SDBU001A

SN74AS888 SN74AS890 Bit-Slice Processor User's Guide

8-Bit Family

SN74AS888/SN74AS890 User's Guide

TEXAS INSTRUMENTS

SN74AS888/SN74AS890 Bit-Slice Processor

User's Guide

TEXAS INSTRUMENTS

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes in the devices or the device specifications identified in this publication without notice. TI advises its customers to obtain the latest version of device specifications to verify, before placing orders, that the information being relied upon by the customer 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 such testing necessary to support this warranty. Unless mandated by government requirements, specific testing of all parameters of each device is not necessarily performed.

In the absence of written agreement to the contrary, TI assumes no liability for TI applications assistance, customer's product design, or infringement of patents or copyrights of third parties by or arising from use of semiconductor devices described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor devices might be or are used.

First printed April 1985
Revised August 1985

TABLE OF CONTENTS

LIST OF APPENDICES

APPENDIX **PAGE**

LIST OF ILLUSTRATIONS

FIGURE **PAGE**

LIST OF TABLES

1 Introduction

With the introduction of the 'AS888, Texas Instruments Incorporated offers an LSI building block that can be cascaded to form an ALU of any word width with a significant increase in efficiency and speed over older 4-bit-slice systems. The 8-bit slice and its companion microsequencer, the 'AS890, increase processing throughput per unit area to an extent never before realized in bit-slice systems.

These innovations are the result of a new Texas Instruments technology called IMPACT. The new processing technique reduced feature size to two microns, enabling the development of about six to eight times the number of gates possible with Schottky and low-power Schottky TTL. The increased gate density permitted expansion of the slice to an 8-bit width and the development of special on-board circuitry for decoding high-level operations into microoperations. The result is a flexible, multi-function chip that provides rapid multiplication and division; supports sign-magnitude, BCD, excess-3, single- or double-precision arithmetic; and offers additional specialized features such as operations on selected bits or bytes.

This section of the User's Guide introduces the 'AS888 and 'AS890 and outlines the support tools available for system development. Section 2 looks at the architecture and instruction set of the 'AS888. The microsequencer is the subject of section 3, beginning with a functional description of the chip and looking at its flexible instruction set. Possible applications for the 'AS888/'AS890 are explored in sections 4 and 5. The first approaches high-speed CPU design using the 8-bit slice; the second develops a design for a floating point processor.

1.1 Understanding Bit-Slice Architecture

Figure 1.1 illustrates a simple bit-slice system. The three basic components are an arithmetic/logic unit, a sequencer and a memory. The program that resides in this memory is commonly called the microprogram, while the memory 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), while the sequencer is dedicated to generating the next address to the micromemory. The ALU and sequencer 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 sequencer. Unlike a microprocessor opcode, 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 parallelism offers greater speed and flexibility than a typical microprocessor, although the microinstruction serves the same purpose as a microprocessor instruction: it specifies control information by which the user is able to implement desired data processing operations in a desired 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.

1.2 The 'AS888 8-bit processor slice

The 'AS888 is engineered to support high-speed, high-level operations. The slice, described in detail in section 2, contains an 8-bit ALU, a 16-word by 8-bit register file, two shifters to support double-precision arithmetic and three independent, bidirectional data ports.

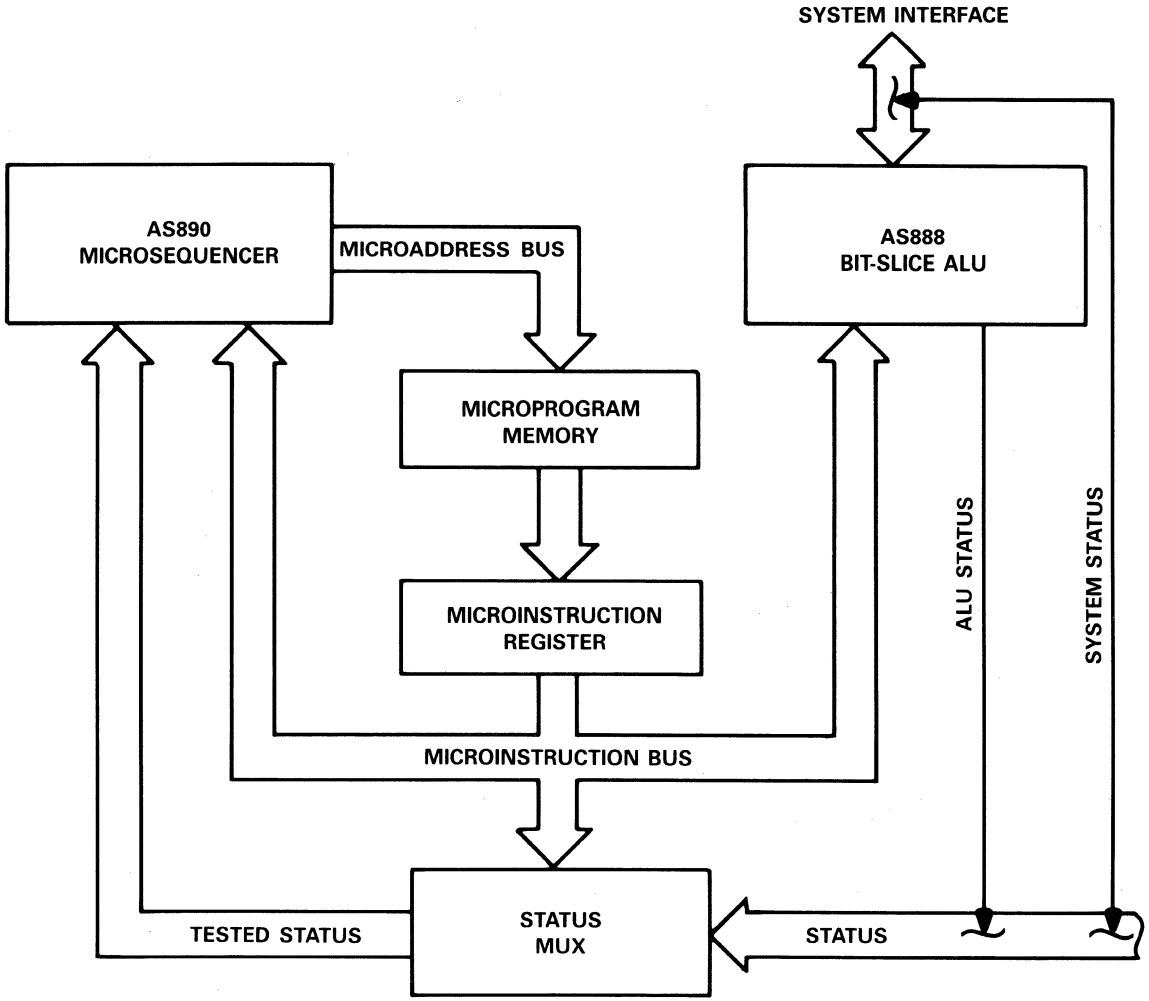


Figure 1-1. Bit-Slice System Block Diagram

The slice's thirteen 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.

The chip's configuration enhances processing throughout 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.

1.3 'AS890 Microsequencer

To complement these innovations in bit-slice processor technology, Texas Instruments also developed the 'AS890. Implemented with Advanced Schottky and Schottky-transistor logic, the microsequencer performs double-nested loops, multiway branching, do-while loops, compound if...then...else expressions, interrupt processing and other complex instructions. Fast memory devices and the high-speed microsequencer make possible to construct from a handful of components a bit-slice system executing high-level operations.

Like the 'AS888, the 'AS890 is expandable. The 9-word stack can be increased externally using a stack status pin. Two register/counters can be read or loaded externally to permit operations such as array indexing while looping in a microprogram.

Diagnostics are also supported. Conditions that cause stack overflow can be traced by reading the stack, reducing software development time and monitoring runtime errors.

1.4 Support Tools

Texas Instruments provides a low-cost, real-time development and evaluation module (EVM) to aid initial hardware and software design. The 16-bit, self-contained system provides a quick and easy way to test and debug simple microcode, allowing software and hardware evaluation at or near rated execution speeds.

The EVM incorporates a single-chip 8-bit microcomputer to handle user interface and communications. An EPROM-based monitor program gives the user complete control over all important functions, registers and buses of the target system, as well as the high-speed writable control store. Further information is given in the document, *74AS EVM-1 Bit-Slice Evaluation System User's Guide.*

1.5 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 courses for bit-slice design using the 'AS888 and 'AS890 can be obtained from the following Regional Technology Centers:

Atlanta
Texas Instruments Incorporated
3300 N.E. Expressway, Building 8
Atlanta, GA 30341
404/452-4682

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

Boston
Texas Instruments Incorporated
400-2 Totten Pond Rd.
Waltham, MA 02154
617/890-6671

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

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

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

The VLSI Systems Engineering Group maintains a computer bulletin board to assist bit-slice users. The board can be accessed by dialing 214/995-4569.

1.6 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 214/995-4720.

2 'AS888 8-Bit Processor Slice

The 'AS888 is an 8-bit ALU/register slice designed for use in high-performance digital computers or controllers. Slices can be cascaded to any word width 16 bits or greater.

Key elements include a 16-word by 8-bit register file and a high-speed ALU. Three independent 4-bit port addresses allow a two-operand fetch and an operand write to be performed at the register file simultaneously. The 8-bit ALU can perform seven arithmetic and six logical instructions, followed by conditional arithmetic, logical or circular shifts. The result can be returned to the register file or output through the Y port.

The ALU also supports a wide range of arithmetic and logical functions, such as multiplication, division, normalization, add and subtract immediate, cyclic redundancy character accumulation, and data conversions such as BCD, excess-3, USASCII and sign magnitude. Double precision operations can be implemented using a multiplier-quotient register and shifter designed to operate alone or in parallel with the register file and ALU shifter.

An internal ALU bypass path increases the speeds of multiply, divide and normalize instructions by eliminating many common types of test and branch instructions. The path is also used by 'AS888 instructions that permit bits and bytes to be manipulated.

2.1 Architecture

2.1.1 Data Flow

Data flow through the 'AS888 is shown in Figure 2-1. Data enters the chip 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 R and S buses serving the ALU. Data enters the ALU through two multiplexers: R MUX, which selects the R bus operand from the DA port or the register file addressed by A3-A0; and S MUX, which selects data from the DB port, the register file addressed by B3-B0, or the multiplier-quotient (MQ) register.

The result of the ALU operation is passed on the F bus to the ALU shifter, where it can be shifted or passed without shift to the Y bus for output from the 'AS888 and/or storage in the internal register file. The MQ shifter, which operates in parallel with the ALU shifter, can be loaded from the ALU via the F bus, 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.

Data can be output from three bidirectional ports: the Y port and the DA and DB ports. DA and DB can be used to read ALU input data on the R and S buses for debug or other special purposes.

2.1.2 Architectural Elements

Figure 2-2 is a functional diagram of the 'AS888. Key elements of the slice are discussed below.

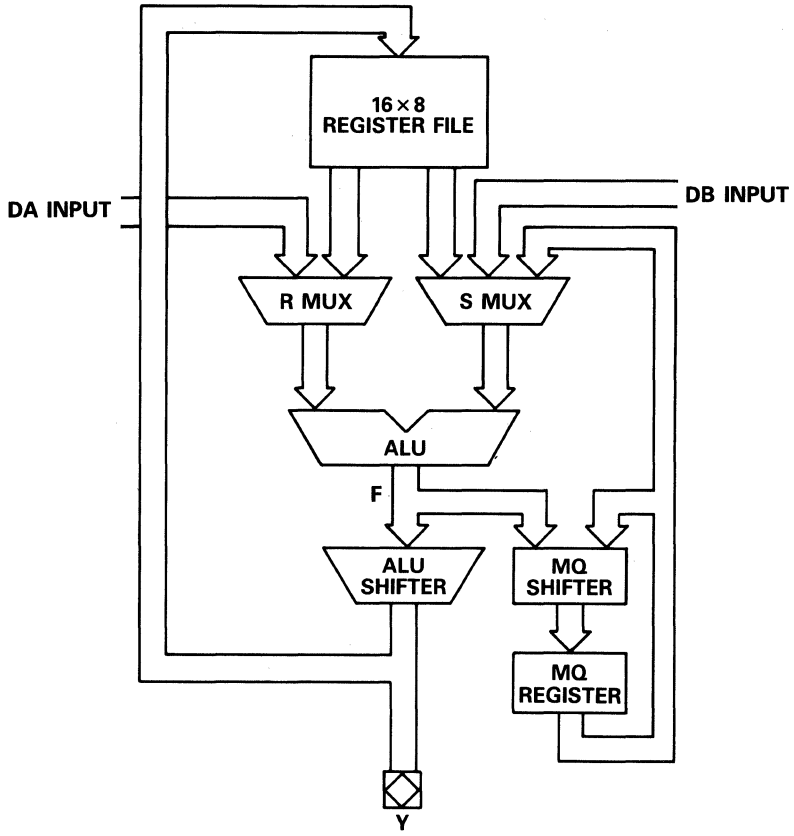


Figure 2-1. Internal Data Flow for 'AS888

2.1.2.1 Three-Port Register File

Sixteen 8-bit registers are accessed by three address ports. C3-C0 address the destination register during write operations; A3-A0 and B3-B0 address any two registers during read operations. Data is written into the register file when $\overline{WE}$ is low and a low-to-high clock transition occurs. Under certain conditions, the address buses are used to furnish immediate data to the ALU: A3-A0 to provide constant data for the add and subtract immediate instructions; A3-A0 and C3-C0 to provide masks for set, reset and test bit operations.

2.1.2.2 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 2-1.

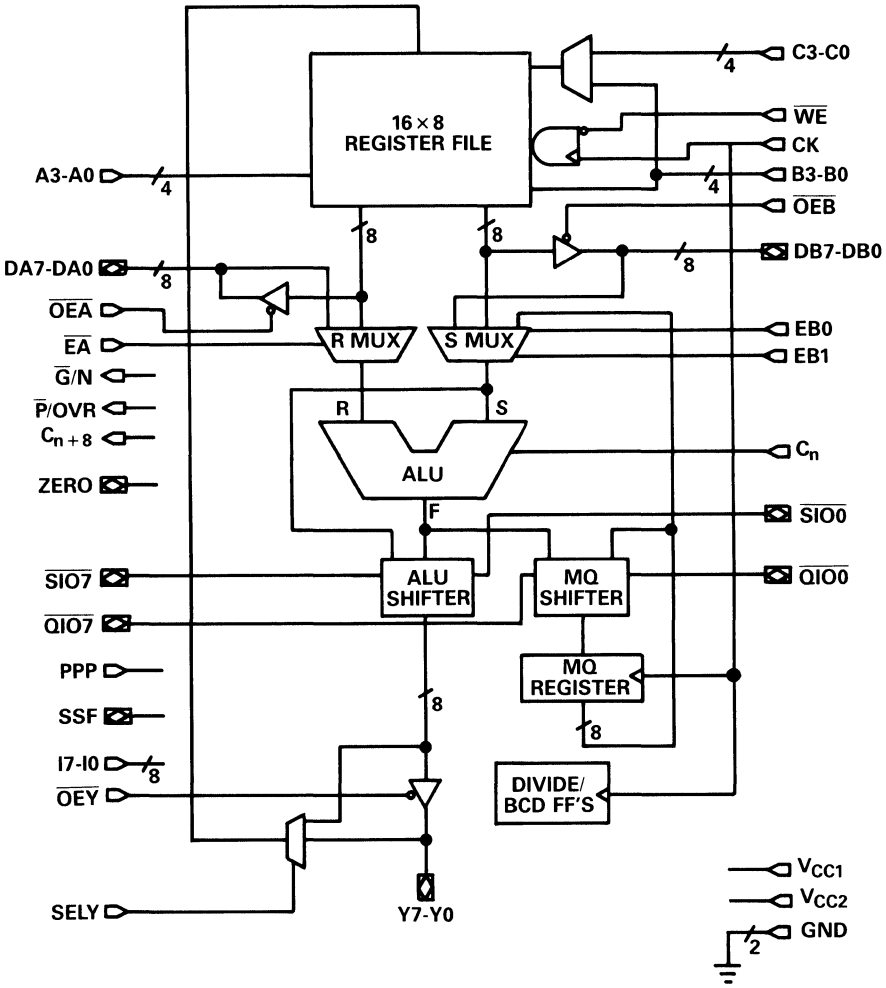


Figure 2-2. Functional Block Diagram of 'AS888

Table 2-1. ALU Source Operand Selects

R-BUS OPERAND SELECT $\overline{EA}$	S-BUS OPERAND SELECT EB1-EB0	RESULT DESTINATION ← SOURCE OPERAND
0		R bus ← Register File addressed by A3-A0
1		R bus ← DA port
	0 0	S bus ← Register File addressed by B3-B0
	0 1	S bus ← MQ Register
	1 0	S bus ← DB Port
	1 1	S bus ← MQ Register

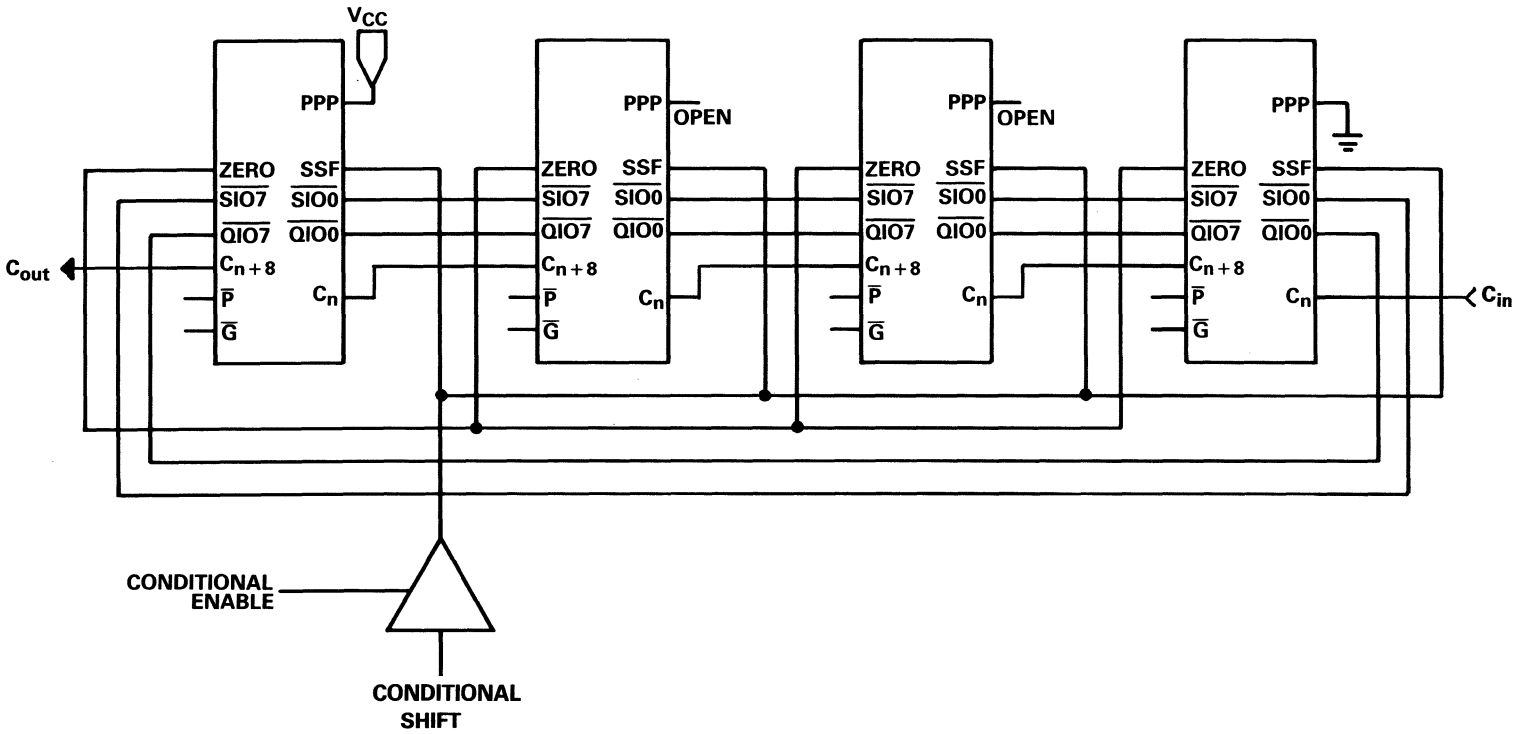


Figure 2-3. Essential 'AS888 Interconnections

2.1.2.3 *DA and DB Buses*

The DA and DB buses can be used to read S bus or R bus inputs from the register file or to load the S bus and/or R bus directly from an external source. See Tables 2-1 and 2-2 for the selects and enables which affect DA and DB.

Table 2-2. Destination Operand Select/Enables

REGISTER FILE WRITE ENABLE $\overline{WE}$	Y BUS OUTPUT ENABLE $\overline{OEY}$	Y BUS SELECT SELY	DA PORT OUTPUT ENABLE $\overline{OEA}$	DB PORT OUTPUT ENABLE $\overline{OEB}$	RESULT DESTINATION ← SOURCE OPERAND
0	0	X			Y port and Register File ← ALU Shifter
1	0	X			Y port ← ALU Shifter
0	1	0			Register File ← ALU Shifter
0	1	1			Register File ← Y port
			1		DA port ← R bus
			0		DA port ← Hi-Z
				1	DB port ← S bus
				0	DB port ← Hi-Z

2.1.2.4 *ALU*

The ALU can perform seven arithmetic and six logical instructions on two 8-bit operands. It also supports multiplication, division, normalization, bit and byte operations and data conversion, including excess-3 BCD arithmetic. The 'AS888 instruction set is discussed in section 2.2 and presented in detail in section 2.5.

2.1.2.5 *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 exclusively for single precision or concurrently for double precision shifts. Shifts can be made conditional, using the Special Shift Function (SSF) pin.

The shifters of adjacent slices are connected by four bidirectional pins: $\overline{SIO0}$ and $\overline{SIO7}$, $\overline{QIO0}$ and $\overline{QIO7}$. These pins allow serial data to be shifted between packages and also serve to transfer data between the MQ and ALU shifters for double precision and other operations. Figure 2-3 shows four interconnected packages. The shift pins on all cascaded 'AS888s must be wired as shown in the figure and in Table 2-3.

Status connections will vary according to system design. The system shown uses ripple carry. For large word widths (four or more slices), a look-ahead carry generator may be desired. This can be implemented using the generate ($\overline{G}$) and propagate ($\overline{P}$) signals. A schematic using carry look-ahead can be found in Section 4.

2.1.2.6 *MQ Register*

The MQ register and the MQ shifter function as a shift register and can be loaded from the ALU, register file or external data buses. The register has specific functions in multiplication, division, and data conversion and can also be used as a temporary storage register.

Table 2-3. Required 'AS888 Shift Pin Connections (External)

INTERMEDIATE PACKAGES	END PACKAGES
$\overline{SIO7}$ to $\overline{SIO0}$ of next most significant package	$\overline{SIO7}$ on most significant package to $\overline{SIO0}$ of least significant package
$\overline{QIO7}$ to $\overline{QIO0}$ of next most significant package	$\overline{QIO7}$ on most significant package to $\overline{QIO0}$ of least significant package
$\overline{SIO0}$ to $\overline{SIO7}$ of next least significant package $\overline{QIO0}$ to $\overline{QIO7}$ of next least significant package	

2.1.2.7 Y Bus

The Y bus contains the output of the ALU shifter if $\overline{OEY}$ is low and can be used as an input if $\overline{OEY}$ is high. SELY controls the flow of data to the register file. If SELY is low, ALU shifter output will be passed to the register file; if SELY is high, the Y port becomes an input to the register file.

2.1.2.8 Status

Four status signals are generated by the most significant slice: overflow (OVR), sign (N), carry-out (C_{n+8}) and ZERO. C_{n+8} indicates carry-out of the ALU, regardless of shift. OVR, N and ZERO indicate status from the ALU shifter. ZERO must be wire-ANDed as shown in Figure 2-3.

2.1.2.9 Package Position Pin

The package position pin (PPP) defines the position of the slice in the system. Intermediate positions are selected by leaving the pin open. Tying the pin to V_{CC} makes the slice the most significant package; tying the pin to GND makes it the least significant.

2.1.2.10 Special Shift Function Pin

Conditional shifting algorithms may be implemented using the SSF pin under hardware or firmware control. SSF is a bidirectional pin and is used in certain 'AS888 instructions to transmit information between slices, eliminating many types of test and branch instructions. During multiplication, for example, the least significant bit of the multiplier determines whether an add/shift or shift operation is to be performed. In this case, the SSF pin of the least significant package is used as an output pin, while all other packages become input pins. Similarly, during normalization, the required operation depends on whether the two most significant bits of the operand are the same or different. Here, the SSF pin of the most significant package becomes an output pin while those on all other packages become input pins.

During instructions that force the SSF pin during execution, SSF must be left in the high-Z state, as shown in Figure 2-3. Use of SSF is discussed for individual instructions in section 2.5.

2.1.2.11 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. These flip-flops are not directly accessible by the user.

2.2 Instruction Set Overview

Bits I7-I0 are used as instruction inputs to the slice. Instructions are summarized in Tables 2-4 and 2-5. Table 2-6 lists all instructions, divided into five groups, with their opcodes and mnemonics. 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.

A brief overview of the instruction set follows. Details about individual instructions, including operand, status and control information, can be found in section 2.5.

Table 2-4. Combined 'AS888 Arithmetic-Logical/Shift Operations

GROUP 1		GROUP 2	
Arithmetic		**Shift ALU**	**Shift MQ Register**
Add	$R + S + C_n$	Arithmetic Right	Arithmetic Right
Subtract	$\overline{R} + S + C_n$	Arithmetic Left	Logical Right
	$R + \overline{S} + C_n$	Logical Right	Logical Left
Increment	$R + C_n$	Circular Left	Circular Left
	$S + C_n$	Circular Right	
	$\overline{R} + C_n$		**Load MQ Register**
	$\overline{S} + C_n$	**Shift ALU and MQ Register**	Load MQ with ALU
Logical	R AND S	Arithmetic Right	
	$\overline{R}$ AND S	Arithmetic Left	**Pass ALU Result Unshifted**
	R OR S	Logical Right	Pass ALU to specified output
	R XOR S	Circular Left	destination without shift
	R NAND S	Circular Right	
	R NOR S		

Table 2-5. Other 'AS888 Instructions

GROUPS 3–5		
Arithmetic Operations	**Bit Operations**	**Data Conversion**
Add Immediate	Set Bit	Absolute Value
Subtract Immediate	Reset Bit	Sign Magnitude/Two's Complement
Signed Divide	Test Bit (One)	Single Length Normalize
Signed Divide Initialize	Test Bit (Zero)	Double Length Normalize
Signed Divide Overflow Check		BCD to Binary
Signed Divide Start	**Byte Operations**	Binary to Excess-3
Signed Divide Iterate	Add R to S	Excess-3 Byte Correction
Signed Divide Terminate	Subtract S from R	Excess-3 Word Correction
Signed Divide Quotient Fix	Subtract R from S	
Divide Remainder Fix	Increment S	**Other**
Unsigned Divide	Increment Negative S	Select S or R
Unsigned Divide Start	XOR R and S	
Unsigned Divide Iterate	AND R and S	
Unsigned Divide Terminate	OR R and S	
Divide Remainder Fix		
Multiply		
Signed Multiply Iterate		
Signed Multiply Terminate		
Unsigned Multiply Iterate		

2.2.1 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. Group 1 logic instructions do not use carry-in; carry-out is forced to zero.

Fourteen single- and double-precision shifts can be specified, or the ALU result can be passed unshifted to the MQ shifter or to the specified output destination by using the LOADMQ or PASS instructions. Table 2-7 summarizes possible shift instructions. 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 register file for passage to the ALU shifter.

All shift operations require that cascaded packages be wired as shown in Figure 2-3.

Table 2-6. 'AS888 Instruction Set

GROUP 1 INSTRUCTIONS		
INSTRUCTION BITS (I3-I0) HEX CODE	**MNEMONIC**	**FUNCTION**
0		Used to access Group 4 instructions
1	ADD	$R + S + C_n$
2	SUBR	$\overline{R} + S + C_n$
3	SUBS	$R + \overline{S} + C_n$
4	INCS	$S + C_n$
5	INCNS	$\overline{S} + C_n$
6	INCR	$R + C_n$
7	INCNR	$\overline{R} + C_n$
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
GROUP 2 INSTRUCTIONS		
INSTRUCTION BITS (I7-I4) HEX CODE	**MNEMONIC**	**FUNCTION**
0	SRA	Arithmetic Right Single
1	SRAD	Arithmetic Right Double
2	SRL	Logical Right Single
3	SRLD	Logical Right Double
4	SLA	Arithmetic Left Single
5	SLAD	Arithmetic Left Double
6	SLC	Circular Left Single
7	SLCD	Circular Left Double
8	SRC	Circular Right Single
9	SRCD	Circular Right Double
A	MQSRA	Pass (Y ← F) and Arithmetic Right MQ
B	MQSRL	Pass (Y ← F) and Logical Right MQ
C	MQSLL	Pass (Y ← F) and Logical Left MQ
D	MQSLC	Pass (Y ← F) and Circular Left MQ
E	LOADMQ	Pass (Y ← F) and Load MQ (MQ ← F)
F	PASS	Pass (Y ← F)

Table 2-6. 'AS888 Instruction Set (Continued)

GROUP 3 INSTRUCTIONS		
INSTRUCTION BITS (I7-I0) HEX CODE	**MNEMONIC**	**FUNCTION**
08	SET1	Set Bit
18	SET0	Reset Bit
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
GROUP 4 INSTRUCTIONS		
00		Reserved
10	SEL	Select S/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
GROUP 5 INSTRUCTIONS		
0F	CLR	Clear
1F	CLR	Clear
2F	CLR	Clear
3F	CLR	Clear
4F	CLR	Clear
5F	CLR	Clear
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

Table 2-7. 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
Arithmetic left	May lose the sign bit if an overflow occurs. A 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 set externally
Logical left	Fills a zero in the least significant bit position unless the bit is set externally

2.2.2 Other Arithmetic Instructions

The 'AS888 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 section 2.3.2 and include: signed multiplication, which performs an $8N+2$ clock two's complement multiplication; unsigned multiplication, which produces an unsigned times unsigned product in $8N+2$ clocks; and mixed multiplication which multiplies a signed multiplicand by an unsigned multiplier to produce a signed result in $8N+2$ clocks, where N is the number of cascaded packages.

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 (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 section 2.3.1. These use a nonrestoring technique to divide a 16N-bit integer dividend by an 8-bit integer divisor to produce an 8N-bit integer quotient and remainder.

2.2.3 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. Zeros are shifted 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.

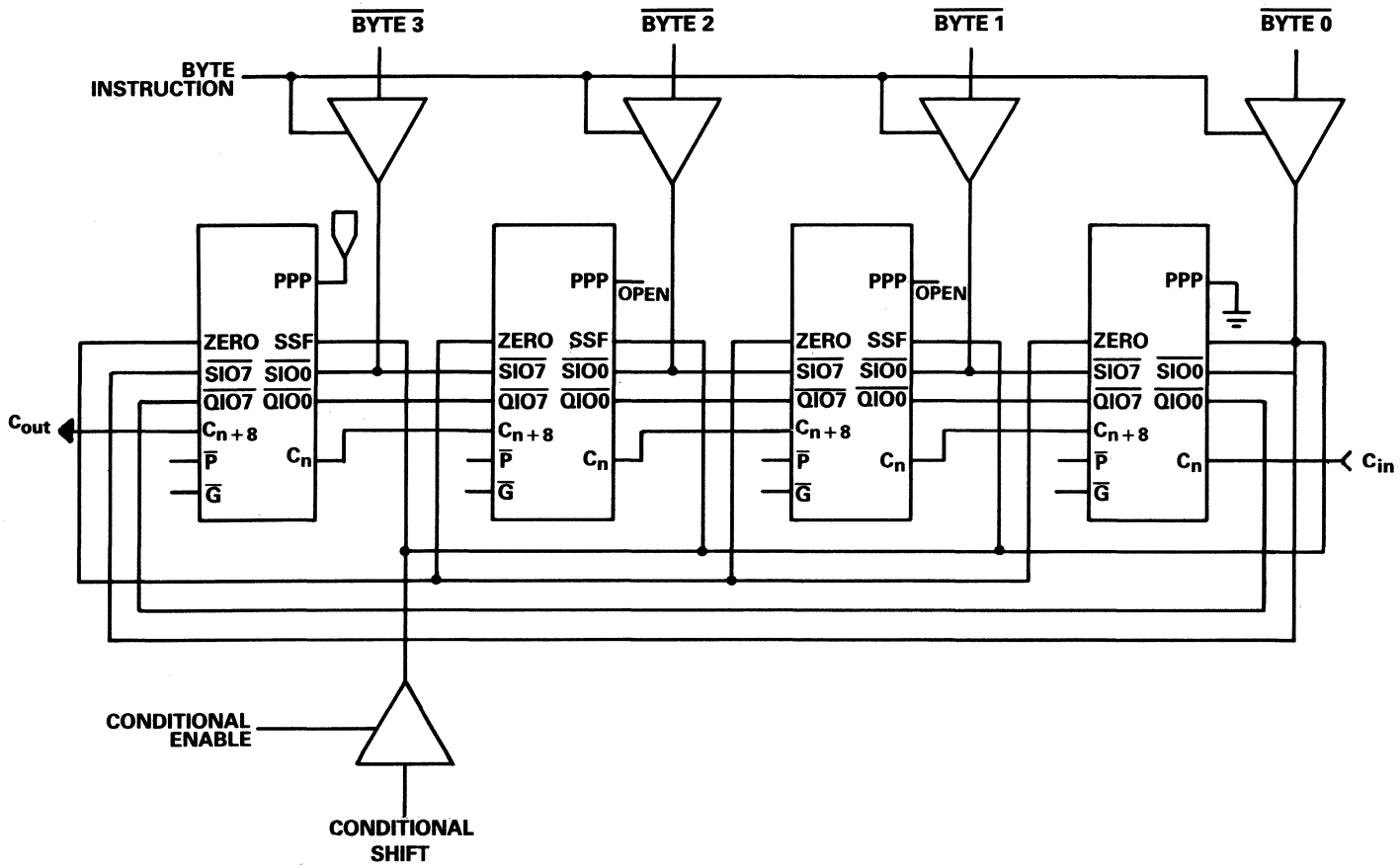


Figure 2-4. 'AS888 Package Connections for Bit and Byte Instructions

Data stored in binary-coded decimal form can be converted to binary using BCDBIN (Group 5). A routine for this conversion, which accompanies the discussion of BCDBIN in section 2.5, allows the user to convert an N-digit BCD number to a 4N-bit binary number in 4N + 8 clock cycles.

