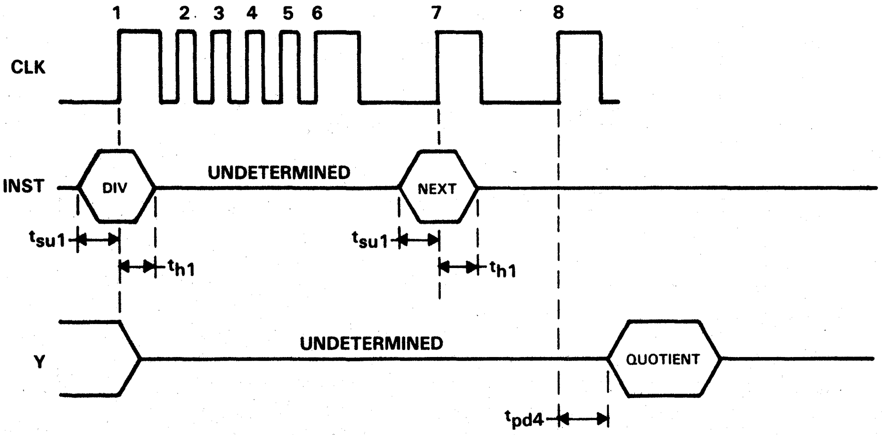


Figure 25. Single-Precision Floating Point Division (PIPES2-PIPES0 = 000)

Double-Precision Floating Point Division

The next four examples show double-precision floating point division. The operands may be single or double precision, or a combination of both. However, wrapped single precision numbers may not be used. A single-precision number should be loaded in the most significant half of the double precision operand.

Each example includes a timing diagram for a different setting of the internal register controls (PIPES2-PIPES0). The examples use both clock modes 0 and 1.

The first example shows a mixed-precision, mixed-mode operation with input registers RA and RB enabled. A single precision number is divided by a double precision wrapped number. The output, in double-precision format is available after thirteen rising clock edges, and may be valid for that cycle only. If the next operation is double precision, the first half of the operands may be loaded on the rising edge of clock 13 (using clock mode 0). The second half operands and the instruction are loaded on the fourteenth clock. A single-precision operation may only be loaded on the fourteenth rising edge.

CLKMODE = 0 PIPES = 110 Operation: A / B(wr)

I10-I0	CLKMODE	CONFIG1-0	PIPES2-0	SELOP7-0	RND1-0	FAST	ENRA	ENRB	SRCC	SELMS/$\overline{\text{LS}}$	$\overline{\text{OEY}}$	$\overline{\text{OEC}}$	$\overline{\text{OES}}$	BYTEP	SELST1-0	$\overline{\text{RESET}}$	$\overline{\text{HALT}}$	TP1-0
000 1110 0001	0	01	110	1111 1111	00	0	1	1	x	x	0	0	0	x	11	1	1	11

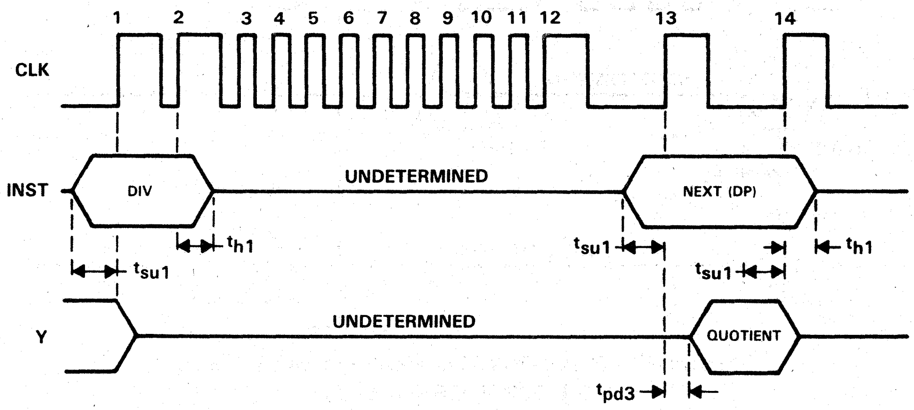


Figure 26. Double-Precision Floating Point Division (PIPES2-PIPES0 = 110)

In the next example, the divisor is a wrapped number. Since both numbers are double precision, the result may overflow, setting the OVR flag. With the multiplier pipeline and input registers enabled, the next double-precision operands and instruction may be loaded on the twelfth and thirteenth rising clock edges (clock mode 0), as shown in Figure 27.

CLKMODE = 0 PIPES = 100 Operation: A / B(wr)

I 10-0	CLKMODE	CONFIG 1-0	PIPES 2-0	SELOP 7-0	RND 1-0	FAST	ENRA	ENRB	SRCC	SELMS/$\overline{\text{LS}}$	$\overline{\text{OEY}}$	$\overline{\text{OEC}}$	$\overline{\text{OES}}$	BYTEP	SELST 1-0	$\overline{\text{RESET}}$	$\overline{\text{HALT}}$	TP 1-0
001 1110 0001	0	01	100	1111 1111	00	0	1	1	x	x	0	0	0	x	11	1	1	11

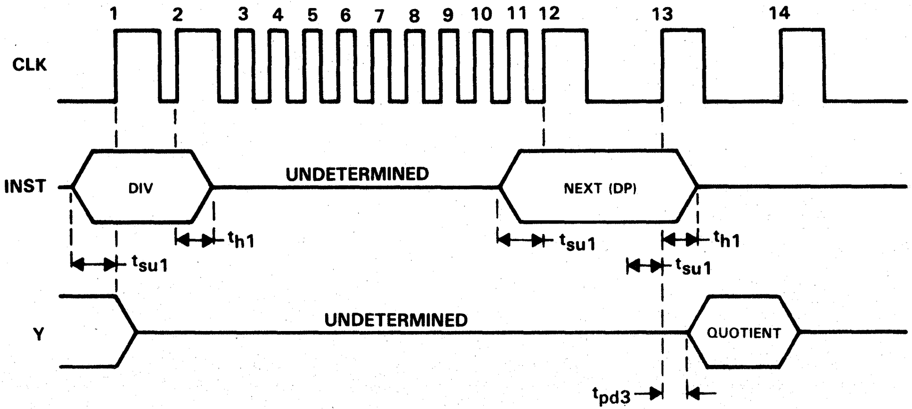


Figure 27. Double-Precision Floating Point Division (PIPES2-PIPES0 = 100)

A double-precision number is divided by a single-precision number in the third example. Clock mode 1 is used, with input and output registers enabled (PIPES2-PIPES0 = 010). The output is available after the fourteenth rising clock edge and may only be valid for that cycle. If the next operation is double precision, half the operands can be loaded on the falling edge of the thirteenth clock, and the second half operands and instruction load on rising edge of the fourteenth clock. If the next operation is single precision, the instruction must be loaded on the fourteenth rising edge.

CLKMODE = 1   PIPES = 010   Operation: A / B

I10–I0	CLKMODE	CONFIG1–0	PIPES2–0	SELOP7–0	RND1–0	FAST	ENRA	ENRB	ENRC	SELMS/$\overline{LS}$	$\overline{OEY}$	$\overline{OEC}$	$\overline{OES}$	BYTEP	SELST1–0	$\overline{RESET}$	$\overline{HALT}$	TP1–0
001 0110 0000	1	01	010	1111 1111	00	0	1	1	x	x	0	0	0	x	11	1	1	11

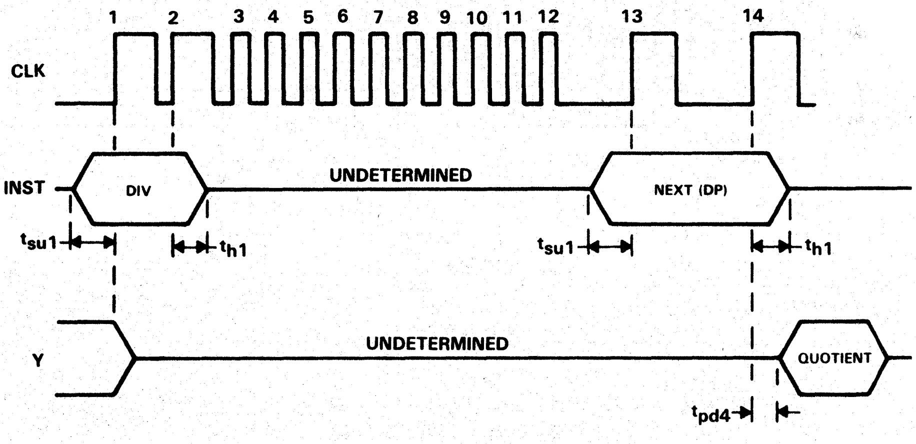


Figure 28. Double-Precision Floating Point Division (PIPES2-PIPES0 = 010)

The last example is a microinstruction for A / B with all registers enabled (PIPES2-PIPES0 = 000). Both operands are double precision. The output is available after the fourteenth rising edge. The next double-precision operation may be loaded on the falling edge of clock 12 and the rising edge of clock 13 using clock mode 1. If the next operation is single precision, the instruction must be loaded on the thirteenth rising clock edge.

CLKMODE = 1 PIPES = 000 Operation: A / B

I10-I0	CLKMODE	CONFIG1-0	PIPES2-0	SELOP7-0	RND1-0	FAST	ENRA	ENRB	SRCC	SELMS/$\overline{\text{LS}}$	$\overline{\text{OEY}}$	$\overline{\text{OEC}}$	$\overline{\text{OES}}$	BYTEP	SELST1-0	$\overline{\text{RESET}}$	$\overline{\text{HALT}}$	TP1-0
001 1110 0000	1	00	000	1111 1111	00	0	1	1	x	x	0	0	0	x	11	1	1	11

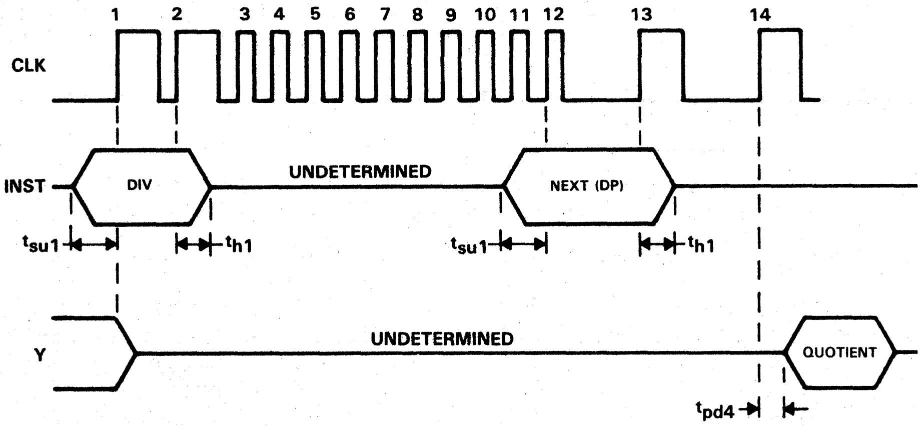


Figure 29. Double-Precision Floating Point Division (PIPES2-PIPES0 = 000)

Integer Division

The following sample microinstructions perform division operations on single-precision integer operands. Both operands must be in the same format (unsigned or two's complement), and absolute value, wrapped numbers, and negated outputs are not allowed. Each example includes a timing diagram for a different setting of the internal register controls (PIPES2-PIPES0).

The first example shows A / B with only the input registers RA and RB enabled. Both operands are in two's complement format. The result is available after the rising edge of the fifteenth clock and may be valid for only that cycle. The next instruction and data may be loaded on the sixteenth clock.

CLKMODE = 0   PIPES = 110   Operation: A / B

I10–I0	CLKMODE	CONFIG1–0	PIPES2–0	SELOP7–0	RND1–0	FAST	ENRA	ENRB	SRCC	SELMS/$\overline{LS}$	$\overline{OEY}$	$\overline{OEC}$	$\overline{OES}$	BYTEP	SELST1–0	$\overline{RESET}$	$\overline{HALT}$	TP1–0
010 0110 0000	0	01	110	1111 1111	00	0	1	1	x	x	0	0	0	x	11	1	1	11

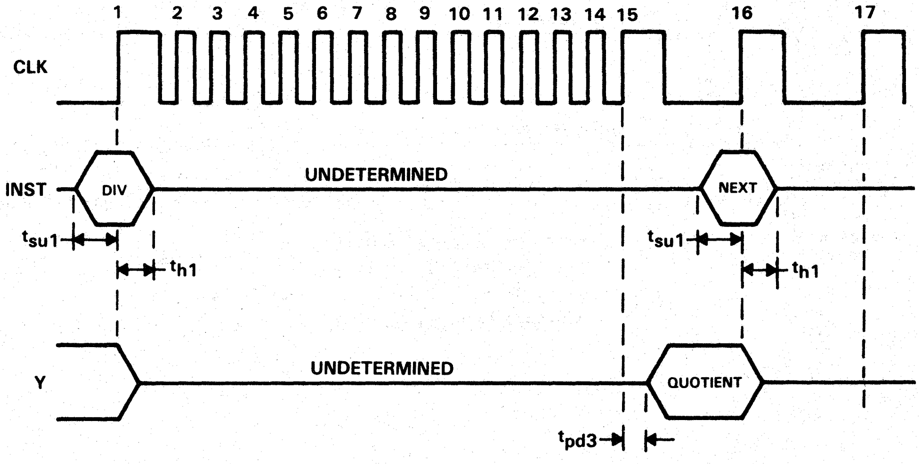


Figure 30. Integer Division
(PIPES2-PIPES0 = 110)

Enabling the multiplier pipeline register in the second example allows the next instruction to be loaded one cycle earlier. The quotient is still not available until after the fifteenth rising clock (see Figure 31).