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 in 2N + 3 clocks; N is the number of cascaded packages. For an algorithm, see the BINEX3 entry in section 2.5.

2.2.4 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 concatenation of register file address ports C3-C0 and A3-A0. The register file addressed by B3-B0 is used as the source and destination for the test bit instructions and as the destination operand for the set bit instructions. Bytes to be operated on are selected by forcing $\overline{SIO0}$ low.

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 $\overline{SIO0}$ low, but multiple bytes can be operated on only if they are adjacent to one another; at least one byte must be non-selected.

To implement bit and byte instructions, tri-state drivers must be connected to the slices, as shown in Figure 2.4.

2.2.5 Other Instructions

SEL (Group 4) selects one of the ALU's two operands, 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.

CLR (Group 5) 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.

2.3 Divison and Multiplication

2.3.1 Division

Ten 'AS888 instructions support binary division of signed or unsigned integers:

Instruction Code (I7-I0) (hex)	Instruction
B0	Unsigned Divide Start (UDIVIS)
C0	Unsigned Divide Iterate (UDIVI)
F0	Unsigned Divide Terminate (UDIVIT)
80	Signed Divide Initialize (SDIVIN)
AF	Signed Divide Overflow Test (SDIVO)
90	Signed Divide Start (SDIVIS)
A0	Signed Divide Iterate (SDIVI)
E0	Signed Divide Terminate (SDIVIT)
40	Divide Remainder Fix (DIVRF)
50	Signed Divide Quotient Fix (SDIVQF)

These are designed for use with an efficient division algorithm known as nonrestoring division:

1. Subtract the divisor from the dividend.
2. If the result is positive, then
   a. Set the quotient bit
   b. Shift the result
   c. Go to Step 1.
3. If the result is negative, then
   a. Clear the quotient bit
   b. Shift the result
   c. Add the divisor to the dividend
   d. Go to Step 2.

The iteration proceeds until the desired number of quotient bits is obtained. Whenever a result is negative, the dividend must be restored by the amount subtracted. Since another shift and subtract must be performed anyway, the restore, shift and subtract can be combined efficiently into a single shift and add operation. These are equivalent, since restore, shift and subtract are identical to add, multiply by two and subtract, which is identical to a single addition.

The division instructions preclude the need to test and branch in the microprogram; whether addition or subtraction is to be carried out is decided by an internal flag which indicates whether or not the previous operation gave a negative result.

Overflow will occur during division whenever the divisor is zero or greater than the dividend. Overflow detection is also built into the division instructions and does not require special test and branch or normalizing instructions in the microprogram.

The following algorithms for signed and unsigned division produce an 8N-bit integer quotient and remainder, given a 16N-bit integer dividend and an 8N-bit integer divisor, where N is the number of cascaded packages.

All algorithms begin with a LOADMQ instruction. This must be implemented even if the proper value is already in the MQ register. The LOADMQ instruction initializes internal flip-flops used by the multiplication and division routines.

2.3.1.1 Signed Division

LOADMQ LSHDIV	Load MQ register with the least significant half of the dividend.
SDIVIN DIVSOR, DIVMSH, REM	Shift dividend and store sign. R bus = Divisor S bus = Most significant half of dividend
SDIVO DIVSOR, REM	Optional test for overflow (may be omitted if OVR pin is ignored; $\overline{WE}$ must be high to avoid writing back to the register file if used). R bus = Divisor S bus = Result of SDIVIN
SDIVIS DIVSOR, REM, REM	Calculate difference between divisor and most significant half of the dividend to compute first quotient bit. R bus = Divisor S bus = Result of SDIVIN
REPEAT 8N – 2 TIMES:	
SDIVI DIVSOR, REM, REM	Calculate difference between divisor and most significant half of the dividend to compute subsequent quotient bits. R bus = Divisor S bus = Result of SDIVIS (or SDIVI)
(END REPEAT)	
SDIVIT DIVSOR, REM, REM	Generate last quotient bit. Test for remainder equal to zero. R bus = Divisor S bus = Result of SDIVI
DIVRF DIVSOR, REM, REM	Correct remainder if needed. R bus = Divisor S bus = Result of SDIVIT
SDIVQF DIVSOR, MQ	Correct quotient if needed. Test for overflow. R bus = Divisor S bus = MQ register

The remainder is correct at the end of the DIVRF instruction. The quotient is correct after the SDIVQF instruction.

The quotient is stored in the MQ register; the remainder is stored in REM. Inputs, outputs and number of cycles required for this algorithm are shown in Table 2.8.

Table 2-8. Signed Division Algorithm

OP CODE	MNEMONIC	CLOCK CYCLES	INPUT S PORT	INPUT R PORT	OUTPUT Y PORT
E4	LOADMQ	1	Dividend (LS Half)	—	Dividend (LS Half)
80	SDIVIN	1	Dividend (MS Half)	Divisor	Remainder (N)
AF	SDIVO	1	Remainder (N)	Divisor	Test Result
90	SDIVIS	1	Remainder (N)	Divisor	Remainder (N)
A0	SDIVI	8N – 2*	Remainder (N)	Divisor	Remainder (N)
E0	SDIVIT	1	Remainder (N)	Divisor	Remainder (Unfixed)
40	DIVRF	1	Remainder (Unfixed)	Divisor	Remainder
50	SDIVQF	1	MQ Register	Divisor	Quotient

* N = Number of cascaded packages.

2.3.1.2 Unsigned Division

LOADMQ LSHDIV	Load MQ with least significant half of dividend.
UDIVIS DIVSOR, MSHDIV, REM	Begin iterate procedure; test for quotient overflow and division by zero. R bus = Divisor S bus = Most significant half of dividend.
REPEAT 8N – 1 TIMES:	
UDIVI DIVSOR, REM, REM	Generates one quotient bit through iterative subtract/shift or add/shift operations of the divisor and dividend. R bus = Divisor S bus = Result of UDIVIS (or UDIVI)
(END REPEAT)	
UDIVIT DIVSOR, REM, REM	Generate last quotient bit. R bus = Divisor S bus = Result of UDIVI
DIVRF DIVSOR, REM, REM	Correct the remainder. R bus = Divisor S bus = Result of UDIVIT

The remainder is correct following the DIVRF instruction. The quotient is stored in the MQ register at the completion of the routine and does not require correction.

Inputs, outputs and number of cycles required for this algorithm are shown in Table 2.9.

Table 2-9. Unsigned Division Algorithm

OP CODE	MNEMONIC	CLOCK CYCLES	INPUT S PORT	INPUT R PORT	OUTPUT Y PORT
E4	LOADMQ	1	Dividend (LS Half)	—	Dividend (LS Half)
B0	UDIVIS	1	Dividend (MS Half)	Divisor	Remainder (N)
C0	UDIVI	8N – 1*	Remainder (N)	Divisor	Remainder (N)
F0	UDIVIT	1	Remainder (N)	Divisor	Remainder (Unfixed)
40	DIVRF	1	Remainder (Unfixed)	Divisor	Remainder

* N = Number of cascaded packages.

2.3.2 Multiplication

The ALU performs three types of N by N multiplication by repeated addition: signed times signed, unsigned times unsigned, and mixed (signed times unsigned). Each produces a 2N-bit result, where N is the number of cascaded 'AS888 packages.

All three types of multiplication proceed by the following recursion:

$$P(J+1) = 2[P(J) + \text{Multiplicand} \times M(8N-J)]$$

where

$P(J+1)$ = partial product at iteration number $J+1$;
J = 0 to 8N – 1, depending on iteration number.
(N = 2 for a 16 by 16 multiplication);
2 = some type of shift (unique to multiplication instructions);
$P(J)$ = partial product at iteration number J;
$M(8N-J)$ = mode bit, depending on multiplication type;
N = number of 'AS888 packages that are cascaded.

Multiplication instructions are listed below, followed by algorithms for signed, unsigned and mixed multiplication.

Instruction Code (I7-I0) (hex)	Instruction
60	Signed Multiply Iterate (SMULI)
70	Signed Multiply Terminate (SMULT)
D0	Unsigned Multiply Iterate (UMULI)

2.3.2.1 *Signed Multiplication*

Signed multiplication performs an 8N + 2 clock, two's complement multiplication.

```
XOR ACC, ACC, ACC          Zero register to be used for accumulator
LOADMQ MUL                 Load MQ with multiplier
REPEAT 8N - 1 TIMES:
    SMULI, MULT, ACC, ACC  Perform a signed times signed iteration.
                               R bus = Multiplicand
                               S bus = Accumulator
(END REPEAT)
SMULT MULT, ACC, ACC       Perform a signed times signed iteration.
                               R bus = Multiplicand
                               S bus = Accumulator
```

The accumulator now contains the 8N most significant bits of the product, and the MQ the 8N least significant bits.

2.3.2.2 *Unsigned Multiplication*

Unsigned multiplication produces an unsigned times unsigned product in 8N + 2 clocks.

```
XOR ACC, ACC, ACC          Zero the register to be used for accumulator.
LOADMQ MUL                 Load MQ register with multiplier.
REPEAT 8N TIMES:
    UMULI MULT, ACC, ACC   Perform an unsigned multiplication iteration.
                               R bus = Multiplicand
                               S bus = Accumulator
(END REPEAT)
```

The accumulator now contains the 8N most significant bits of the product. The MQ register contains the 8N least significant bits.

2.3.2.3 *Mixed Multiplication*

Mixed multiplication computes a signed multiplicand times an unsigned multiplier, producing a signed result in 8N + 2 clocks.

```
XOR ACC, ACC, ACC          Zero the register to be used for accumulator.
LOADMQ MUL                 Load MQ with unsigned multiplier.
REPEAT 8N TIMES:
    SMULI MULT, ACC, ACC   Perform a signed times signed iteration.
                               R bus = Multiplicand
                               S bus = Accumulator
(END REPEAT)
```

The accumulator now contains the 8N most significant bits of the product. The MQ register contains the 8N least significant bits.

2.4 Decimal Arithmetic and Data Conversion

Excess-3 is a binary decimal code in which each digit (0—9) is represented by adding three to its NBCD (natural binary coded decimal) representation, as shown in Table 2.10. Excess-3 code has the useful property that it allows decimal arithmetic to be carried out in binary hardware. Carries from one digit to another during addition in BCD occur when the sum of the two digits plus the carry-in is greater than or equal to ten. If both numbers are excess-3, the sum will be excess-6, which will produce the proper carries. Therefore, every addition or subtraction operation may use the binary adder.

Table 2-10. Excess-3 Representation

DECIMAL	NBCD	EXCESS-3
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

2.4.1 Excess-6 to Excess-3

To convert the result from excess-6 to excess-3, one must consider two cases resulting from a BCD digit add: one where a carry-out is produced, and one where a carry-out is not produced. If a carry-out is not produced, three must be subtracted from the resulting digit. If a carry is produced, the digit is correct as a BCD number. For example, if BCD 5 is added to BCD 6, the excess-3 result would be 8 + 9 = 1 (with a carry). A carry rolls the number through the illegal BCD representations into a correct BCD representation. A binary 3 must be added to digit positions that produce a carry-out to correct the result to an excess-3 representation.

Every addition and subtraction instruction stores the carry generated from each 4-bit digit location for use by the excess-3 correction functions. These correction instructions (EX3BC for byte corrections and EX3C for word corrections) must be executed in the clock cycle immediately after the addition or subtraction operation.

Signed numbers may be represented in ten's complement form by complementing the excess-3 number. An example is given with the BINEX3 instruction in section 2.5. Complements of excess-3 numbers may be generated by subtracting the excess-3 number from an excess-3 zero followed by an excess-3 correct (EX3C).

2.4.2 Binary to Excess-3

Binary numbers can be converted to excess-3 representation using the BINEX3 instruction. An algorithm for this conversion accompanies the discussion of BINEX3 in section 2.5.

2.4.3 BCD to Binary

Binary decimal code can be coverted to binary using BCDBIN. For an algorithm, see BCDBIN in section 2.5.

2.4.4 Excess-3 to USASCII

Input/output devices or files represent numbers differently than high-speed central processing units. I/O devices handle all alphanumeric data similarly. CPUs handle more numeric data than alphabetic data and store numeric data in packed form to minimize calculation throughput and reduce memory requirements.

To represent the amount 1096, the I/O device would handle the four USASCII characters (''1'', ''0'', ''9'' and ''6'') separately, requiring four bytes of storage. In packed BCD, the number could be stored in two bytes of data as 1096 (0001 0000, 1001 0110). The 'AS888 can be programmed to perform data format conversions such as excess-3 BCD to USASCII.

An algorithm to convert a packed word of excess-3 BCD to two unpacked words of USASCII code is given below.

ALGORITHM:

Main 1	READ NUM	Read packed excess-3 number (1096) into NUM.
Main 2	XOR OFFSET, OFFSET, OFFSET	Clear register to hold offset constant 2D2D (Hex) to convert excess-3 numbers to USASCII.
Main 3	SET1 OFFSET, Mask(2D), OFFSET MSH, LSH	Store 2D (Hex) in both bytes of offset.
Main 4	MOVE NUM, TEMP1, JSR UNPACK	Copy NUM into TEMP1 to set up subroutine parameters; call Unpack procedure.
Main 5	MOVE TEMP1, TEMP2	Store TEMP1 in TEMP2.
Main 6	ADD NUM, NUM, NUM, SLC	Rotate NUM two places.
Main 7	ADD NUM, NUM, NUM, SLC	Rotate NUM two places.
Main 8	ADD NUM, NUM, NUM, SLC	Rotate NUM two places.
Main 9	ADD NUM, NUM, NUM(0), SLC, JSR (UNPACK)	Rotate NUM two places; call Unpack procedure.
Main 10	STORE TEMP2	Store two USASCII characters in TEMP2.
Main 11	STORE TEMP1	Store two USASCII characters in TEMP1.
Unpack 1	SET0, TEMP1, Mask(FF), MSH	Clear upper byte of TEMP1, leaving 0C09 (Hex).
Unpack 2	ADD TEMP1, TEMP1, TEMP3, SLC	Rotate TEMP1 two places; Store result in TEMP3.
Unpack 3	ADD TEMP3, TEMP3, TEMP3, SLC	Rotate TEMP3 two places.
Unpack 4	OR TEMP1, TEMP3, TEMP1	OR TEMP1 and TEMP2; Store result (0CD9 Hex) in TEMP1.
Unpack 5	SET0 TEMP1, Mask(F0), LSH, MSH	Clear most significant four bits in each byte leaving 0C09 (Hex).
Unpack 6	ADD TEMP1, OFFSET, TEMP1, RTS	Add 2D2D (Hex) to TEMP1 to produce 3936 (Hex), the USASCII representation of 96. Return to Main 5.

2.5 Instruction Set

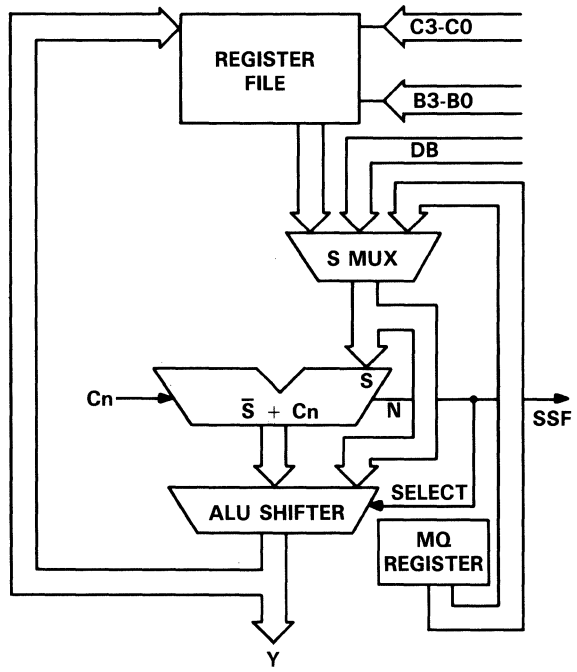
The 'AS888's instruction set is presented in alphabetical order on the following pages. The discussion of each instruction includes a functional description, list of possible operands, data flow schematic and notes on status and control bits affected by the instruction. Microcoded examples or algorithms are also shown.

Mnemonics and op codes are given at the top of each page. An asterisk (*) in the left side of an op code box means that an op code can be selected from the Group 2 instructions on page 2-9; an asterisk in the right side indicates a Group 1 instruction.

FUNCTION

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

DATA FLOW



Available R Bus Source Operands

RF (A3-A0)	A3-A0 Immediate	DA Port	A3-A0 : : C3-C0 Mask

Available S Bus Source Operands

RF (B3-B0)	DB Port	MQ Register
•	•	•

Available Destination Operands

RF (C3-C0)	RF (B3-B0)	Y Port
•		•

Shift Operations

ALU	MQ
None	None

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Carries result of sign bit test from MSP.
$\overline{SIO0}$	No	Inactive
$\overline{SIO7}$	No	Inactive
$\overline{QIO0}$	No	Inactive
$\overline{QIO7}$	No	Inactive
C_n	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 package is 80_{16} and inputs in all other packages are 00_{16}.
C_{n+8} = 1 if S = 0

DESCRIPTION

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

EXAMPLES (assume a 24-bit cascaded system)

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

Instruction Code I7-I0	Operand Address A3-A0	Operand Address B3-B0	Operand Select $\overline{EA}$ EB1-EB0	Destination Address C3-C0	Destination Select $\overline{WE}$ SELY $\overline{OEY}$	Carry-in C_n
0100 1000	XXXX	0001	X 00	0100	0 0 1	1

Example 1: Assume register file 1 holds $F6D840_{16}$.

	Most Significant Package	Next Most Significant Package	Least Significant Package	
Source	1111 0110	1101 1000	0100 0000	S ← RF(1)
	Most Significant Package	Next Most Significant Package	Least Significant Package	
Destination	0000 1001	0010 0111	1100 0000	RF(4) ← S + C_n

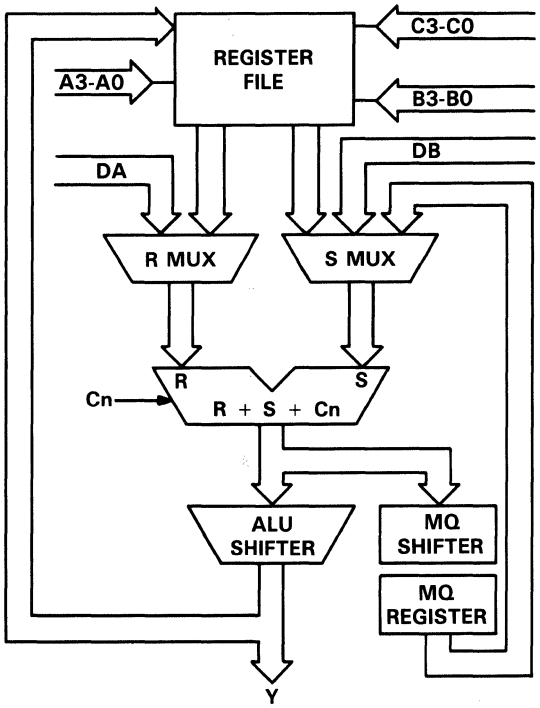
Example 2: Assume register file 1 holds $0927C0_{16}$.

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

FUNCTION

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

DATA FLOW



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 on page 2-9.

Available R Bus Source Operands

RF (A3-A0)	A3-A0 Immediate	DA Port	A3-A0 : : C3-C0 Mask
•		•	

Available S Bus Source Operands

RF (B3-B0)	DB Port	MQ Register
•	•	•

Available Destination Operands

RF (C3-C0)	RF (B3-B0)	Y Port
•		•

Shift Operations

ALU	MQ
•	•

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Affect shift instructions programmed in bits I7-I4 of instruction field.
$\overline{SIO0}$	No	
$\overline{SIO7}$	No	
$\overline{QIO0}$	No	
$\overline{QIO7}$	No	
C_n	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_{n+8} = 1 if carry-out = 1

† C_{n+8} 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 24-bit cascaded system)

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

Instruction Code I7-I0	Operand Address A3-A0	Operand Address B3-B0	Operand Select $\overline{EA}$ EB1-EB0	Destination Address C3-C0	Destination Select $\overline{WE}$ SELY $\overline{OEY}$	Carry-in C_n
1110 0001	0001	XXXX	0 10	XXXX	1 X 1	0

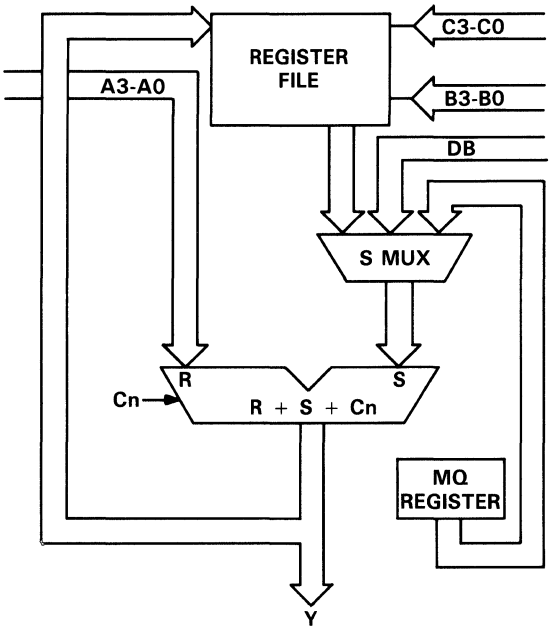
Assume register file 1 holds $08C618_{16}$ and DB bus holds 007530_{16}.

	Most Significant Package	Next Most Significant Package	Least Significant Package	
Source	0000 1000	1100 0110	0001 1000	R ← RF(1)
Source	0000 0000	0111 0101	0011 0000	S ← DB bus
Destination	0000 1001	0011 1011	0100 1000	MQ register ← R + S + C_n

FUNCTION

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

DATA FLOW



Available R Bus Source Operands

RF (A3-A0)	A3-A0 Immediate	DA Port	A3-A0 : : C3-C0 Mask
	•		

Available S Bus Source Operands

RF (B3-B0)	DB Port	MQ Register
•	•	•

Available Destination Operands

RF (C3-C0)	RF (B3-B0)	Y Port
•		•

Shift Operations

ALU	MQ
None	None

Control/Data Signals

Signal	User Programmable	Use
$\overline{SSF}$	No	Inactive
$\overline{SIO0}$	No	Inactive
$\overline{SIO7}$	No	Inactive
$\overline{QIO0}$	No	Inactive
$\overline{QIO7}$	No	Inactive
C_n	Yes	Increments sum if set to one.

Status Signals

ZERO	=	if result = 0
N	=	1 if MSB = 1
OVR	=	1 if signed arithmetic overflow
C_{n+8}	=	1 if carry-out = 1

DESCRIPTION

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

EXAMPLE (assumes a 24-bit cascaded system)

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

Instruction Code I7-I0	Operand Address A3-A0	Operand Address B3-B0	Operand Select $\overline{EA}$ EB1-EB0	Destination Address C3-C0	Destination Select $\overline{WE}$ SELY $\overline{OEY}$	Carry-in C_n
0110 1000	1100	XXXX	X 10	0001	0 0 1	0

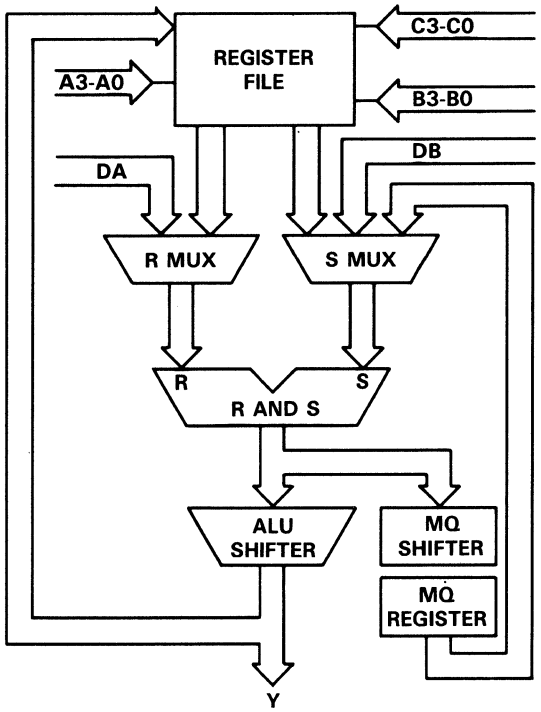
Assume bits A3-A0 hold C_{16} and DB bus holds 000100_{16}.

	Most Significant Package	Next Most Significant Package	Least Significant Package	
Source	0000 0000	0000 0000	0000 1100	R ← A3-A0
Source	0000 0000	0000 0001	0000 0000	S ← DB bus
Destination	0000 0000	0000 0001	0000 1100	RF(1) ← R + S + C_n

FUNCTION

Evaluates the logical expression R AND S.

DATA FLOW



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 on page 2-9.

Available R Bus Source Operands

RF (A3-A0)	A3-A0 Immediate	DA Port	A3-A0 : : C3-C0 Mask
•		•	

Available S Bus Source Operands

RF (B3-B0)	DB Port	MQ Register
•	•	•

Available Destination Operands

RF (C3-C0)	RF (B3-B0)	Y Port
•		•

Shift Operations

ALU	MQ
•	•

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Affect shift instructions programmed in bits I7-14 of instruction field.
$\overline{SIO0}$	No	
$\overline{SIO7}$	No	
$\overline{QIO0}$	No	
$\overline{QIO7}$	No	
C_n	No	Inactive

Status Signals†

ZERO = 1 if result = 0
N = 1 if MSB = 1
OVR = 0
C_{n+8} = 0

† C_{n+8} 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 24-bit cascaded system)

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

Instruction Code I7-I0	Operand Address A3-A0	Operand Address B3-B0	Operand Select $\overline{EA}$ EB1-EB0	Destination Address C3-C0	Destination Select $\overline{WE}$ SELY $\overline{OEY}$	Carry-in C_n
1111 1010	0011	0101	0 00	0101	0 0 1	X

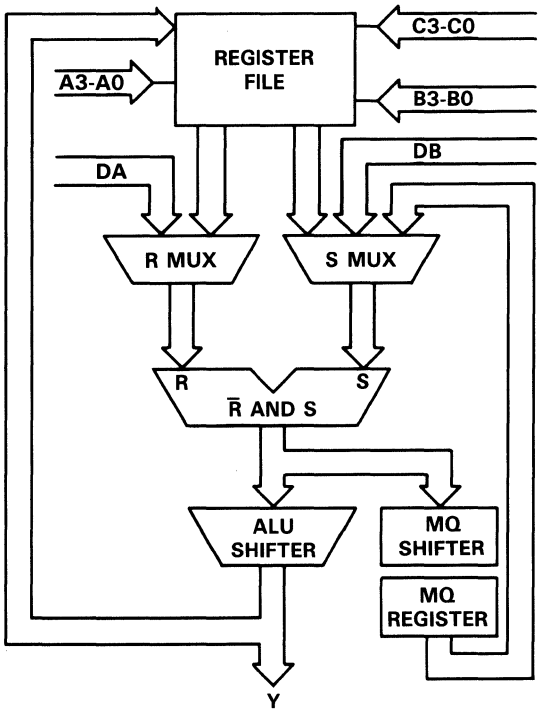
Assume register file 3 holds $F6D840_{16}$ and register file 5 holds $F6D842_{16}$.

	Most Significant Package	Next Most Significant Package	Least Significant Package	
Source	1111 0110	1101 1000	0100 0000	R ← RF(3)
Source	1111 0110	1101 1000	0100 0010	S ← RF(5)
Destination	1111 0110	1101 1000	0100 0000	RF(5) ← R AND S

FUNCTION

Computes the logical expression S AND NOT R.

DATA FLOW



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 on page 2-9.

Available R Bus Source Operands

RF (A3-A0)	A3-A0 Immediate	DA Port	A3-A0 : : C3-C0 Mask
•		•	

Available S Bus Source Operands

RF (B3-B0)	DB Port	MQ Register
•	•	•

Available Destination Operands

RF (C3-C0)	RF (B3-B0)	Y Port
•		•

Shift Operations

ALU	MQ
•	•

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Affect shift instructions programmed in bits I7-I4 of instruction field.
$\overline{SIO0}$	No	
$\overline{SIO7}$	No	
$\overline{QIO0}$	No	
$\overline{QIO7}$	No	
C_n	No	Inactive

Status Signals†

ZERO = 1 if result = 0
N = 1 if MSB = 1
OVR = 0
C_{n+8} = 0

† C_{n+8} 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 24-bit cascaded system)

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

Instruction Code I7-I0	Operand Address A3-A0	Operand Address B3-B0	Operand Select $\overline{EA}$ EB1-EB0	Destination Address C3-C0	Destination Select $\overline{WE}$ SELY $\overline{OEY}$	Carry-in C_n
1111 1110	0011	0101	0 00	0101	0 0 1	X

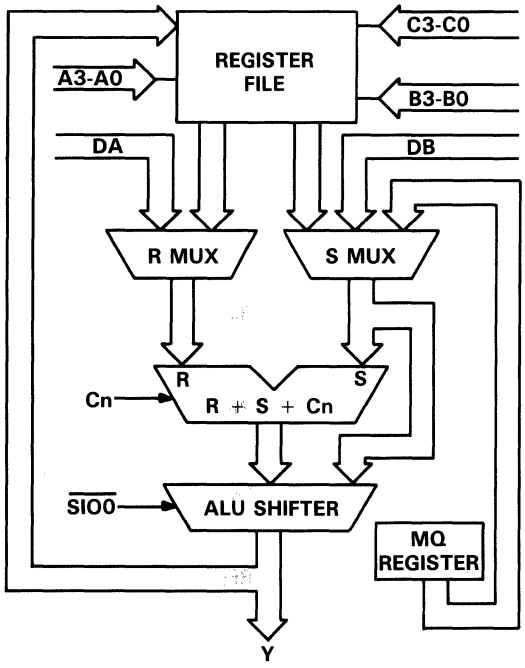
Assume register file 3 holds $F6D840_{16}$ and register file 5 holds $F6D842_{16}$.

	Most Significant Package	Next Most Significant Package	Least Significant Package	
Source	1111 0110	1101 1000	0100 0000	R ← RF(3)
Source	1111 0110	1101 1000	0100 0010	S ← RF(5)
Destination	0000 0000	0000 0000	0000 0010	RF(5) ← $\overline{R}$ AND S

FUNCTION

Adds S with carry-in to a selected slice or selected adjacent slices of R.

DATA FLOW



Available R Bus Source Operands

RF (A3-A0)	A3-A0 Immediate	DA Port	A3-A0 : : C3-C0 Mask
•		•	

Available S Bus Source Operands

RF (B3-B0)	DB Port	MQ Register
•	•	•

Available Destination Operands

RF (C3-C0)	RF (B3-B0)	Y Port
•		•

Shift Operations

ALU	MQ
None	None

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Passes overflow from most-significant selected byte
$\overline{SIO0}$	Yes	Byte select
SIO7	No	Senses most-significant selected byte
$\overline{QIO0}$	No	Inactive
$\overline{QIO7}$	No	Inactive
C_n	Yes	Propagates through non-selected packages; increments selected byte(s) if programmed high.

Status Signals

ZERO	=	1 if result (selected bytes) = 0
N	=	1 if MSB of most-significant selected byte = 1
OVR	=	1 if signed arithmetic overflow (selected bytes)
C_{n+8}	=	1 if carry-out (most-significant selected byte) = 1

DESCRIPTION

Slices with $\overline{SIO0}$ programmed low compute R + S + C_n. Slices with $\overline{SIO0}$ programmed high or floating pass S unaltered. Multiple slices can be selected only if they are adjacent to one another. At least one slice must be non-selected.

EXAMPLE (assumes a 24-bit cascaded system)

Add bytes 1 and 2 of register 3 with carry to bytes 0, 1 and 2 of register 1; store the result in register 11.

Instruction Code I7-I0	Operand Address A3-A0	Operand Address B3-B0	Operand Select $\overline{EA}$ EB1-EB0	Byte Select $\overline{SIO0}$	Destination Address C3-C0	Destination Select $\overline{WE}$	SELY	$\overline{OEY}$	Carry-in C_n
0100 1000	0011	0001	0 00	001	1011	0	0	1	1

Assume register file 3 holds 018181_{16} and register file 1 holds $8FBE3E_{16}$.

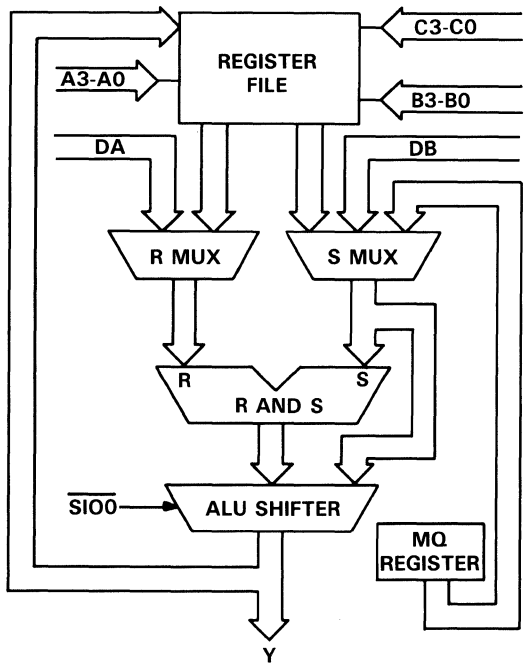
	Most Significant Package Byte 2 (selected)	Next Most Significant Package Byte 1 (selected)	Least Significant Package Byte 0 (not selected)	
Source	0000 0001	1000 0001	1000 0001	$R_n \leftarrow RF(1)_n$
Source	1000 1111	1011 1110	0011 1110	$S_n \leftarrow RF(1)_n$
ALU	1001 0001	0100 0000	1011 1111	$F_n \leftarrow R_n + S_n + C_n$
Destination	1001 0001	0011 1111	0011 1110	$RF(11)_n \leftarrow F_n$ or S_n†

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

FUNCTION

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

DATA FLOW



Available R Bus Source Operands

RF (A3-A0)	A3-A0 Immediate	DA Port	A3-A0 : : C3-C0 Mask
•		•	

Available S Bus Source Operands

RF (B3-B0)	DB Port	MQ Register
•	•	•

Available Destination Operands

RF (C3-C0)	RF (B3-B0)	Y Port
•		•

Shift Operations

ALU	MQ
None	None

Control/Data Signals

Signal	User Programmable	Use
$\overline{SSF}$	No	Forced low
$\overline{SIO0}$	Yes	Byte select
$\overline{SIO7}$	No	Senses most-significant selected byte
$\overline{QIO0}$	No	Inactive
$\overline{QIO7}$	No	Inactive
C_n	No	Inactive

Status Signals

ZERO = 1 if result (selected bytes) = 0
N = 0
OVR = 0
C_{n+8} = 0

DESCRIPTION

Slices with $\overline{SIO0}$ programmed low compute R AND S. Slices with $\overline{SIO0}$ programmed high or floating pass S unaltered. Multiple slices can be selected only if they are adjacent to one another. At least one slice must be non-selected.