CLKMODE = 0 PIPES = 100 Operation: A / B

I10-I0	CLKMODE	CONFIG1-0	PIPES2-0	SELOP7-0	RND1-0	FAST	ENRA	ENRB	SRC	SELMS/$\overline{\text{LS}}$	$\overline{\text{OEY}}$	$\overline{\text{OEC}}$	$\overline{\text{OES}}$	BYTEP	SELST1-0	$\overline{\text{RESET}}$	$\overline{\text{HALT}}$	TP1-0
010 0110 0000	0	01	100	1111 1111	00	0	1	1	x	x	0	0	0	x	11	1	1	11

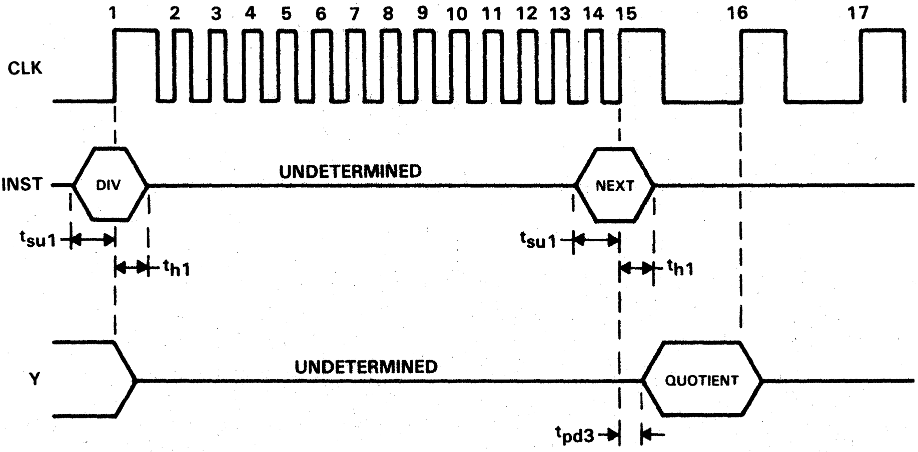


Figure 31. Integer Division (PIPES2-PIPES0 = 100)

7

SN74ACT8847

In the third example, unsigned integer numbers are divided. The pipeline settings (PIPES2-PIPES0 = 010) enable the input and output registers. Timing for this microinstruction is shown in Figure 32.

CLKMODE = 0   PIPES = 010   Operation: A / B

I10-I0	CLKMODE	CONFIG1-0	PIPES2-0	SELOP7-0	RND1-0	FAST	ENRA	ENRB	SRCC	SELMS/$\overline{LS}$	$\overline{OEY}$	$\overline{OEC}$	$\overline{OES}$	BYTEP	SELST1-0	SELECT	$\overline{RESET}$	$\overline{HALT}$	TP1-0
010 1110 0000	0	01	010	1111 1111	00	0	1	1	x	x	0	0	0	x	11		1	1	11

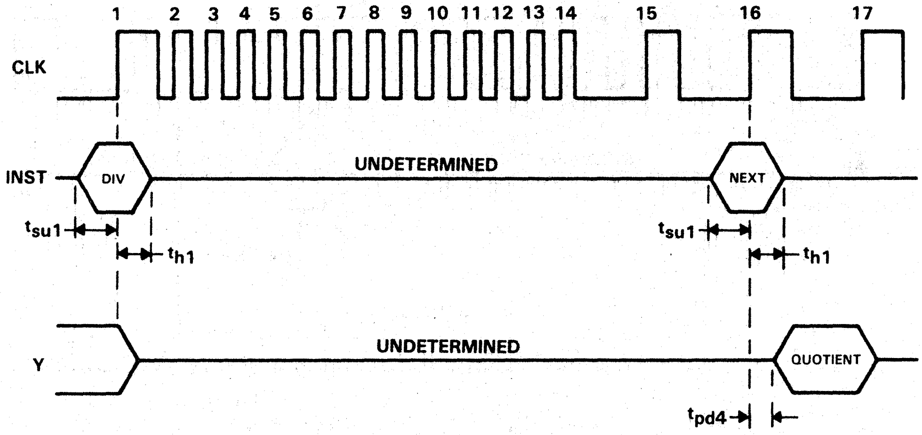


Figure 32. Integer Division (PIPES2-PIPES0 = 010)

The final example has all internal registers enabled. The output is valid after the sixteenth clock edge, but the next instruction can be loaded one cycle earlier.

CLKMODE = 0   PIPES = 000   Operation: A / B

I10-0	CLKMODE	CONFIG1-0	PIPES2-0	SELOP7-0	RND1-0	FAST	ENRA	ENRB	SRCC	SELMS/$\overline{\text{LS}}$	$\overline{\text{OEY}}$	$\overline{\text{OEC}}$	$\overline{\text{OES}}$	BYTEP	SELST1-0	$\overline{\text{RESET}}$	$\overline{\text{HALT}}$	TP1-0
010 1110 0000	0	01	000	1111 1111	00	0	1	1	x	x	0	0	0	x	11	1	1	11

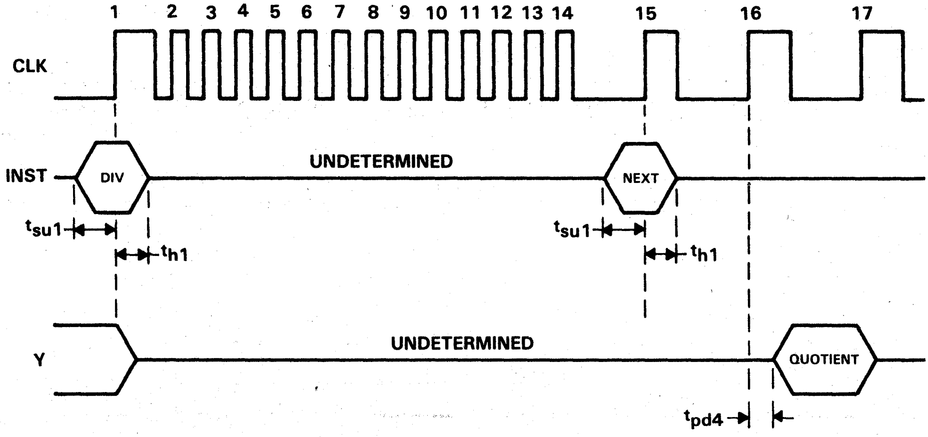


Figure 33. Integer Division (PIPES2-PIPES0 = 000)

Square Root Microinstructions

Square root calculations are performed as independent multiplier operations, as determined by the settings of instruction bits I10, I5, and I3. Independent multiplier operation is selected by setting I10 low, while both I5 and I3 must be set high to select a square root operation.

Taking the square root of a nonzero negative number sets the IVAL flag and produces an NaN result. Square root of negative zero does not set IVAL and produces a negative zero result. Only the A operand is used in a square root calculation. The operand may be a single-precision integer, single-precision floating point number, or double-precision floating point number. Since the B operand is unused, the I7 input should be set low or set to match the I8 input, and the I0 input is a "Don't Care"

Single-Precision Floating Point Square Root

The following four sample microinstructions select square root operations on single-precision floating point operands. Each example includes a timing diagram for a different setting of the internal register controls (PIPES2-PIPES0).

The first example shows an instruction to perform SQRT |A| on a wrapped number with only the RA and RB input registers enabled (PIPES2-PIPES0 = 110). The result is available after the tenth rising clock edge and may be valid only for that cycle. The next instructions and data operand may be loaded on the following rising clock.

CLKMODE = 0   PIPES = 110   Operation: SQRT |A|

```
                                                      S
                                                      E
            C C C                                     L
            L O O P P    S S                          M       S S
            K N N I I    E E                          S     B E E R̄
            M F F P P    L L     R R F E E S          /     Y L L E H̄
   I        O I I E E    O O     N N A N N R   Ō Ō Ō  T S S S A T T
  1 I       D G G S S    P P     D D S R R C L̄ E E E  E T T E L P P
  0-0       E 1-0 2-0    7-0     1-0 T A B C S Y C S  P 1-0 T T 1-0

000 0111 101x 0 01  110 1111 1111 00   0 1 1 x x 0 0 0 x  11  1 1 11
```

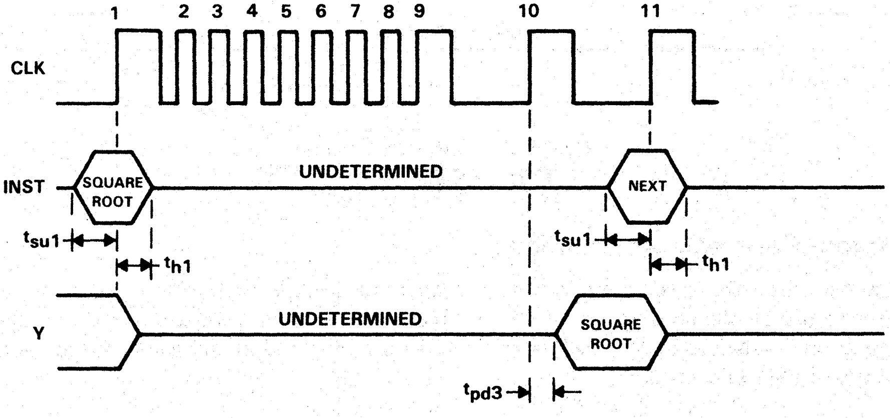


Figure 34. Single-Precision Floating Point Square Root (PIPES2-PIPES0 = 110)

The square root of a wrapped number is calculated in the second example. Both the input registers and multiplier pipeline are enabled, allowing the next instruction to be loaded on the tenth rising clock (see Figure 35).

CLKMODE = 0 PIPES = 100 Operation: SQRT A(wr)

I10-I0	CLKMODE	CONFIG1-0	PIPES2-0	SELOP7-0	RND1-0	FAST	ENRA	ENRB	SRCC	SELMS/$\overline{\text{LS}}$	$\overline{\text{OEY}}$	$\overline{\text{OEC}}$	$\overline{\text{OES}}$	BYTEP	SELST1-0	$\overline{\text{RESET}}$	$\overline{\text{HALT}}$	TP1-0
000 0110 001x	0	01	100	1111 1111	00	0	1	1	x	x	0	0	0	x	11	1	1	11

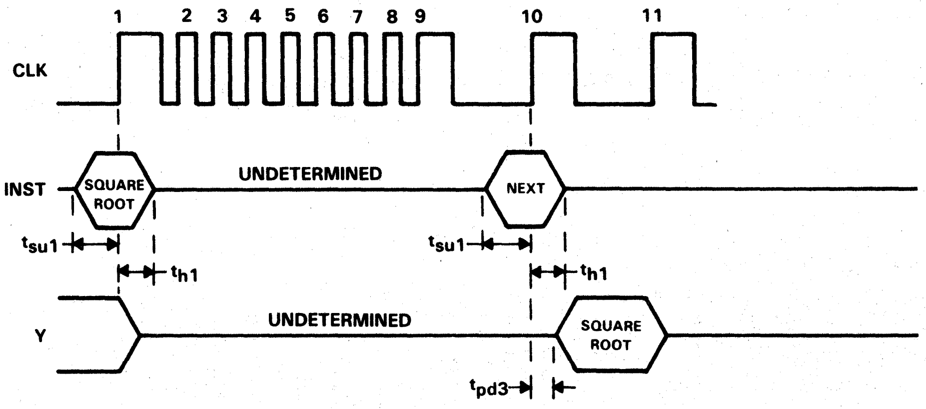


Figure 35. Single-Precision Floating Point Square Root (PIPES2-PIPES0 = 100)

Enabling both the input and output registers (PIPES2-PIPES0 = 010) in the next example delays the output by one cycle. This instruction calculates −SQRT |A|

CLKMODE = 0 PIPES = 010 Operation: − SQRT |A|

I10-I0	CLKMODE	CONFIG1-0	PIPES2-0	SELOP7-0	RND1-0	FAST	ENRA	ENRB	SRCC	SELMS/$\overline{\text{LS}}$	$\overline{\text{OEY}}$	$\overline{\text{OEC}}$	$\overline{\text{OES}}$	BYTEP	SELST1-0	SRCEST	$\overline{\text{RESET}}$	$\overline{\text{HALT}}$	TP1-0
000 0110 010x	0	01	010	1111 1111	00	0	1	1	x	x	0	0	0	x	11		1	1	11

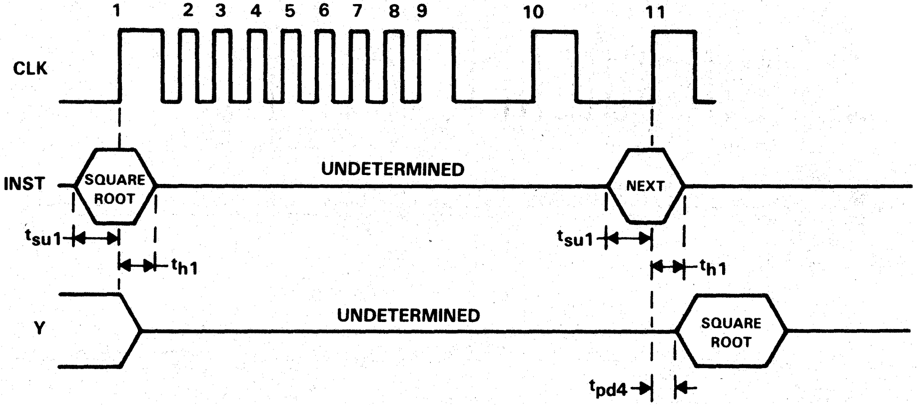


Figure 36. Single-Precision Floating Point Square Root (PIPES2-PIPES0 = 010)

The fourth instruction performs SQRT A with all registers enabled. The output is valid after the eleventh rising clock edge, and the next instruction may be loaded on the tenth rising clock.