EXAMPLE (assumes a 24-bit cascaded system)

Logically AND bytes 1 and 2 of register 3 with bytes 0, 1 and 2 on the DB bus; store the result in register 3.

Instruction Code I7-I0	Operand Address A3-A0	Operand Address B3-B0	Operand Select $\overline{EA}$ EB1-EB0	Byte Select $\overline{SIO0}$	Destination Address C3-C0	Destination Select $\overline{WE}$	SELY	$\overline{OEY}$	Carry-in C_n
1110 1000	0011	XXXX	0 10	001	0011	0	0	1	X

Assume register file 3 holds $8FBEBE_{16}$ and the DB bus holds $90BFBF_{16}$.

	Most Significant Package Byte 2 (selected)	Next Most Significant Package Byte 1 (selected)	Least Significant Package Byte 0 (not selected)	
Source	1000 1111	1011 1110	1011 1110	$R_n \leftarrow RF(3)_n$
Source	1001 0000	1011 1111	1011 1111	$S_n \leftarrow DB_n$
Destination	1000 0000	1011 1110	1011 1111	$RF(3)_n \leftarrow F_n$ or S_n†

† F = ALU result
n = nth package
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) + 8 clocks. The instruction sums the R and S buses with carry.

An arithmetic left shift is performed on the ALU result. The contents of the ALU are shifted one bit to the left. A zero is filled into bit 0 of the least significant package unless $\overline{SIO0}$ is set to zero; this will force bit 0 to one. Bit 7 is passed through $\overline{SIO7}$-$\overline{SIO0}$ to bit 0 of the next-most-significant package. Bit 7 of the most-significant package is dropped.

Simultaneously, the contents of the MQ register are rotated one bit to the left. Bit 7 of the least-significant package is passed through $\overline{QIO7}$-$\overline{QIO0}$ to bit 0 of the next-most-significant package. Bit 7 of the most-significant package is passed through $\overline{SIO7}$-$\overline{SIO0}$ to bit 0 of the least-significant package.

Recommended R Bus Source Operands

RF (A3-A0)	A3-A0 Immediate	DA Port	A3-A0 : : C3-C0 Mask
•			

Recommended S Bus Source Operands

RF (B3-B0)	DB Port	MQ Register
•	•	

Recommended Destination Operands

RF (C3-C0)	RF (B3-B0)	Y Port
•		

Shift Operations

ALU	MQ
Left	Left

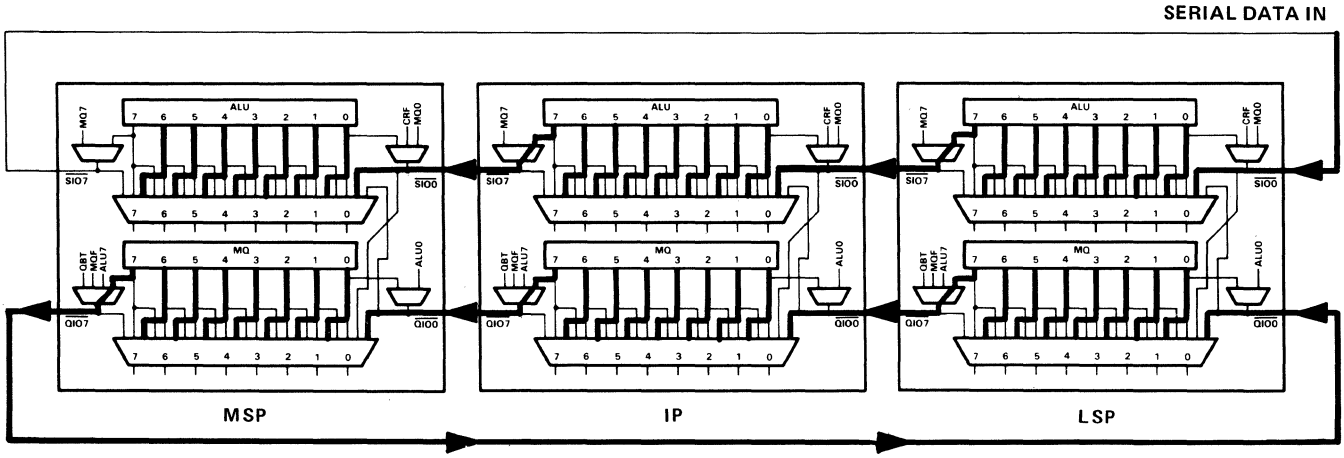
Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inactive
$\overline{SIO0}$	Yes	Link cascaded ALU shifters. $\overline{SIO0}$ fills a zero in LSB of ALU shifter if high or floating; sets MSB to one if low.
$\overline{SIO7}$	No	
$\overline{QIO0}$	No	Link cascaded MQ shifters. Output of MSP's $\overline{QIO7}$ is MSB of MQ shifter (inverted).
$\overline{QIO7}$	No	
C_n	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_{n+8}	=	1 if carry-out = 1

DATA FLOW



The following code converts an N-digit BCD number to a 4N-bit binary number in $4(N-1) + 8$ clocks. It employs the standard conversion formula for a BCD number (shown here for 32 bits):

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

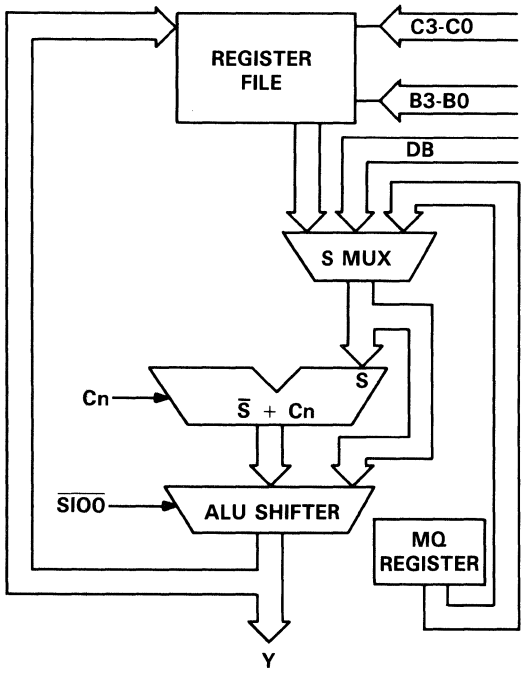
The conversion begins with the most-significant BCD digit. Addition is performed in radix 2.

LOADMQ NUM	Load MQ with BCD number.
SUBR ACC, ACC, ACC, MQSLC	Clear accumulator; Circular left shift MQ.
SUB MSK, MSK, MSK, MQSLC	Clear mask register; Circular left shift MQ.
MQSLC	Circular left shift MQ.
MQSLC	Circular left shift MQ.
ADDI ACC, MSK, 15_{10}	Store 15_{10} in mask register.
Repeat N − 1 times: (N = number of BCD digits)	
AND MQ, MSK, R1, MQSLC	Extract one digit; Circular left shift MQ.
ADD, ACC, R1, R1, MQSLC	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 + digit)]; Circular left shift MQ.
BCDBIN ACC, R1, ACC	Perform BCDBIN instruction, and store result in accumulator [10 × (ACC + digit)]; Circular left shift MQ.
(END REPEAT)	
AND MQ, MSK, R1	Fetch last digit.
ADD ACC, R1, ACC	Add in last digit and store result in accumulator.

FUNCTION

Computes $\overline{S}$ + C_n for selected slices of S.

DATA FLOW



DESCRIPTION

Slices with $\overline{SIO0}$ programmed low compute $\overline{S}$ + C_n. Slices with $\overline{SIO0}$ programmed high or floating pass S unaltered. Multiple slices can be selected only if they are adjacent to one another. At least one slice must be non-selected.

Available R Bus Source Operands

RF (A3-A0)	A3-A0 Immediate	DA Port	A3-A0 : : C3-C0 Mask

Available S Bus Source Operands

RF (B3-B0)	DB Port	MQ Register
•	•	•

Available Destination Operands

RF (C3-C0)	RF (B3-B0)	Y Port
•		•

Shift Operations

ALU	MQ
None	None

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Passes overflow from most-significant selected byte
$\overline{SIO0}$	Yes	Byte select
$\overline{SIO7}$	No	Inactive
$\overline{QIO0}$	No	Inactive
$\overline{QIO7}$	No	Inactive
C_n	Yes	Propagates through non-selected packages; increments selected byte(s) if programmed high.

Status Signals

ZERO	=	1 if result (selected bytes) = 0
N	=	1 if MSB of most-significant selected byte = 1
OVR	=	1 if signed arithmetic overflow (selected bytes)
C_{n+8}	=	1 if carry-out (most-significant selected byte) = 1

EXAMPLE (assumes a 24-bit cascaded system)

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

Instruction Code I7-I0	Operand Address A3-A0	Operand Address B3-B0	Operand Select $\overline{EA}$ EB1-EB0	Byte Select $\overline{SIO0}$	Destination Address C3-C0	Destination Select $\overline{WE}$	SELY	$\overline{OEY}$	Carry-in C_n
1100 1000	XXXX	0011	X 00	100	0011	0	0	1	1

Example 1: Assume register file 3 holds 018181_{16}.

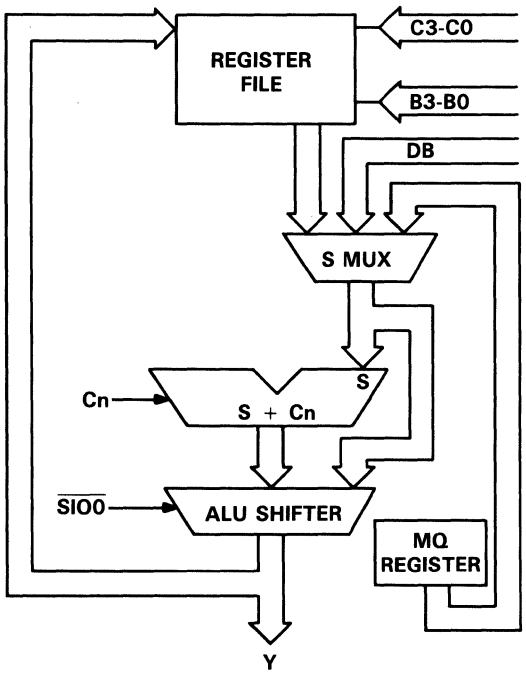
	Most Significant Package Byte 2 (not selected)	Next Most Significant Package Byte 1 (selected)	Least Significant Package Byte 0 (selected)	
Source	0000 0001	1000 0001	1000 0001	$S_n \leftarrow RF(3)_n$
ALU	1111 1111	0111 1110	0111 1111	$F_n \leftarrow \overline{S}_n + C_n$
Destination	0000 0001	0111 1110	0111 1111	$RF(3)_n \leftarrow F_n$ or S_n†

† F = ALU result
n = nth package
Register file 3 gets F if byte selected, S if byte not selected.

FUNCTION

Increments selected slices of S if the carry is set.

DATA FLOW



Available R Bus Source Operands

RF (A3-A0)	A3-A0 Immediate	DA Port	A3-A0 : : C3-C0 Mask

Available S Bus Source Operands

RF (B3-B0)	DB Port	MQ Register
•	•	•

Available Destination Operands

RF (C3-C0)	RF (B3-B0)	Y Port
•		•

Shift Operations

ALU	MQ
None	None

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Passes overflow from most-significant selected byte
$\overline{SIO0}$	Yes	Byte select
$\overline{SIO7}$	No	Senses most-significant selected byte
$\overline{QIO0}$	No	Inactive
$\overline{QIO7}$	No	Inactive
C_n	Yes	Propagates through non-selected packages; increments selected byte(s) if programmed high.

Status Signals

ZERO	=	1 if result (selected bytes) = 0
N	=	1 if MSB of most-significant selected byte = 1
OVR	=	1 if signed arithmetic overflow (selected bytes)
C_{n+8}	=	1 if carry-out (most-significant selected byte) = 1

DESCRIPTION

Slices with $\overline{SIO0}$ programmed low compute S + C_n. Slices with $\overline{SIO0}$ programmed high or floating pass S unaltered. Multiple slices can be selected only if they are adjacent to one another. At least one slice must be non-selected.

EXAMPLE (assumes a 24-bit cascaded system)

Add bytes 1 and 2 of register 7 to the carry; store the result in register 2 (byte 0 is not changed).

Instruction Code I7-I0	Operand Address A3-A0	Operand Address B3-B0	Operand Select $\overline{EA}$ EB1-EB0	Byte Select $\overline{SIO0}$	Destination Address C3-C0	Destination Select $\overline{WE}$	SELY	$\overline{OEY}$	Carry-in C_n
1011 1000	XXXX	0111	X 00	001	0010	0	0	1	1

Assume register file 3 holds $8FBEBE_{16}$.

	Most Significant Package Byte 2 (selected)	Next Most Significant Package Byte 1 (selected)	Least Significant Package Byte 0 (not selected)	
Source	1000 1111	1011 1110	1011 1110	$S_n \leftarrow RF(7)_n$
ALU	1000 1111	1011 1111	1011 1111	$F_n \leftarrow S_n + C_n$
Destination	1000 1111	1011 1111	1011 1110	$RF(2)_n \leftarrow F_n$ or S_n†

† F = ALU result
n = nth package
Register file (3) gets F if byte selected, S if byte not selected.

FUNCTION

Converts a binary number to excess-3 representation.

DESCRIPTION

This instruction allows the user to convert an N-bit binary number to N/4 bit excess-3 number representation in 2N + 3 clocks. The data on the R and S buses are added to the carry-in, which contains bit 7 of the most-significant package's MQ register.

The contents of the MQ register are rotated one bit to the left. Bit 7 of the least-significant package is passed through $\overline{QIO7}$-$\overline{QIO0}$ to bit 0 of the next-most-significant package. Bit 7 of the most-significant package is passed through $\overline{SIO7}$-$\overline{SIO0}$ to bit 0 of the least-significant package.

If this instruction is used with carry look-ahead, data on the R and S buses should be the same, as in the accompanying algorithm. Otherwise, incorrect carry look-ahead will be generated.

Recommended R Bus Source Operands

RF (A3-A0)	A3-A0 Immediate	DA Port	A3-A0 : : C3-C0 Mask
•			

Recommended S Bus Source Operands

RF (B3-B0)	DB Port	MQ Register
•	•	

Recommended Destination Operands

RF (C3-C0)	RF (B3-B0)	Y Port
•		

Shift Operations

ALU	MQ
None	Left

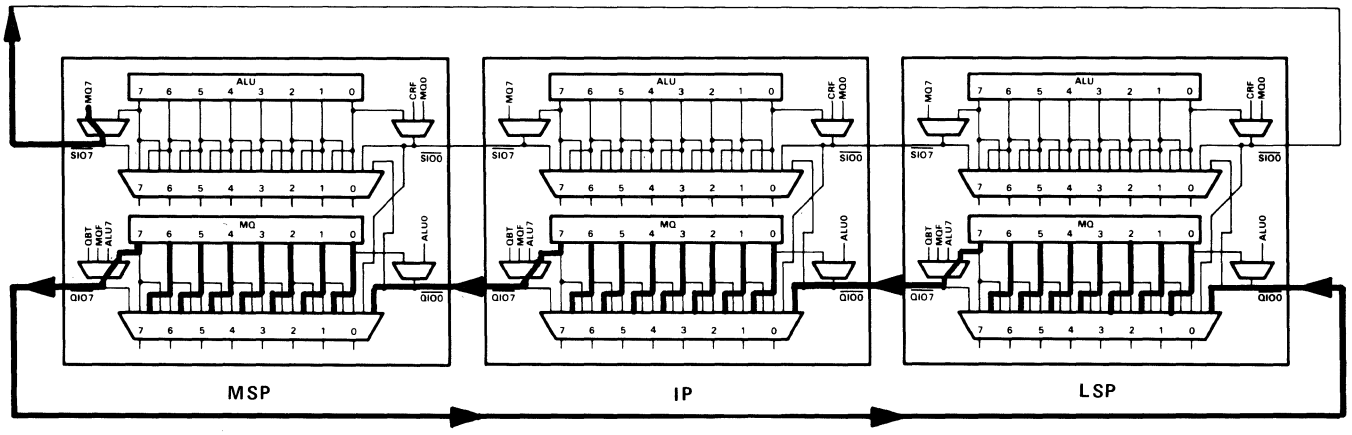
Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inactive
$\overline{SIO0}$	No	Link cascaded ALU shifters. Ouput value of MSP's $\overline{SIO7}$ is MSB of MQ shifter (inverted).
$\overline{SIO7}$	No	
$\overline{QIO0}$	No	Link cascaded MQ shifters. Output value of MSP's $\overline{QIO7}$ is MSB of MQ shifter (inverted).
$\overline{QIO7}$	No	
C_n	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_{n+8}	=	1 if carry-out = 1

DATA FLOW



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

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

The conversion begins with the most-significant binary 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_{16}	Store 33_{16} in all bytes of accumulator.

Repeat N times:
(N = number of bits in binary number)

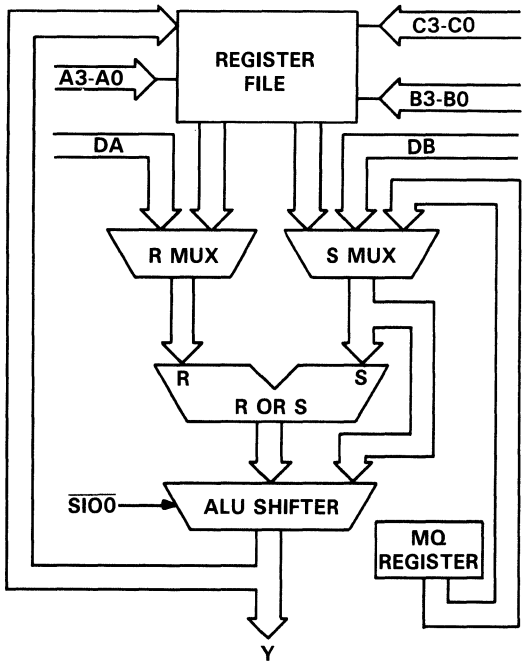
BINEX3 ACC, ACC, ACC	Double accumulator and add in most-significant bit of MQ register. Circular left shift MQ.
EX3C ACC, ACC	Perform excess-3 correction.

(END REPEAT)

FUNCTION

Evaluates R OR S of selected slices of a cascaded system.

DATA FLOW



Available R Bus Source Operands

RF (A3-A0)	A3-A0 Immediate	DA Port	A3-A0 : : C3-C0 Mask
•		•	

Available S Bus Source Operands

RF (B3-B0)	DB Port	MQ Register
•	•	•

Available Destination Operands

RF (C3-C0)	RF (B3-B0)	Y Port
•		•

Shift Operations

ALU	MQ
None	None

Control/Data Signals

Signal	User Programmable	Use
$\overline{SSF}$	No	Forced low
$\overline{SIO0}$	Yes	Byte select
$\overline{SIO7}$	No	Senses most-significant selected byte
$\overline{QIO0}$	No	Inactive
$\overline{QIO7}$	No	Inactive
C_n	No	Inactive

Status Signals

ZERO	=	1 if result (selected bytes) = 0
N	=	1 if MSB of most-significant selected byte = 1
OVR	=	1 if signed arithmetic overflow (selected bytes)
C_{n+8}	=	1 if carry-out (most-significant selected byte) = 1

DESCRIPTION

Slices with $\overline{SIO0}$ programmed low evaluate R OR S. Slices with $\overline{SIO0}$ programmed high or floating pass S unaltered. Multiple slices can be selected only if they are adjacent to one another. At least one slice must be non-selected.

EXAMPLE (assumes a 24-bit cascaded system)

Logically OR bytes 1 and 2 of register 12 with bytes 1 and 2 on the DB bus. Concatenate the result with DB byte 0 and store in register 12.

Instruction Code I7-I0	Operand Address A3-A0	Operand Address B3-B0	Operand Select $\overline{EA}$ EB1-EB0	Byte Select $\overline{SIO0}$	Destination Address C3-C0	Destination Select $\overline{WE}$	SELY	$\overline{OEY}$	Carry-in C_n
1111 1000	1100	XXXX	0 10	001	1100	0	0	1	X

Assume register file 3 holds $8FBEBE_{16}$ and the DB bus holds $90BEBE_{16}$.

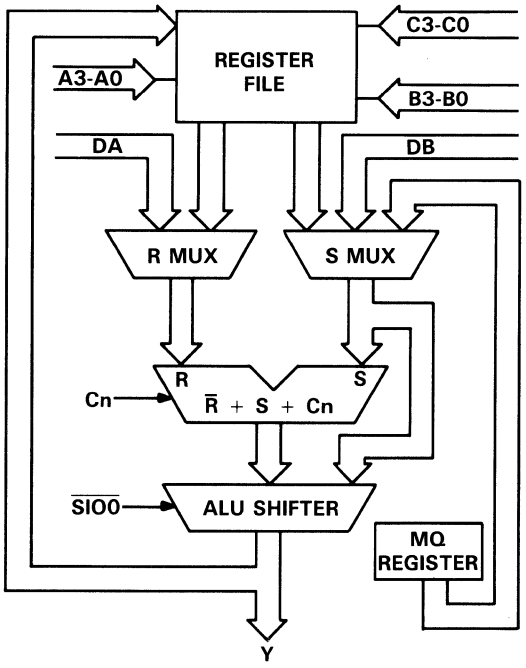
	Most Significant Package Byte 2 (selected)	Next Most Significant Package Byte 1 (selected)	Least Significant Package Byte 0 (not selected)	
Source	1000 1111	1011 1110	1011 1110	$R_n \leftarrow RF(12)_n$
Source	1001 0000	1011 1110	1011 1110	$S_n \leftarrow DB_n$
Destination	1001 1111	1011 1110	1011 1110	$RF(12)_n \leftarrow F_n$ or S_n†

† F = ALU result
n = nth package
Register file 3 gets F if byte selected, S if byte not selected.

FUNCTION

Subtracts R from S in selected slices of a cascaded system.

DATA FLOW



DESCRIPTION

Slices with $\overline{SIO0}$ programmed low compute $\overline{R}$ + S + C_n. Slices with $\overline{SIO0}$ programmed high or floating pass S unaltered. Multiple slices can be selected only if they are adjacent to one another. At least one slice must be non-selected.

Available R Bus Source Operands

RF (A3-A0)	A3-A0 Immediate	DA Port	A3-A0 : : C3-C0 Mask
•		•	

Available S Bus Source Operands

RF (B3-B0)	DB Port	MQ Register
•	•	•

Available Destination Operands

RF (C3-C0)	RF (B3-B0)	Y Port
•		•

Shift Operations

ALU	MQ
None	None

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Passes overflow from most-significant selected byte
$\overline{SIO0}$	Yes	Byte select
$\overline{SIO7}$	No	Senses most-significant selected byte
$\overline{QIO0}$	No	Inactive
$\overline{QIO7}$	No	Inactive
C_n	Yes	Propagates through non-selected packages; increments selected bytes(s) if programmed high.

Status Signals

ZERO	=	1 if result (selected bytes) = 0
N	=	1 if MSB of most-significant selected byte = 1
OVR	=	1 if signed arithmetic overflow (selected bytes)
C_{n+8}	=	1 if carry-out (most-significant selected byte) = 1

EXAMPLE (assumes a 24-bit cascaded system)

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

Instruction Code I7-I0	Operand Address A3-A0	Operand Address B3-B0	Operand Select $\overline{EA}$ EB1-EB0	Byte Select $\overline{SIO0}$	Destination Address C3-C0	Destination Select $\overline{WE}$	SELY	$\overline{OEY}$	Carry-in C_n
1010 1000	0001	0011	0 00	001	1011	0	0	1	1

Assume register file 1 holds $1B5858_{16}$ and register file 3 holds $3A9898_{16}$.

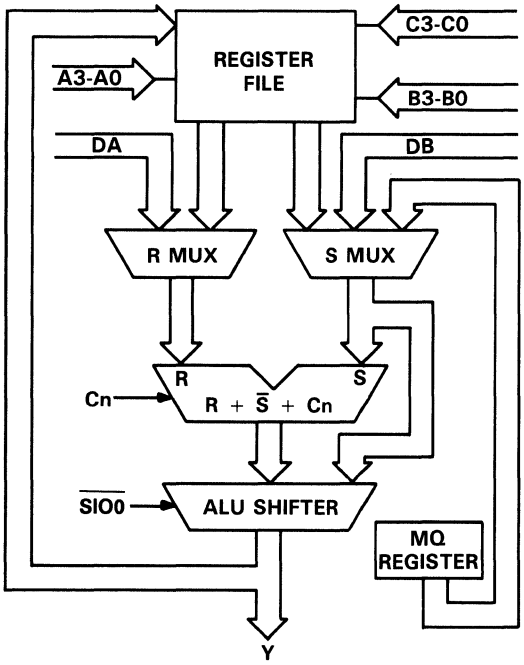
	Most Significant Package Byte 2 (selected)	Next Most Significant Package Byte 1 (selected)	Least Significant Package Byte 0 (not selected)	
Source	0001 1011	0101 1000	0101 1000	$R_n \leftarrow RF(1)_n$
Source	0011 1010	1001 1000	1001 1000	$S_n \leftarrow RF(3)_n$
ALU	0001 1111	0011 1111	0011 1111	$F_n \leftarrow \overline{R}_n + S_n + C_n$
Destination	0001 1111	0011 1111	1001 1000	$RF(11)_n \leftarrow F_n$ or S_n†

† 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 slices of a cascaded system.

DATA FLOW



DESCRIPTION

Slices with $\overline{SIO0}$ programmed low compute R + $\overline{S}$ + C_n. Slices with $\overline{SIO0}$ programmed high or floating pass S unaltered. Multiple slices can be selected only if they are adjacent to one another. At least one slice must be non-selected.

Available R Bus Source Operands

RF (A3-A0)	A3-A0 Immediate	DA Port	A3-A0 : : C3-C0 Mask
•		•	

Available S Bus Source Operands

RF (B3-B0)	DB Port	MQ Register
•	•	•

Available Destination Operands

RF (C3-C0)	RF (B3-B0)	Y Port
•		•

Shift Operations

ALU	MQ
None	None

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Passes overflow from most-significant selected byte
$\overline{SIO0}$	Yes	Byte select
$\overline{SIO7}$	No	Senses most-significant selected byte
$\overline{QIO0}$	No	Inactive
$\overline{QIO7}$	No	Inactive
C_n	Yes	Propagates through non-selected packages; increments selected byte(s) if programmed high.

Status Signals

ZERO	=	1 if result (selected bytes) = 0
N	=	1 if MSB of most-significant selected byte = 1
OVR	=	1 if signed arithmetic overflow (selected bytes)
C_{n+8}	=	1 if carry-out (most-significant selected byte) = 1

EXAMPLE (assumes a 24-bit cascaded system)

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

Instruction Code I7-I0	Operand Address A3-A0	Operand Address B3-B0	Operand Select $\overline{EA}$ EB1-EB0	Byte Select $\overline{SIO0}$	Destination Address C3-C0	Destination Select $\overline{WE}$ SELY $\overline{OEY}$	Carry-in C_n
1010 1000	0001	0011	0 00	001	1011	0 0 1	1

Assume register file 1 holds $88B8B8_{16}$ and register file 3 holds $3A9898_{16}$.

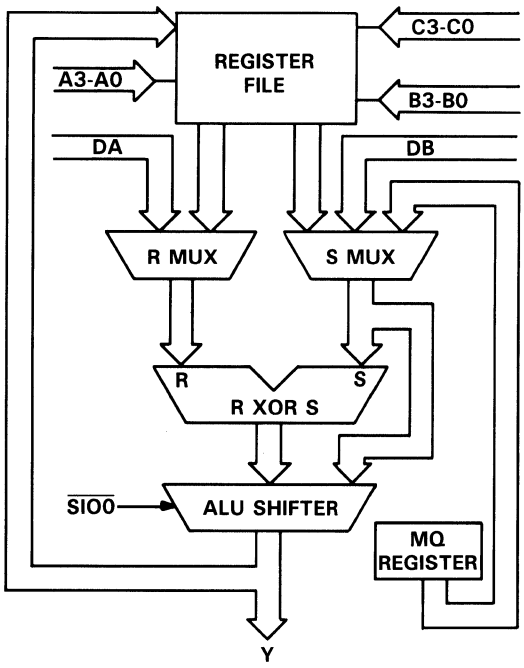
	Most Significant Package Byte 2 (selected)	Next Most Significant Package Byte 1 (selected)	Least Significant Package Byte 0 (not selected)	
Source	1000 1000	1011 1000	1011 1000	$R_n \leftarrow RF(1)_n$
Source	0011 1010	1001 1000	1001 1000	$S_n \leftarrow RF(3)_n$
ALU	0100 1110	0001 1111	0001 1111	$F_n \leftarrow R_n + \overline{S}_n + C_n$
Destination	0100 1110	0001 1111	1001 1000	$RF(11)_n \leftarrow F_n$ or S_n†

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

FUNCTION

Evaluates R exclusive OR S in selected slices of a cascaded system.

DATA FLOW



Available R Bus Source Operands

RF (A3-A0)	A3-A0 Immediate	DA Port	A3-A0 : : C3-C0 Mask
•		•	

Available S Bus Source Operands

RF (B3-B0)	DB Port	MQ Register
•	•	•

Available Destination Operands

RF (C3-C0)	RF (B3-B0)	Y Port
•		•

Shift Operations

ALU	MQ
None	None

Control/Data Signals

Signal	User Programmable	Use
$\overline{SSF}$	No	Forced low
$\overline{SIO0}$	Yes	Byte select
$\overline{SIO7}$	No	Senses most-significant selected byte
$\overline{QIO0}$	No	Inactive
$\overline{QIO7}$	No	Inactive
C_n	No	Inactive

Status Signals

ZERO	=	1 if result (selected bytes) = 0
N	=	1 if MSB of most-significant selected byte = 1
OVR	=	0
C_{n+8}	=	0

DESCRIPTION

Slices with $\overline{SIO0}$ programmed low evaluate R exclusive OR S. Slices with $\overline{SIO0}$ programmed high or floating pass S unaltered. Multiple slices can be selected only if they are adjacent to one another. At least one slice must be non-selected.

EXAMPLE (assumes a 24-bit cascaded system)

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

Instruction Code I7-I0	Operand Address A3-A0	Operand Address B3-B0	Operand Select $\overline{EA}$ EB1-EB0	Byte Select $\overline{SIO0}$	Destination Address C3-C0	Destination Select $\overline{WE}$	SELY	$\overline{OEY}$	Carry-in C_n
1101 1000	0110	XXXX	0 10	001	1010	0	0	1	X

Assume register file 3 holds $8FBEBE_{16}$ and the DB bus holds $90BEBE_{16}$.

	Most Significant Package Byte 2 (selected)	Next Most Significant Package Byte 1 (selected)	Least Significant Package Byte 0 (not selected)	
Source	1000 1111	1011 1110	1011 1110	$R_n \leftarrow RF(6)_n$
Source	1001 0000	1011 1110	1011 1110	$S_n \leftarrow DB_n$
Destination	0001 1111	0000 0000	1011 1110	$RF(10)_n \leftarrow F_n$ or S_n†

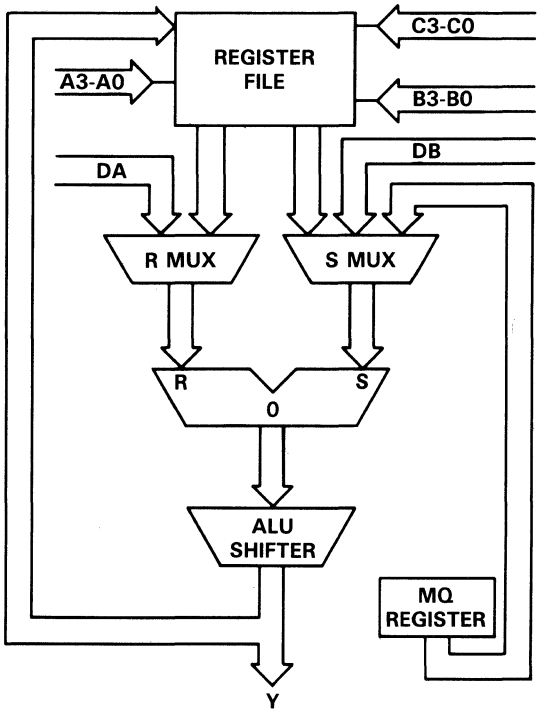
† F = ALU result
n = nth package
Register file 3 gets F if byte selected, S if byte not selected.

CLR — Clear — 0F †

FUNCTION

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

DATA FLOW



Available R Bus Source Operands

RF (A3-A0)	A3-A0 Immediate	DA Port	A3-A0 : : C3-C0 Mask

Available S Bus Source Operands

RF (B3-B0)	DB Port	MQ Register

Available Destination Operands

RF (C3-C0)	RF (B3-B0)	Y Port
•		•

Shift Operations

ALU	MQ
None	None

Status Signals

ZERO = 1
N = 0
OVR = 0
C_{n+8} = 0

DESCRIPTION

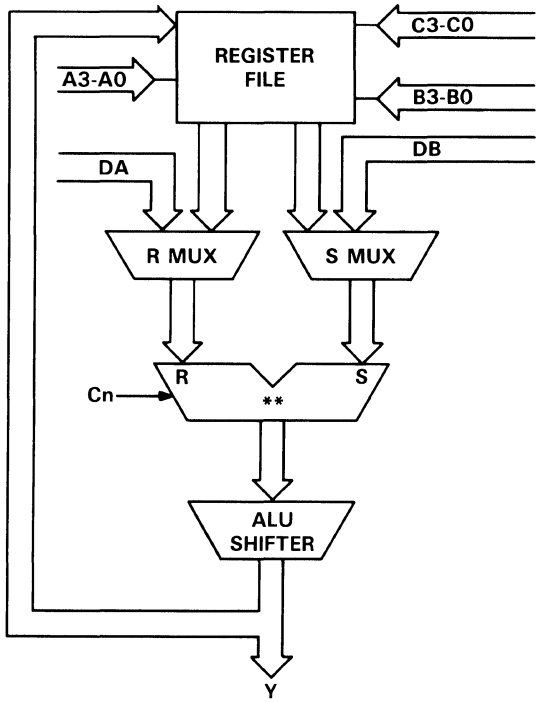
ALU output is forced to zero and the BCD flip-flops are cleared.

† This instruction may also be coded with the following opcodes:
1F, 2F, 3F, 4F, 5F, 6F, BF, CF, EF

FUNCTION

Corrects the remainder of nonrestoring division routine if correction is needed. A description of nonrestoring division and an algorithm using this instruction are given in section 2.3.1.

DATA FLOW



** $S + \overline{R} + 1$ if SSF = 1
$S + R + 0$ if SSF = 0

Recommended R Bus Source Operands

RF (A3-A0)	A3-A0 Immediate	DA Port	A3-A0 : : C3-C0 Mask
•			

Recommended S Bus Source Operands

RF (B3-B0)	DB Port	MQ Register
•	•	

Recommended Destination Operands

RF (C3-C0)	RF (B3-B0)	Y Port
•		

Shift Operations

ALU	MQ
None	None

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Indicates whether quotient fix is required in next instruction.
$\overline{SIO0}$	No	Inactive
$\overline{SIO7}$	No	Inactive
$\overline{QIO0}$	No	Inactive
$\overline{QIO7}$	No	Inactive
C_n	Yes	Should be programmed high

Status Signals

ZERO = 1 if remainder = 0
N = 0
OVR = 0
C_{n+8} = 1 if carry-out = 1

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 non-zero and the signs of the remainder and dividend are different. SSF is used to indicate that a fix is required.

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 SSF (used to signal whether a fix is required) and evaluates:

$Y \leftarrow S + \overline{R} + C_n$ if SSF = 1
$Y \leftarrow S + R$ if SSF = 0.

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

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 via the $\overline{QIO0}$ input. The S bus holds the most-significant half; the MQ register holds the least-significant half.

Normalization is complete when overflow occurs. The SSF pin inhibits the shift whenever normalization is attempted on a number already normalized.

Available R Bus Source Operands

RF (A3-A0)	A3-A0 Immediate	DA Port	A3-A0 : : C3-C0 Mask

Recommended S Bus Source Operands

RF (B3-B0)	DB Port	MQ Register
•		

Recommended Destination Operands

RF (C3-C0)	RF (B3-B0)	Y Port
•		

Shift Operations

ALU	MQ
Left	Left

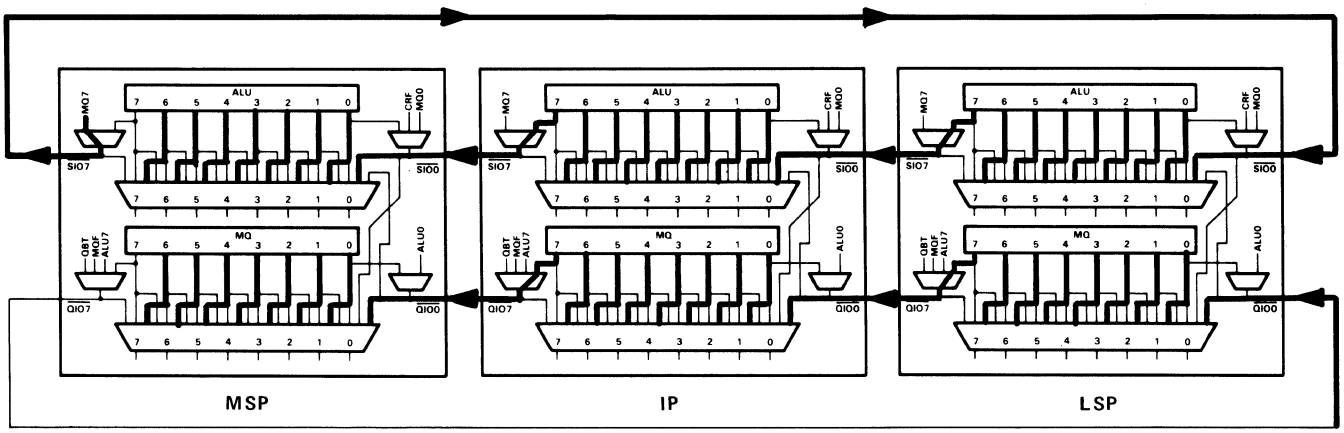
Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inhibits shift if normalization is complete.
$\overline{SIO0}$	No	Link cascaded ALU shifters. Output of MSP's $\overline{SIO7}$ is MSB of MQ shifter (inverted).
$\overline{SIO7}$	No	
$\overline{QIO0}$	No	Link cascaded MQ shifters. $\overline{QIO0}$ of LSP fills a zero into LSB of MQ shifter.
$\overline{QIO7}$	No	
C_n	No	Inactive

Status Signals

ZERO	=	1 if result = 0
N	=	1 if MSB = 1
OVR	=	1 if MSB XOR 2nd MSB = 1
C_{n+8}	=	1 if carry-out = 1

DATA FLOW



EXAMPLE (assumes a 24-bit cascaded system)

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).

Instruction Code I7-I0	Operand Address A3-A0	Operand Address B3-B0	Operand Select $\overline{EA}$ EB1-EB0	Destination Address C3-C0	Destination Select $\overline{WE}$ SELY $\overline{OEY}$	Carry-in C_n
0011 0000	0011	XXXX	0 XX	0011	0 0 1	X

Assume register file 3 holds $F6D84E_{16}$ and MQ register holds $F6D843_{16}$.

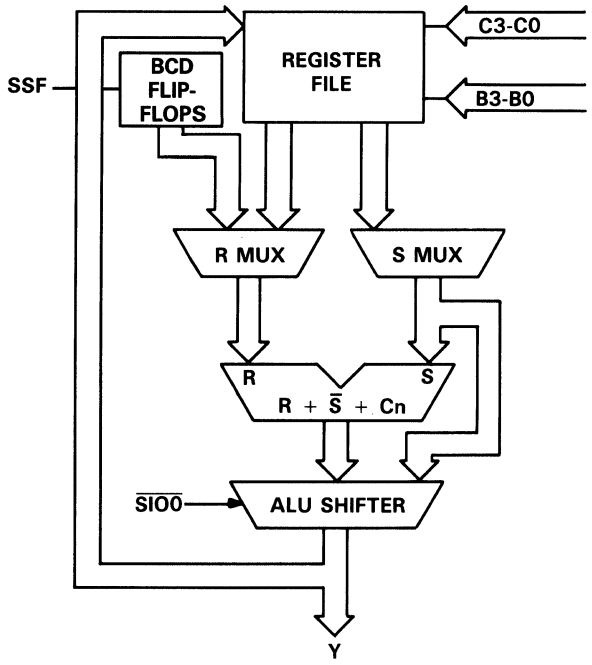
	Most Significant Package	Next Most Significant Package	Least Significant Package	
Source	1111 0110	1101 1000	0100 1110	ALU shifter ← RF(3)
Source	1111 0110	1101 1000	0100 0011	MQ shifter ← MQ register
Destination	1110 1101	1011 0000	1001 1101	RF(3) ← Result (MSH)
Destination	1110 1101	1011 0000	1000 0110	MQ register ← Result (LSH)
			0	OVF ← 0†

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

FUNCTION

Corrects the result of excess-3 addition or subtraction.

DATA FLOW



Available R Bus Source Operands

RF (A3-A0)	A3-A0 Immediate	DA Port	A3-A0 : : C3-C0 Mask

Recommended S Bus Source Operands

RF (B3-B0)	DB Port	MQ Register
•		

Available Destination Operands

RF (C3-C0)	RF (B3-B0)	Y Port
•		•

Shift Operations

ALU	MQ
Left	Left

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Passes overflow from most-significant selected byte.
$\overline{SIO0}$	Yes	Byte select
$\overline{SIO7}$	No	Inactive
$\overline{QIO0}$	No	Inactive
$\overline{QIO7}$	No	Inactive
C_n	No	Inactive

Status Signals

ZERO = 0
N = 1 if MSB = 1
OVR = 1 if arithmetic signed overflow
C_{n+8} = 1 if carry-out = 1

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. Slices with $\overline{SIO0}$ programmed low evaluate the correct excess-3 representation. Slices with $\overline{SIO0}$ programmed high or floating pass S unaltered.

EXAMPLE (assumes a 24-bit cascaded system)

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

1) Clear accumulator (SUB ACC, ACC, ACC)
2) Store 33_{16} in all bytes of register (SET1 R2, H/33/)
3) Add 33_{16} to first BCD number (ADDI DB, R2, R1)
4) Add 33_{16} to second BCD number (ADDI DB, R2, R3)
5) Add selected bytes of registers 1 and 3 (BADD, R1, R3, R3)
6) Correct the result (EX3BC, R3, R3)

Instruction Code I7-I0	Operand Address A3-A0	Operand Address B3-B0	Operand Select $\overline{EA}$	EB1-EB0	Byte Select $\overline{SIO0}$	Destination Address C3-C0	Destination Select $\overline{WE}$	SELY	$\overline{OEY}$	Carry-in C_n
1111 0010	0010	0010	0	00	XXX	0010	0	0	1	1
0000 1000	0010	XXXX	0	XX	XXX	0010	0	0	1	X
1111 0001	0010	XXXX	0	10	XXX	0001	0	0	1	0
1111 0001	0010	XXXX	0	10	XXX	0011	0	0	1	0
1000 1000	0001	0011	0	00	100	0011	0	0	1	0
1000 1111	XXXX	0011	X	00	100	0011	1	0	1	0

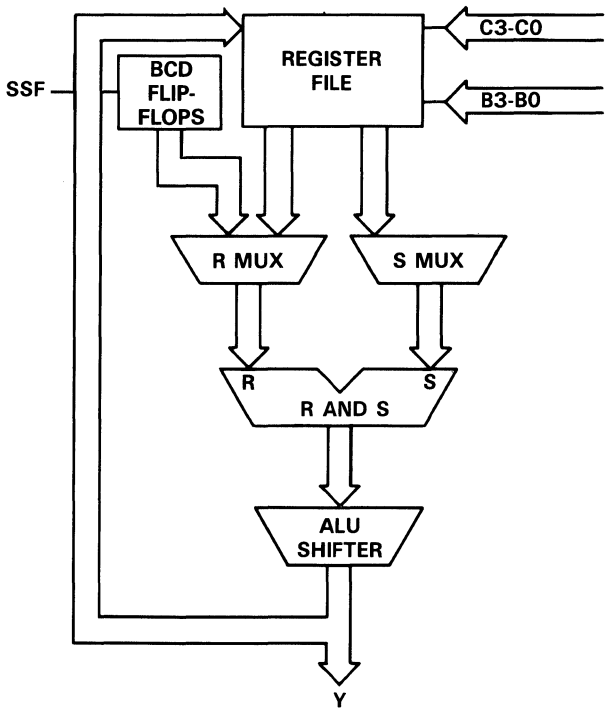
Assume DB bus holds 336912_{10} at second instruction and 437162_{10} at fourth instruction.

Result of Instruction Cycle:	Most Significant Package (not selected)	Next Most Significant Package (selected)	Least Significant Package (selected)	
1	0000 0000	0000 0000	0000 0000	RF(2) ← 0
2	0011 0011	0011 0011	0011 0011	RF(2) ← 333333_{16}
3	0110 0110	1001 1100	0100 0101	RF(1) ← RF(2) + DB
4	0111 0110	1010 0100	1001 0101	RF(3) ← RF(2) + DB
5	0111 0110	0100 0000	1101 1010	$RF(3)_n \leftarrow RF(1)_n + RF(3)_n$
6	0111 0110	0100 0000	0111 0100	$RF(3)_n \leftarrow$ Corrected $RF(3)_n$ result

FUNCTION

Corrects the result of excess-3 addition or subtraction.

DATA FLOW



Available R Bus Source Operands

RF (A3-A0)	A3-A0 Immediate	DA Port	A3-A0 : : C3-C0 Mask

Recommended S Bus Source Operands

RF (B3-B0)	DB Port	MQ Register
•		

Available Destination Operands

RF (C3-C0)	RF (B3-B0)	Y Port
•		•

Shift Operations

ALU	MQ
Left	Left

Control/Data Signals

Signal	User Programmable	Use
$\overline{SSF}$	No	Passes overflow.
$\overline{SIO0}$	No	Inactive
$\overline{SIO7}$	No	Inactive
$\overline{QIO0}$	No	Inactive
$\overline{QIO7}$	No	Inactive
C_n	No	Inactive

Status Signals

ZERO = 0
N = 1 if MSB = 1
OVR = 1 if arithmetic signed overflow
C_{n+8} = 1 if carry-out = 1

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 flop and previous overflow condition reported on the SSF pin.

EXAMPLE (assumes a 24-bit cascaded system)

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

1) Clear accumulator (SUB ACC, ACC, ACC)
2) Store 33_{16} in all bytes of register (SET1 R2, H/33/)
3) Add 33_{16} to first BCD number (ADDI DB, R2, R1)
4) Add 33_{16} to second BCD number (ADDI DB, R2, R3)
5) Add the excess-3 data (ADD R1, R3, R3)
6) Correct the result (EX3C, R3, R3)

Instruction Code I7-I0	Operand Address A3-A0	Operand Address B3-B0	Operand Select $\overline{EA}$	EB1-EB0	Destination Address C3-C0	Destination Select $\overline{WE}$	SELY	$\overline{OEY}$	Carry-in C_n
1111 0010	0010	0010	0	00	0010	0	0	1	1
0000 1000	0011	0010	0	XX	0011	0	0	1	X
1111 0001	0010	XXXX	0	10	0001	0	0	1	0
1111 0001	0010	XXXX	0	10	0011	0	0	1	0
1111 0001	0001	0011	0	00	0011	0	0	1	0
1001 1111	XXXX	0011	X	00	0011	1	0	1	0

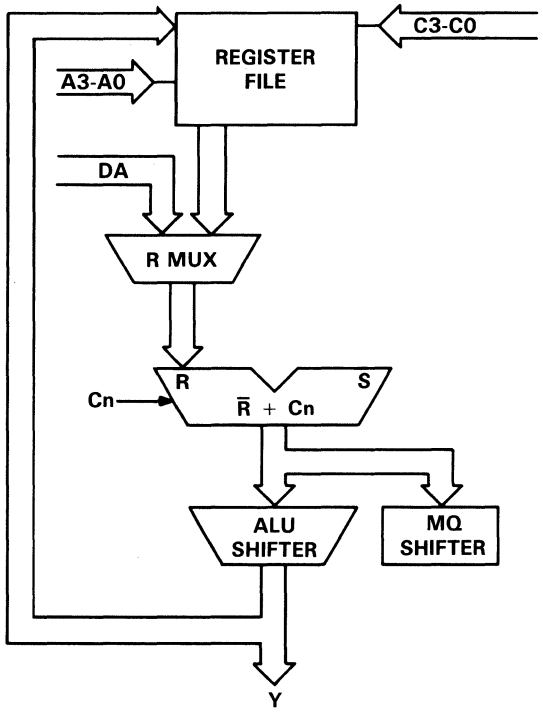
Assume DB bus holds 336912_{10} at second instruction and 437162_{10} at fourth instruction.

Result of Instruction Cycle:	Most Significant Package	Next Most Significant Package	Least Significant Package	
1	0000 0000	0000 0000	0000 0000	RF(2) ← 0
2	0011 0011	0011 0011	0011 0011	RF(2) ← 333333_{16}
3	0110 0110	1001 1100	0100 0101	RF(1) ← RF(2) + DB
4	0111 0110	1010 0100	1001 0101	RF(3) ← RF(2) + DB
5	1101 1101	0100 0000	1101 1010	RF(3) ← RF(1) + RF(3)
6	1010 1010	0111 0011	1010 0111	RF(3) ← Corrected RF(3) result
7	0111 0111	0100 0000	0111 0100	RF(4) ← RF(3) − RF(2)

FUNCTION

Evaluates $\overline{R}$ + C_n.

DATA FLOW



Available R Bus Source Operands

RF (A3-A0)	A3-A0 Immediate	DA Port	A3-A0 : : C3-C0 Mask
•		•	

Available S Bus Source Operands

RF (B3-B0)	DB Port	MQ Register

Available Destination Operands

RF (C3-C0)	RF (B3-B0)	Y Port
•		•

Shift Operations

ALU	MQ
•	•

Control/Data Signals

Signal	User Programmable	Use
$\overline{SSF}$	No	Affect shift instructions programmed in bits I7-I4
$\overline{SIO0}$	No	
$\overline{SIO7}$	No	
$\overline{QIO0}$	No	
$\overline{QIO7}$	No	
C_n	Yes	Increments if programmed high.

Status Signals†

ZERO	=	1 if result = 0
N	=	1 if MSB = 1
OVR	=	1 if signed arithmetic overflow
C_{n+8}	=	1 if carry-out = 1

† C_{n+8} 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.

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 on page 2-9.

EXAMPLE (assumes a 24-bit cascaded system)

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

Instruction Code I7-I0	Operand Address A3-A0	Operand Address B3-B0	Operand Select $\overline{EA}$ EB1-EB0	Destination Address C3-C0	Destination Select $\overline{WE}$ SELY $\overline{OEY}$	Carry-in C_n
1111 0111	XXXX	XXXX	1 XX	0100	0 0 1	1

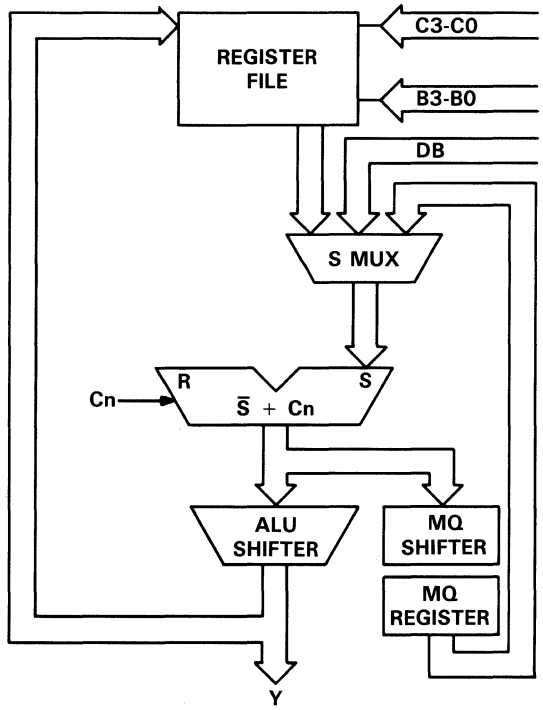
Assume register file 1 holds $91FEF6_{16}$.

	Most Significant Package	Next Most Significant Package	Least Significant Package	
Source	1001 0001	1111 1110	1111 0110	R ← DA
Destination	0110 1110	0000 0001	0000 1010	RF(4) ← $\overline{R}+C_n$

FUNCTION

Evaluates $\overline{S}$ + C_n.

DATA FLOW



Available R Bus Source Operands

RF (A3-A0)	A3-A0 Immediate	DA Port	A3-A0 : : C3-C0 Mask

Available S Bus Source Operands

RF (B3-B0)	DB Port	MQ Register
•	•	•

Available Destination Operands

RF (C3-C0)	RF (B3-B0)	Y Port
•		•

Shift Operations

ALU	MQ
•	•

Control/Data Signals

Signal	User Programmable	Use
$\overline{SSF}$	No	Affect shift instructions programmed in bits I7-I4 of instruction field.
$\overline{SIO0}$	No	
$\overline{SIO7}$	No	
$\overline{QIO0}$	No	
$\overline{QIO7}$	No	
C_n	Yes	Increments if programmed high.

Status Signals†

ZERO	=	1 if result = 0
N	=	1 if MSB = 1
OVR	=	1 if signed arithmetic overflow
C_{n+8}	=	1 if carry-out = 1

† C_{n+8} is ALU carry out and is evaluated before shift operation. ZERO and N (negative) are evaluated after ALU operation and after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

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 on page 2-9.

EXAMPLE (assumes a 24-bit cascaded system)

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

Instruction Code I7-I0	Operand Address A3-A0	Operand Address B3-B0	Operand Select $\overline{EA}$ EB1-EB0	Destination Address C3-C0	Destination Select $\overline{WE}$	SELY	$\overline{OEY}$	Carry-in C_n
1111 0101	XXXX	XXXX	X 11	0100	0	0	1	0

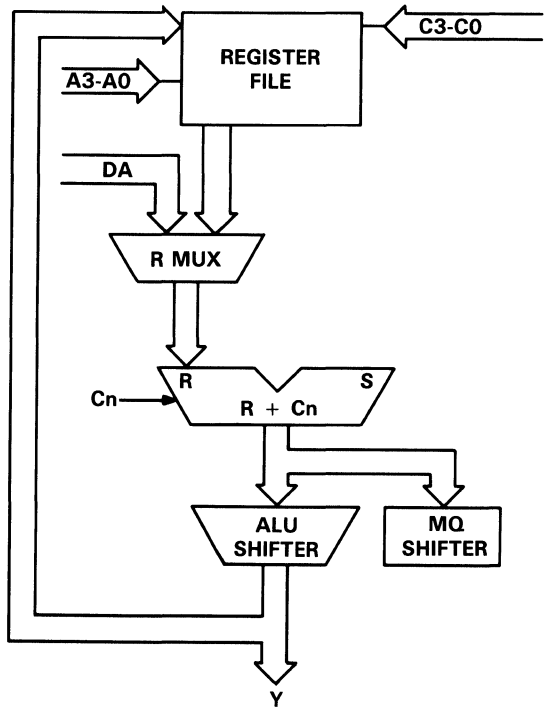
Assume MQ register holds $91FEF6_{16}$.

	Most Significant Package	Next Most Significant Package	Least Significant Package	
Source	1001 0001	1111 1110	1111 0110	S ← MQ register
Destination	0110 1110	0000 0001	0000 1001	RF(4) ← $\overline{S}$ + C_n

FUNCTION

Increments R if the carry is set.

DATA FLOW



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 on page 2-9.

Available R Bus Source Operands

RF (A3-A0)	A3-A0 Immediate	DA Port	A3-A0 : : C3-C0 Mask
•		•	

Available S Bus Source Operands

RF (B3-B0)	DB Port	MQ Register

Available Destination Operands

RF (C3-C0)	RF (B3-B0)	Y Port
•		•

Shift Operations

ALU	MQ
•	•

Control/Data Signals

Signal	User Programmable	Use
$\overline{SSF}$	No	Affect shift instructions programmed in bits I7-I4 of instruction field.
$\overline{SIO0}$	No	
$\overline{SIO7}$	No	
$\overline{QIO0}$	No	
$\overline{QIO7}$	No	
C_n	Yes	Increments R if programmed high.

Status Signals†

ZERO	=	1 if result = 0
N	=	1 if MSB = 1
OVR	=	1 if signed arithmetic overflow
C_{n+8}	=	1 if carry-out = 1

† C_{n+8} 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 24-bit cascaded system)

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

Instruction Code I7-I0	Operand Address A3-A0	Operand Address B3-B0	Operand Select $\overline{EA}$ EB1-EB0	Destination Address C3-C0	Destination Select $\overline{WE}$ SELY $\overline{OEY}$	Carry-in C_n
1111 0110	XXXX	XXXX	1 XX	0100	0 0 1	1

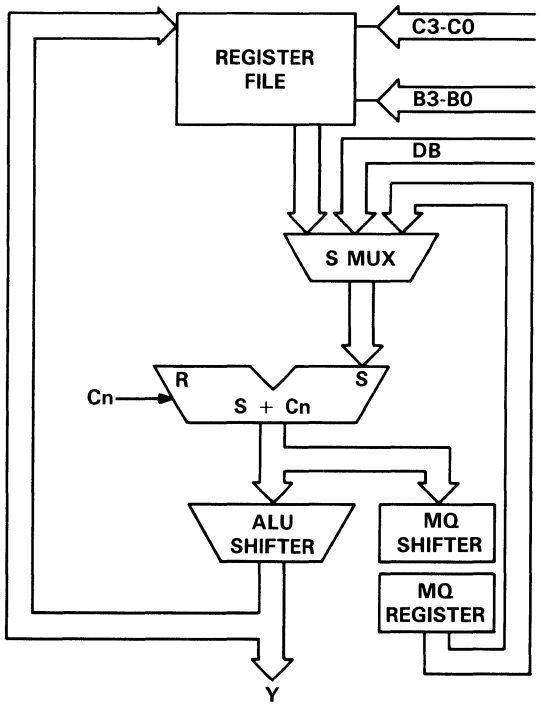
Assume register file 1 holds $91FEF6_{16}$.

	Most Significant Package	Next Most Significant Package	Least Significant Package	
Source	1001 0001	1111 1110	1111 0110	$R \leftarrow RF(1)$
Destination	1001 0001	1111 1110	1111 0111	$RF(4) \leftarrow R + C_n$

FUNCTION

Increments S if the carry is set.

DATA FLOW



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 on page 2-9.

Available R Bus Source Operands

RF (A3-A0)	A3-A0 Immediate	DA Port	A3-A0 : : C3-C0 Mask

Available S Bus Source Operands

RF (B3-B0)	DB Port	MQ Register
•	•	•

Available Destination Operands

RF (C3-C0)	RF (B3-B0)	Y Port
•		•

Shift Operations

ALU	MQ
•	•

Control/Data Signals

Signal	User Programmable	Use
$\overline{SSF}$	No	Affect shift instructions programmed in bits I7-I4 of instruction field.
$\overline{SIO0}$	No	
$\overline{SIO7}$	No	
$\overline{QIO0}$	No	
$\overline{QIO7}$	No	
C_n	Yes	Increments S if programmed high.

Status Signals†

ZERO	=	1 if result = 0
N	=	1 if MSB = 1
OVR	=	1 if signed arithmetic overflow
C_{n+8}	=	1 if carry-out = 1

† C_{n+8} 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 24-bit cascaded system)

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

Instruction Code I7-I0	Operand Address A3-A0	Operand Address B3-B0	Operand Select $\overline{EA}$ EB1-EB0	Destination Address C3-C0	Destination Select $\overline{WE}$ SELY $\overline{OEY}$	Carry-in C_n
1111 0010	XXXX	XXXX	X 11	0100	0 0 1	1

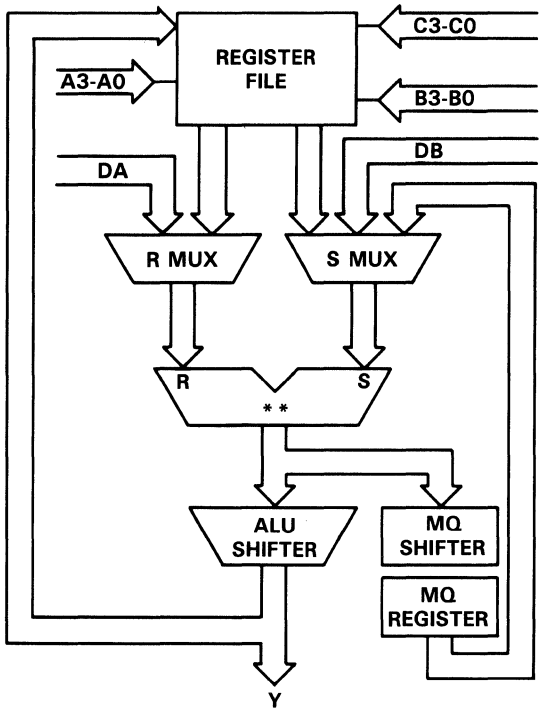
Assume MQ register holds $FF00FF_{16}$.

	Most Significant Package	Next Most Significant Package	Least Significant Package	
Source	1111 1111	0000 0000	1111 1111	S ← MQ register
Destination	1111 1111	0000 0001	0000 0000	RF(4) ← S + C_n

FUNCTION

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

DATA FLOW



*** *Arithmetic/logic function specified in I3-I0**

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 on page 2-9.

Shift Operations

ALU Shifter	MQ Shifter
None	None

Available Destination Operands

RF (C3-C0)	RF (B3-B0)	Y Port
•		•

Control/Data Signals

Signal	User Programmable	Use
$\overline{SSF}$	No	Outputs MQ0 of LSP
$\overline{SIO0}$	No	Inactive
$\overline{SIO7}$	No	Inactive
$\overline{QIO0}$	No	Inactive
$\overline{QIO7}$	No	Inactive
C_n	No	Inactive

Status Signals†

If arithmetic instruction specified in I3-I0:

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_{n+8} = 1 if carry-out = 1

If logic instruction specified in I3-I0:

ZERO = 1 if result = 0
N = 1 if MSB of result = 1
= 0 if MSB of result = 0
OVR = 0
C_{n+8} = 0

† C_{n+8} 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 24-bit cascaded system)

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 an INCR instruction without carry-in).

Instruction Code I7-I0	Operand Address A3-A0	Operand Address B3-B0	Operand Select $\overline{EA}$ EB1-EB0	Destination Address C3-C0	Destination Select $\overline{WE}$ SELY $\overline{OEY}$	Carry-in C_n
1110 0110	0001	XXXX	0 XX	XXXX	1 0 1	0

Assume register file 1 holds $08C618_{16}$.

	Most Significant Package	Next Most Significant Package	Least Significant Package	
Source	0000 1000	1100 0110	0001 1000	R ← RF(1)
Destination	0000 1000	1100 0110	0001 1000	MQ register ← R + C_n

FUNCTION

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

The contents of the MQ register are rotated one bit to the left. Bit 7 of the least-significant package is passed through $\overline{QIO7}$-$\overline{QIO0}$ to bit 0 of the next-most-significant package. Bit 7 of the most-significant package is passed through $\overline{SIO7}$-$\overline{SIO0}$ to bit 0 of the least-significant package.

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 on page 2-9.

Shift Operations

ALU Shifter	MQ Shifter
None	Circular Left

Available Destination Operands
ALU Shifter:

RF (C3-C0)	RF (B3-B0)	Y-Port
•		•

Control/Data Signals

Signal	User Programmable	Use
SSF	Yes	Passes shift result if high or floating; retains MQ without shift if low.
$\overline{SIO0}$ $\overline{SIO7}$	No No	Link cascaded ALU shifters. Output value of MSP's $\overline{SIO7}$ is MSB of MQ shifter (inverted).
$\overline{QIO0}$ $\overline{QIO7}$	No No	Link cascaded MQ shifters. Output value of MSP's $\overline{QIO7}$ is MSB of MQ shifter (inverted).
C_n	No	Affects arithmetic operation programmed in bits I3-I0 of instruction field.

Status Signals†

If arithmetic instruction specified in I3-I0:

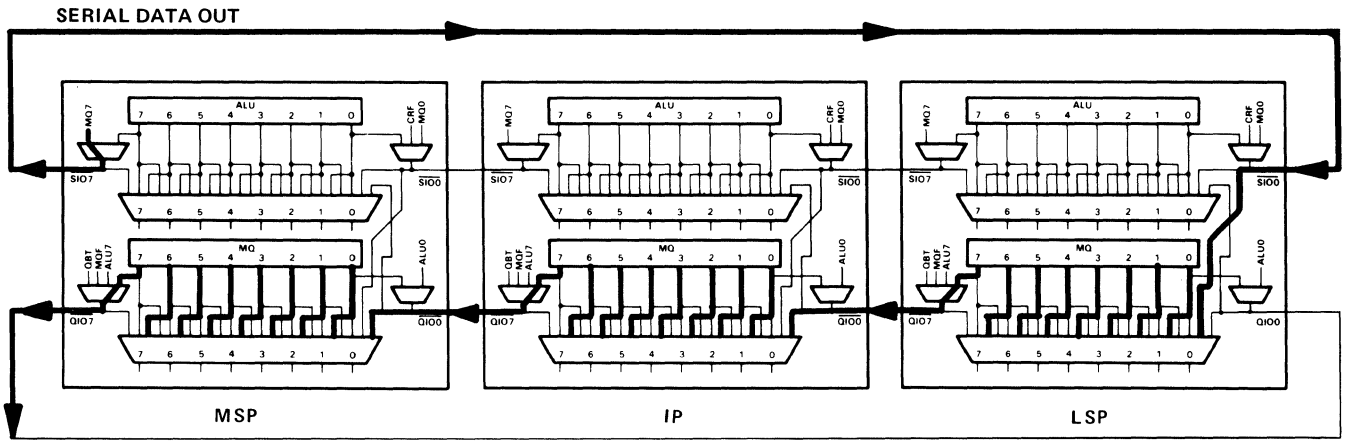
ZERO = 1 if result = 0
N = 1 if MSB of result = 1
= 0 if MSB of result = 0
OVR = 1 if signed arithmetic overflow
C_{n+8} = 1 if carry-out = 1

If logic instruction specified in I3-I0:

ZERO = 1 if result = 0
N = 1 if MSB of result = 1
= 0 if MSB of result = 0
OVR = 0
C_{n+8} = 0

† C_{n+8} 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.

DATA FLOW



EXAMPLE (assumes a 24-bit cascaded system)

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.

Instruction Code I7-I0	Operand Address A3-A0	Operand Address B3-B0	Operand Select $\overline{EA}$ EB1-EB0	Destination Address C3-C0	Destination Select $\overline{WE}$ SELY $\overline{OEY}$	Carry-in C_n
1101 0001	0001	XXXX	0 10	0001	0 0 1	1

Assume register file 1 holds $08C618_{16}$, DB bus holds 007530_{16}, and MQ register holds $A99A0E_{16}$.

	Most Significant Package	Next Most Significant Package	Least Significant Package	
Source	0000 1000	1100 0110	0001 1000	R ← RF(1)
Source	0000 0000	0111 0101	0011 0000	S ← DB bus
Destination	0000 1001	0011 1011	0100 1001	RF(1) ← R + S + C_n
Source	1010 1001	1001 1010	0000 1110	MQ shifter ← MQ register
Destination	0101 0011	0011 0100	0001 1101	MQ register ← MQ shifter

FUNCTION

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

The contents of the MQ register are shifted one bit to the left. A zero is filled into bit 0 of the least-significant package unless $\overline{SIO0}$ is programmed low; this will force the least-significant bit to one. Bit 7 is passed through $\overline{QIO7}$-$\overline{QIO0}$ to bit 0 of the next-most-significant package. Bit 7 of the most-significant package 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 on page 2-9.

Shift Operations

ALU Shifter	MQ Shifter
None	Logical Left

Available Destination Operands
ALU Shifter:

RF (C3-C0)	RF (B3-B0)	Y-Port
•		•

Control/Data Signals

Signal	User Programmable	Use
SSF	Yes	Passes shift result if high or floating; retains MQ without shift if low.
$\overline{SIO0}$	Yes	$\overline{SIO0}$ fills a zero in LSB of MQ shifter if high or floating; sets LSB to one if low.
$\overline{SIO7}$	No	Inactive
$\overline{QIO0}$	No	Link cascaded MQ shifters. Output of MSP's $\overline{QIO7}$ is MSB of MQ shifter (inverted).
$\overline{QIO7}$	No	
C_n	No	Affects arithmetic operation programmed in bits I3-10 of instruction field.