CLKMODE = 0   PIPES = 000   Operation: SQRT A

I10-I0	CLKMODE	CONFIG1-0	PIPES2-0	SELOP7-0	RND1-0	FAST	ENRA	ENRB	SRCC	SELMS/$\overline{LS}$	$\overline{OEY}$	$\overline{OEC}$	$\overline{OES}$	BYTEP	SELST1-0	$\overline{R}$ESET	$\overline{H}$ALT	TP1-0
000 0110 100x	0	01	000	1111 1111	00	0	1	1	x	x	0	0	0	x	11	1	1	11

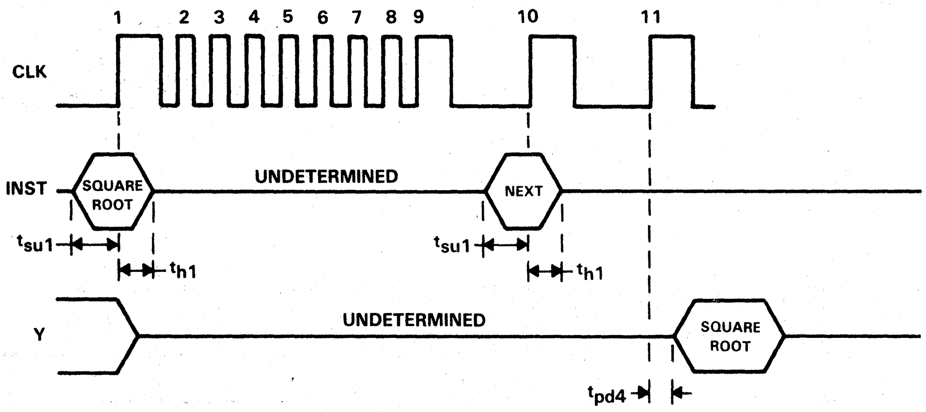


Figure 37. Single-Precision Floating Point Square Root (PIPES2-PIPES0 = 000)

Double-Precision Floating Point Square Root

The next four examples show square root operations on double-precision floating point operands. Each example includes a timing diagram for a different setting of the internal register controls (PIPES2-PIPES0). Both clock modes 0 and 1 are used in the examples.

The first sample instruction calculates − SQRT A. Clock mode 1 is shown. With only the input registers RA and RB enabled, the output is valid after the sixteenth rising clock edge and may be stable for that cycle only. If the next instruction is a double-precision operation, the first half of the data operands may be loaded on the falling edge of clock 16 (assuming clock mode 1). The second half of the data and the instruction are loaded on the seventeenth rising clock edge. A single-precision operation must be loaded on the seventeenth rising edge.

CLKMODE = 1   PIPES = 110   Operation: − SQRT A

I10-I0	CLKMODE	CONFIG1-0	PIPES2-0	SELOP7-0	RND1-0	FAST	ENRA	ENRB	SRCC	SELMS/$\overline{\text{LS}}$	$\overline{\text{OEY}}$	$\overline{\text{OEC}}$	$\overline{\text{OES}}$	BYTEP	SELST1-0	$\overline{\text{RESET}}$	$\overline{\text{HALT}}$	TP1-0
001 0110 110x	1	11	110	1111 1111	00	0	1	1	x	x	0	0	0	x	11	1	1	11

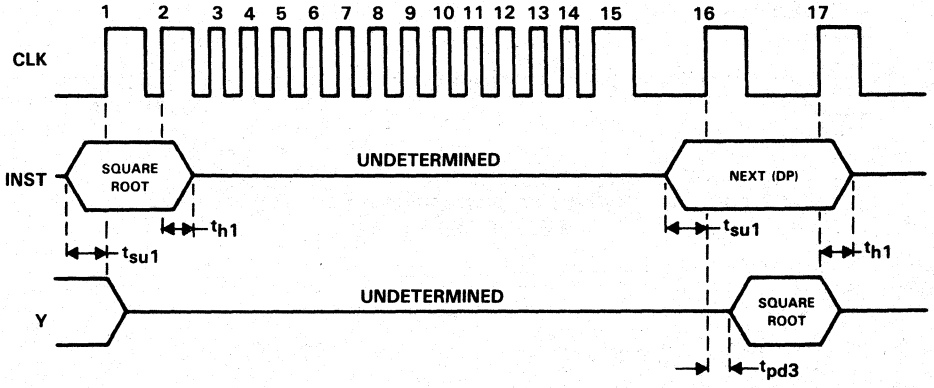


Figure 38. Double-Precision Floating Point Square Root (PIPES2-PIPES0 = 110)

The second example shows an instruction to perform SQRT |A|. Because the multiplier pipeline is enabled (PIPES2-PIPES0 = 100), the next double-precision operand and instruction may be loaded on the falling edge of clock 15 (using clock mode 1) and the rising edge of clock 16. If the next operation is single precision, it must be loaded on the sixteenth rising clock.

CLKMODE = 0   PIPES = 100   Operation: SQRT |A|

I10–I0	CLKMODE	CONFIG1–0	PIPES2–0	SELOP7–0	RND1–0	FAST	ENRA	ENRB	SRCC	SELMS/$\overline{\text{LS}}$	$\overline{\text{OEY}}$	$\overline{\text{OEC}}$	$\overline{\text{OES}}$	BYTEP	SELST1–0	$\overline{\text{RESET}}$	$\overline{\text{HALT}}$	TP1–0
001 0111 100x	0	01	100	1111 1111	00	0	1	1	x	x	0	0	0	x	11	1	1	11

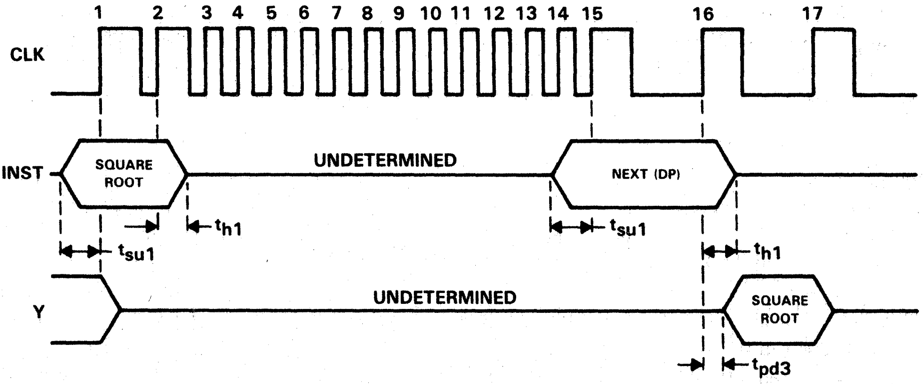


Figure 39. Double-Precision Floating Point Square Root (PIPES2-PIPES0 = 100)

In the third example, the operand is a wrapped number. With both input and output registers enabled (PIPES2-PIPES0 = 010), the output is valid after the seventeenth clock cycle. If the next operation is double precision, the first half operands may be loaded on the sixteenth rising clock edge, using clock mode 0. The remaining operands and instruction are then loaded on the seventeenth rising edge. If the next operation is single precision, the instruction and operands are loaded on the rising edge of clock 17.

CLKMODE = 1 PIPES = 010 Operation: SQRT A(wr)

I10-I0	CLKMODE	CONFIG1-0	PIPES2-0	SELOP7-0	RND1-0	FAST	ENRA	ENRB	SRCC	SELMS/$\overline{\text{LS}}$	$\overline{\text{OEY}}$	$\overline{\text{OEC}}$	$\overline{\text{OES}}$	BYTEP	SELST1-0	$\overline{\text{RESET}}$	$\overline{\text{HALT}}$	TP1-0
001 0110 101x	1	10	010	1111 1111	00	0	1	1	x	x	0	0	0	x	11	1	1	11

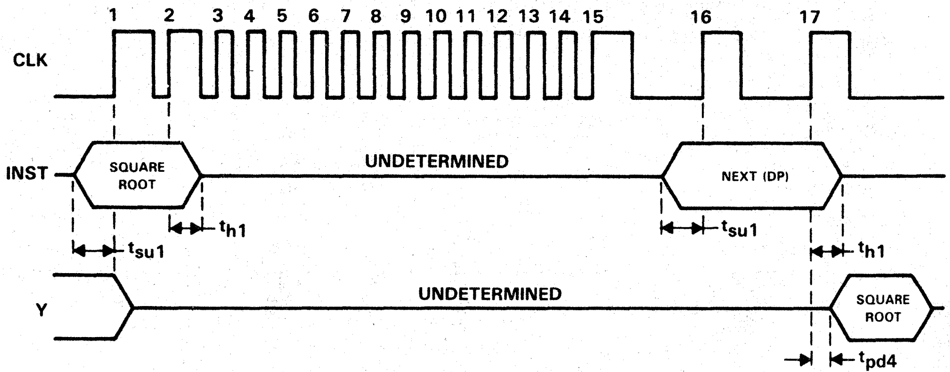


Figure 40. Double-Precision Floating Point Square Root (PIPES2-PIPES0 = 010)

All registers are enabled in the fourth example, which calculates SQRT A. Using clock mode 0, the next double precision operation may be loaded on the fifteenth and sixteenth rising clock edges. A single precision operation would load on the sixteenth clock. The output is available after the sixteenth rising clock edge and may only be valid for that cycle (see Figure 41).

CLKMODE = 0 PIPES = 000 Operation: SQRT A

I10-I0	CLKMODE	CONFIG1-0	PIPES2-0	SELOP7-0	RND1-0	FAST	ENRA	ENRB	SRCC	SELMS/$\overline{\text{LS}}$	$\overline{\text{OEY}}$	$\overline{\text{OEC}}$	$\overline{\text{OES}}$	BYTEP	SELST1-0	$\overline{\text{RESET}}$	$\overline{\text{HALT}}$	PPT1-0
001 0110 100x	0	01	000	1111 1111	00	0	1	1	x	x	0	0	0	x	11	1	1	11

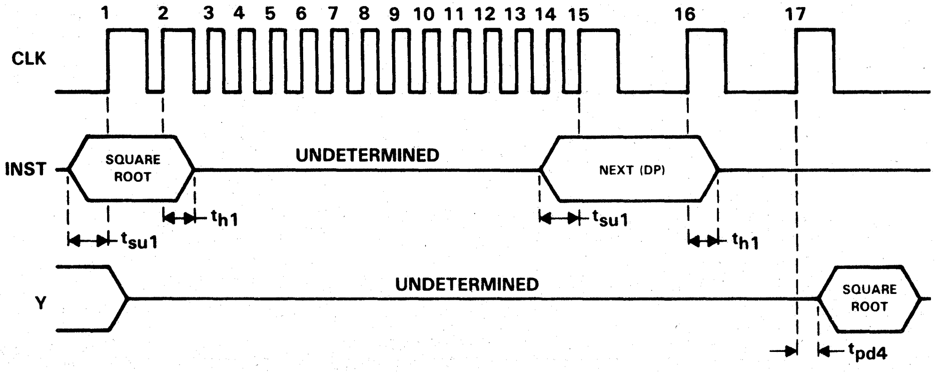


Figure 41. Double-Precision Floating Point Square Root (PIPES2-PIPES0 = 000)

Integer Square Root

The following example microinstructions perform square root operations on single precision integer operands. Absolute values and negated results are not valid options for integer calculations. Each example includes a timing diagram for a different setting of the internal register controls (PIPES2-PIPES0).

The first instruction performs the square root of an unsigned integer with only the RA and RB input registers enabled. The output is available after the nineteenth rising clock edge and may only be valid for that cycle. The next instruction may be loaded on the twentieth rising edge.

CLKMODE = 0 PIPES = 110 Operation: SQRT A

```
                                      S
                                      E
        C C C                         L
        L O O P P   S S               M         S S
        K N N I I   E E               S       B E E R̄
        M F F P P   L L   R R F E E S /       Y L L E H̄
I       O I I E E   O O   N N A N N R   Ō Ō Ō T S S S A T T
1 I     D G G S S   P P   D D S R R C L̄ E E E E T T E L P P
0-0     E 1-0 2-0   7-0   1-0 T A B C S Y C S P 1-0 T T 1-0

010 1100 000x 0 01 110 1111 1111 00 0 1 1 x x 0 0 0 x 11 1 1 11
```

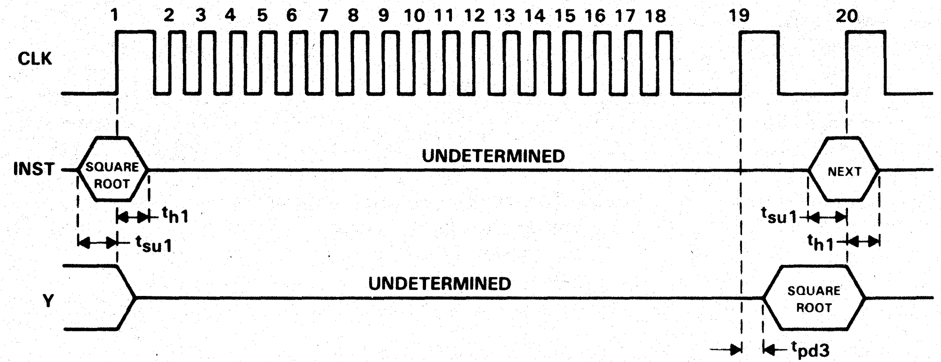


Figure 42. Integer Square Root
(PIPES2-PIPES0 = 110)

Enabling the multiplier pipeline register (PIPES2-PIPES0 = 100) in the second example allows the next instruction to be loaded one cycle earlier, on the nineteenth clock. The output will be valid after the nineteenth rising clock edge. (See Figure 43).