Status Signals†

If arithmetic instruction specified in I3-I0:

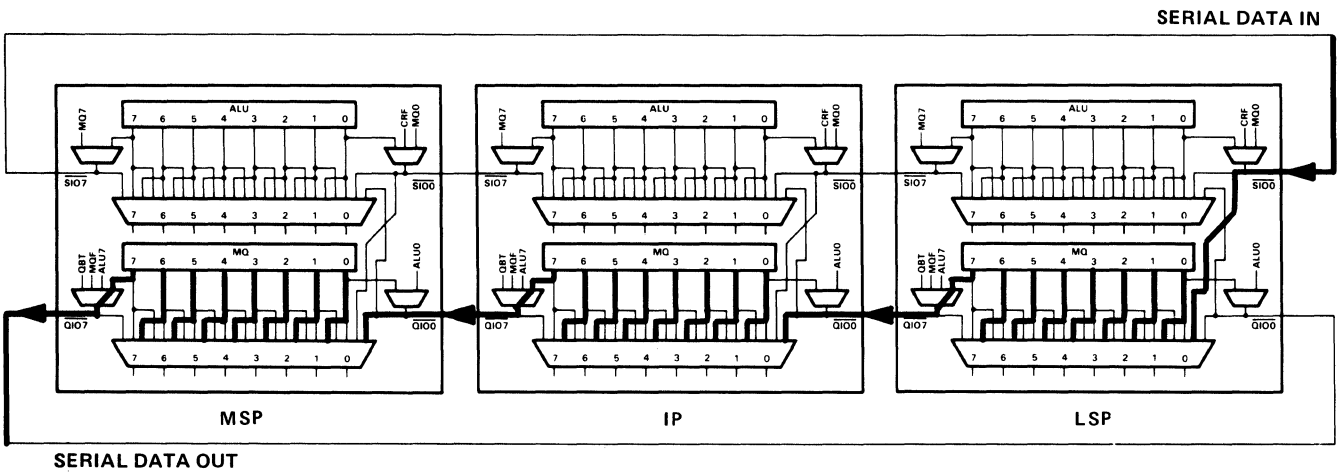
ZERO = 1 if result = 0
N = 1 if MSB of result = 1
= 0 if MSB of result = 0
OVR = 1 if signed arithmetic overflow
C_{n+8} = 1 if carry-out = 1

If logic instruction specified in I3-I0:

ZERO = 1 if result = 0
N = 1 if MSB of result = 1
= 0 if MSB of result = 0
OVR = 0
C_{n+8} = 0

† C_{n+8} 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.

DATA FLOW



EXAMPLE (assumes a 24-bit cascaded system)

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.

Instruction Code I7-I0	Operand Address A3-A0	Operand Address B3-B0	Operand Select C3-C0	End Fill $\overline{EA}$ EB1-EB0	Destination Address $\overline{SIOO}$	Destination Select $\overline{WE}$	SELY	$\overline{OEY}$	Carry-in C_n
1100 0001	0111	XXXX	0 10	1	0111	0	0	1	1

Assume register file 1 holds $08C618_{16}$, DB bus holds 007530_{16} and MQ register holds $A99A0E_{16}$.

	Most Significant Package	Next Most Significant Package	Least Significant Package	
Source	0000 1000	1100 0110	0001 1000	R ← RF(7)
Source	0000 0000	0111 0101	0011 0000	S ← DB bus
Destination	0000 1001	0011 1011	0100 1001	RF(7) ← R + S + C_n
Source	1010 1001	1001 1010	0000 1110	MQ shifter ← MQ register
Destination	0101 0011	0011 0100	0001 1100	MQ register ← MQ shifter

FUNCTION

Passes the result of the ALU instruction specified in the lower nibble of the instruction field to Y. Performs a 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.

The contents of the MQ register are shifted one bit to the right. The sign bit of the most-significant package is retained. Bit 0 is passed through $\overline{QIO0}$-$\overline{QIO7}$ to bit 7 of the next-most-significant package. Bit 0 of the least-significant package 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 on page 2-9.

Shift Operations

ALU Shifter	MQ Shifter
None	Arithmetic Right

Available Destination Operands
ALU Shifter:

RF (C3-C0)	RF (B3-B0)	Y-Port
•		•

Control/Data Signals

Signal	User Programmable	Use
SSF	Yes	Passes shift result if high or floating; retains MQ without shift if low.
$\overline{SIO0}$	No	Output value of LSP's $\overline{SIO0}$ is LSB of MQ shifter (inverted).
$\overline{SIO7}$	No	Inactive
$\overline{QIO0}$	No	Link cascaded MQ shifters. Output value of LSP's $\overline{QIO0}$ is LSB of MQ shifter (inverted).
$\overline{QIO7}$	No	
C_n	No	Affects arithmetic operation specified in bits I3-I0 of instruction field.

Status Signals†

If arithmetic instruction specified in I3-I0:

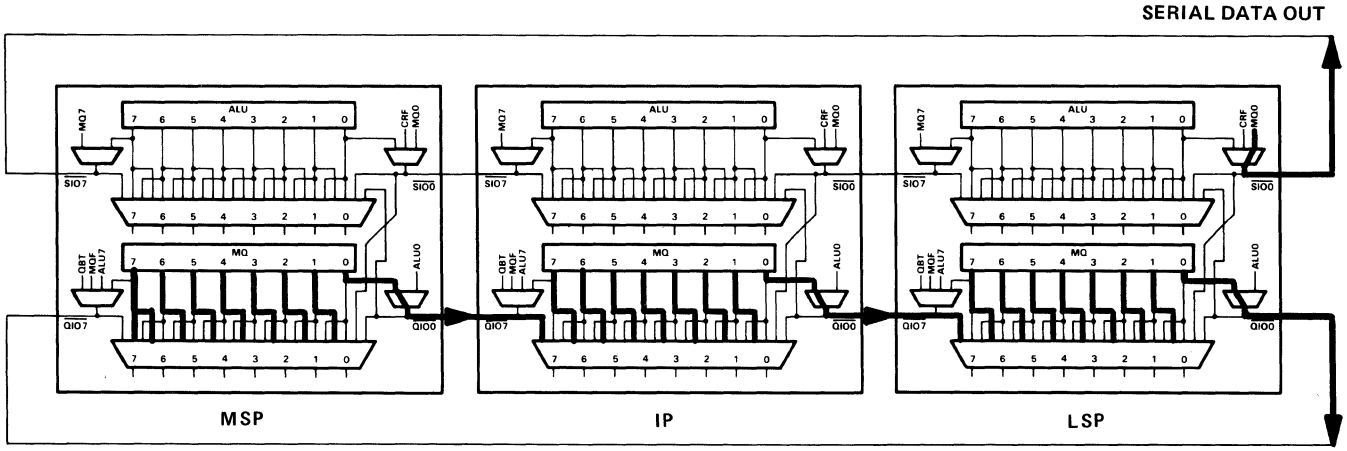
ZERO = 1 if result = 0
N = 1 if MSB of result = 1
= 0 if MSB of result = 0
OVR = 1 if signed arithmetic overflow
C_{n+8} = 1 if carry-out = 1

If logic instruction specified in I3-I0:

ZERO = 1 if result = 0
N = 1 if MSB of result = 1
= 0 if MSB of result = 0
OVR = 0
C_{n+8} = 0

† C_{n+8} 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.

DATA FLOW



EXAMPLE (assumes a 24-bit cascaded system)

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.

Instruction Code I7-I0	Operand Address A3-A0	Operand Address B3-B0	Operand Select $\overline{EA}$ EB1-EB0	Destination Address C3-C0	Destination Select $\overline{WE}$ SELY $\overline{OEY}$	Carry-in C_n
1010 0001	0001	1010	0 00	0001	0 0 1	1

Assume register file 1 holds $08C618_{16}$, DB bus holds 007530_{16}, and MQ register holds $A99A0E_{16}$.

	Most Significant Package	Next Most Significant Package	Least Significant Package	
Source	0000 1000	1100 0110	0001 1000	R ← RF(1)
Source	0000 0000	0111 0101	0011 0000	S ← RF(10)
Destination	0000 1001	0011 1011	0100 1001	RF(1) ← R + S + C_n
Source	1010 1001	1001 1010	0000 1110	MQ shifter ← MQ register
Destination	1101 0100	1100 1101	0000 0111	MQ register ← MQ shifter

FUNCTION

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

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 package unless $\overline{QIO7}$ is set to zero; this will force the sign bit to 1. Bit 0 is passed through $\overline{QIO0}$-$\overline{QIO7}$ to bit 7 of the next-most-significant package. Bit 0 of the least-significant package 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 on page 2-9.

Shift Operations

ALU Shifter	MQ Shifter
None	Logical Right

Available Destination Operands
ALU Shifter:

RF (C3-C0)	RF (B3-B0)	Y-Port
•		•

Control/Data Signals

Signal	User Programmable	Use
SSF	Yes	Passes shift result if high or floating; retains MQ without shift if low.
$\overline{SIO0}$	No	Output value of LSP's $\overline{SIO0}$ is LSB of MQ shifter (inverted).
$\overline{SIO7}$	No	Inactive
$\overline{QIO0}$	No	Link cascaded MQ shifters. $\overline{QIO7}$ fills a zero into MSB of MQ register if high or floating; sets MSB to one if low.
$\overline{QIO7}$	Yes	
C_n	No	Affects arithmetic operation specified in bits I3-I0 of instruction field.

Status Signals†

If arithmetic instruction specified in I3-I0:

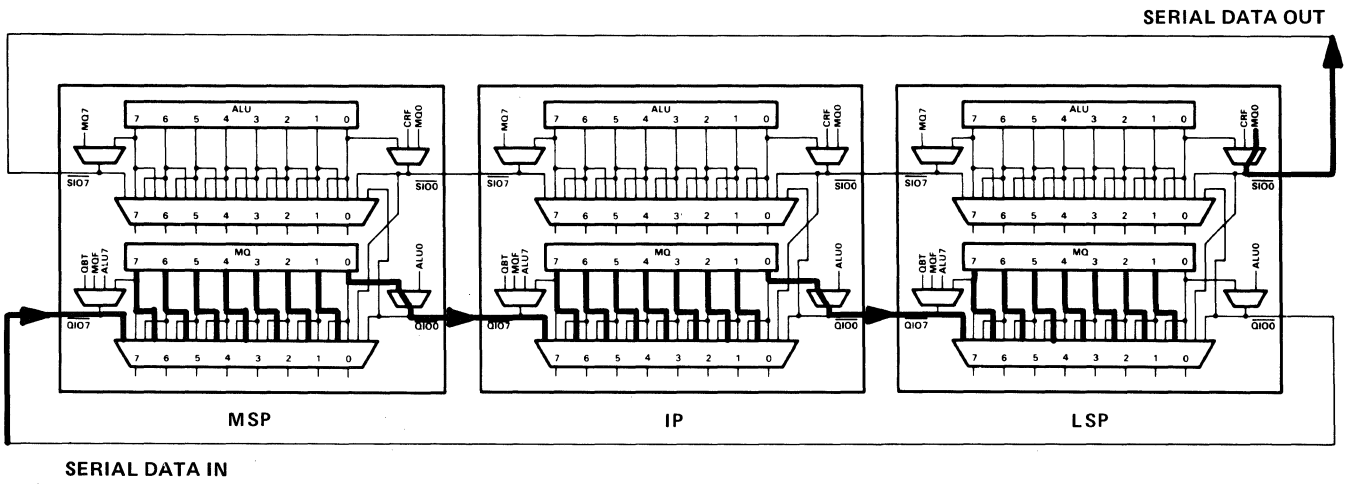
ZERO = 1 if result = 0
N = 1 if MSB of result = 1
= 0 if MSB of result = 0
OVR = 1 if signed arithmetic overflow
C_{n+8} = 1 if carry-out = 1

If logic instruction specified in I3-I0:

ZERO = 1 if result = 0
N = 1 if MSB of result = 1
= 0 if MSB of result = 0
OVR = 0
C_{n+8} = 0

† C_{n+8} 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.

DATA FLOW



EXAMPLE (assumes a 24-bit cascaded system)

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.

Instruction Code I7-I0	Operand Address A3-A0	Operand Address B3-B0	Operand Select C3-C0	End Fill $\overline{EA}$ EB1-EB0	Destination Address $\overline{SIO0}$	Destination Select $\overline{WE}$	SELY	$\overline{OEY}$	Carry-in C_n
1011 0001	0001	XXXX	0 10	1	0001	0	0	1	1

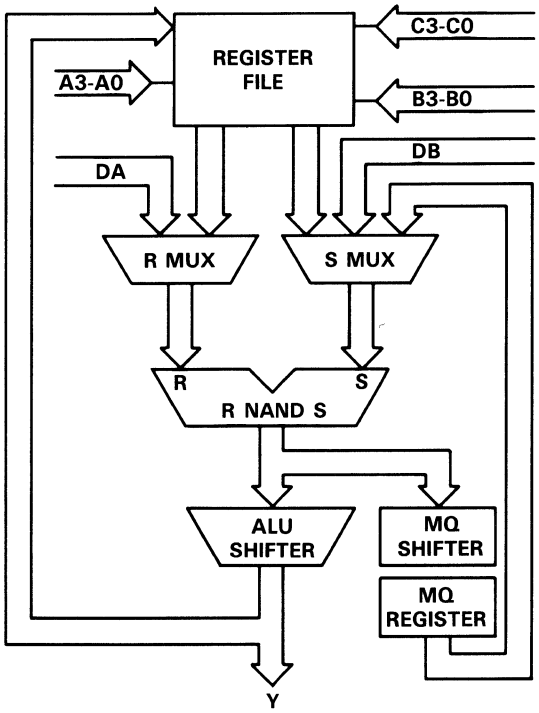
Assume register file 1 holds $08C618_{16}$, DB bus holds 007530_{16}, and MQ register holds $A99A0E_{16}$.

	Most Significant Package	Next Most Significant Package	Least Significant Package	
Source	0000 1000	1100 0110	0001 1000	R ← RF(1)
Source	0000 0000	0111 0101	0011 0000	S ← DB bus
Destination	0000 1001	0011 1011	0100 1001	RF(1) ← R + S + C_n
Source	1010 1001	1001 1010	0000 1110	MQ shifter ← MQ register
Destination	0101 0100	1100 1101	0000 0111	MQ register ← MQ shifter

FUNCTION

Evaluates the logical expression R NAND S.

DATA FLOW



Available R Bus Source Operands

RF (A3-A0)	A3-A0 Immediate	DA Port	A3-A0 : : C3-C0 Mask
•		•	

Available S Bus Source Operands

RF (B3-B0)	DB Port	MQ Register
•	•	•

Available Destination Operands

RF (C3-C0)	RF (B3-B0)	Y Port
•		•

Shift Operations

ALU	MQ
•	•

Control/Data Signals

Signal	User Programmable	Use
$\overline{SSF}$	No	Affect shift instructions programmed in bits I7-I4 of instruction field.
$\overline{SIO0}$	No	
$\overline{SIO7}$	No	
$\overline{QIO0}$	No	
$\overline{QIO7}$	No	
C_n		Inactive

Status Signals†

ZERO = 1 if result = 0
N = 1 if MSB = 1
OVR = 0
C_{n+8} = 0

† C_{n+8} 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.

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 on page 2-9.

EXAMPLE (assumes a 24-bit cascaded system)

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

Instruction Code I7-I0	Operand Address A3-A0	Operand Address B3-B0	Operand Select $\overline{EA}$ EB1-EB0	Destination Address C3-C0	Destination Select $\overline{WE}$ SELY $\overline{OEY}$	Carry-in C_n
1111 1100	0011	0101	0 00	0101	0 0 1	X

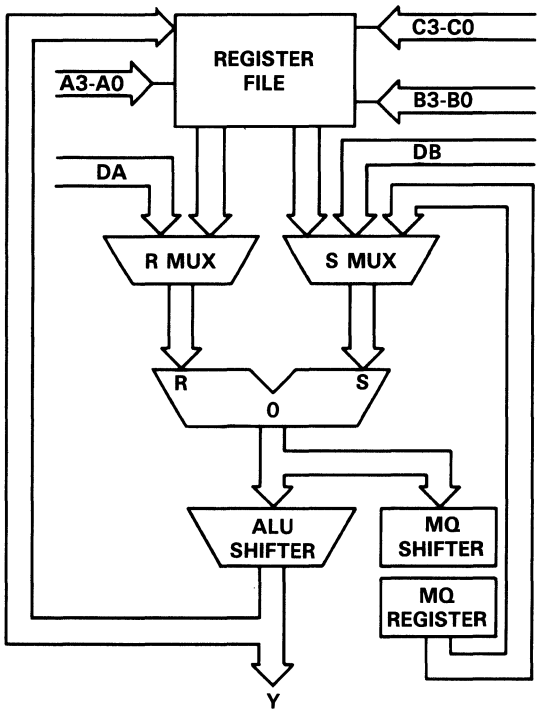
Assume register file 3 holds $F6D840_{16}$ and register file 5 holds $F6D842_{16}$.

	Most Significant Package	Next Most Significant Package	Least Significant Package	
Source	1111 0110	1101 1000	0100 0000	R ← RF(3)
Source	1111 0110	1101 1000	0100 0010	S ← RF(5)
Destination	0000 1001	0010 0111	1011 1111	RF(5) ← R NAND S

FUNCTION

Forces ALU output to zero.

DATA FLOW



Available R Bus Source Operands

RF (A3-A0)	A3-A0 Immediate	DA Port	A3-A0 : : C3-C0 Mask

Available S Bus Source Operands

RF (B3-B0)	DB Port	MQ Register

Available Destination Operands

RF (C3-C0)	RF (B3-B0)	Y Port
•		•

Shift Operations

ALU	MQ
None	None

Status Signals

ZERO = 1
N = 0
OVR = 0
C_{n+8} = 0

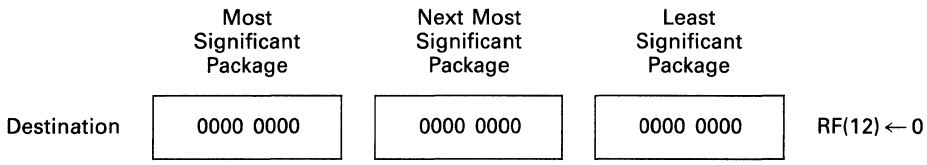
DESCRIPTION

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.

EXAMPLE (assumes a 24-bit cascaded system)

Clear register 12.

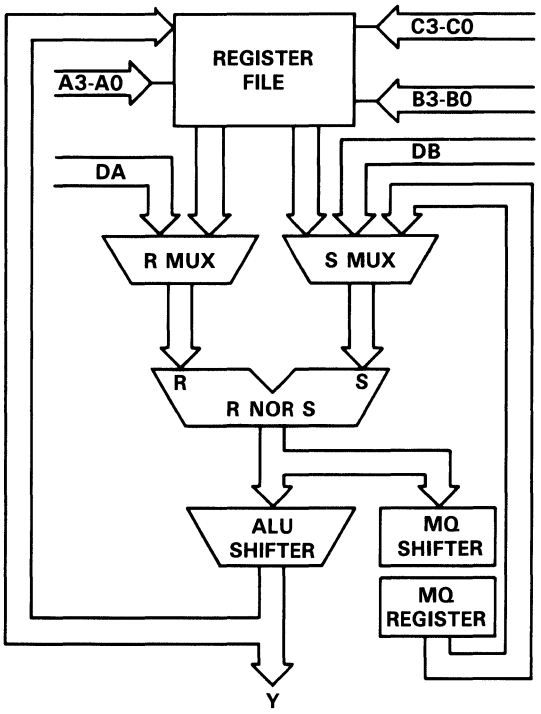
Instruction Code I7-I0	Operand Address A3-A0	Operand Address B3-B0	Operand Select $\overline{EA}$ EB1-EB0	Destination Address C3-C0	Destination Select $\overline{WE}$ SELY $\overline{OEY}$	Carry-in C_n
1111 1111	XXXX	XXXX	X XX	1100	0 0 1	X



FUNCTION

Evaluates the logical expression R NOR S.

DATA FLOW



Available R Bus Source Operands

RF (A3-A0)	A3-A0 Immediate	DA Port	A3-A0 : : C3-C0 Mask
•		•	

Available S Bus Source Operands

RF (B3-B0)	DB Port	MQ Register
•	•	•

Available Destination Operands

RF (C3-C0)	RF (B3-B0)	Y Port
•		•

Shift Operations

ALU	MQ
•	•

Control/Data Signals

Signal	User Programmable	Use
$\overline{SSF}$	No	Affect shift instructions programmed in bits I7-I4 of instruction field.
$\overline{SIO0}$	No	
$\overline{SIO7}$	No	
$\overline{QIO0}$	No	
$\overline{QIO7}$	No	
C_n	No	Inactive

Status Signals†

ZERO = 1 if result = 0
N = 1 if MSB = 1
OVR = 0
C_{n+8} = 0

† C_{n+8} 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.

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 on page 2-9.

EXAMPLE (assumes a 24-bit cascaded system)

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

Instruction Code I7-I0	Operand Address A3-A0	Operand Address B3-B0	Operand Select $\overline{EA}$ EB1-EB0	Destination Address C3-C0	Destination Select $\overline{WE}$ SELY $\overline{OEY}$	Carry-in C_n
1111 1011	0011	0101	0 00	0101	0 0 1	X

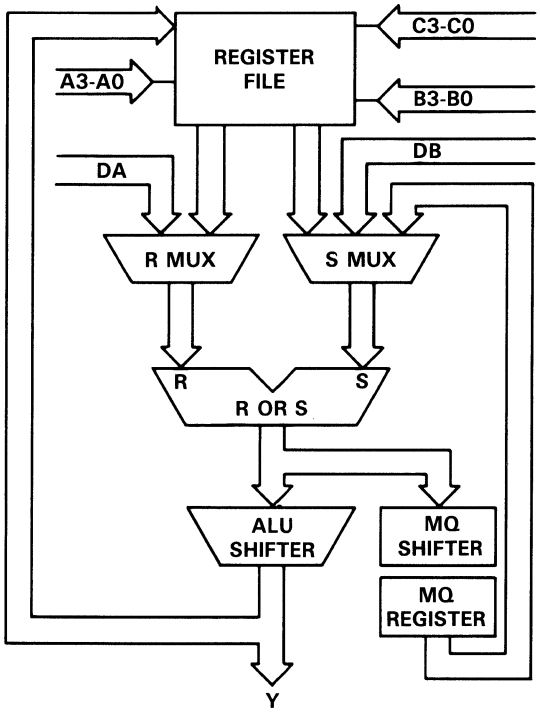
Assume register file 3 holds $F6D840_{16}$ and register file 5 holds $F6D842_{16}$.

	Most Significant Package	Next Most Significant Package	Least Significant Package	
Source	1111 0110	1101 1000	0100 0000	R ← RF(3)
Source	1111 0110	1101 1000	0100 0010	S ← RF(5)
Destination	0000 1001	0010 0111	1011 1101	RF(5) ← R NOR S

FUNCTION

Evaluates the logical expression R OR S.

DATA FLOW



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 on page 2-9.

Available R Bus Source Operands

RF (A3-A0)	A3-A0 Immediate	DA Port	A3-A0 : : C3-C0 Mask
•		•	

Available S Bus Source Operands

RF (B3-B0)	DB Port	MQ Register
•	•	•

Available Destination Operands

RF (C3-C0)	RF (B3-B0)	Y Port
•		•

Shift Operations

ALU	MQ
•	•

Control/Data Signals

Signal	User Programmable	Use
$\overline{SSF}$	No	Affect shift instructions programmed in bits I7-I4 of instruction field.
$\overline{SIO0}$	No	
$\overline{SIO7}$	No	
$\overline{QIO0}$	No	
$\overline{QIO7}$	No	
C_n	No	Inactive

Status Signals†

ZERO	=	1 if result = 0
N	=	1 if MSB = 1
OVR	=	0
C_{n+8}	=	0

† C_{n+8} 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 24-bit cascaded system)

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

Instruction Code I7-I0	Operand Address A3-A0	Operand Address B3-B0	Operand Select $\overline{EA}$ EB1-EB0	Destination Address C3-C0	Destination Select $\overline{WE}$ SELY $\overline{OEY}$	Carry-in C_n
1111 1011	0101	0011	0 00	0011	0 0 1	X

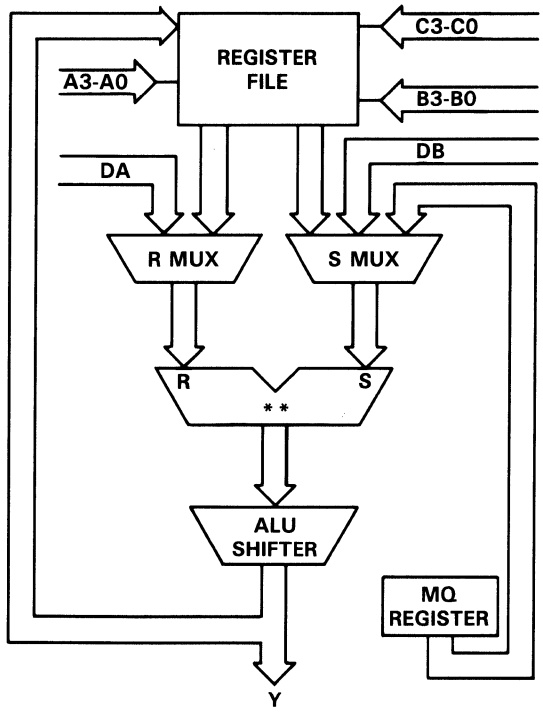
Assume register file 3 holds $F6D840_{16}$ and register file 5 holds $F6D842_{16}$.

	Most Significant Package	Next Most Significant Package	Least Significant Package	
Source	1111 0110	1101 1000	0100 0010	R ← RF(5)
Source	1111 0110	1101 1000	0100 0000	S ← RF(3)
Destination	1111 0110	1101 1000	0100 0010	RF(3) ← R OR S

FUNCTION

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

DATA FLOW



* *Arithmetic/logic function specified in I3-I0

Available Destination Operands

RF (C3-C0)	RF (B3-B0)	Y Port
•		•

Shift Operations

ALU	MQ
None	None

Control/Data Signals

Signal	User Programmable	Use
$\overline{SSF}$	No	Inactive
$\overline{SIO0}$	No	Inactive
$\overline{SIO7}$	No	Inactive
$\overline{QIO0}$	No	Inactive
$\overline{QIO7}$	No	Inactive
C_n	No	Affects arithmetic operation specified in bits I3-I0 of instruction field.

Status Signals†

If arithmetic instruction specified in I3-I0:

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_{n+8} = 1 if carry-out condition

If logic instruction specified in I3-I0:

ZERO = 1 if result = 0
N = 1 if MSB of result = 1
= 0 if MSB of result = 0
OVR = 0
C_{n+8} = 0

† C_{n+8} 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.

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.

* A list of ALU operations that can be used with this instruction is given on page 2-9.

EXAMPLE (assumes a 24-bit cascaded system)

Add data in register 1 to data on the DB bus with carry-in and store the unshifted result in register 10.

Instruction Code I7-I0	Operand Address A3-A0	Operand Address B3-B0	Operand Select $\overline{EA}$ EB1-EB0	Destination Address C3-C0	Destination Select $\overline{WE}$ SELY $\overline{OEY}$	Carry-in C_n
1111 0001	0001	XXXX	0 10	1010	0 0 1	1

Assume register file 3 holds $08C618_{16}$ and DB bus holds 007530_{16}.

	Most Significant Package	Next Most Significant Package	Least Significant Package	
Source	0000 1000	1100 0110	0001 1000	R ← RF(1)
Source	0000 0000	0111 0101	0011 0000	S ← DB bus
Destination	0000 1001	0011 1011	0100 1001	RF(10) ← R + S + C_n

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. A description of nonrestoring signed division and an algorithm using this instruction are given in section 2.3.1.

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 tests SSF, which holds the pass/fail result of the test subtraction from the previous instruction, and evaluates

$F \leftarrow R + S$ if SSF = 0
$F \leftarrow \overline{R} + S + C_n$ if SSF = 1.

A double precision left shift is performed; bit 7 of the most-significant package of the MQ shifter is transferred through $\overline{SIO7}$-$\overline{SIO0}$ to bit 1 of the least-significant package of the ALU shifter. Bit 7 of the most-significant package of the ALU shifter is lost. The unfixed quotient bit is circulated into the least-significant bit of MQ through $\overline{QIO7}$-$\overline{QIO0}$.

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 OVF at the end of the signed divide routine (after SDIVQF).

Available R Bus Source Operands

RF (A3-A0)	A3-A0 Immediate	DA Port	A3-A0 : : C3-C0 Mask
•		•	

Recommended S Bus Source Operands

RF (B3-B0)	DB Port	MQ Register
•	•	

Recommended Destination Operands

RF (C3-C0)	RF (B3-B0)	Y Port
•		

Shift Operations

ALU	MQ
Left	Left

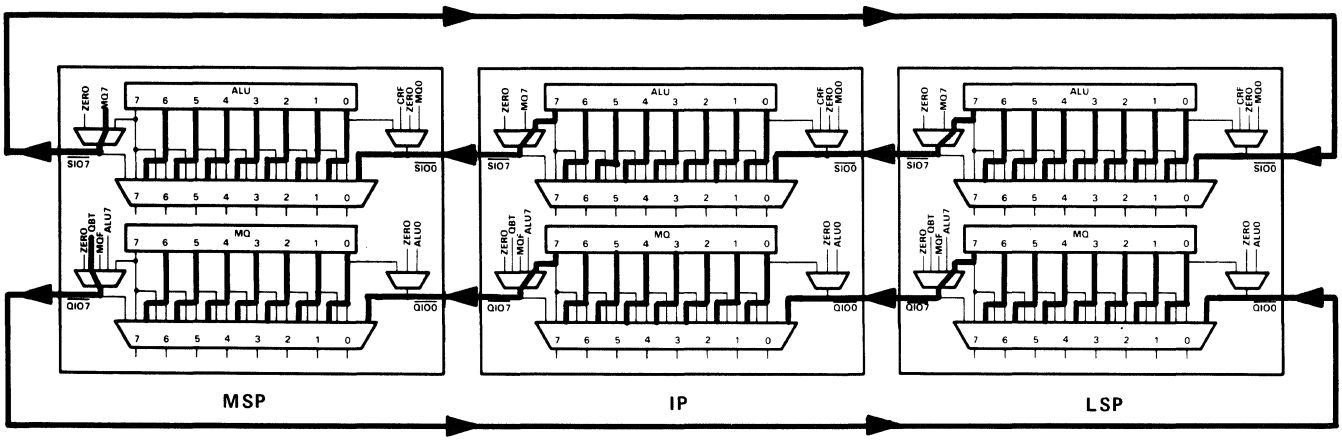
Control/Data Signals

Signal	User Programmable	Use
SSF	No	Preserves result of test subtraction for next instruction.
$\overline{SIO0}$	No	Link cascaded ALU shifters. Output value of MSP's $\overline{SIO7}$ is MSB of MQ shifter (inverted).
$\overline{SIO7}$	No	
$\overline{QIO0}$	No	Link cascaded MQ shifters. Output value of MSP's $\overline{QIO7}$ is unfixed quotient sign result.
$\overline{QIO7}$	No	
C_n	Yes	Should be programmed high.

Status Signals

ZERO	= 1 if intermediate result = 0
N	= 0
OVR	= 0
C_{n+8}	= 1 if carry-out

DATA FLOW



FUNCTION

Initializes 'AS888 for nonrestoring signed division by shifting the dividend left and internally preserving the sign bit. A description of nonrestoring signed division and an algorithm using this instruction are given in section 2.3.1.

DESCRIPTION

This instruction prepares for signed divide iteration operations by shifting the dividend and storing the sign for future use.

The preceding instruction should load the MQ register with the least-significant half of the dividend. During SDIVIN, the S bus should be loaded with the most-significant half of the dividend, and the R bus with the divisor. Y-output should be written back to the register file for use in the next instruction.

A double precision logical left shift is performed; bit 7 of the most-significant package of the MQ shifter is transferred through $\overline{SIO7}$-$\overline{SIO0}$ to bit 0 of the least-significant package of the ALU shifter. Bit 7 of the most-significant package of the ALU shifter is lost. The unfixed quotient sign bit (QBT) is shifted into the least-significant bit of MQ through $\overline{QIO7}$-$\overline{QIO0}$. SSF preserves the dividend's sign bit.

Available R Bus Source Operands

RF (A3-A0)	A3-A0 Immediate	DA Port	A3-A0 : : C3-C0 Mask
•		•	

Recommended S Bus Source Operands

RF (B3-B0)	DB Port	MQ Register
•	•	

Recommended Destination Operands

RF (C3-C0)	RF (B3-B0)	Y Port
•		

Shift Operations

ALU	MQ
Left	Left

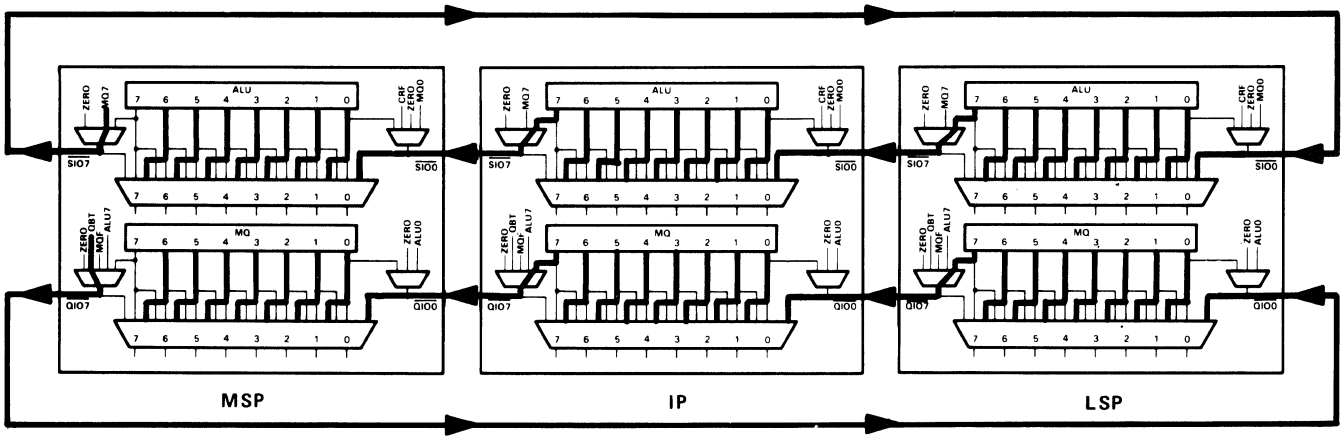
Control/Data Signals

Signal	User Programmable	Use
SSF	No	Preserves dividend's sign bit.
$\overline{SIO0}$	No	Link cascaded ALU shifters. Output value of MSP's $\overline{SIO7}$ is MSB of MQ shifter (inverted).
$\overline{SIO7}$	No	
$\overline{QIO0}$	No	Link cascaded MQ shifters. Output value of MSP's $\overline{QIO7}$ is unfixed quotient sign (QBT).
$\overline{QIO7}$	No	
C_n	No	Inactive

Status Signals

ZERO = 1 if divisor = 0
N = 0
OVR = 0
C_{n+8} = 0

DATA FLOW



FUNCTION

Computes the first quotient bit of nonrestoring signed division. A description of nonrestoring signed division and an algorithm using this instruction are given in section 2.3.1.

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; SSF is used to signal the subsequent SDIVI instruction to 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.

SDIVIS computes

$F \leftarrow R + S$ if SSF = 0
$F \leftarrow \overline{R} + S + C_n$ if SSF = 1

A double precision left shift is performed; bit 7 of the most-significant package of the MQ shifter is transferred through $\overline{SIO7}$-$\overline{SIO0}$ to bit 0 of the least-significant package of the ALU shifter. Bit 7 of the most-significant package of the ALU shifter is lost. The unfixed quotient bit is circulated into the least-significant bit of MQ through $\overline{QIO7}$-$\overline{QIO0}$.

Overflow occurring during SDIVIS is reported to OVF at the end of the signed division routine (after SDIVQF).

Available R Bus Source Operands

RF (A3-A0)	A3-A0 Immediate	DA Port	A3-A0 : : C3-C0 Mask
•		•	

Recommended S Bus Source Operands

RF (B3-B0)	DB Port	MQ Register
•	•	

Recommended Destination Operands

RF (C3-C0)	RF (B3-B0)	Y Port
•		

Shift Operations

ALU	MQ
Left	Left

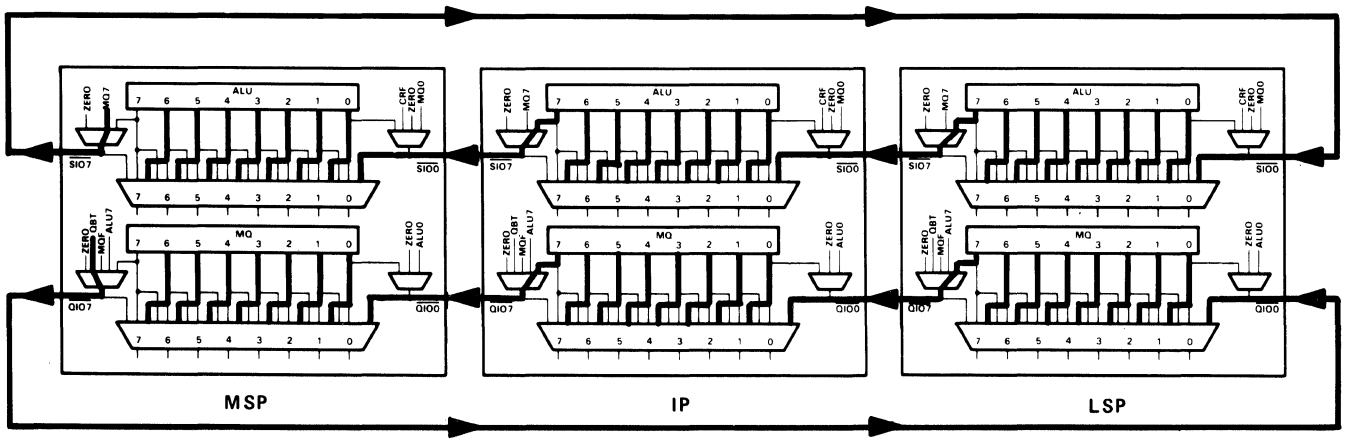
Control/Data Signals

Signal	User Programmable	Use
SSF	No	Preserves result of test subtraction for next instruction.
$\overline{SIO0}$	No	Link cascaded ALU shifters. Output value of MSP's $\overline{SIO7}$ is MSB of MQ shifter (inverted).
$\overline{SIO7}$	No	
$\overline{QIO0}$	No	Link cascaded MQ shifters. Output value of MSP's $\overline{QIO7}$ is unfixed quotient sign (QBT).
$\overline{QIO7}$	No	
C_n	Yes	Should be programmed high.

Status Signals

ZERO = 1 if intermediate result = 0
N = 0
OVR = 0
C_{n+8} = 1 if carry out

DATA FLOW



FUNCTION

Solves the final quotient bit during nonrestoring signed division. A description of nonrestoring signed division and an algorithm using this instruction are given in section 2.3.1.

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, 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 tests SSF, which holds the pass/fail result of the previous instruction's test subtraction, and evaluates

$Y \leftarrow R + S$ if SSF = 0
$Y \leftarrow \overline{R} + S + C_n$ if SSF = 1.

The contents of the MQ register are shifted one bit to the left; the unfixed quotient bit is circulated into the least-significant bit through $\overline{QIO7}$-$\overline{QIO0}$.

SSF is used to indicate to all slices whether the remainder must be corrected in the subsequent instruction. Overflow during this instruction is reported to OVF at the end of the signed division routine (after SDIVQF).

Available R Bus Source Operands

RF (A3-A0)	A3-A0 Immediate	DA Port	A3-A0 : : C3-C0 Mask
•		•	

Recommended S Bus Source Operands

RF (B3-B0)	DB Port	MQ Register
•	•	

Recommended Destination Operands

RF (C3-C0)	RF (B3-B0)	Y Port
•		

Shift Operations

ALU	MQ
Left	Left

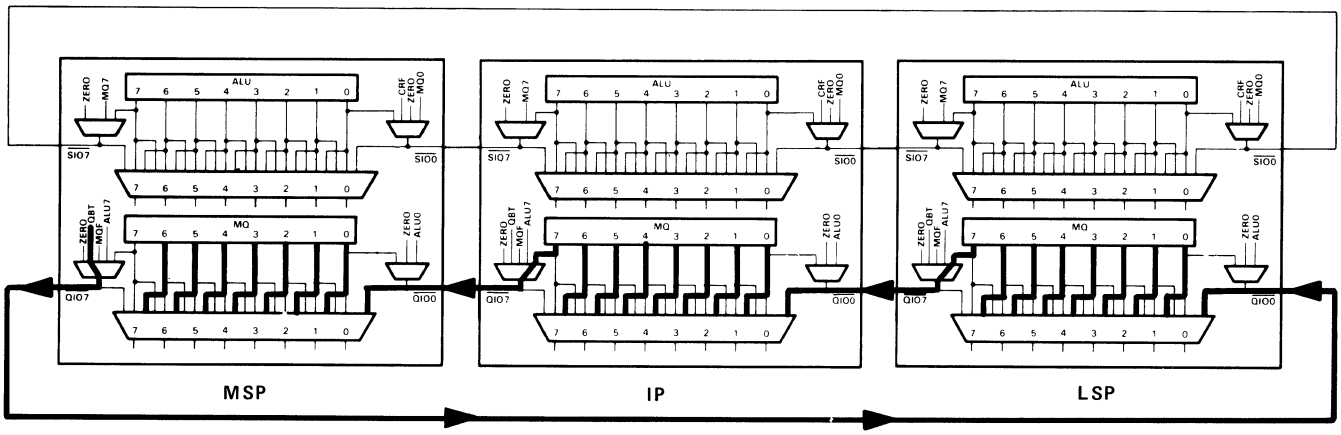
Control/Data Signals

Signal	User Programmable	Use
SSF	No	Indicates whether remainder fix is required in next instruction.
$\overline{SIO0}$	No	Inactive
$\overline{SIO7}$	No	Inactive
$\overline{QIO0}$	No	Link cascaded MQ shifters. Output value of MSP's $\overline{QIO7}$ is unfixed quotient sign (QBT).
$\overline{QIO7}$	No	
C_n	Yes	Should be programmed high

Status Signals

ZERO = 1 if intermediate result = 0
N = 0
OVR = 0
C_{n+8} = 1 if carry-out

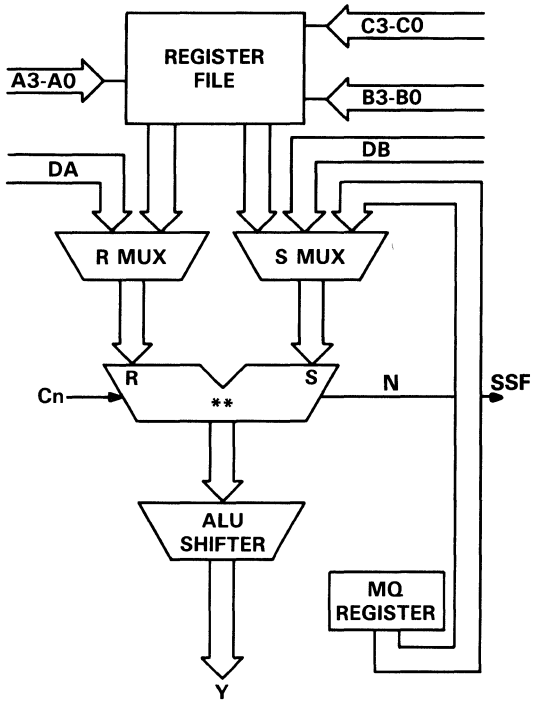
DATA FLOW



FUNCTION

Tests for overflow during nonrestoring signed division. A description of nonrestoring signed division and an algorithm using this instruction are given in section 2.3.1.

DATA FLOW



** $\overline{R} + S + 0$ if SSF = 0
$\overline{R} + S + 1$ if SSF = 1

Available R Bus Source Operands

RF (A3-A0)	A3-A0 Immediate	DA Port	A3-A0 : : C3-C0 Mask
•		•	

Recommended S Bus Source Operands

RF (B3-B0)	DB Port	MQ Register
•	•	

Recommended Destination Operands

RF (C3-C0)	RF (B3-B0)	Y Port
•		

Shift Operations

ALU	MQ
None	None

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Preserves dividend's sign bit from previous instruction.
$\overline{SIO0}$	No	Inactive
$\overline{SIO7}$	No	Inactive
$\overline{QIO0}$	No	Inactive
$\overline{QIO7}$	No	Inactive
C_n	Yes	Should be programmed high.

Status Signals

ZERO = 1 if divisor = 0
N = 0
OVR = 0
C_{n+8} = 1 if carry out

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 instruction tests SSF (sign of dividend) and then evaluates

$Y \leftarrow R + S$ if SSF = 0
$Y \leftarrow \overline{R} + S + C_n$ if SSF = 1.

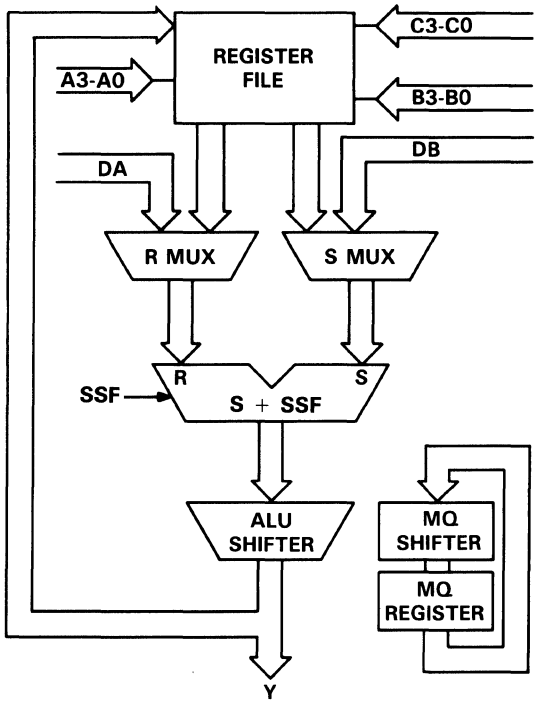
The result on the Y bus should not be stored back into the register file; $\overline{WE}$ should be programmed high.

Carry-in should also be programmed high. SSF is used to preserve the sign bit.

FUNCTION

Tests the quotient result after nonrestoring signed division and corrects it if necessary. A description of nonrestoring signed division and an algorithm using this instruction are given in section 2.3.1.

DATA FLOW



Available R Bus Source Operands

RF (A3-A0)	A3-A0 Immediate	DA Port	A3-A0 : : C3-C0 Mask
•		•	

Recommended S Bus Source Operands

RF (B3-B0)	DB Port	MQ Register
•	•	

Recommended Destination Operands

RF (C3-C0)	RF (B3-B0)	Y Port
•		•

Shift Operations

ALU	MQ
None	None

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Indicates whether quotient fix is required in this instruction; inactive at end of instruction cycle.
$\overline{SIO0}$	No	Inactive
$\overline{SIO7}$	No	Inactive
$\overline{QIO0}$	No	Inactive
$\overline{QIO7}$	No	Inactive
C_n	No	Inactive

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_{n+8}	= 1 if carry-out

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.

SSF is used to signal to all slices that correction is needed. The fix is implemented by adding SSF to S:

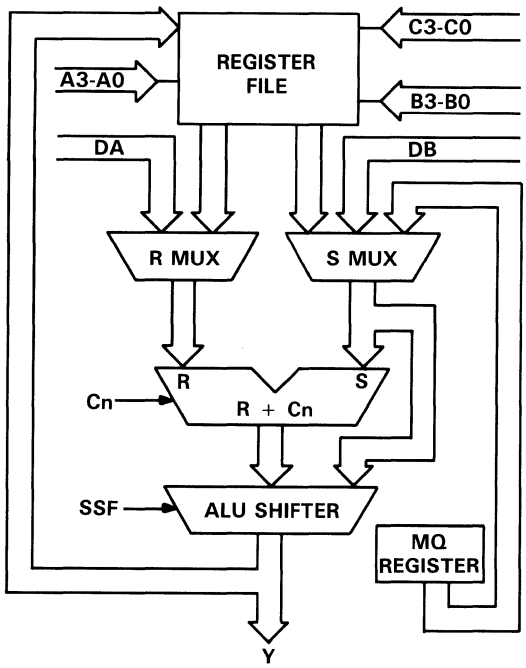
$Y \leftarrow S + 1$ if SSF = 1
$Y \leftarrow S + 0$ if SSF = 0.

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.

FUNCTION

Selects S if SSF is high; otherwise selects R.

DATA FLOW



Available R Bus Source Operands

RF (A3-A0)	A3-A0 Immediate	DA Port	A3-A0 : : C3-C0 Mask
•		•	

Available S Bus Source Operands

RF (B3-B0)	DB Port	MQ Register
•	•	•

Available Destination Operands

RF (C3-C0)	RF (B3-B0)	Y Port
•		•

Shift Operations

ALU	MQ
None	None

Control/Data Signals

Signal	User Programmable	Use
$\overline{SSF}$	Yes	Selects S if high, R if low.
$\overline{SIO0}$	No	Inactive
$\overline{SIO7}$	No	Inactive
$\overline{QIO0}$	No	Inactive
$\overline{QIO7}$	No	Inactive
C_n	Yes	Increments R if programmed high.

Status Signals

ZERO = 1 if result = 0
N = 1 if MSB = 1
OVR = 0
C_{n+8} = 0

DESCRIPTION

Data on the S bus is passed to Y if SSF is programmed high or floating; data on the R bus is passed with carry to Y if SSF is programmed low.

EXAMPLE (assumes a 24-bit cascaded system)

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 that SSF is set by the negative status (N) from the previous instruction).

Instruction Code I7-I0	Operand Address A3-A0	Operand Address B3-B0	Operand Select $\overline{EA}$ EB1-EB0	Destination Address C3-C0	Destination Select $\overline{WE}$ SELY $\overline{OEY}$	Carry-in C_n
1111 0011	0001	0011	0 00	XXXX	1 X X	1
0001 0000	0001	0011	0 00	0101	0 0 1	0

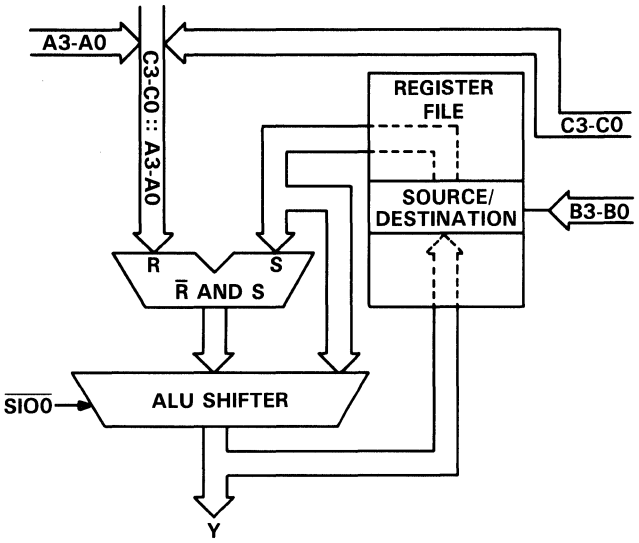
Assume register file 1 holds $0084D0_{16}$ and register file 3 holds $01C350_{16}$.

		Most Significant Package Byte 2 (selected)	Next Most Significant Package Byte 1 (selected)	Least Significant Package Byte 0 (not selected)	
Instruction Cycle 1:					
	Source	0000 0000	1000 0100	1101 0000	R ← RF(1)
	Source	0000 0001	1100 0011	0101 0000	S ← RF(3)
	Destination	1111 1110	1100 0001	1000 0000	Y Bus ← R + $\overline{S}$ + C_n
				1	N ← 1
Instruction Cycle 2:					
	Source	0000 0000	1000 0100	1101 0000	R ← RF(1)
				1	SSF ← 1
	Source	0000 0001	1100 0011	0101 0000	S ← RF(3)
	Destination	0000 0001	1100 0011	0101 0000	RF(5) ← S

FUNCTION

Resets bits in selected bytes of S-bus data using mask in C3-C0::A3-A0.

DATA FLOW



DESCRIPTION

The register addressed by B3-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. Slices with $\overline{SIO0}$ programmed low perform the Reset Bit instruction. Slices with $\overline{SIO0}$ programmed high or floating pass S unaltered.

Available R Bus Source Operands

RF (A3-A0)	A3-A0 Immediate	DA Port	A3-A0 : : C3-C0 Mask
			•

Available S Bus Source Operands

RF (B3-B0)	DB Port	MQ Register
•		

Available Destination Operands

RF (C3-C0)	RF (B3-B0)	Y Port
•		•

Shift Operations

ALU	MQ
None	None

Control/Data Signals

Signal	User Programmable	Use
$\overline{SSF}$	No	Inactive
$\overline{SIO0}$	No	Byte-select
$\overline{SIO7}$	No	Inactive
$\overline{QIO0}$	No	Inactive
$\overline{QIO7}$	No	Inactive
C_n	No	Inactive

Status Signals

ZERO = 1 if result (selected bytes) = 0
N = 0
OVR = 0
C_{n+8} = 0

EXAMPLE (assumes a 24-bit cascaded system)

Set bits 3-0 of bytes 1 and 2 of register file 8 to zero and store the result back in register 8.

Instruction Code I7-I0	Mask (LSH) A3-A0	Operand and Destination Address B3-B0	Mask (MSH) C3-C0	Operand Select $\overline{EA}$ EB1-EB0	Byte Select $\overline{SIO0}$	Destination Select $\overline{WE}$ SELY $\overline{OEY}$	Carry-in C_n
1001 1000	1111	1000	0000	X 00	001	0 0 1	X

Assume register file 8 holds $83BEBE_{16}$.

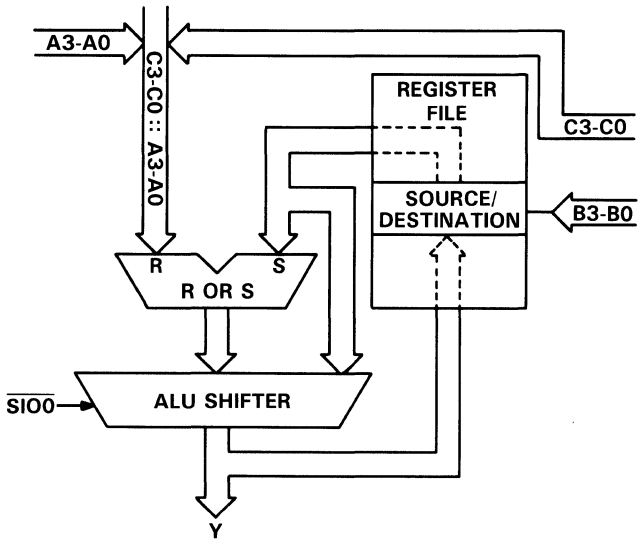
	Most Significant Package Byte 2 (selected)	Next Most Significant Package Byte 1 (selected)	Least Significant Package Byte 0 (not selected)	
Mask	0000 1111	0000 1111	0000 1111	$R_n \leftarrow$ C3-C0::A3-A0
Source	1000 0011	1011 1110	1011 1110	$S_n \leftarrow RF(3)_h$
ALU	1000 0000	1011 0000	1011 0000	$F_n \leftarrow S_n$ AND R_n
Destination	1000 0000	1011 0000	1011 1110	$RF(8)_n \leftarrow F_n$ or S_n†

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

FUNCTION

Sets bits in selected bytes of S-bus data using mask in C3-C0::A3-A0.

DATA FLOW



DESCRIPTION

The register addressed by B3-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. Slices with $\overline{SIO0}$ programmed low perform the Set Bit instruction. Slices with $\overline{SIO0}$ programmed high or floating pass S unaltered.

Available R Bus Source Operands

RF (A3-A0)	A3-A0 Immediate	DA Port	A3-A0 : : C3-C0 Mask
			•

Available S Bus Source Operands

RF (B3-B0)	DB Port	MQ Register
•		

Available Destination Operands

RF (C3-C0)	RF (B3-B0)	Y Port
•		•

Shift Operations

ALU	MQ
None	None

Control/Data Signals

Signal	User Programmable	Use
$\overline{SSF}$	No	Inactive
$\overline{SIO0}$	Yes	Byte-select
$\overline{SIO7}$	No	Inactive
$\overline{QIO0}$	No	Inactive
$\overline{QIO7}$	No	Inactive
C_n	No	Inactive

Status Signals

ZERO = 1 if result (selected bytes) = 0
N = 0
OVR = 0
C_{n+8} = 0

EXAMPLE (assumes a 24-bit cascaded system)

Set bits 3-0 of byte 1 of register file 1 to one and store the result back in register 1.

Instruction Code I7-I0	Mask (LSH) A3-A0	Operand and Destination Address B3-B0	Mask (MSH) C3-C0	Operand Select $\overline{EA}$ EB1-EB0	Byte Select $\overline{SIO0}$	Destination Select $\overline{WE}$ SELY $\overline{OEY}$	Carry-in C_n
0000 1000	1111	0001	0000	X 00	101	0 0 1	X

Assume register file 1 holds $83BEBE_{16}$.

	Most Significant Package Byte 2 (not selected)	Next Most Significant Package Byte 1 (selected)	Least Significant Package Byte 0 (not selected)	
Mask	0000 1111	0000 1111	0000 1111	$R_n \leftarrow$ (C3-C0::A3-A0)
Source	1000 0011	1011 1110	1011 1110	$S_n \leftarrow RF(1)_n$
ALU	1001 1111	1011 1111	1011 1111	$F_n \leftarrow S_n$ OR R_n
Destination	1000 0011	1011 1111	1011 1110	$RF(1)_n \leftarrow F_n$ or S_n†

† F = ALU result
n = nth package
Register file 1 gets F if byte selected, S if byte not selected.

SHIFT 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 package unless $\overline{SIO0}$ is programmed low; this will force bit 0 to one. Bit 7 is passed through $\overline{SIO7}$-$\overline{SIO0}$ to bit 0 of the next-most-significant package. Bit 7 of the most-significant package is dropped.

The shift may be made conditional on SSF. If SSF is high or floating, the shift result will be sent to Y. If SSF is low, F will be passed unaltered.

* A list of ALU operations that can be used with this instruction is given on page 2-9.

Shift Operations

ALU Shifter	MQ Shifter
Arithmetic Left	None

Available Destination Operands
ALU Shifter:

RF (C3-C0)	RF (B3-B0)	Y-Port
•		•

Control/Data Signals

Signal	User Programmable	Use
SSF	Yes	Passes shift result if high or floating; passes ALU result if low.
$\overline{SIO0}$	Yes	Link cascaded ALU shifters. $\overline{SIO0}$ fills a zero in LSB of ALU shifter if high or floating; sets MSB to one if low.
$\overline{SIO7}$	No	
$\overline{QIO0}$	No	Link cascaded MQ shifters. Output value of MSP's $\overline{QIO7}$ is MSB of ALU shifter (inverted).
$\overline{QIO7}$	No	
C_n	No	Affects arithmetic operation specified in bits I3-I0 of instruction field.

Status Signals†

If arithmetic instruction specified in I3-10:

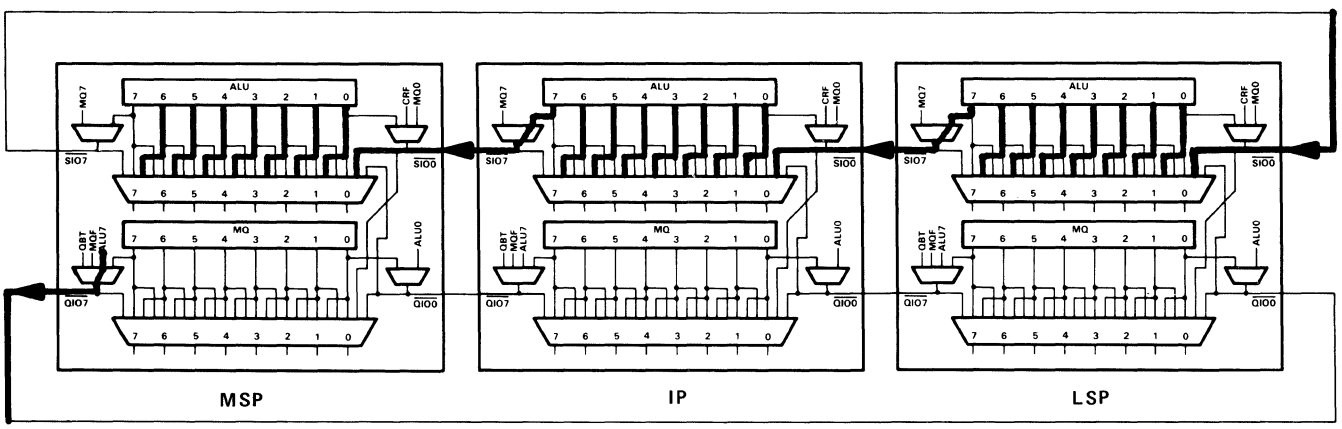
ZERO = 1 if result = 0
N = 1 if MSB of result = 1
= 0 if MSB of result = 0
OVR = 1 if signed arithmetic overflow or if ALU result MSB XOR MSB-1 = 1
C_{n+8} = 1 if carry-out condition

If logic instruction specified in I3-I0:

ZERO = 1 if result = 0
N = 1 if MSB of result = 1
= 0 if MSB of result = 0
OVR = 0
C_{n+8} = 0

† C_{n+8} 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.

DATA FLOW



EXAMPLE (assumes a 24-bit cascaded system)

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.

Instruction Code I7-I0	Operand Address 0001	Operand Address B3-B0	Operand Select $\overline{EA}$ EB1-EB0	End Fill $\overline{SIO0}$	Destination Address C3-C0	Destination Select $\overline{WE}$ SELY $\overline{OEY}$	Carry-in C_n
0100 0001	0001	XXXX	0 10	0	0001	0 0 1	0

Assume register file 1 holds $08C618_{16}$ and DB bus holds 007530_{16}.

	Most Significant Package	Next Most Significant Package	Least Significant Package	
Source	0000 1000	1100 0110	0001 1000	R ← RF(1)
Source	0000 0000	0111 0101	0011 0000	S ← DB bus
Intermediate Result	0000 1001	0011 1011	0100 1000	ALU Shifter ← R + S + C_n
Destination	0001 0010	0111 0110	1001 0000	RF(1) ← ALU shift result

FUNCTION

Performs arithmetic 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 register are shifted one bit to the left. A zero is filled into bit 0 of the least-significant package unless $\overline{SIO0}$ is set to zero; this will force bit 0 to one. Bit 7 is passed through $\overline{QIO7}$-$\overline{QIO0}$ to bit 0 of the next-most significant package. Bit 7 of the most-significant package is passed through $\overline{QIO7}$-$\overline{QIO0}$ to to bit 0 of the least-significant package of the ALU. Bit 7 of the least-significant package's ALU is passed through $\overline{SIO7}$-$\overline{SIO0}$ to bit 0 of the next-most-significant package. Bit 7 of the most-significant-package is dropped.

The shift may be made conditional on SSF. If SSF is high or floating, the shift result will be sent to Y and the MQ register. If SSF is low, F will be passed unaltered, and the MQ register will not be changed.

* A list of ALU operations that can be used with this instruction is given on page 2-9.

Shift Operations

ALU Shifter	MQ Shifter
Arithmetic Left	Arithmetic Left

Available Destination Operands
ALU Shifter:

RF (C3-C0)	RF (B3-B0)	Y-Port
•		•

Control/Data Signals

Signal	User Programmable	Use
SSF	Yes	Passes shift result if high or floating; passes ALU result and retains MQ register if low.
$\overline{SIO0}$	Yes	Link cascaded ALU shifters. $\overline{SIO0}$ fills a zero in LSB of MQ shifter if high or floating; sets LSB to one if low.
$\overline{SIO7}$	No	
$\overline{QIO0}$	No	Link cascaded MQ shifters. Output value of MSP's $\overline{QIO7}$ is MSB of MQ shifter (inverted).
$\overline{QIO7}$	No	
C_n	No	Affects arithmetic operation specified in bits I3-I0 of instruction field.

Status Signals†

If arithmetic instruction specified in I3-10:

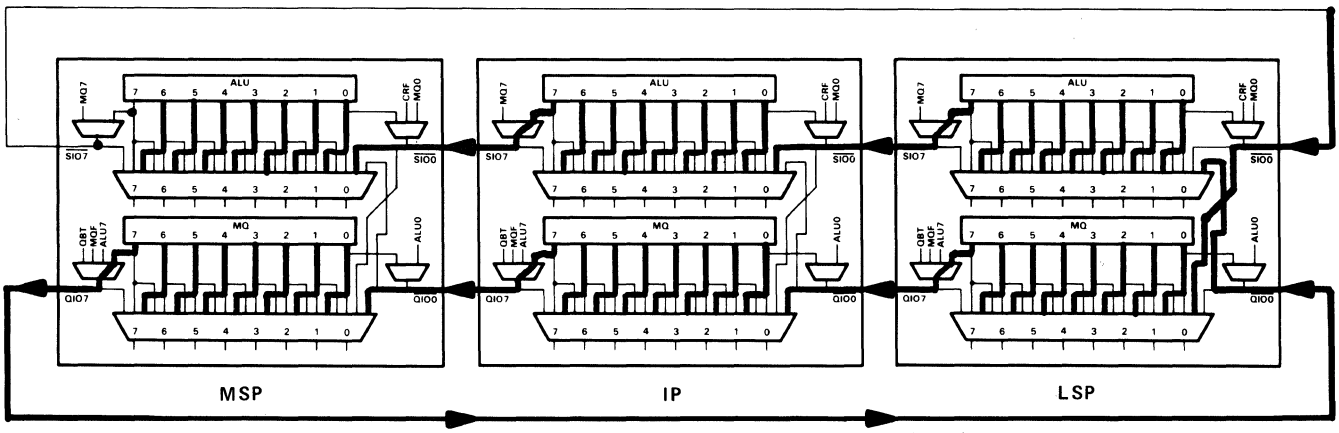
ZERO = 1 if result = 0
N = 1 if MSB of result = 1
= 0 if MSB of result = 0
OVR = 1 if signed arithmetic overflow or if ALU result MSB XOR MSB−1 = 1
C_{n+8} = 1 if carry-out condition

If logic instruction specified in I3-I0:

ZERO = 1 if result = 0
N = 1 if MSB of result = 1
= 0 if MSB of result = 0
OVR = 0
C_{n+8} = 0

† C_{n+8} 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.

DATA FLOW



EXAMPLE (assumes a 24-bit cascaded system)

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.

Instruction Code I7-I0	Operand Address A3-A0	Operand Address B3-B0	Operand Select $\overline{EA}$ EB1-EB0	End Fill $\overline{SIO0}$	Destination Address C3-C0	Destination Select $\overline{WE}$ SELY $\overline{OEY}$	Carry-in C_n
0101 0001	0001	XXXX	0 10	0	0001	0 0 1	0

Assume register file 1 holds $08C618_{16}$, DB bus holds 007530_{16} and MQ register holds $A99A0E_{16}$.

		Most Significant Package	Next Most Significant Package	Least Significant Package	
MSH:					
	Source	0000 1000	1100 0110	0001 1000	R ← RF(1)
	Source	0000 0000	0111 0101	0011 0000	S ← DB bus
	Intermediate Result	0000 1001	0011 1011	0100 1000	ALU Shifter ← R + S + C_n
	Destination	0001 0010	0111 0110	1001 0001	RF(1) ← ALU shift result
LSH:					
	Source	1010 1001	1001 1010	0000 1110	MQ shifter ← MQ register
	Destination	0101 0011	0111 0100	0001 1100	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 least-significant package is passed through $\overline{SIO7}$-$\overline{SIO0}$ to bit 0 of the next-most significant package. Bit 7 of the most-significant package is passed to bit 0 of the least-significant package.

The shift may be made conditional on SSF. If SSF is high or floating, the shift result will be sent to Y. If SSF is low, F will be passed unaltered.

* A list of ALU operations that can be used with this instruction is given on page 2-9.

Shift Operations

ALU Shifter	MQ Shifter
Circular Left	None

Available Destination Operands
ALU Shifter:

RF (C3-C0)	RF (B3-B0)	Y-Port
•		•

Control/Data Signals

Signal	User Programmable	Use
SSF	Yes	Passes shift result if high or floating; passes ALU result if low.
$\overline{SIO0}$	No	Link cascaded ALU shifters. Output value of MSP's $\overline{SIO7}$ is MSB of ALU (inverted).
$\overline{SIO7}$	No	
$\overline{QIO0}$	No	Inactive
$\overline{QIO7}$	No	Inactive
C_n	No	Affects arithmetic operation specified in bits I3-I0 of instruction field.