CLKMODE = 0   PIPES = 100   Operation: SQRT A

```
                                        S
                                        E
        C C C                           L
        L O O P P   S S                 M         S S
        K N N I I   E E                 S       B E E  R̅
        M F F P P   L L    R R  F E E S /       Y L L  E H̅
 I      O I I E E   O O    N N  A N N R   O̅ O̅ O̅ T S S  S A T T
1 I     D G G S S   P P    D D  S R R C L̅ E E E E T T  E L P P
0-0     E 1-0 2-0   7-0    1-0  T A B C S Y C S P 1-0  T T 1-0

010 1100 000x  0 01  100 1111 1111  00  0 1 1 x x 0 0 0 x 11  1 1 11
```

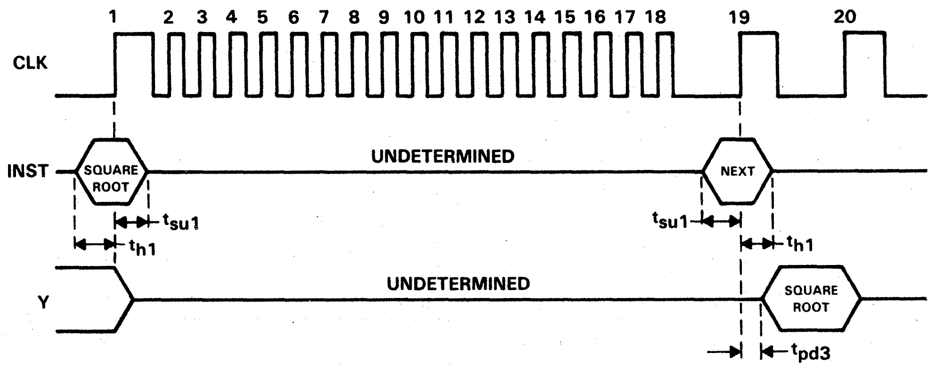


Figure 43. Integer Square Root (PIPES2-PIPES0 = 100)

The third example instruction performs SQRT A on a two's complement integer operand. With both input and output registers enabled (PIPE2-PIPE0 = 010), the output is valid after the twentieth rising clock edge. The next instruction may be loaded on the twentieth clock.

CLKMODE = 0   PIPES = 010   Operation: SQRT A

I10-I0	CLKMODE	CONFIG1-0	PIPES2-0	SELOP7-0	RND1-0	FAST	ENRA	ENRB	SRCC	SELMS/$\overline{\text{LS}}$	$\overline{\text{OEY}}$	$\overline{\text{OEC}}$	$\overline{\text{OES}}$	BYTEP	SELST1-0	$\overline{\text{RESET}}$	$\overline{\text{HALT}}$	TP1-0
010 0100 000x	0	01	010	1111 1111	00	0	1	1	x	x	0	0	0	x	11	1	1	11

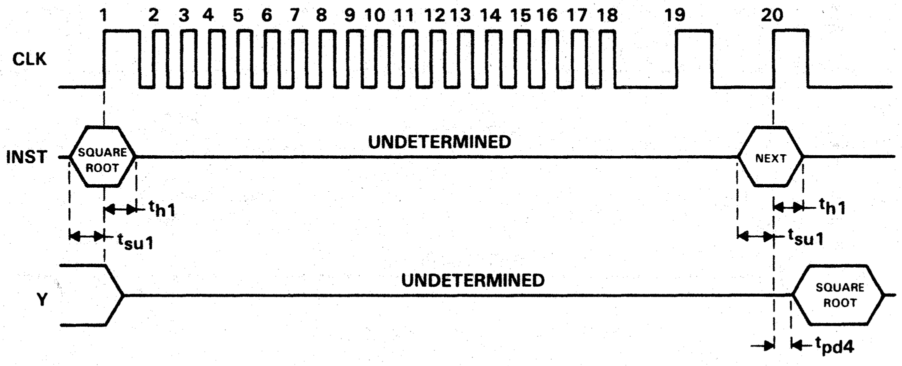


Figure 44. Integer Square Root (PIPES2-PIPES0 = 010)

The final example shows SQRT A with all registers enabled. The next instruction may be loaded on the nineteenth rising clock edge, but the output will not be available until after the twentieth rising clock.

CLKMODE = 0   PIPES = 000   Operation: SQRT A

I10-I0	CLKMODE	CONFIG1-0	PIPES2-0	SELOP7-0	RND1-0	FAST	ENRA	ENRB	SRCC	SELMS/$\overline{\text{LS}}$	$\overline{\text{OEY}}$	$\overline{\text{OEC}}$	$\overline{\text{OES}}$	BYTEP	SELST1-0	SELS	$\overline{\text{RESET}}$	$\overline{\text{HALT}}$	TP1-0
010 0100 000x	0	01	000	1111 1111	00	0	1	1	x	x	0	0	0	x	11		1	1	11

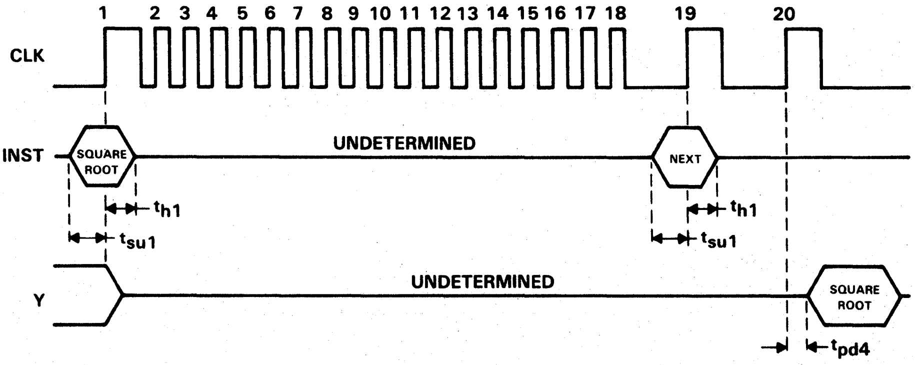


Figure 45. Integer Square Root (PIPES2-PIPES0 = 000)

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 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.

The sample microinstruction sequence shown in Tables 36 and 37 performs the operations for multiplying sets of data operands and accumulating the results, the basic operations involved in computing a sum of products.

Table 36 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 36. 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 37. Only three instructions in chained mode are required, since the multiplier begins the calculation independently and the ALU completes it independently.

Table 37. Sample Microinstructions for Single-Precision Sum of Products

I 9-0	CLKMODE	CONFIG 1-0	PIPES 2-0	SELOP 7-0	RND 1-0	FAST	ENRA	ENRB	SRCC	SELMS/$\overline{LS}$	$\overline{OEY}$	$\overline{OEC}$	$\overline{OES}$	BYTEP	SELST 1-0	$\overline{RESET}$	$\overline{HALT}$	TP 1-0
00 0100 0000	0	01	010	1111 xxxx	00	0	1	1	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	xx	1	1	11
10 0000 0000	0	01	010	1111 1010	00	0	1	1	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	xx	1	1	11
00 0000 0000	0	01	010	xxxx 1010	00	0	x	x	x	x	x	x	x	x	xx	1	1	11
xx xxxx xxxx	x	xx	xxx	xxxx xxxx	xx	x	x	x	x	x	0	0	0	x	xx	1	1	11

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.

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 HL to load operands by MSH and LSH, all registers enabled (PIPES2 - PIPES0 = LLL), and the C register clock tied to the system clock.

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

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

Table 38. Pseudocode for Fully Pipelined Double-Precision Sum of Products†
(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 MSH	B1 MSH	A1,B1MSH	A1 * B1									
↑ 2	A1 LSH	B1 LSH	A1,B1MSH	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,B2MSH	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,B3MSH	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,B4MSH	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,B5MSH	PR + SR A5 * B5	PR + SR, A5 * B5	A5	B5	A4 * B4	P3	P3	S1 + P3	S1	
↑ 11	A6 MSH	B6 MSH	A6,B6(M)	PR + SR A6 * B6	PR + SR, A5 * B5	A5	B5	A5 * B5	P4	P3	XXXXX	S2	
↑ 12													

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

Table 39. 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			$\overline{\text{ENRC}}$ = L 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	NOP	CR ∗ SR A3 + B3	A3	B3	S1 ∗ S2		S1	A3 + B3	S2	
⎍ 8	A4(M)	B4(M)	A4,B4(M)	PR ∗ SR A4 + B4	NOP	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	NOP	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	NOP	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	X	S4	

NOTE: NOP instruction is 011 0000 0000.

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

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 NOPs (no operations). The NOPs 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. 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.

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 46 shows a simplified timing waveform for a series of mixed operations:

CLOCK CYCLE	1	2	3	4	5	6	7	8	9	10	11	12	13
FUNCTION AND DATA	A,B	A,B	C,D	C,D		E,F	G,H	G,H	I,J	I,J	K,L	M,N	
RESULTS AND STATUS			XXXX	A,B	XXXX	C,D	E,F	E,F	G,H	G,H	I,J	K,L	M,N

A,B,C,D — double precision multiply; E,F — single precision operation; G,H,I,J — double precision add; K,L — single precision opration. 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 46. 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 47, a similar sequence of mixed operations is set up for execution in fully pipelined mode:

CLOCK CYCLE	1	2	3	4	5	6	7	8	9	10	11	12	13
FUNCTION AND DATA	A,B		C,D		E,F	G,H	I,J	K,L	M,N	O,P		Q,R	
RESULTS AND STATUS					A,B	A,B	C,D	E,F	G,H	I,J	K,L	M,N	M,N

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 47. 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.

Matrix Operations

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

Representation of Variables

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

$$\frac{dx1}{dt} = x\,2, \ldots, \quad \frac{dx(n-1)}{dt} = xn$$

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

$$\frac{d}{dt}\boxed{\begin{matrix} x1 \\ x2 \\ : \\ : \\ xn \end{matrix}} = \boxed{\begin{matrix} a11 & a12 & \ldots. & a1n \\ : & : & & : \\ : & : & & : \\ : & : & & : \\ an1 & an2 & \ldots. & ann \end{matrix}} \boxed{\begin{matrix} x1 \\ x2 \\ : \\ : \\ xn \end{matrix}} + \boxed{\begin{matrix} b11 & \ldots & b1n \\ : & & : \\ : & & : \\ : & & : \\ bn1 & \ldots & bnn \end{matrix}} \boxed{\begin{matrix} u1 \\ u2 \\ : \\ : \\ un \end{matrix}}$$

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

Expanding the matrix equation for one state variable, dx1/dt, results in the following expression:

$$\dot{X}1 = (a11 * x1 + \ldots + a1n * xn) + (b11 * u1 + \ldots + b1n * un)$$

where $\dot{X}1 = dx1/dt$.

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

$$\mathrm{cij} = \sum_{k=1}^{n} \mathrm{aik} * \mathrm{xkj} \quad \text{for } i=1,\ .\ .\ .\ ,n \quad j=1,\ .\ .\ .\ ,n \qquad (1)$$

For the cij elements to be calculated by the 'ACT8847, the corresponding elements aik and xkj must be stored outside the 'ACT8847 and fed to the 'ACT8847 in the proper order required to effect a matrix multiplication such as the state space system representation just discussed.

Sample Matrix Transformation

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

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

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

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

$$\left|\begin{array}{c:c} 3 \times 3 & \begin{matrix} 3 \\ \times \\ 1 \end{matrix} \\ \hdashline 1 \times 3 & 1 \times 1 \end{array}\right|$$

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

The rotation of an object about an arbitrary axis in a three-dimensional space can be carried out by first translating the object such that the desired axis of rotation passes through the origin of the coordinate system, then rotating the object about the axis through the origin, and finally translating the rotated object such that the axis of rotation resumes its initial position. If the axis of rotation passes through the point P = [a b c 1], then the transformation matrix is representable in this form:

$$[x\ y\ z\ h] = [x\ y\ z\ 1] \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -a & -b & -c & 1 \end{bmatrix}}_{\text{translation to origin}} \underbrace{\boxed{R}}_{\text{rotation about origin}} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ a & b & c & 1 \end{bmatrix}}_{\text{translation back to initial position}} \tag{2}$$

where R may be expressed as:

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

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

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

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

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

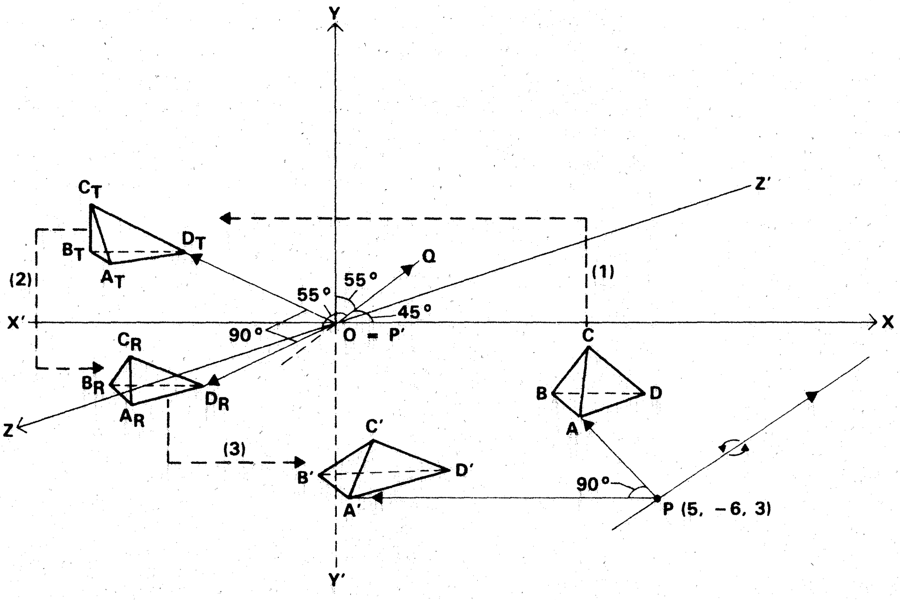
$\overline{Q}$ = vector defining axis of rotation = [q1 q2 q3]

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

A general rotation using equation (2) is effected by determining the [x y z] coordinates of a point A to be rotated on the object, the direction cosines of the axis of rotation [n1, n2, n3], and the angle ϕ of rotation about the axis, all of which are needed to define matrix [R]. Suppose, for example, that a tetrahedron ABCD, represented by the coordinate matrix below is to be rotated about an axis of rotation RX which passes through a point P = [5 −6 3 1] and whose direction cosines are given by unit vector [n1 = 0.866, n2 = 0.5, n3 = 0.707]. The angle of rotation 0 is 90 degrees (see Figure 24). The rotation matrix [R] becomes

$$\begin{array}{c} \begin{bmatrix} 2 & -3 & 3 & 1 \\ 1 & -2 & 2 & 1 \\ 2 & -1 & 2 & 1 \\ 2 & -2 & 2 & 1 \end{bmatrix} \end{array} \begin{array}{l} \text{—— A} \\ \text{—— B} \\ \text{—— C} \\ \text{—— D} \end{array}$$