Status Signals†

If arithmetic instruction specified in I3-I0:

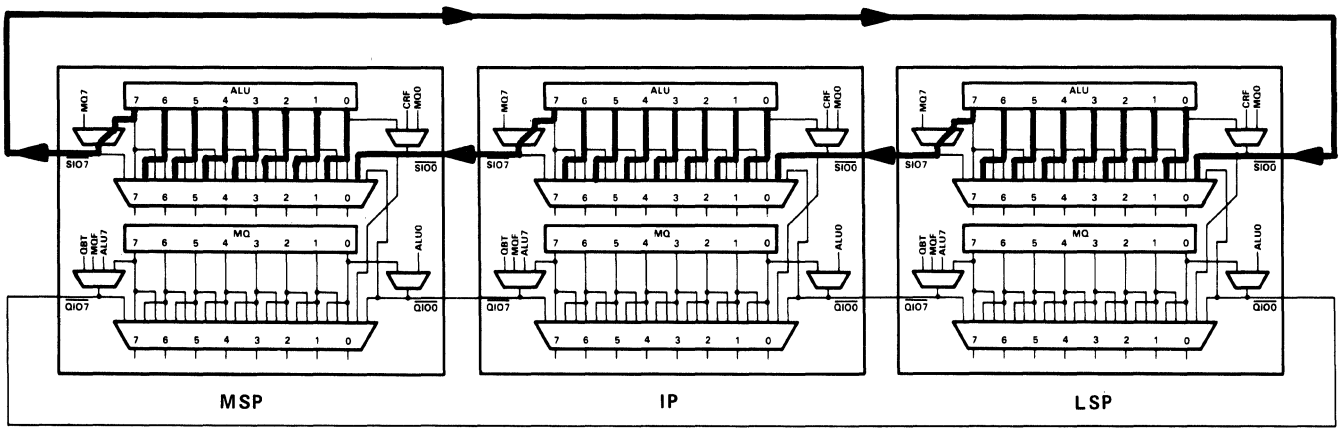
ZERO = 1 if result = 0
N = 1 if MSB of result = 1
= 0 if MSB of result = 0
OVR = 1 if signed arithmetic overflow
C_{n+8} = 1 if carry-out condition

If logic instruction specified in I3-I0:

ZERO = 1 if result = 0
N = 1 if MSB of result = 1
= 0 if MSB of result = 0
OVR = 0
C_{n+8} = 0

† C_{n+8} 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.

DATA FLOW



EXAMPLE (assumes a 24-bit cascaded system)

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

Instruction Code I7-I0	Operand Address A3-A0	Operand Address B3-B0	Operand Select $\overline{EA}$ EB1-EB0	Destination Address C3-C0	Destination Select $\overline{WE}$ SELY $\overline{OEY}$	Carry-in C_n
0110 0110	0110	XXXX	0 X	0001	0 0 1	0

Assume register file 6 holds $88C618_{16}$.

	Most Significant Package	Next Most Significant Package	Least Significant Package	
Source	1000 1000	1100 0110	0001 1000	R ← RF(6)
Intermediate Result	1000 1000	1100 0110	0001 1000	ALU Shifter ← R + C_n
Destination	0001 0001	1000 1100	0011 0001	RF(1) ← ALU shift 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 least-significant package of the MQ register is passed through $\overline{QIO7}$-$\overline{QIO0}$ to bit 0 of the next-most-significant package. Bit 7 of the most-significant package is passed through $\overline{QIO7}$-$\overline{QIO0}$ to bit 0 of the least-significant package of the ALU. Bit 7 of the least-significant package's ALU is passed through $\overline{SIO7}$-$\overline{SIO0}$ to bit 0 of the next-most-significant package. Bit 7 of the most-significant package is passed through $\overline{SIO7}$-$\overline{SIO0}$ to bit 0 of the least-significant package's MQ register.

The shift may be made conditional on SSF. If SSF is high or floating, the shift result will be sent to Y and to the MQ register. If SSF is low, F is passed unaltered, and the MQ register will not be changed.

* A list of ALU operations that can be used with this instruction is given on page 2-9.

Shift Operations

ALU Shifter	MQ Shifter
Circular Left	Circular Left

Available Destination Operands ALU Shifter:

RF (C3-C0)	RF (B3-B0)	Y-Port
•		•

Control/Data Signals

Signal	User Programmable	Use
SSF	Yes	Passes shift result if high or floating; passes ALU result and retains MQ register if low.
$\overline{SIO0}$	No	Link cascaded ALU shifters. Output value of MSP's $\overline{SIO7}$ is MSB of ALU shifter (inverted).
$\overline{SIO7}$	No	
$\overline{QIO0}$	No	Link cascaded MQ shifters. Output value of MSP's $\overline{QIO7}$ is MSB of MQ shifter (inverted).
$\overline{QIO7}$	No	
C_n	No	Affects arithmetic operation specified in bits I3-I0 of instruction field.

Status Signals†

If arithmetic instruction specified in I3-I0:

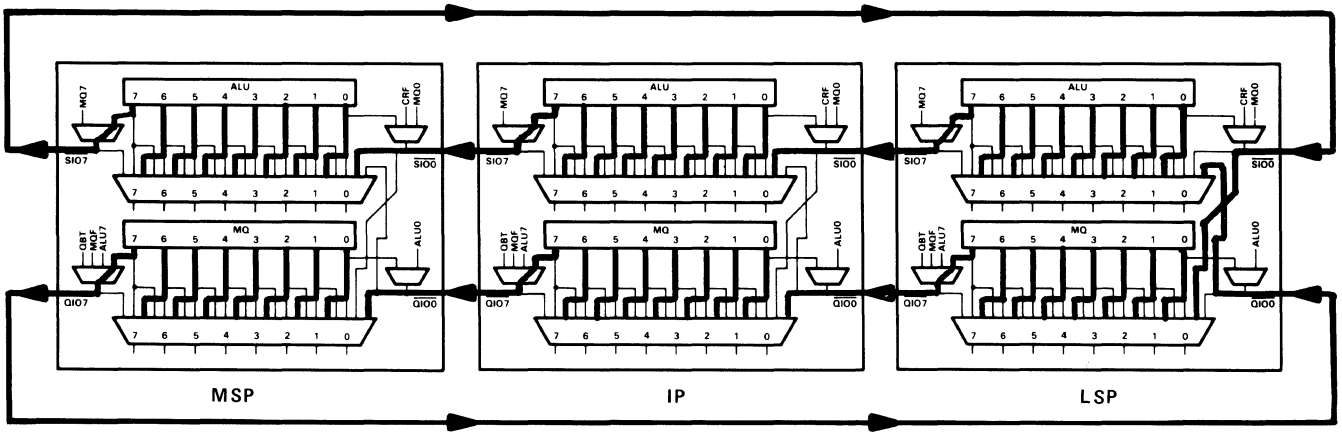
ZERO = 1 if result = 0
N = 1 if MSB of result = 1
= 0 if MSB of result = 0
OVR = 1 if signed arithmetic overflow
C_{n+8} = 1 if carry-out condition

If logic instruction specified in I3-I0:

ZERO = 1 if result = 0
N = 1 if MSB of result = 1
= 0 if MSB of result = 0
OVR = 0
C_{n+8} = 0

† C_{n+8} 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.

DATA FLOW



EXAMPLE (assumes a 24-bit cascaded system)

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.

Instruction Code I7-I0	Operand Address A3-A0	Operand Address B3-B0	Operand Select $\overline{EA}$ EB1-EB0	Destination Address C3-C0	Destination Select $\overline{WE}$ SELY $\overline{OEY}$	Carry-in C_n
0111 0110	0110	XXXX	0 XX	0110	0 0 1	0

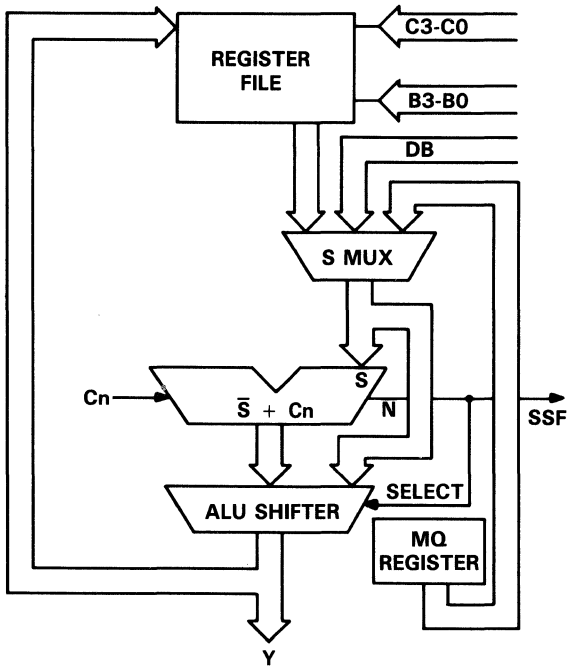
Assume register file 6 holds $08C618_{16}$ and MQ register holds $A99A0E_{16}$.

		Most Significant Package	Next Most Significant Package	Least Significant Package	
MSH:					
	Source	0000 1000	1100 0110	0001 1000	R ← RF(6)
	Intermediate Result	0000 1000	1100 0110	0001 1000	ALU Shifter ← R + C_n
	Destination	0001 0001	1000 1100	0011 0001	RF(6) ← ALU shift result
LSH:					
	Source	1010 1001	1001 1010	0000 1110	MQ shifter ← MQ register
	Destination	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.

DATA FLOW



Available R Bus Source Operands

RF (A3-A0)	A3-A0 Immediate	DA Port	A3-A0 : : C3-C0 Mask

Available S Bus Source Operands

RF (B3-B0)	DB Port	MQ Register
•	•	•

Available Destination Operands

RF (C3-C0)	RF (B3-B0)	Y Port
•		•

Shift Operations

ALU	MQ
None	None

Control/Data Signals

Signal	User Programmable	Use
SSF	No	Carries result of sign bit test from MSP
$\overline{SIO0}$	No	Inactive
$\overline{SIO7}$	No	Inactive
$\overline{QIO0}$	No	Inactive
$\overline{QIO7}$	No	Inactive
C_n	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 package is 80_{16} and results in all other packages are 00_{16}.
C_{n+8}	=	1 if S = 0

DESCRIPTION

The S bus is 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 package is tested; if it is high, the SSF pin is pulled low, and the converted number is passed to the destination operands. If the SSF pin is high, the operand is passed unaltered. The SSF signal from the most-significant package is used as an input to all other packages to determine whether the operand is passed altered or unaltered.

EXAMPLES (assume a 24-bit cascaded system)

Convert the two's complement number in register 1 to signed magnitude representation and store the result in register 4.

Instruction Code I7-I0	Operand Address A3-A0	Operand Address B3-B0	Operand Select $\overline{EA}$ EB1-EB0	Destination Address C3-C0	Destination Select $\overline{WE}$ SELY $\overline{OEY}$	Carry-in C_n
0101 1000	XXXX	0001	X 00	0100	0 0 1	1

Example 1: Assume register file 1 holds $F6D840_{16}$.

	Most Significant Package	Next Most Significant Package	Least Significant Package	
Source	1111 0110	1101 1000	0100 0000	S ← RF(1)
Destination	1000 1001	0010 0111	1100 0000	RF(4) ← $\overline{S} + C_n$ (sign bit retained)

Example 2: Assume register file 1 holds $0927C0_{16}$.

Source	0000 1001	0010 0111	1100 0000	S ← RF(1)
Destination	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 section 2.3.2.

DESCRIPTION

SMULI tests SSF to determine whether the multiplicand should be added with the present partial product. The instruction evaluates

$F \leftarrow R + S + C_n$ if SSF = 1
$F \leftarrow S$ if SSF = 0

A double precision right shift is performed. Bit 0 of the least-significant package of the ALU shifter is passed through $\overline{QIO0}$-$\overline{QIO7}$ to bit 7 of the most-significant package 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 (A3-A0)	A3-A0 Immediate	DA Port	A3-A0 : : C3-C0 Mask
•		•	

Recommended S Bus Source Operands

RF (B3-B0)	DB Port	MQ Register
•	•	

Recommended Destination Operands

RF (C3-C0)	RF (B3-B0)	Y Port
•		

Shift Operations

ALU	MQ
Right	Right

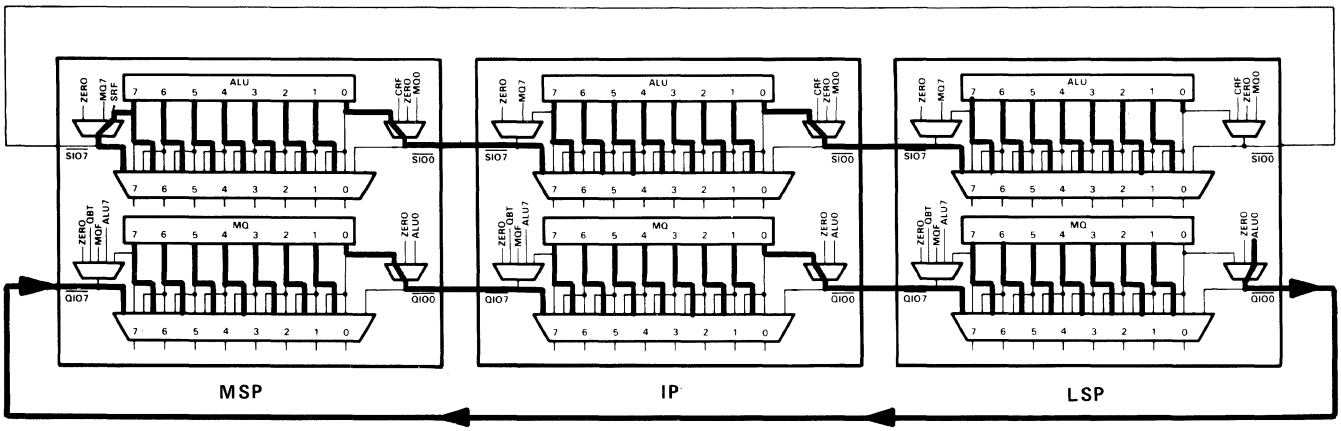
Control/Data Signals

Signal	User Programmable	Use
SSF	No	Indicates whether multiplicand should be added with partial product.
$\overline{SIO0}$	No	Link cascaded ALU shifters.
$\overline{SIO7}$	No	
$\overline{QIO0}$	No	Link cascaded MQ shifters. Output value of MSP's $\overline{QIO7}$ is LSB of ALU shifter (inverted).
$\overline{QIO7}$	No	
C_n	Yes	Should be programmed low.

Status Signals

ZERO	=	1 if result = 0
N	=	1 if MSB = 1
OVR	=	0
C_{n+8}	=	1 if carry-out

DATA FLOW



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 section 2.3.2.

DESCRIPTION

SMULT tests SSF, which holds 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

$F \leftarrow \overline{R} + S + C_n$ if SSF = 1
$F \leftarrow S + 0$ if SSF = 0

with the correct sign in the product.

A double precision right shift is performed. Bit 0 of the least-significant package of the ALU shifter is passed through $\overline{QIO0}$-$\overline{QIO7}$ to bit 7 of the most-significant package of the MQ shifter.

The S bus must be loaded with the contents of the 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 (A3-A0)	A3-A0 Immediate	DA Port	A3-A0 : : C3-C0 Mask
•		•	

Recommended S Bus Source Operands

RF (B3-B0)	DB Port	MQ Register
•	•	

Available Destination Operands

RF (C3-C0)	RF (B3-B0)	Y Port
•		•

Shift Operations

ALU	MQ
Right	Right

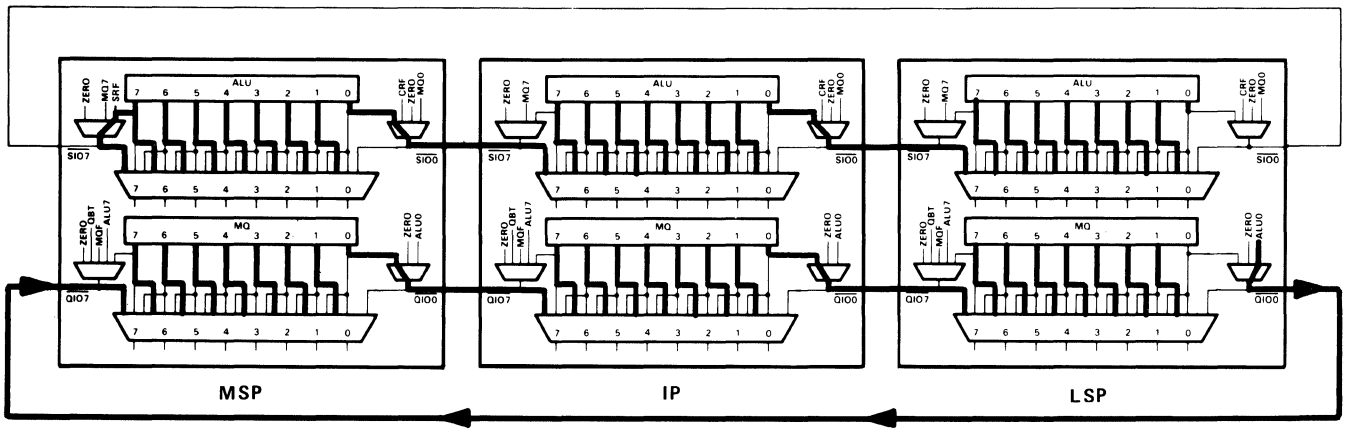
Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inactive
$\overline{SIO0}$	No	Link cascaded ALU shifters. Input value of MSP's $\overline{SIO7}$ is sign remainder fix (SRF).
$\overline{SIO7}$	No	
$\overline{QIO0}$	No	Link cascaded MQ shifters. Output value of MSP's $\overline{QIO0}$ is LSB of ALU shifter.
$\overline{QIO7}$	No	
C_n	Yes	Should be programmed high.

Status Signals

ZERO	=	1 if result = 0
N	=	1 if MSB = 1
OVR	=	0
C_{n+8}	=	1 if carry-out

DATA FLOW



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 to the left and filling a zero into the LSB via the $\overline{QIO0}$ input. 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 SSF pin inhibits the shift and the S bus increment whenever normalization is attempted on a number already normalized. Normalization is complete when overflow occurs.

Available R Bus Source Operands

RF (A3-A0)	A3-A0 Immediate	DA Port	A3-A0 : : C3-C0 Mask

Available S Bus Source Operands

RF (B3-B0)	DB Port	MQ Register
•		

Available Destination Operands

RF (C3-C0)	RF (B3-B0)	Y Port
•		

Shift Operations

ALU	MQ
None	Left

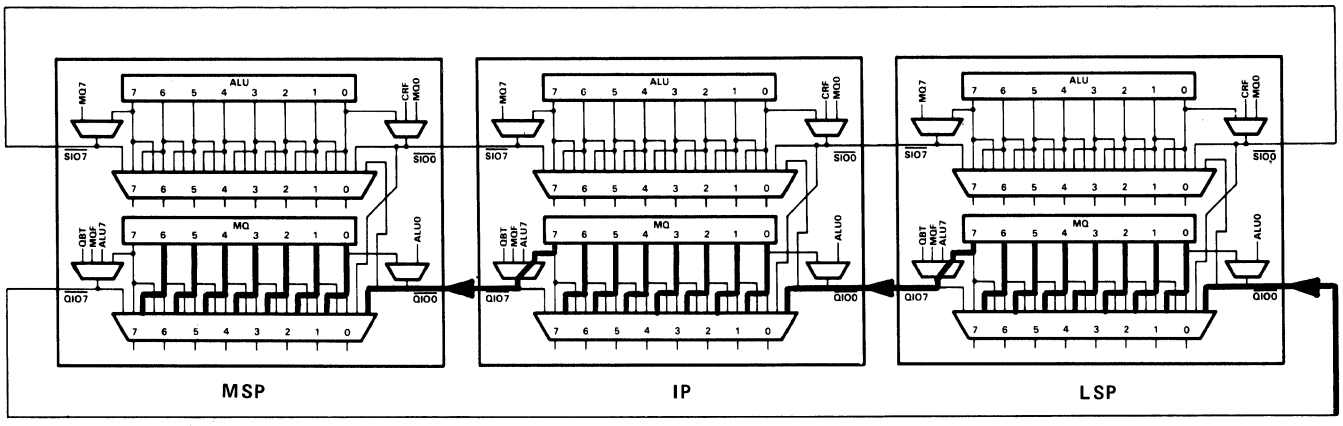
Control/Data Signals

Signal	User Programmable	Use
SSF	No	Inhibits shift if normalization is complete.
$\overline{SIO0}$	No	Inactive
$\overline{SIO7}$	No	Inactive
$\overline{QIO0}$	No	Link cascaded MQ shifters. $\overline{QIO0}$ fills a zero into LSB of MQ shifter.
$\overline{QIO7}$	No	
C_n	Yes	Increments S bus (shift count) if set to one.

Status Signals

ZERO	=	1 if MQ result = 0
N	=	1 if MSB of MQ register = 1
OVR	=	1 if MSB of MQ register XOR 2nd MSB = 1
C_{n+8}	=	1 if carry-out = 1

DATA FLOW



EXAMPLE (assumes a 24-bit cascaded system)

Normalize the number in the MQ register, storing the number of shifts in register 3.

Instruction Code I7-I0	Operand Address A3-A0	Operand Address B3-B0	Operand Select $\overline{EA}$ EB1-EB0	Destination Address C3-C0	Destination Select $\overline{WE}$ SELY $\overline{OEY}$	Carry-in C_n
0010 0000	0011	XXXX	0 XX	0011	1 0 1	1

Assume register file 3 holds 000003_{16} and MQ register holds $36D84E_{16}$.

		Most Significant Package	Next Most Significant Package	Least Significant Package	
Operand:					
	Source	0011 0110	1101 1000	0100 1110	MQ shifter ← MQ register
	Destination	0110 1101	1011 0000	1001 1100	MQ register ← MQ shift result
Count:					
	Source	0000 0000	0000 0000	0000 0011	S ← RF(3)
	Destination	0000 0000	0000 0000	0000 0100	RF(3) ← S + C_n

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 package is retained if the ALU calculation does not produce an overflow. If an overflow condition occurs, the sign bit is inverted. Bit 0 is passed through $\overline{SIO0}$-$\overline{SIO7}$ to bit 7 of the next-most-significant package. Bit 0 of the least-significant package is dropped.

The shift may be made conditional on SSF. If SSF is high or floating, the shift result will be sent to Y. If SSF is low, F will be passed unaltered.

* A list of ALU operations that can be used with this instruction is given on page 2-9.

Shift Operations

ALU Shifter	MQ Shifter
Arithmetic Right	None

Available Destination Operands
ALU Shifter:

RF (C3-C0)	RF (B3-B0)	Y-Port
•		•

Control/Data Signals

Signal	User Programmable	Use
SSF	Yes	Passes shifted output if high or floating; passes ALU result if low.
$\overline{SIO0}$	No	Link cascaded ALU shifters. Output value of LSP's $\overline{SIO0}$ is LSB of ALU (inverted).
$\overline{SIO7}$	No	
$\overline{QIO0}$	No	Inactive
$\overline{QIO7}$	No	Inactive
C_n	No	Affects arithmetic operation specified in bits I3-I0 of instruction field.

Status Signals†

If arithmetic instruction specified in I3-I0:

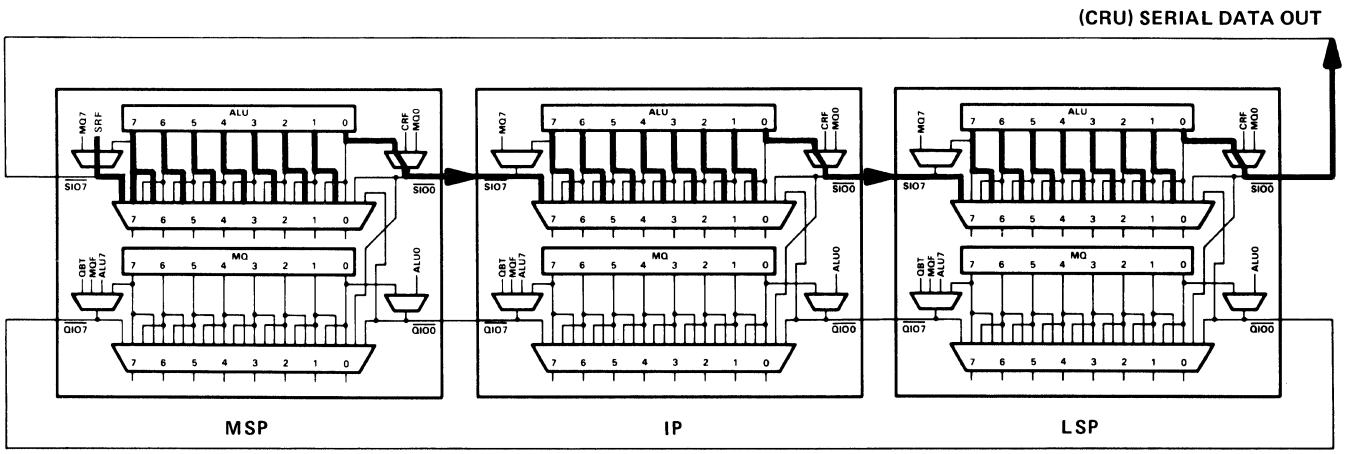
ZERO = 1 if result = 0
N = 1 if MSB of result = 1
= 0 if MSB of result = 0
OVR = 1 if signed arithmetic overflow
C_{n+8} = 1 if carry-out condition

If logic instruction specified in I3-I0:

ZERO = 1 if result = 0
N = 1 if MSB of result = 1
= 0 if MSB of result = 0
OVR = 0
C_{n+8} = 0

† C_{n+8} 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.

DATA FLOW



EXAMPLE (assumes a 24-bit cascaded system)

Perform the computation A = (A + B)/2, where A and B are single-precision two's complement numbers. Let A be residing in register 1 and B be input via the DB bus.

Instruction Code I7-I0	Operand Address A3-A0	Operand Address B3-B0	Operand Select $\overline{EA}$ EB1-EB0	Destination Address C3-C0	Destination Select $\overline{WE}$ SELY $\overline{OEY}$	Carry-in C_n
0000 0001	0001	XXXX	0 10	0001	0 0 1	0

Assume register file 1 holds $08C618_{16}$ and DB bus holds 007530_{16}.

	Most Significant Package	Next Most Significant Package	Least Significant Package	
Source	0000 1000	1100 0110	0001 1000	R ← RF(1)
Source	0000 0000	0111 0101	0011 0000	S ← DB bus
Intermediate Result	0000 1001	0011 1011	0100 1000	ALU Shifter ← R + S + C_n
Destination	0000 0100	1001 1101	1010 0100	RF(1) ← ALU shift result

FUNCTION

Performs arithmetic right 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 ALU are shifted one bit to the right. The sign bit of the most-significant package is retained if the ALU operation does not produce an overflow. If an overflow condition occurs, the sign bit is inverted. Bit 0 is passed through $\overline{SIO0}$-$\overline{SIO7}$ to bit 7 of the next-most-significant package. Bit 0 of the least-significant package is passed through $\overline{QIO0}$-$\overline{QIO7}$ to bit 7 of the most-significant package of the MQ register. Bit 0 of the MQ register's most-significant package is passed through $\overline{QIO0}$-$\overline{QIO7}$ to bit 7 of the next-most-significant package. Bit 0 of the MQ register's least significant package is dropped.

The shift may be made conditional on SSF. If SSF is high or floating, the shift result will be sent to Y and the MQ register. If SSF is low, F will be passed unaltered, and the MQ register will not be changed.

* A list of ALU operations that can be used with this instruction is given on page 2-9.

Shift Operations

ALU Shifter	MQ Shifter
Arithmetic Right	Arithmetic Right

Available Destination Operands
ALU Shifter:

RF (C3-C0)	RF (B3-B0)	Y-Port
•		•

Control/Data Signals

Signal	User Programmable	Use
SSF	Yes	Passes shift result if high or floating; passes ALU result and retains MQ register if low.
$\overline{SIO0}$	No	Link cascaded ALU shifters. Output value of LSP's $\overline{SIO0}$ is LSB of MQ shifter (inverted).
$\overline{SIO7}$	No	
$\overline{QIO0}$	No	Link cascaded MQ shifters. Output value of LSP's $\overline{QIO0}$ is LSB of ALU shifter (inverted).
$\overline{QIO7}$	No	
C_n	No	Affects arithmetic operation specified in bits I3-I0 of instruction field.

Status Signals†

If arithmetic instruction specified in I3-I0:

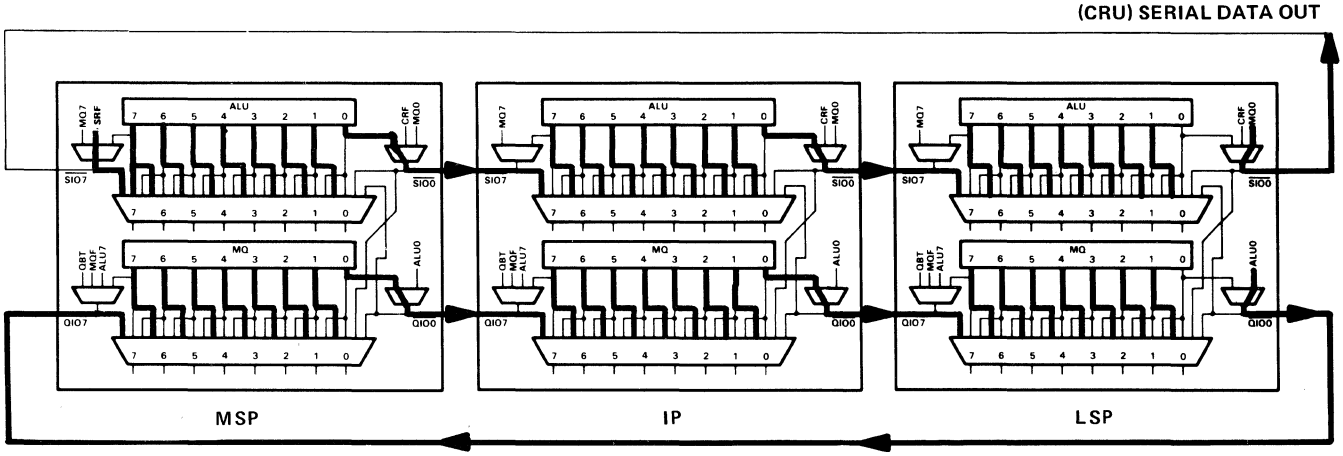
ZERO = 1 if result = 0
N = 1 if MSB of result = 1
= 0 if MSB of result = 0
OVR = 1 if signed arithmetic overflow
C_{n+8} = 1 if carry-out condition

If logic instruction specified in I3-I0:

ZERO = 1 if result = 0
N = 1 if MSB of result = 1
= 0 if MSB of result = 0
OVR = 0
C_{n+8} = 0

† C_{n+8} 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.

DATA FLOW



EXAMPLE (assumes a 24-bit cascaded system)

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.

Instruction Code I7-I0	Operand Address A3-A0	Operand Address B3-B0	Operand Select $\overline{EA}$ EB1-EB0	Destination Address C3-C0	Destination Select $\overline{WE}$	SELY	$\overline{OEY}$	Carry-in C_n
0001 0001	0001	XXXX	0 10	0001	0	0	1	0

Assume register file 1 holds $08C618_{16}$, DB bus holds 007530_{16}, and MQ register holds $299A0F_{16}$.

		Most Significant Package	Next Most Significant Package	Least Significant Package	
MSH:					
	Source	0000 1000	1100 0110	0001 1000	R ← RF(1)
	Source	0000 0000	0111 0101	0011 0000	S ← DB bus
	Intermediate Result	0000 1001	0011 1011	0100 1000	ALU Shifter ← R+S+C_n
	Destination	0000 0100	1001 1101	1010 0100	RF(1) ← ALU shift result
LSH:					
	Source	0010 1001	1001 1010	0000 1111	MQ shifter ← MQ register
	Destination	0001 0100	1100 1101	0000 0111	MQ register ← MQ shift result

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 most-significant package is passed through $\overline{SIO0}$-$\overline{SIO7}$ to bit 7 of the next-most-significant package. Bit 0 of the least-significant package is passed through $\overline{SIO0}$-$\overline{SIO7}$ to bit 7 of the most-significant package.

The shift may be made conditional on SSF. If SSF is high or floating, the shift result will be sent to Y. If SSF is low, F will be passed unaltered.

* A list of ALU operations that can be used with this instruction is given on page 2-9.

Shift Operations

ALU Shifter	MQ Shifter
Circular Right	None

Available Destination Operands
ALU Shifter:

RF (C3-C0)	RF (B3-B0)	Y-Port
•		•

Control/Data Signals

Signal	User Programmable	Use
SSF	Yes	Passes shift result if high or floating; passes ALU result if low.
$\overline{SIO0}$	No	Link cascaded ALU shifters. Output value of LSP's $\overline{SIO0}$ is LSB of ALU (inverted).
$\overline{SIO7}$	No	
$\overline{QIO0}$	No	Inactive
$\overline{QIO7}$	No	Inactive
C_n	No	Affects arithmetic operation specified in bits I3-I0 of instruction field.

Status Signals†

If arithmetic instruction specified in I3-I0:

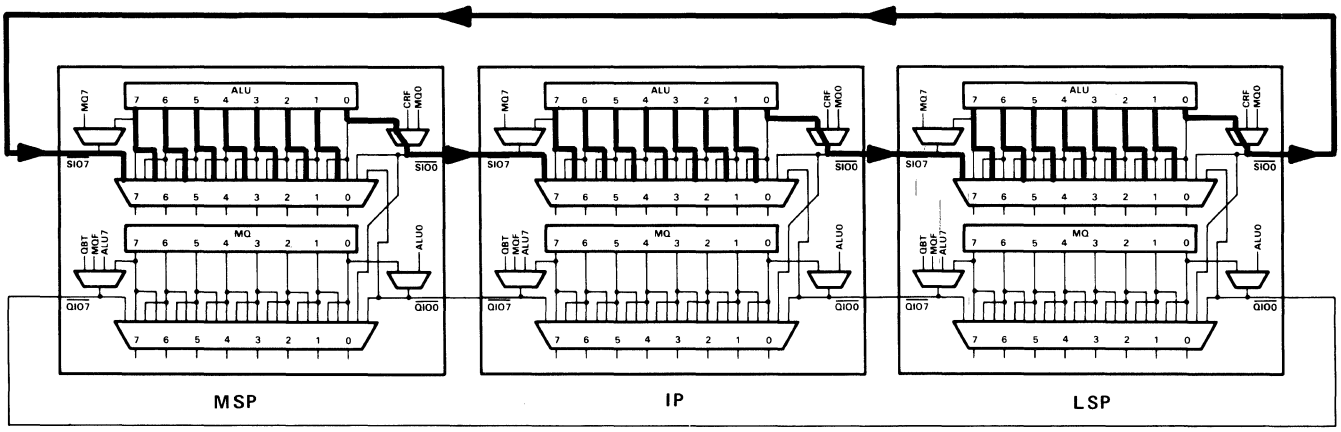
ZERO = 1 if result = 0
N = 1 if MSB of result = 1
= 0 if MSB of result = 0
OVR = 1 if signed arithmetic overflow
C_{n+8} = 1 if carry-out condition

If logic instruction specified in I3-I0:

ZERO = 1 if result = 0
N = 1 if MSB of result = 1
= 0 if MSB of result = 0
OVR = 0
C_{n+8} = 0

† C_{n+8} 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.