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



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

Figure 48. Sequence of Matrix Operations

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

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

$$\begin{bmatrix} 2 & -3 & 3 & 1 \\ 1 & -2 & 2 & 1 \\ 2 & -1 & 2 & 1 \\ 2 & -2 & 2 & 1 \end{bmatrix} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -5 & 6 & -3 & 1 \end{bmatrix}}_{\substack{\text{translation} \\ \text{to origin}}} \underbrace{\begin{bmatrix} 0.750 & 1.140 & 0.112 & 0 \\ -0.274 & 0.250 & 1.22 & 0 \\ 1.112 & -0.513 & 0.500 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\text{rotation about origin}} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 5 & -6 & 3 & 1 \end{bmatrix}}_{\substack{\text{translation} \\ \text{back to} \\ \text{initial} \\ \text{position}}} \tag{3}$$

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:

$$\boxed{\begin{matrix} 2 & -3 & 3 & 1 \\ 1 & -2 & 2 & 1 \\ 2 & -1 & 2 & 1 \\ 2 & -2 & 2 & 1 \end{matrix}} \; \underbrace{\boxed{\begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -5 & 6 & -3 & 1 \end{matrix}}}_{\substack{\text{translation} \\ \text{to origin}}} = \underbrace{\boxed{\begin{matrix} (2-5) & (-3+6) & (3-3) & 1 \\ (1-5) & (-2+6) & (2-3) & 1 \\ (2-5) & (-1+6) & (2-3) & 1 \\ (2-5) & (-2+6) & (2-3) & 1 \end{matrix}}}_{\substack{\text{vertices of translated} \\ \text{tetrahedron}}}$$

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

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:

$$\boxed{\begin{matrix} -3 & 3 & 0 & 1 \\ -4 & 4 & -1 & 1 \\ -3 & 5 & -1 & 1 \\ -3 & 4 & -1 & 1 \end{matrix}} \underbrace{\boxed{\begin{matrix} 0.750 & 1.140 & 0.112 & 0 \\ -0.274 & 0.250 & 1.22 & 0 \\ 1.112 & -0.513 & 0.500 & 0 \\ 0 & 0 & 0 & 1 \end{matrix}}}_{\text{rotation about origin}} = \underbrace{\boxed{\begin{matrix} -3.072 & -2.670 & 3.324 & 1 \\ -5.208 & -3.047 & 3.932 & 1 \\ -4.732 & -1.657 & 5.264 & 1 \\ -4.458 & -1.907 & 4.044 & 1 \end{matrix}}}_{\text{rotated coordinates}}$$

Step 3

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

$$\boxed{\begin{matrix} -3.072 & -2.670 & 3.324 & 1 \\ -5.208 & -3.047 & 3.932 & 1 \\ -4.732 & -1.657 & 5.264 & 1 \\ -4.458 & -1.907 & 4.044 & 1 \end{matrix}} \underbrace{\boxed{\begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 5 & -6 & 3 & 1 \end{matrix}}}_{\text{translate back}} = \underbrace{\boxed{\begin{matrix} 1.928 & -8.670 & 6.324 & 1 \\ -0.208 & -9.047 & 6.932 & 1 \\ 0.268 & -7.657 & 8.264 & 1 \\ 0.542 & -7.907 & 7.044 & 1 \end{matrix}}}_{\text{final rotated coordinates}}$$

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:

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

$$\boxed{\begin{matrix} 2 & -3 & 3 & 1 \\ 1 & -2 & 2 & 1 \\ 2 & -1 & 2 & 1 \\ 2 & -2 & 2 & 1 \end{matrix}} \quad \underbrace{\boxed{\begin{matrix} 0.750 & 1.140 & 0.112 & 0 \\ -0.274 & 0.250 & 1.220 & 0 \\ 1.112 & -0.513 & 0.500 & 0 \\ -3.730 & -8.661 & 8.260 & 1 \end{matrix}}}_{\text{transformation matrix}}$$

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:

$$cij = \sum_{k=1}^{n} aik * xkj \quad \text{for } i=1, \ldots, n \quad j=1, \ldots, n \tag{1}$$

For example, the coordinates may be computed as follows:

$$\begin{aligned} xa = c11 &= a11 * x11 + a12 * x21 + a13 * x31 + a14 * x41 \\ &= 2 * 0.750 + (-3) * (-0.274) + 3 * 1.112 + 1 * (-3.73) \\ &= 1.5 + 0.822 + 3.336 - 3.73 \\ &= 1.928 \end{aligned}$$

$$\begin{aligned} ya = c12 &= a11 * x12 + a12 * x22 + a13 * x32 + a14 * x42 \\ &= 2 * 1.140 + (-3) * 0.250 + 3 * (-0.513) + 1x(-8.661) \\ &= 2.28 - 0.75 - 1.539 - 8.661 \\ &= -8.67 \end{aligned}$$

$$\begin{aligned} za = c13 &= a11 * x13 + a12 * x23 + a13 * x33 + a14 * x43 \\ &= 2 * 0.112 + (-3) * 1.220 + 3 * 0.500 + 1 * 8.260 \\ &= 0.224 - 3.66 + 1.5 + 8.260 \\ &= 6.324 \end{aligned}$$

$$\begin{aligned} h1 = c14 &= a11 * x14 + a12 * x24 + a13 * x34 + a14 * x44 \\ &= 2 * 0 + (-3) * 0 + 3 * 0 + 1 * 1 \\ &= 0 + 0 + 0 + 1 \\ &= 1 \end{aligned}$$

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

The other rotated vertices are computed in a similar manner:

$$\begin{aligned} B' &= [-5.208 \; -3.047 \; 3.932 \; 1] \\ C' &= [-4.732 \; -1.657 \; 5.264 \; 1) \\ D' &= [-4.458 \; -1.907 \; 4.044 \; 1] \end{aligned}$$

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 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 40. Table 41 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 40. Microinstructions for Sample Matrix Multiplication

II 10-0	CLKMODE	CONFIG 1-0	PIPES 2-0	SELOP 7-0	RND 1-0	FAST	ENRA	ENRB	SRCC	SELMS/$\overline{\text{L}}$S	$\overline{\text{O}}$EY	$\overline{\text{O}}$EC	$\overline{\text{O}}$ES	BYTEP	SELST 1-0	$\overline{\text{R}}$ESET	$\overline{\text{H}}$ALT	TP 1-0
000 0100 0000	0	01	010	1111 xxxx	00	0	1	1	x	x	x	x	x	x	xx	1	1	11
100 0110 0000	0	01	010	1111 xxxx	00	0	1	1	x	x	x	x	x	x	xx	1	1	11
100 0000 0000	0	01	010	1111 1010	00	0	1	1	x	x	x	x	x	x	xx	1	1	11
100 0000 0000	0	01	010	1111 1010	00	0	1	1	x	x	x	x	x	x	xx	1	1	11
100 0000 0000	0	01	010	1111 1010	00	0	1	1	x	x	x	x	x	x	xx	1	1	11
100 0110 0000	0	01	010	1111 xxxx	00	0	1	1	x	x	x	x	x	x	xx	1	1	11
100 0000 0000	0	01	010	1111 1010	00	0	1	1	x	x	x	x	x	x	xx	1	1	11
100 0000 0000	0	01	010	1111 1010	00	0	1	1	x	x	x	x	x	x	xx	1	1	11
100 0000 0000	0	01	010	1111 1010	00	0	1	1	x	x	x	x	x	x	xx	1	1	11
100 0110 0000	0	01	010	1111 xxxx	00	0	1	1	x	x	x	x	x	x	xx	1	1	11

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

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

$$\boxed{\begin{matrix}0\\0\\0\\1\end{matrix}}$$

Table 41. Single-Precision Matrix Multiplication (PIPES2-PIPES0 = 010)

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

The h-scalars h1, h2, h3, and h4 are equal to 1. The number of clock cycles to generate each 4-tuple can then be decreased from 16 to 13 cycles. Total number of clock cycles to calculate all four vertices is reduced from 66 to 54 clocks. Figure 49 summarizes the overall matrix transformation.

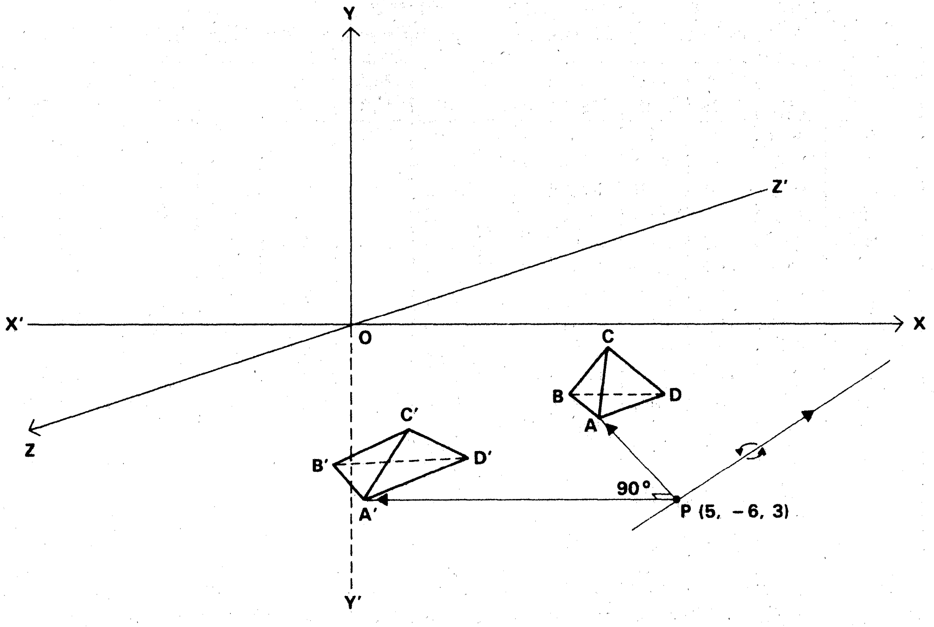


Figure 49. Resultant Matrix Transformation

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 42 presents the sequence of calculations for the first two coordinates, xa and ya.

Table 42. 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		p1			
4	Chn	x12	a11	Mul	x41	a14	p3	s1	p2			
5	Chn	x22	a12	Chn	x12	a11	p4	†	p3	s1	p2	
6	Chn	x32	a13	Chn	x22	a12	p5	s2	p4	†	p2	
7	Chn	x42	a14	Chn	x32	a13	p6	s3	p5	s2	p2	
8	Chn	x13	a11	Chn	x42	a14	p7	s4	p6	s3	s2	
9	Chn	x23	a12	Chn	x13	a11	p8	xA	p7	s4	p6	
10	Chn	x33	a13	Chn	x23	a12	p9	s5	p8	xA	p6	xA
11	Chn	x43	a14	Chn	x33	a13	p10	s6	p9	s5	p6	
12	Chn	x14	a11	Chn	x43	a14	p11	s7	p10	s6	s5	
13	Chn	x24	a12	Chn	x14	a11	p12	yA	p11	s7	p10	
14	Chn	x34	a13	Chn	x24	a12	p13	s8	p12	yA	p10	yA
15	Chn	x44	a14	Chn	x34	a13	p14	s9	p13	s8	p10	

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

Products in Table 42 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 42 are listed below:

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

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 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.

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.

Microsoft C is a registered trademark of Microsoft Corporation

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

Quadtree and Designer's Choice are trademarks of Quadtree Software Corporation
Logic Automation and Smartmodel are trademarks of Logic Automation Incorporated

'8800 SDB Kit

TI offers an '8800 Software Development Board (SDB) kit as an evaluation and training tool. The '8800 SDB kit allows users to evaluate performance and write microprograms for several of the '8800 building blocks, using a range of software development tools.

Built on a PC/AT card occupying a single slot, the '8800 SDB includes an 'ACT8818 microsequencer, 'ACT8832 registered ALU, and an 'ACT8847 floating point unit, along with 32K deep by 128 bits wide microcode memory, and 32K deep by 32 bits wide local data memory. Interface software is provided so that microcode and local data memory may be loaded and read from the PC/AT bus. Documentation is also included in the kit.

Users may write microcode source with the TI Meta Assembler, Metastep™ Assembler, or Hi-Level software tools. Pre-built '8800 SDB definition files are available with each of these so that users can begin developing microcode as soon as possible.

A software simulation of the '8800 SDB is currently being developed. The simulator operates in an MS-DOS environment.

For additional technical information, please contact VLSI Systems Engineering at (214) 997-3970. For ordering information, please call your local field sales representative.

MetaStep is a trademark of STEP Engineering, Inc.

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

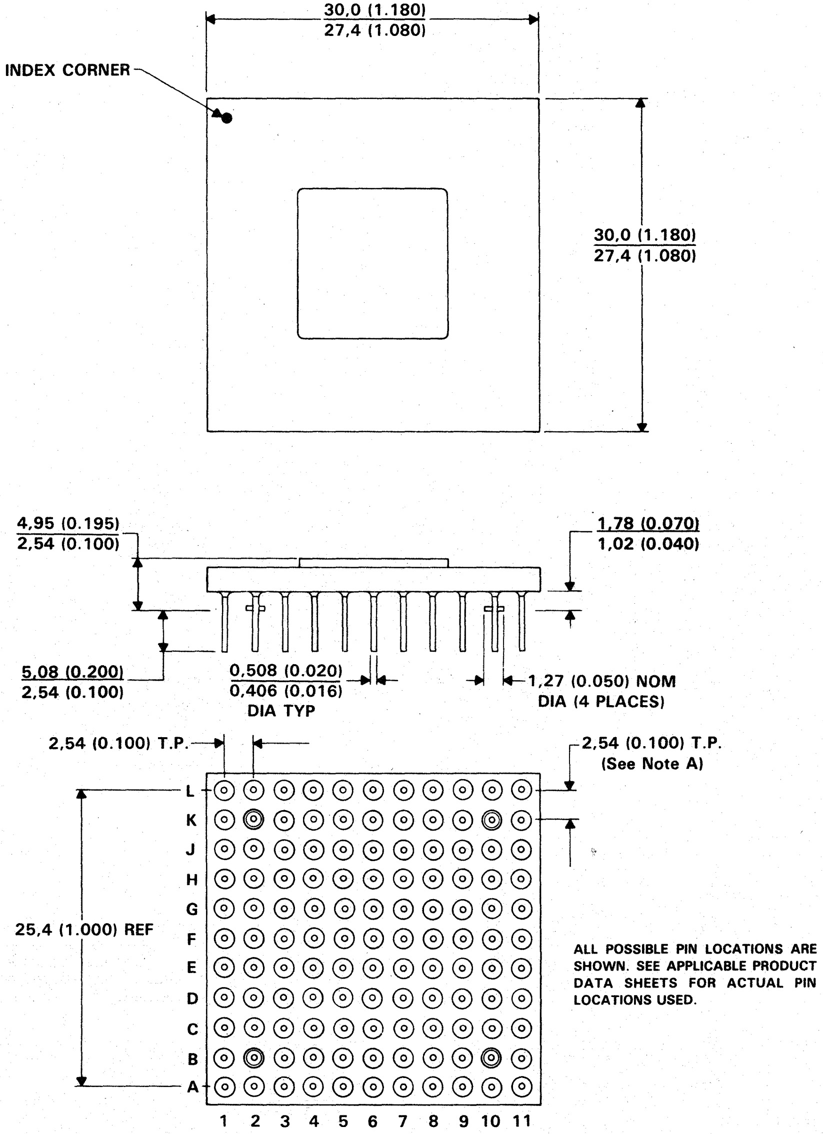
Texas Instruments VLSI Logic applications group is available to help designers analyze TI's high-performance VLSI products, such as the '8800 32-bit processor family. The group works directly with designers to provide ready answers to device-related questions and also prepares a variety of applications documentation.

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

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

11 × 11 GB pin grid array ceramic package

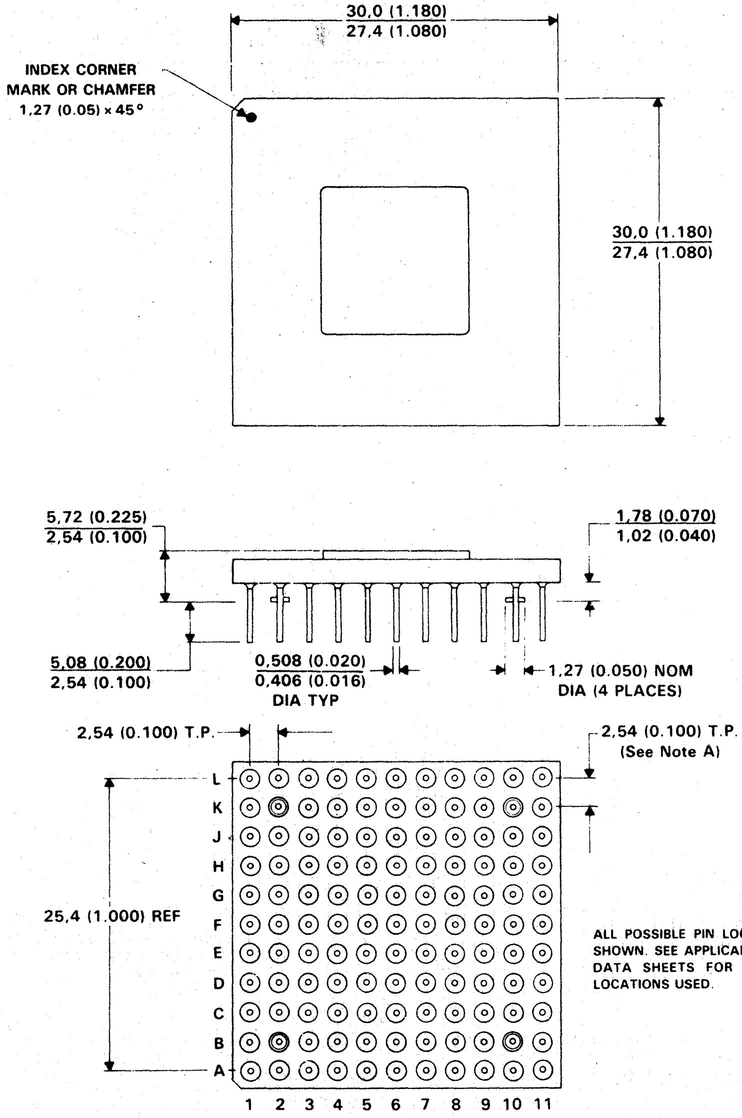


ALL POSSIBLE PIN LOCATIONS ARE SHOWN. SEE APPLICABLE PRODUCT DATA SHEETS FOR ACTUAL PIN LOCATIONS USED.

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

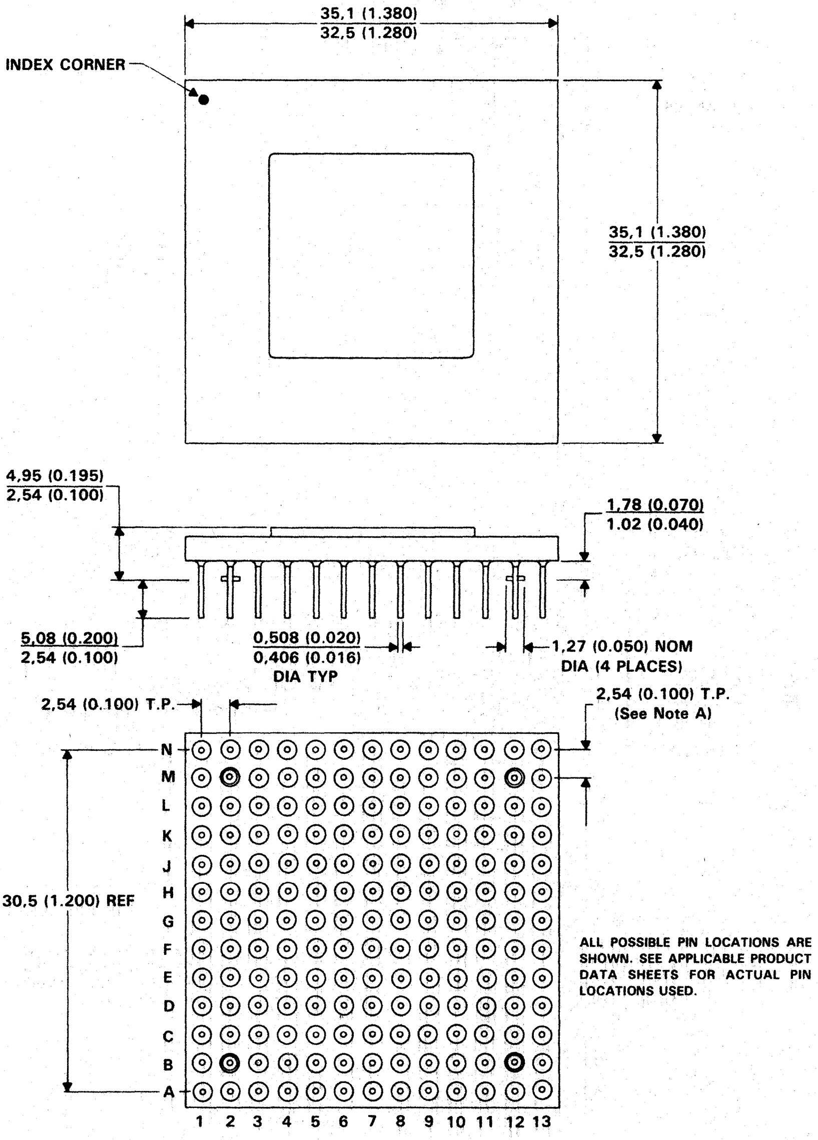


ALL POSSIBLE PIN LOCATIONS ARE SHOWN. SEE APPLICABLE PRODUCT DATA SHEETS FOR ACTUAL PIN LOCATIONS USED.

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

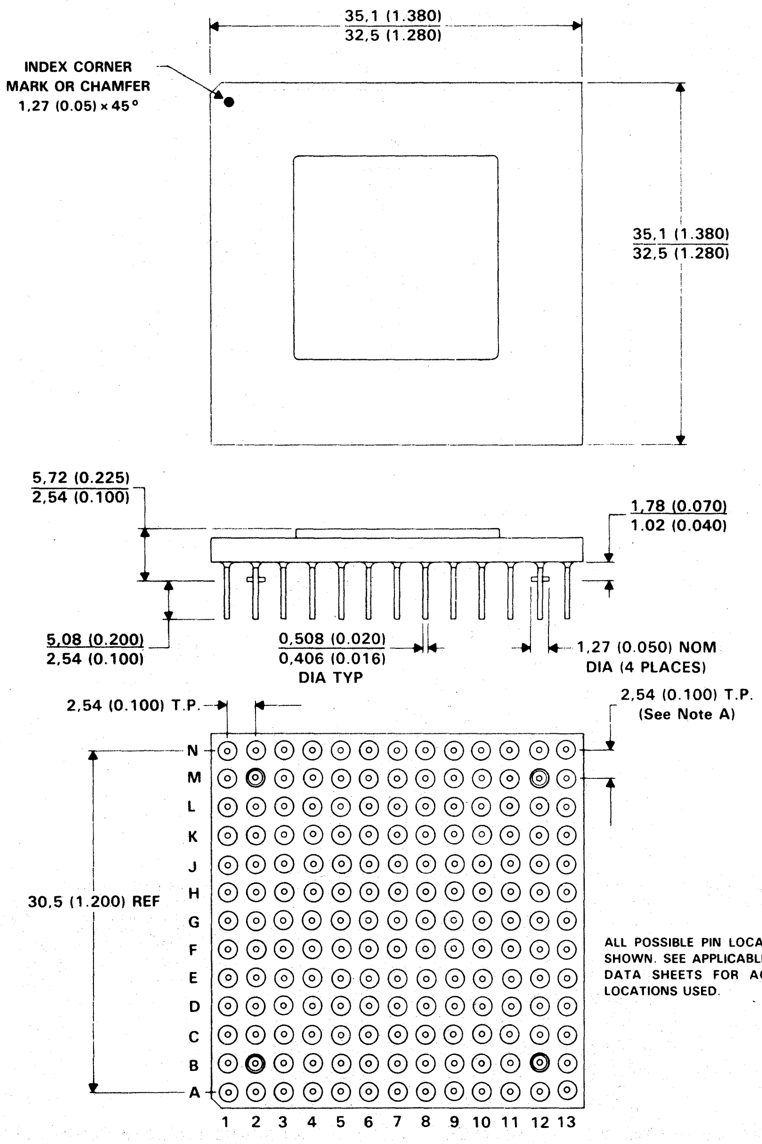
13 × 13 GB pin grid array ceramic package



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

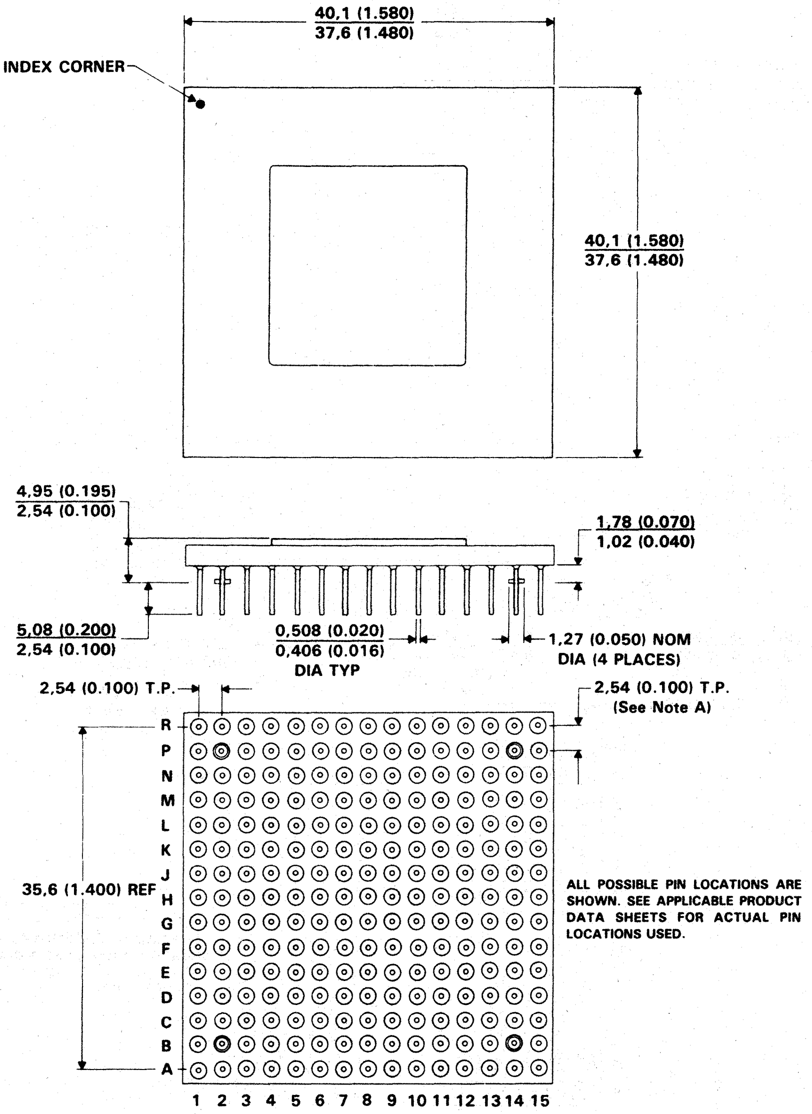


ALL POSSIBLE PIN LOCATIONS ARE SHOWN. SEE APPLICABLE PRODUCT DATA SHEETS FOR ACTUAL PIN LOCATIONS USED.

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

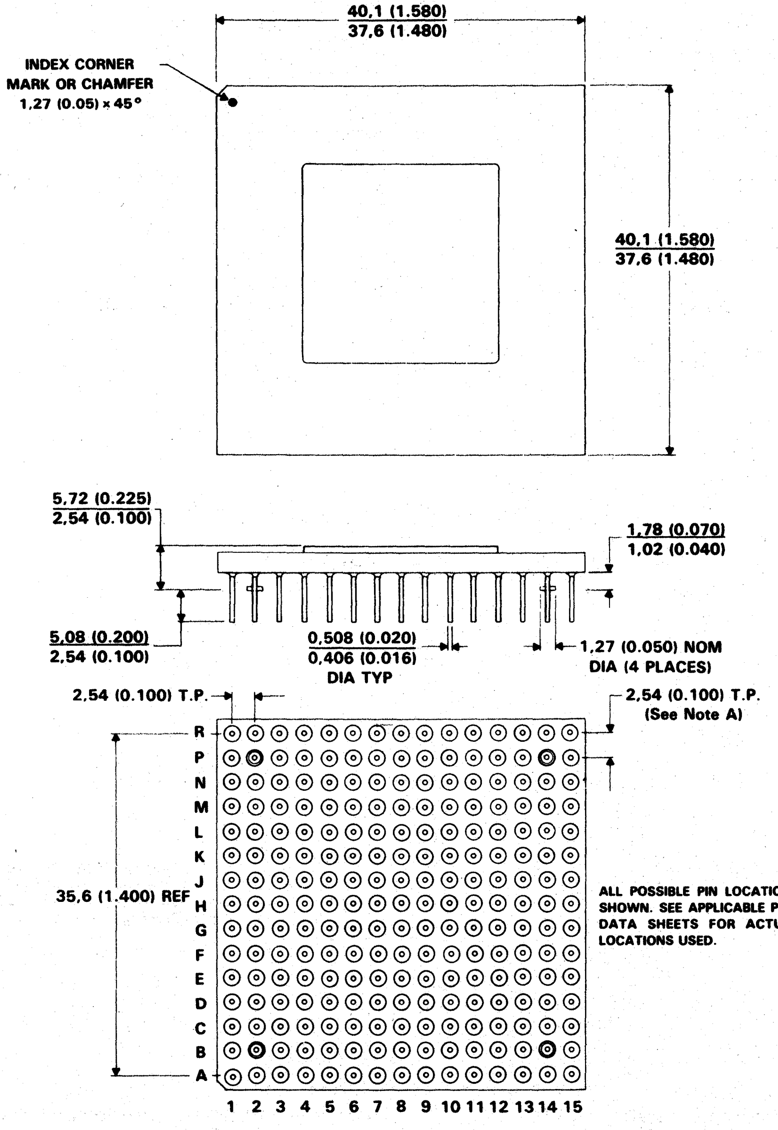
15 × 15 GB pin grid array ceramic package



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

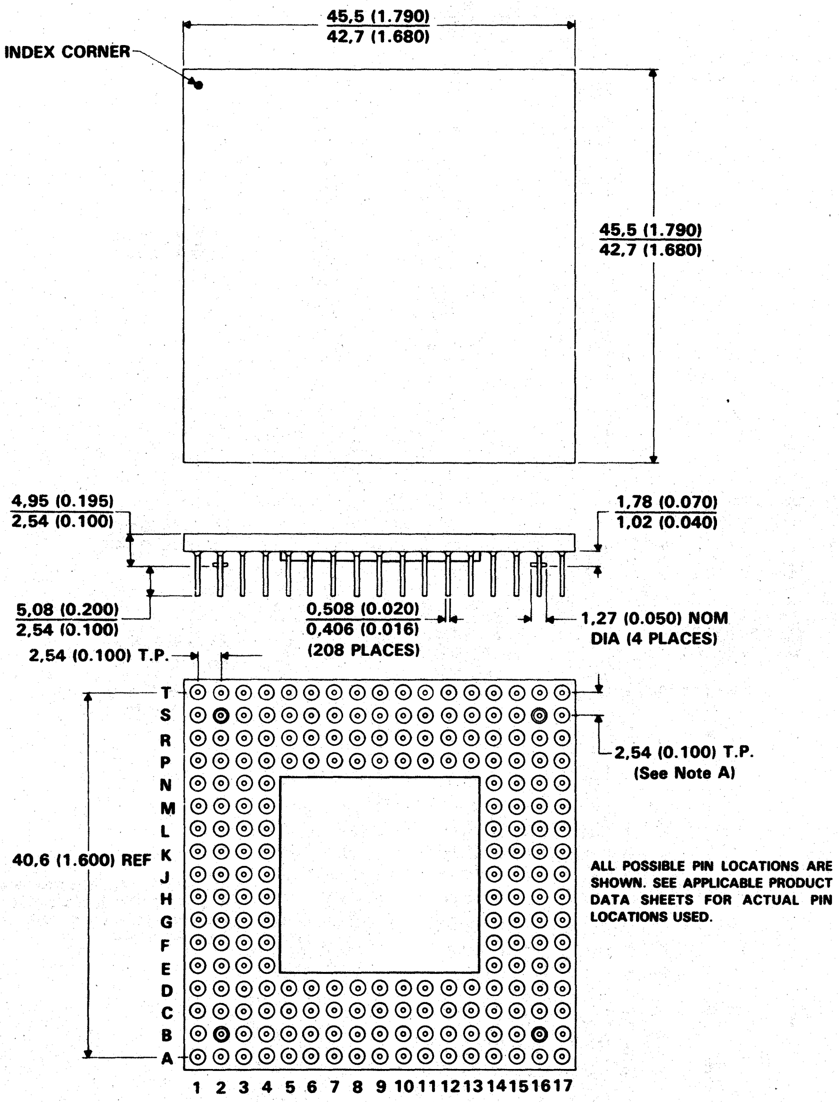


ALL POSSIBLE PIN LOCATIONS ARE SHOWN. SEE APPLICABLE PRODUCT DATA SHEETS FOR ACTUAL PIN LOCATIONS USED.

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

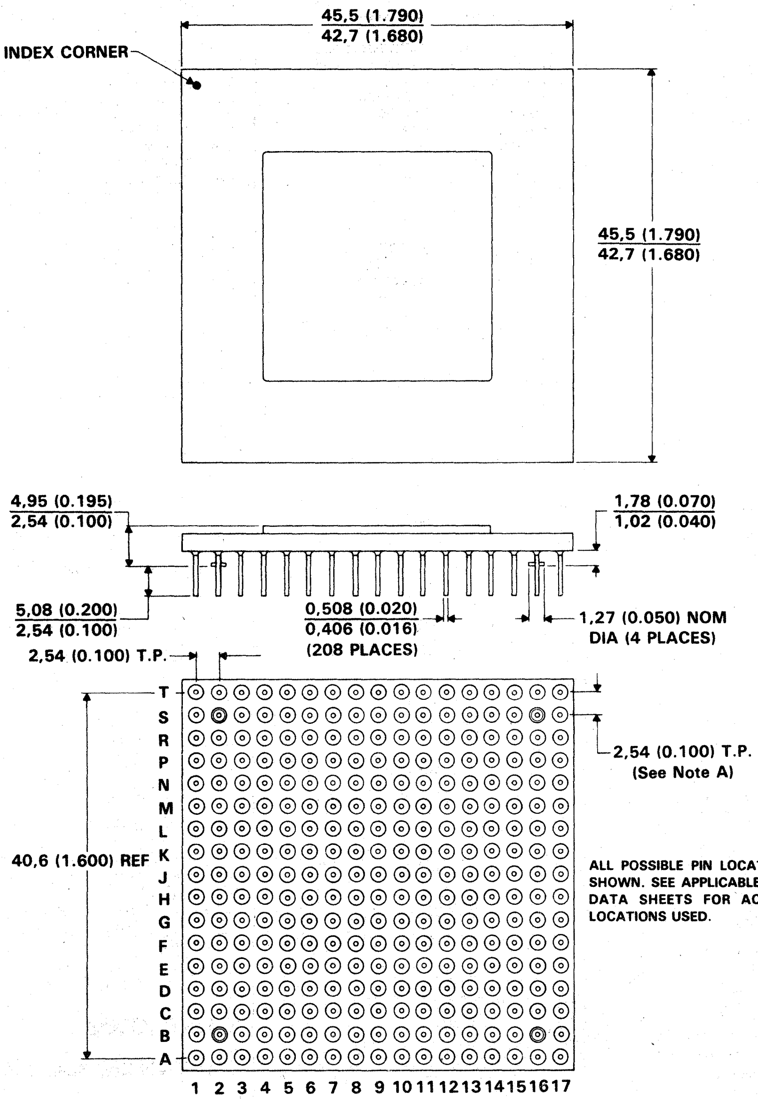
17 × 17 GA pin grid array ceramic package



NOTE A: Pins are located within 0,13 (0.005) radius of true position relative to each other at meximum 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



ALL POSSIBLE PIN LOCATIONS ARE SHOWN. SEE APPLICABLE PRODUCT DATA SHEETS FOR ACTUAL PIN LOCATIONS USED.

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) 624-3276.

CALIFORNIA: Irvine (714) 660-1200;
Sacramento (916) 929-0197;
San Diego (619) 278-9600;
Santa Clara (408) 980-9000;
Torrance (213) 217-7000;
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) 286-0420.

GEORGIA: Norcross (404) 662-7900.

ILLINOIS: Arlington Heights (312) 640-3000.

INDIANA: Carmel (317) 573-6400;
Ft. Wayne (219) 424-5174.

IOWA: Cedar Rapids (319) 395-9550.

KANSAS: Overland Park (913) 451-4511.

MARYLAND: Baltimore (301) 944-8600.

MASSACHUSETTS: Waltham (617) 895-9100.

MICHIGAN: Farmington Hills (313) 553-1500;
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) 527-0930;
Raleigh (919) 876-2725.

OHIO: Beachwood (216) 464-6100;
Dayton (513) 258-3877.

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) 250-6769;
Houston (713) 778-6592; **Richardson** (214) 680-5082;
San Antonio (512) 496-1779.

UTAH: Murray (801) 266-8972.

VIRGINIA: Fairfax (703) 849-1400.

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) 660-8140;
Santa Clara (408) 748-2220;
Torrance (213) 217-7019.

COLORADO: Aurora (303) 368-8000.

GEORGIA: Norcross (404) 662-7945.

ILLINOIS Arlington Heights (313) 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 (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
(617) 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) 997-4874; Wyle (602) 866-2888.

CALIFORNIA: Los Angeles/Orange County:
Arrow/Kierulff (818) 701-7500, (714) 838-5422;
Hall-Mark (818) 716-7300, (714) 669-4100,
(213) 217-8400; Marshall (818) 407-0101, (818) 459-5500,
(714) 458-5395; Schweber (818) 999-4702;
(714) 863-0200, (213) 320-8090; Wyle (213) 322-9953,
(818) 880-9000, (714) 863-9953; Zeus (714) 921-9000;
Sacramento: Hall-Mark (916) 722-8600;
Marshall (916) 635-9700; Schweber (916) 929-9732;
Wyle (916) 638-5282;
San Diego: Arrow/Kierulff (619) 565-4800;
Hall-Mark (619) 268-1201; Marshall (619) 578-9600;
Schweber (619) 450-0454; Wyle (619) 565-9171;
San Francisco Bay Area: Arrow/Kierulff (408) 745-6600,
Hall-Mark (408) 432-0900; Marshall (408) 942-4600;
Schweber (408) 432-7171; Wyle (408) 727-2500;
Zeus (408) 998-5121.

COLORADO: Arrow/Kierulff (303) 790-4444;
Hall-Mark (303) 790-1662; Marshall (303) 451-8383;
Schweber (303) 799-0258; Wyle (303) 457-9953.

CONNETICUT: Arrow/Kierulff (203) 265-7741;
Hall-Mark (203) 269-0100; Marshall (203) 265-3822;
Schweber (203) 748-7080.

FLORIDA: Ft. Lauderdale:
Arrow/Kierulff (305) 429-8200; Hall-Mark (305) 971-9280;
Marshall (305) 977-4880; Schweber (305) 977-7511;
Orlando: Arrow/Kierulff (305) 725-1480, (305) 682-6923;
Hall-Mark (305) 855-4020; Marshall (305) 767-8585;
Schweber (305) 331-7555; Zeus (305) 365-3000;
Tampa: Hall-Mark (813) 530-4543;
Marshall (813) 576-1399.

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) 872-8875; Marshall (317) 297-0483.

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) 840-9450;
Schweber (301) 840-5900; Zeus (301) 997-1118.

MASSACHUSETTS Arrow/Kierulff (617) 935-5134;
Hall-Mark (617) 667-0902; Marshall (617) 658-0810;
Schweber (617) 275-5100, (617) 657-0760;
Time (617) 532-6200; Zeus (617) 863-8800.

MICHIGAN: Detroit: Arrow/Kierulff (313) 971-8220;
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) 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, (609) 235-1900;
Marshall (201) 882-0320, (609) 234-9100;
Schweber (201) 227-7880.

NEW MEXICO: Arrow/Kierulff (505) 243-4566.

NEW YORK: Long Island:
Arrow/Kierulff (516) 231-1000; Hall-Mark (516) 737-0600;
Marshall (516) 273-2424; Schweber (516) 334-7555;
Zeus (914) 937-7400;
Rochester: Arrow/Kierulff (716) 427-0300;
Hall-Mark (716) 244-9290; Marshall (716) 235-7620;
Schweber (716) 424-2222;
Syracuse: Marshall (607) 798-1611.

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) 464-2970;
Columbus: Arrow/Kierulff (614) 436-0928;
Hall-Mark (614) 888-3313;
Dayton: Arrow/Kierulff (513) 435-5563;
Marshall (513) 898-4480; Schweber (513) 439-1800.

OKLAHOMA: Arrow/Kierulff (918) 252-7537;
Schweber (918) 622-8003.

OREGON: Arrow/Kierulff (503) 645-6456;
Marshall (503) 644-5050; Wyle (503) 640-6000.

PENNSYLVANIA: Arrow/Kierulff (412) 856-7000,
(215) 928-1800; GRS Electronics (215) 922-7037;
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) 661-5010; Wyle (214) 235-9953;
Zeus (214) 783-7010;
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) 747-9100; Wyle (206) 453-8300.

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;
Edmonton: Future (403) 438-2858;
Montreal: Arrow Canada (514) 735-5511;
Future (514) 694-7710;
Ottawa: Arrow Canada (613) 226-6903;
Future (613) 820-8313;
Quebec City: Arrow Canada (418) 687-4231;
Toronto: Arrow Canada (416) 672-7769;
Future (416) 638-4771;
Vancouver: Future (604) 294-1166;
Winnipeg: Future (204) 339-0554.

Customer Response Center

TOLL FREE: (800) 232-3200

OUTSIDE USA: (214) 995-6611
(8:00 a.m. – 5:00 p.m. CST)

TEXAS INSTRUMENTS

BU

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.

CALIFORNIA: Irvine: 17891 Cartwright Rd., Irvine, CA 92714, (714) 660-8187; **Sacramento:** 1900 Point West Way, Suite 171, Sacramento, CA 95815, (916) 929-1521; **San Diego:** 4333 View Ridge Ave., Suite B., San Diego, CA 92123, (619) 278-9601; **Santa Clara:** 5353 Betsy Ross Dr., Santa Clara, CA 95054, (408) 980-9000; **Torrance:** 690 Knox St., Torrance, CA 90502, (213) 217-7010; **Woodland Hills:** 21220 Erwin St., Woodland Hills, CA 91367, (818) 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: Ft. Lauderdale: 2765 N.W. 62nd St., Ft. Lauderdale, FL 33309, (305) 973-8502; **Maitland:** 2601 Maitland Center Parkway, Maitland, FL 32751, (305) 660-4600; **Tampa:** 5010 W. Kennedy Blvd., Suite 101, Tampa, FL 33609, (813) 870-6420.

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) 424-5174; **Indianapolis:** 2346 S. Lynhurst, Suite J-400, Indianapolis, IN 46241, (317) 248-8555.

IOWA: Cedar Rapids: 373 Collins Rd. NE, Suite 200, Cedar Rapids, IA 52402, (319) 395-9550.

MARYLAND: Baltimore: 1 Rutherford Pl., 7133 Rutherford Rd., Baltimore, MD 21207, (301) 944-8600.

MASSACHUSETTS: Waltham: 504 Totten Pond Rd., Waltham, MA 02154, (617) 895-9100.

MICHIGAN: Farmington Hills: 33737 W. 12 Mile Rd., Farmington Hills, MI 48018, (313) 553-1500.

MINNESOTA: Eden Prairie: 11000 W. 78th St., Eden Prairie, MN 55344 (612) 828-9300.

MISSOURI: Kansas City: 8080 Ward Pkwy., Kansas City, MO 64114, (816) 523-2500; **St. Louis:** 11816 Borman Drive, St. Louis, MO 63146, (314) 569-7600.

NEW JERSEY: Iselin: 485E U.S. Route 1 South, Parkway Towers, Iselin, NJ 08830 (201) 750-1050

NEW MEXICO: Albuquerque: 2820-D Broadbent Pkwy NE, Albuquerque, NM 87107, (505) 345-2555.

NEW YORK: East Syracuse: 6365 Collamer Dr., East Syracuse, NY 13057, (315) 463-9291; **Endicott:** 112 Nanticoke Ave., P.O. Box 618, Endicott, NY 13760, (607) 754-3900; **Melville:** 1 Huntington Quadrangle, Suite 3C10, P.O. Box 2936, Melville, NY 11747, (516) 454-6600; **Pittsford:** 2851 Clover St., Pittsford, NY 14534, (716) 385-6770; **Poughkeepsie:** 385 South Rd., Poughkeepsie, NY 12601, (914) 473-2900.

NORTH CAROLINA: Charlotte: 8 Woodlawn Green, Woodlawn Rd., Charlotte, NC 28210, (704) 527-0930; **Raleigh:** 2809 Highwoods Blvd., Suite 100, Raleigh, NC 27625, (919) 876-2725.

OHIO: Beachwood: 23408 Commerce Park Rd., Beachwood, OH 44122, (216) 464-6100; **Dayton:** Kingsley Bldg., 4124 Linden Ave., Dayton, OH 45432, (513) 258-3877.

OREGON: Beaverton: 6700 SW 105th St., Suite 110, Beaverton, OR 97005, (503) 643-6758.

PENNSYLVANIA: Ft. Washington: 260 New York Dr., Ft. Washington, PA 19034, (215) 643-6450; **Coraopolis:** 420 Rouser Rd., 3 Airport Office Park, Coraopolis, PA 15108, (412) 771-8550.

PUERTO RICO: Hato Rey: Mercantil Plaza Bldg., Suite 505, Hato Rey, PR 00919, (809) 753-8700.

TEXAS: Austin: P.O. Box 2909, Austin, TX 78769, (512) 250-7655; **Richardson:** 1001 E. Campbell Rd., Richardson, TX 75080, (214) 680-5082; **Houston:** 9100 Southwest Frwy., Suite 237, Houston, TX 77036, (713) 778-6592; **San Antonio:** 1000 Central Parkway South, San Antonio, TX 78232, (512) 496-1779.

UTAH: Murray: 5201 South Green SE, Suite 200, Murray, UT 84107, (801) 266-8972.

VIRGINIA: Fairfax: 2750 Prosperity, Fairfax, VA 22031, (703) 849-1400.

WASHINGTON: Redmond: 5010 148th NE, Bldg B, Suite 107, Redmond, WA 98052, (206) 881-3080.

WISCONSIN: Brookfield: 450 N. Sunny Slope, Suite 150, Brookfield, WI 53005, (414) 785-7140.

CANADA: Nepean: 301 Moodie Drive, Mallorn Center, Nepean, Ontario, Canada, K2H9C4, (613) 726-1970. **Richmond Hill:** 280 Centre St. E., Richmond Hill L4C1B1, Ontario, Canada (416) 884-9181; **St. Laurent:** Ville St. Laurent Quebec, 9460 Trans Canada Hwy., St. Laurent, Quebec, Canada H4S1R7, (514) 335-8392.

ARGENTINA: Texas Instruments Argentina S.A.I.C.F.: Esmeralda 130, 15th Floor, 1035 Buenos Aires, Argentina, 1 + 394-3008.

AUSTRALIA (& NEW ZEALAND): Texas Instruments Australia Ltd.: 6-10 Talavera Rd., North Ryde (Sydney), New South Wales, Australia 2113, 2 + 887-1122; 5th Floor, 418 St. Kilda Road, Melbourne, Victoria, Australia 3004, 3 + 267-4677; 171 Philip Highway, Elizabeth, South Australia 5112, 8 + 255-2066.

AUSTRIA: Texas Instruments Ges.m.b.H.: Industriestrabe B/16, A-2345 Brunn/Gebirge, 2236-846210.

BELGIUM: Texas Instruments N.V. Belgium S.A.: Mercure Centre, Raketstraat 100, Rue de la Fusee, 1130 Brussels, Belgium, 2/720.80.00.

BRAZIL: Texas Instruments Electronicos do Brasil Ltda.: Rua Paes Leme, 524-7 Andar Pinheiros, 05424 Sao Paulo, Brazil, 0815-6166.

DENMARK: Texas Instruments A/S, Mairelundvej 46E, DK-2730 Herlev, Denmark, 2 - 91 74 00.

FINLAND: Texas Instruments Finland OY: Teollisuuskatu 19D 00511 Helsinki 51, Finland, (90) 701-3133.

FRANCE: Texas Instruments France: Headquarters and Prod. Plant, BP 05, 06270 Villeneuve-Loubet, (93) 20-01-01; Paris Office, BP 67 8-10 Avenue Morane-Saulnier, 78141 Velizy-Villacoublay, (3) 946-97-12; Lyon Sales Office, L'Oree D'Ecully, Batiment B, Chemin de la Forestiere, 69130 Ecully, (7) 833-04-40; Strasbourg Sales Office, Le Sebastopol 3, Quai Kleber, 67055 Strasbourg Cedex, (88) 22-12-66; Rennes, 23-25 Rue du Puits Mauger, 35100 Rennes, (99) 31-54-86; Toulouse Sales Office, Le Peripole—2, Chemin du Pigeonnier de la Cepiere, 31100 Toulouse, (61) 44-18-19; Marseille Sales Office, Noilly Paradis—146 Rue Paradis, 13006 Marseille, (91) 37-25-30.

TEXAS
INSTRUMENTS

GERMANY (Fed. Republic of Germany): Texas Instruments Deutschland GmbH: Haggertystrasse 1, D-8050 Freising, 8161 + 80-4591; Kurfuerstendamm 195/196, D-1000 Berlin 15, 30 + 882-7365; III, Hagen 43/Kibbelstrasse, .19, D-4300 Essen, 201-24250; Frankfurter Allee 6-8, D-6236 Eschborm 1, 06196 + 8070; Hamburgerstrasse 11, D-2000 Hamburg 76, 040 + 220-1154, Kirchhorsterstrasse 2, D-3000 Hannover 51, 511 + 648021; Maybachstrabe 11, D-7302 Ostfildern 2-Nelingen, 711 + 547001; Mixikoring 19, D-2000 Hamburg 60, 40 + 637 + 0061; Postfach 1309, Roonstrasse 16, D-5400 Koblenz, 261 + 35044.

HONG KONG (+ PEOPLES REPUBLIC OF CHINA): Texas Instruments Asia Ltd., 8th Floor, World Shipping Ctr., Harbour City, 7 Canton Rd., Kowloon, Hong Kong. 3 + 722-1223.

IRELAND: Texas Instruments (Ireland) Limited: Brewery Rd., Stillorgan, County Dublin, Eire, 1 831311.

ITALY: Texas Instruments Semiconduttori Italia Spa: Viale Delle Scienze, 1, 02015 Cittaducale (Rieti), Italy, 746 694.1; Via Salaria KM 24 (Palazzo Cosma), Monterotondo Scalo (Rome), Italy, 6 + 9003241; Viale Europa, 38-44, 20093 Cologno Monzese (Milano), 2 2532541; Corso Svizzera, 185, 10100 Torino, Italy, 11 774545; Via J. Barozzi 6, 40100 Bologna, Italy, 51 355851.

JAPAN: Texas Instruments Asia Ltd.: 4F Aoyama Fuji Bldg., 6-12, Kita Aoyama 3-Chome, Minato-ku, Tokyo, Japan 107, 3-498-2111; Osaka Branch, 5F, Nissho Iwai Bldg., 30 Imabashi 3- Chome, Higashi-ku, Osaka, Japan 541, 06-204-1881; Nagoya Branch, 7F Daini Toyota West Bldg., 10-27, Meieki 4-Chome, Nakamura-ku Nagoya, Japan 450, 52-583-8691.

KOREA: Texas Instruments Supply Co.: 3rd Floor, Samon Bldg., Yuksam-Dong, Gangnam-ku, 135 Seoul, Korea, 2 + 462-8001.

MEXICO: Texas Instruments de Mexico S.A.: Mexico City, AV Reforma No. 450 — 10th Floor, Mexico, D.F., 06600, 5 + 514-3003.

MIDDLE EAST: Texas Instruments: No. 13, 1st Floor Mannai Bldg., Diplomatic Area, P.O. Box 26335, Manama Bahrain, Arabian Gulf, 973 + 274681.

NETHERLANDS: Texas Instruments Holland B.V., P.O. Box 12995, (Bullewijk) 1100 CB Amsterdam, Zuid-Oost, Holland 20 + 5602911.

NORWAY: Texas Instruments Norway A/S: PB106, Refstad 131, Oslo 1, Norway, (2) 155090.

PHILIPPINES: Texas Instruments Asia Ltd.: 14th Floor, Ba- Lepanto Bldg., 8747 Paseo de Roxas, Makati, Metro Manila, Philippines, 2 + 8188987.

PORTUGAL: Texas Instruments Equipamento Electronico (Portugal), Lda.: Rua Eng. Frederico Ulrich, 2650 Moreira Da Maia, 4470 Maia, Portugal, 2-948-1003.

SINGAPORE (+ INDIA, INDONESIA, MALAYSIA, THAILAND): Texas Instruments Asia Ltd.: 12 Lorong Bakar Batu, Unit 01-02, Kolam Ayer Industrial Estate, Republic of Singapore, 747-2255.

SPAIN: Texas Instruments Espana, S.A.: C/Jose Lazaro Galdiano No. 6, Madrid 16, 1/458.14.58.

SWEDEN: Texas Instruments International Trade Corporation (Sverigefilialen): Box 39103, 10054 Stockholm, Sweden, 8 - 235480.

SWITZERLAND: Texas Instruments, Inc., Reidstrasse 6, CH-8953 Dietikon (Zuerich) Switzerland, 1-740 2220.

TAIWAN: Texas Instruments Supply Co.: Room 903, 205 Tun Hwan Rd., 71 Sung-Kiang Road, Taipei, Taiwan, Republic of China, 2 + 521-9321.

UNITED KINGDOM: Texas Instruments Limited: Manton Lane, Bedford, MK41 7PA, England, 0234 67466; St. James House, Wellington Road North, Stockport, SK4 2RT, England, 61 + 442-7162.

BM

Erratum to SN74ACT8837 User's Guide

Format conversion from floating point to integer for numbers between 1.111 . . . 111 × 2^{31} and 2^{32}, when rounding up, causes the resulting integer to go to zero without causing the device to signal an overflow. This condition occurs whenever rounding causes the integer operand to be incremented. If the user anticipates that this condition may occur in a system being designed, a software trap can be implemented to monitor correct operation of the format conversion.

TEXAS
INSTRUMENTS

Printed in U.S.A.
788—12