DATA FLOW



EXAMPLE (assumes a 24-bit cascaded system)

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

Instruction Code I7-I0	Operand Address A3-A0	Operand Address B3-B0	Operand Select $\overline{EA}$ EB1-EB0	Destination Address C3-C0	Destination Select $\overline{WE}$ SELY $\overline{OEY}$	Carry-in C_n
1000 0110	0110	XXXX	0 XX	0001	0 0 1	0

Assume register file 6 holds $88C618_{16}$.

	Most Significant Package	Next Most Significant Package	Least Significant Package	
Source	1000 1000	1100 0110	0001 1000	R ← RF(6)
Intermediate Result	1000 1000	1100 0110	0001 1000	ALU Shifter ← R + C_n
Destination	0100 0100	0110 0011	0000 1100	RF(1) ← ALU shift result

FUNCTION

Performs circular right 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 ALU and MQ shifters are rotated one bit to the right. Bit 0 of the most-significant package's ALU shifter is passed through $\overline{SIO0}$-$\overline{SIO7}$ to bit 7 of the next-most-significant package. Bit 0 of the least-significant package is passed through $\overline{QIO0}$-$\overline{QIO7}$ to bit 7 of the most-significant package of the MQ register. Bit 0 of the least-significant package is passed through $\overline{SIO0}$-$\overline{SIO7}$ to bit 7 of the most-significant package's ALU.

The shift may be made conditional on SSF. If SSF is high or floating, the shift result will be sent to Y and the MQ register. If SSF is low, F will be passed unaltered, and the MQ register will not be changed.

* A list of ALU operations that can be used with this instruction is given on page 2-9.

Shift Operations

ALU Shifter	MQ Shifter
Circular Right	Circular Right

Available Destination Operands
ALU Shifter:

RF (C3-C0)	RF (B3-B0)	Y-Port
•		•

Control/Data Signals

Signal	User Programmable	Use
SSF	Yes	Passes shift result if high or floating; passes ALU result and retains MQ register if low.
$\overline{SIO0}$ $\overline{SIO7}$	No No	Link cascaded ALU shifters. Output value of LSP's $\overline{SIO0}$ is LSB of MQ shifter (inverted).
$\overline{QIO0}$ $\overline{QIO7}$	No No	Link cascaded MQ shifters. Output value of LSP's $\overline{QIO0}$ is LSB of ALU shifter (inverted).
C_n	No	Affects arithmetic operation specified in bits I3-I0 of instruction field.

Status Signals†

If arithmetic instruction specified in I3-I0:

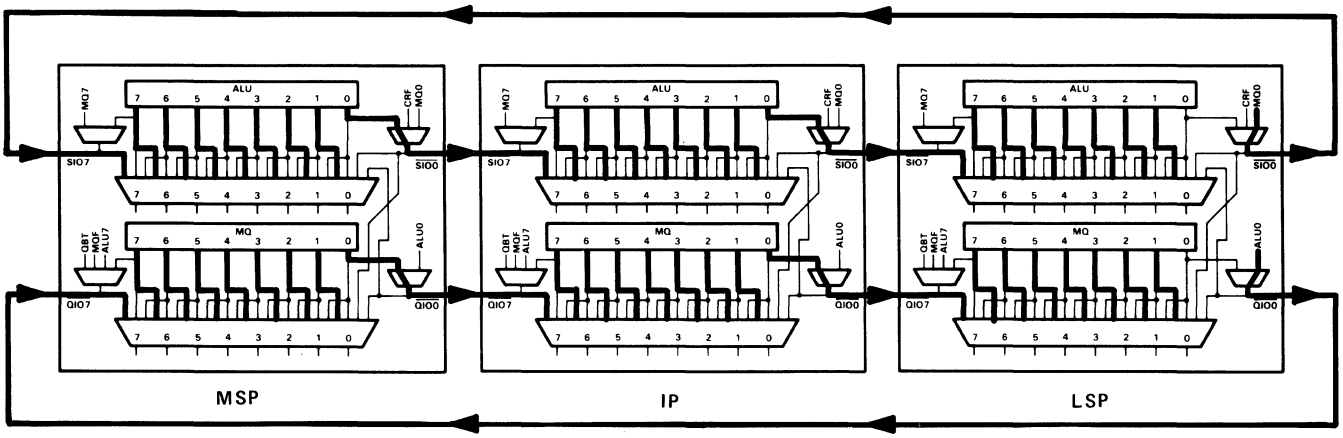
ZERO = 1 if result = 0
N = 1 if MSB of result = 1
= 0 if MSB of result = 0
OVR = 1 if signed arithmetic overflow
C_{n+8} = 1 if carry-out condition

If logic instruction specified in I3-I0:

ZERO = 1 if result = 0
N = 1 if MSB of result = 1
= 0 if MSB of result = 0
OVR = 0
C_{n+8} = 0

† C_{n+8} 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.

DATA FLOW



EXAMPLE (assumes a 24-bit cascaded system)

Perform a circular right double precision shift of the data in register 6 (MSH) and MQ (LSH), and store the result back in register 5 and the MQ register.

Instruction Code I7-I0	Operand Address A3-A0	Operand Address B3-B0	Operand Select $\overline{EA}$ EB1-EB0	Destination Address C3-C0	Destination Select $\overline{WE}$ SELY $\overline{OEY}$	Carry-in C_n
1001 0110	0110	XXXX	0 XX	0110	0 0 1	0

Assume register file holds $88C618_{16}$ and MQ register holds $A99AOF_{16}$.

		Most Significant Package	Next Most Significant Package	Least Significant Package	
MSH:					
	Source	0000 1000	1100 0110	0001 1000	RF ← RF(6)
	Intermediate Result	0000 1000	1100 0110	0001 1000	ALU Shifter ← R + C_n
	Destination	1000 0100	0110 0011	0000 1100	RF(6) ← ALU shift result
LSH:					
	Source	1010 1001	1001 1010	0000 1111	MQ shifter ← MQ shifter
	Destination	0101 0100	1100 1101	0000 0111	MQ shifter ← MQ shift result

FUNCTION

Performs logical right shift on result of ALU operation specified in lower nibble of instruction field.

DESCRIPTION

The result of the ALU operation specified in instruction bits I3-I0 is shifted one bit to the right. A zero is placed in bit 7 of the most-significant package unless $\overline{SIO7}$ is programmed low; this will force the sign bit to one. Bit 0 of the most-significant package is passed through $\overline{SIO0}$-$\overline{SIO7}$ to bit 7 of the next-most-significant package. Bit 0 of the least-significant package is dropped.

The shift may be made conditional on SSF. If SSF is high or floating, the shift result will be sent to Y. If SSF is low, F will be passed unaltered.

* A list of ALU operations that can be used with this instruction is given on page 2-9.

Shift Operations

ALU Shifter	MQ Shifter
Logical Right	None

Available Destination Operands
ALU Shifter:

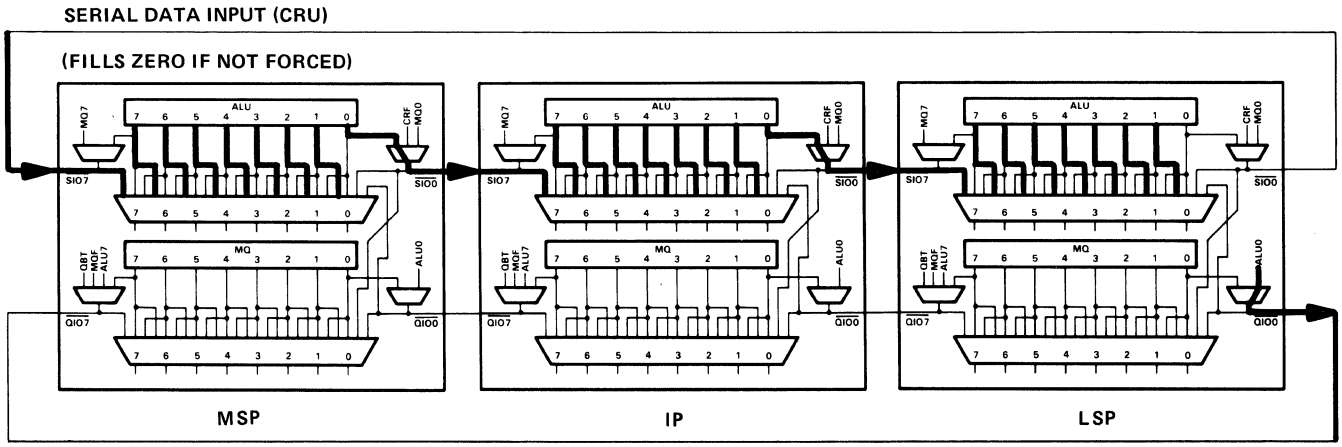
RF (C3-C0)	RF (B3-B0)	Y-Port
•		•

Control/Data Signals

Signal	User Programmable	Use
SSF	Yes	Passes shift result if high or floating; passes ALU result if low.
$\overline{SIO0}$	No	Link cascaded ALU shifters. $\overline{SIO7}$ fills a zero in MSB of ALU shifter if high or floating; sets MSB to one if low.
$\overline{SIO7}$	Yes	
$\overline{QIO0}$	No	Link cascaded MQ shifters. Output of LSP's $\overline{QIO0}$ is LSB of ALU shifter (inverted).
$\overline{QIO7}$	No	
C_n	No	Inactive

* C_{n+8} 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.

DATA FLOW



EXAMPLE (assumes a 24-bit cascaded system)

Perform a logical right single precision shift on data on the DA bus, and store the result in register 1.

Instruction Code I7-I0	Operand Address A3-A0	Operand Address B3-B0	Operand Select $\overline{EA}$ EB1-EB0	END Fill $\overline{SIO7}$	Destination Address C3-C0	Destination Select $\overline{WE}$	SELY	$\overline{OEY}$	Carry-in C_n
0010 0110	XXXX	XXXX	1 XX	1	0001	0	0	1	0

Assume DA bus holds $A8C615_{16}$.

	Most Significant Package	Next Most Significant Package	Least Significant Package	
Source	1010 1000	1100 0110	0001 0101	R ← DA
Intermediate Result	1010 1000	1100 0110	0001 0101	ALU Shifter ← R + C_n
Destination	0101 0100	0110 0011	0000 1010	RF(1) ← ALU shift result

FUNCTION

Performs logical right 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 ALU are shifted one bit to the right. A zero is placed in the sign bit of the most-significant package unless $\overline{SIO7}$ is programmed low; this will force the sign bit to one. Bit 0 of the most-significant package's ALU is passed through $\overline{SIO0}$-$\overline{SIO7}$ to bit 7 of the next-most-significant package. Bit 0 of the least-significant package is passed through $\overline{QIO0}$-$\overline{QIO7}$ to bit 7 of the most-significant package of the MQ register. Bit 0 of the MQ register's most-significant package is passed through $\overline{QIO0}$-$\overline{QIO7}$ of the next-most-significant package. Bit 0 of the least-significant package is dropped.

The shift may be made conditional on SSF. If SSF is high or floating, the shift result will be sent to Y and the MQ register. If SSF is low, F will be passed unaltered, and the MQ register will not be changed.

* A list of ALU operations that can be used with this instruction is given on page 2-9.

Shift Operations

ALU Shifter	MQ Shifter
Logical Right	Logical Right

Available Destination Operands
ALU Shifter:

RF (C3-C0)	RF (B3-B0)	Y-Port
•		•

Control/Data Signals

Signal	User Programmable	Use
SSF	Yes	Passes shift result if high or floating; passes ALU result and retains MQ register if low.
$\overline{SIO0}$	No	Link cascaded ALU shifters. $\overline{SIO7}$ fills a zero in MSB or ALU shifter if high or floating; sets MSB to one if low.
$\overline{SIO7}$	Yes	
$\overline{QIO0}$	No	Link cascaded MQ shifters. Output value of LSP's $\overline{QIO0}$ is LSB of ALU shifter (inverted).
$\overline{QIO7}$	No	
C_n	No	Affects arithmetic operation specified in bits I3-I0 of instruction field.

Status Signals†

If arithmetic instruction specified in I3-I0:

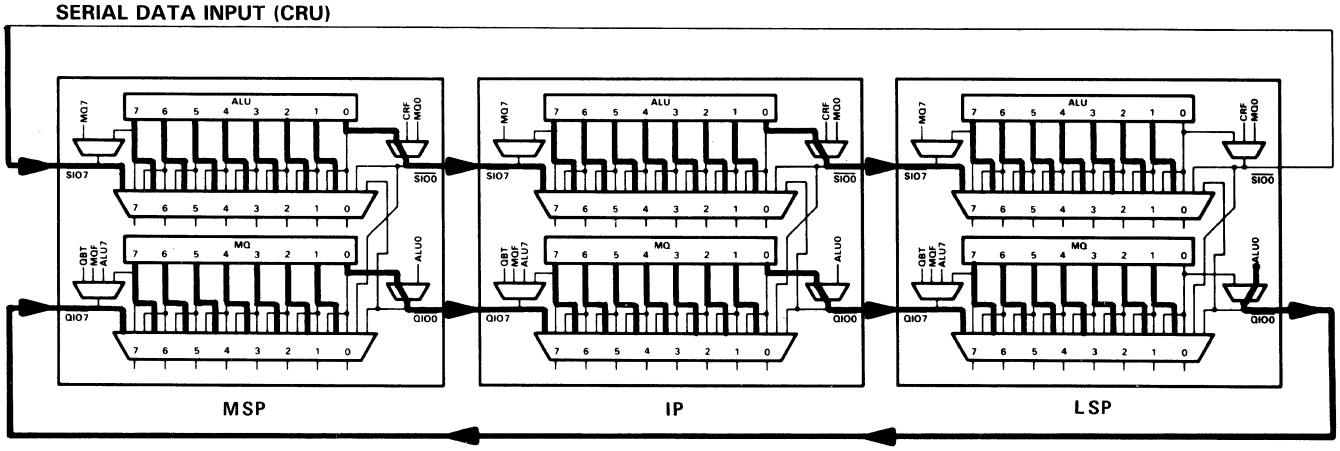
ZERO = 1 if result = 0
N = 1 if MSB of result = 1
= 0 if MSB of result = 0
OVR = 1 if signed arithmetic overflow
C_{n+8} = 1 if carry-out condition

If logic instruction specified in I3-I0:

ZERO = 1 if result = 0
N = 1 if MSB of result = 1
= 0 if MSB of result = 0
OVR = 0
C_{n+8} = 0

† C_{n+8} 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.

DATA FLOW



EXAMPLE (assumes a 24-bit cascaded system)

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.

Instruction Code I7-I0	Operand Address A3-A0	Operand Address B3-B0	Operand Select $\overline{EA}$ EB1-EB0	END Fill $\overline{SIO7}$	Destination Address C3-C0	Destination Select $\overline{WE}$	SELY	$\overline{OEY}$	Carry-in C_n
0011 0110	XXXX	0001	X 00	0	0001	0	0	1	0

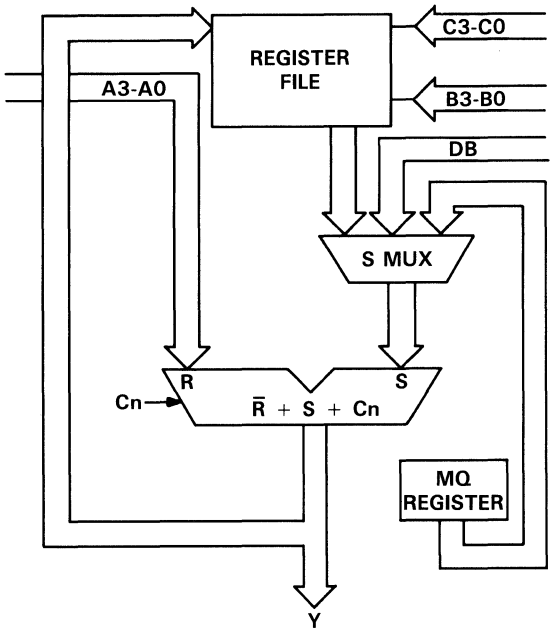
Assume register file 1 holds $A8C615_{16}$ and MQ register holds $A99AOE_{16}$.

		Most Significant Package	Next Most Significant Package	Least Significant Package	
MSH:					
	Source	1010 1000	1100 0110	0001 0101	S ← RF(1)
	Intermediate Result	1010 1000	1100 0110	0001 0101	ALU Shifter ← S + C_n
	Destination	1101 0100	0110 0011	0000 1010	RF(1) ← ALU shift result
LSH:					
	Source	1010 1001	1001 1010	0000 1110	MQ shifter ← MQ register
	Destination	1101 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.

DATA FLOW



Available R Bus Source Operands

RF (A3-A0)	A3-A0 Immediate	DA Port	A3-A0 : : C3-C0 Mask
	•		

Available S Bus Source Operands

RF (B3-B0)	DB Port	MQ Register
•	•	•

Available Destination Operands

RF (C3-C0)	RF (B3-B0)	Y Port
•		•

Shift Operations

ALU	MQ
None	None

Control/Data Signals

Signal	User Programmable	Use
$\overline{SSF}$	No	Inactive
$\overline{SIO0}$	No	Inactive
$\overline{SIO7}$	No	Inactive
$\overline{QIO0}$	No	Inactive
$\overline{QIO7}$	No	Inactive
C_n	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_{n+8}	=	1 if carry-out = 1

DESCRIPTION

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

EXAMPLE (assumes a 24-bit cascaded system)

Subtract the value 12 from data on the DB bus, and store the result in register file 1.

Instruction Code I7-I0	Operand Address A3-A0	Operand Address B3-B0	Operand Select $\overline{EA}$ EB1-EB0	Destination Address C3-C0	Destination Select $\overline{WE}$ SELY $\overline{OEY}$	Carry-in C_n
0111 1000	1100	XXXX	X 10	0001	0 0 1	1

Assume bits A3-A0 hold C_{16} and DB bus holds 000100_{16}.

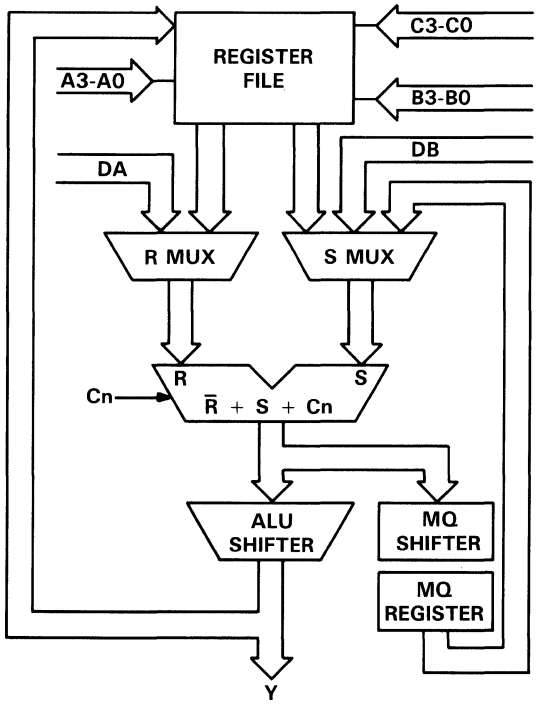
	Most Significant Package	Next Most Significant Package	Least Significant Package	
Source	0000 0000	0000 0000	0000 1100	R ← 12
Source	0000 0000	0000 0001	0000 0000	S ← DB bus
Destination	0000 0000	0000 0000	1111 0100	RF(1) ← $\overline{R}+S+C_n$

SUBR — Subtract R with Carry ($\overline{R}$ + S + C_n) — * 2

FUNCTION

Subtracts data on the R bus from S with carry.

DATA FLOW



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 on page 2-9.

Available R Bus Source Operands

RF (A3-A0)	A3-A0 Immediate	DA Port	A3-A0 : : C3-C0 Mask
•		•	

Available S Bus Source Operands

RF (B3-B0)	DB Port	MQ Register
•	•	•

Available Destination Operands

RF (C3-C0)	RF (B3-B0)	Y Port
•		•

Shift Operations

ALU	MQ
•	•

Control/Data Signals

Signal	User Programmable	Use
$\overline{SSF}$	No	Affect shift instructions programmed in bits I7-I4 of instruction field.
$\overline{SIO0}$	No	
$\overline{SIO7}$	No	
$\overline{QIO0}$	No	
$\overline{QIO7}$	No	
C_n	Yes	Two's complement subtraction if programmed high.

Status Signals†

ZERO	=	1 if result = 0
N	=	1 if MSB = 1
OVR	=	1 if signed arithmetic overflow
C_{n+8}	=	1 if carry-out

† C_{n+8} 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 24-bit cascaded system)

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

Instruction Code I7-I0	Operand Address A3-A0	Operand Address B3-B0	Operand Select $\overline{EA}$ EB1-EB0	Destination Address C3-C0	Destination Select $\overline{WE}$ SELY $\overline{OEY}$	Carry-in C_n
1110 0010	0001	XXXX	0 10	XXXX	0 X X	1

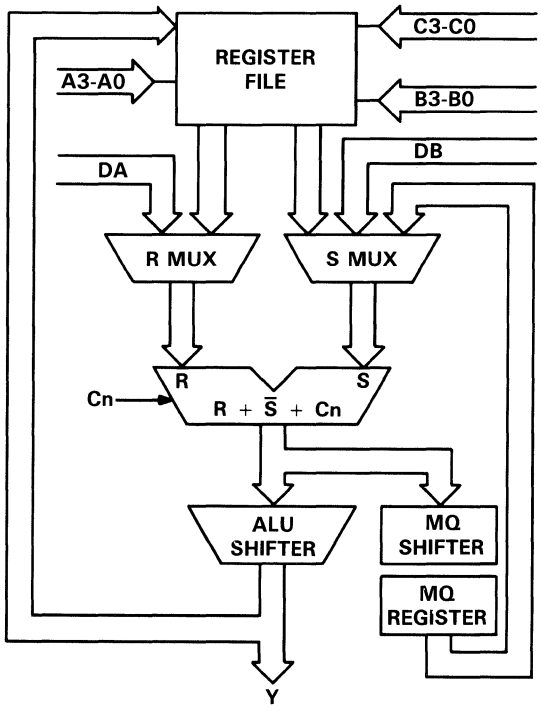
Assume register file 1 holds $0084D0_{16}$ and DB bus holds $00C350_{16}$.

	Most Significant Package	Next Most Significant Package	Least Significant Package	
Source	0000 0000	1000 0100	1101 0000	R ← RF(1)
Source	0000 0000	1100 0011	0101 0000	S ← DB bus
Destination	0000 0000	0011 1110	1000 0000	MQ ← $\overline{R}$ + S + C_n

FUNCTION

Subtracts data on the S bus from R with carry.

DATA FLOW



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 on page 2-9.

Available R Bus Source Operands

RF (A3-A0)	A3-A0 Immediate	DA Port	A3-A0 : : C3-C0 Mask
•		•	

Available S Bus Source Operands

RF (B3-B0)	DB Port	MQ Register
•	•	•

Available Destination Operands

RF (C3-C0)	RF (B3-B0)	Y Port
•		•

Shift Operations

ALU	MQ
•	•

Control/Data Signals

Signal	User Programmable	Use
$\overline{SSF}$	No	Affect shift instructions programmed in bits I7-I4 of instruction field.
$\overline{SIO0}$	No	
$\overline{SIO7}$	No	
$\overline{QIO0}$	No	
$\overline{QIO7}$	No	
C_n	Yes	Two's complement subtraction if programmed high.

Status Signals†

ZERO	=	1 if result = 0
N	=	1 if MSB = 1
OVR	=	1 if signed arithmetic overflow
C_{n+8}	=	1 if carry-out

† C_{n+8} 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 24-bit cascaded system)

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

Instruction Code I7-I0	Operand Address A3-A0	Operand Address B3-B0	Operand Select $\overline{EA}$ EB1-EB0	Destination Address C3-C0	Destination Select $\overline{WE}$ SELY $\overline{OEY}$	Carry-in C_n
1110 0011	0001	XXXX	0 10	XXXX	1 X X	1

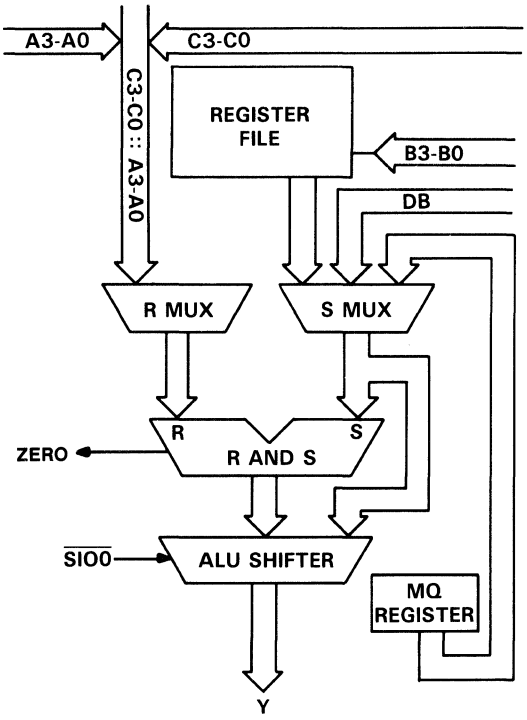
Assume register file 1 holds $0084D0_{16}$ and DB bus holds $00C350_{16}$.

	Most Significant Package	Next Most Significant Package	Least Significant Package	
Source	0000 0000	1000 0100	1101 0000	$R \leftarrow RF(1)$
Source	0000 0000	1100 0011	0101 0000	$S \leftarrow$ DB bus
Destination	1111 1111	1100 0001	1000 0000	$MQ \leftarrow R + \overline{S} + C_n$

FUNCTION

Tests bits in selected bytes of S-bus data for zeros using mask in C3-C0::A3-A0.

DATA FLOW



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 $\overline{SIO0}$ low. Test results are indicated on the ZERO output, which goes to one if the test passes and $\overline{SIO0}$ is low. If $\overline{SIO0}$ is high, a zero will be output on the ZERO pin. The write enable pin ($\overline{WE}$) is internally disabled during this instruction.

Available R Bus Source Operands

RF (A3-A0)	A3-A0 Immediate	DA Port	A3-A0 : : C3-C0 Mask
			•

Available S Bus Source Operands

RF (B3-B0)	DB Port	MQ Register
•	•	•

Control/Data Signals

Signal	User Programmable	Use
$\overline{SSF}$	No	Inactive
$\overline{SIO0}$	Yes	Byte-Select
$\overline{SIO7}$	No	Inactive
$\overline{QIO0}$	No	Inactive
$\overline{QIO7}$	No	Inactive
C_n	No	Inactive

Status Signals

ZERO = 1 if result (selected bytes) = Pass
N = 0
OVR = 0
C_{n+8} = 0

EXAMPLE (assumes a 24-bit cascaded system)

Test bits 7, 6 and 5 of bytes 0 and 2 of data in register 3 for zeros.

Instruction Code I7-I0	Mask (LSH) A3-A0	Operand and Destination Address B3-B0	Mask (MSH) C3-C0	Operand Select $\overline{EA}$ EB1-EB0	Byte Select $\overline{SIO0}$	Destination Select $\overline{WE}$ SELY $\overline{OEY}$	Carry-in C_n
0011 1000	0000	0011	1110	X 00	010	X X X	X

Assume register file 3 holds $1CD003_{16}$.

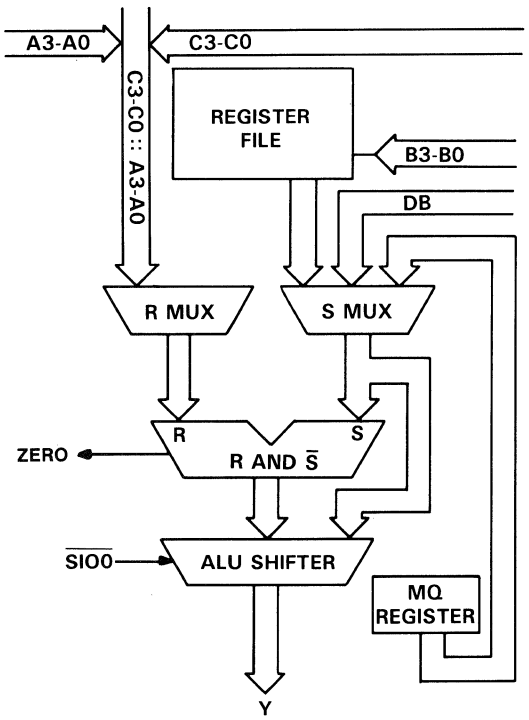
	Most Significant Package Byte 2 (selected)	Next Most Significant Package Byte 1 (not selected)	Least Significant Package Byte 0 (selected)	
Mask	1110 0000	1110 0000	1110 0000	$R_n \leftarrow$ Mask (C3-C0::A3-A0)
Source	0001 1100	1101 0000	0000 0011	$S_n \leftarrow RF(3)_n$†
			Output 1	ZERO ← 1

† n = nth package

FUNCTION

Tests bits in selected bytes of S-bus data for ones using mask in C3-C0::A3-A0.

DATA FLOW



Available R Bus Source Operands

RF (A3-A0)	A3-A0 Immediate	DA Port	A3-A0 : : C3-C0 Mask
			•

Available S Bus Source Operands

RF (B3-B0)	DB Port	MQ Register
•	•	•

Control/Data Signals

Signal	User Programmable	Use
$\overline{SSF}$	No	Inactive
$\overline{SIO0}$	Yes	Byte-Select
$\overline{SIO7}$	No	Inactive
$\overline{QIO0}$	No	Inactive
$\overline{QIO7}$	No	Inactive
C_n	No	Inactive

Status Signals

ZERO = 1 if result (selected bytes) = Pass
N = 0
OVR = 0
C_{n+8} = 0

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 $\overline{SIO0}$ low. Test results are indicated on the ZERO output, which goes to one if the test passes and $\overline{SIO0}$ is low. If $\overline{SIO0}$ is high, a zero will be output on the ZERO pin. The write enable pin ($\overline{WE}$) is internally disabled during this instruction.

EXAMPLE (assumes a 24-bit cascaded system)

Test bits 7, 6 and 5 of bytes 1 and 2 of register 3 for ones.

Instruction Code I7-I0	Mask (LSH) A3-A0	Operand and Destination Address B3-B0	Mask (MSH) C3-C0	Operand Select $\overline{EA}$ EB1-EB0	Byte Select $\overline{SIO0}$	Destination Select $\overline{WE}$ SELY $\overline{OEY}$	Carry-in C_n
0010 1000	0000	0011	1110	X 00	0001	X X X	X

Assume register file 3 holds $1CF003_{16}$.

	Most Significant Package Byte 2 (selected)	Next Most Significant Package Byte 1 (selected)	Least Significant Package Byte 0 (not selected)	
Mask	1110 0000	1110 0000	1110 0000	$R_n \leftarrow$ Mask (C3-C0::A3-A0)
Source	0001 1100	1101 0000	0000 0011	$S_n \leftarrow RF(3)_n$†
			Output 0	ZERO ← 0

† n = nth package

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. A description of nonrestoring unsigned division and an algorithm using this instruction are given in section 2.3.1.

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.

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 tests SSF (used to signal pass/fail of previous test) and then evaluates

$F \leftarrow R + S$ if SSF = 0
$F \leftarrow \overline{R} + S + C_n$ if SSF = 1.

A double precision left shift is performed; bit 7 of the most-significant package of the MQ shifter is transferred through $\overline{SIO7}$-$\overline{SIO0}$ to bit 1 of the least-significant package of the ALU shifter. Bit 7 of the most-significant package of the ALU shifter is lost. The unfixed quotient bit is circulated into the least-significant bit of MQ through $\overline{QIO7}$-$\overline{QIO0}$.

Available R Bus Source Operands

RF (A3-A0)	A3-A0 Immediate	DA Port	A3-A0 : : C3-C0 Mask
•		•	

Recommended S Bus Source Operands

RF (B3-B0)	DB Port	MQ Register
•	•	

Recommended Destination Operands

RF (C3-C0)	RF (B3-B0)	Y Port
•		

Shift Operations

ALU	MQ
Left	Left

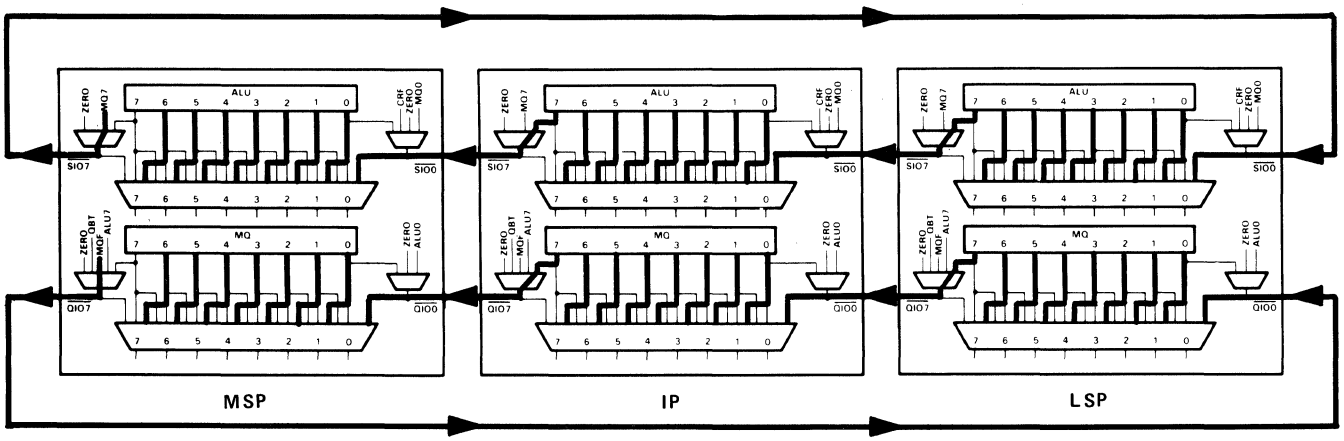
Control/Data Signals

Signal	User Programmable	Use
SSF	No	Preserves result of test subtraction for next instruction.
$\overline{SIO0}$	No	Link cascaded ALU shifters. Output value of MSP's $\overline{SIO7}$ is MSB of MQ shifter (inverted).
$\overline{SIO7}$	No	
$\overline{QIO0}$	No	Link cascaded MQ shifters. Output value of MSP's $\overline{QIO7}$ is unfixed quotient bit (MQF).
$\overline{QIO7}$	No	
C_n	Yes	Should be programmed high.

Status Signals

ZERO	=	1 if intermediate result = 0
N	=	0
OVR	=	0
C_{n+8}	=	1 if carry-out

DATA FLOW



FUNCTION

Computes the first quotient bit of nonrestoring unsigned division. A description of nonrestoring unsigned division and an algorithm using this instruction are given in section 2.3.1.

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; SSF is used to signal the subsequent UDIVI instruction to 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 dividend. 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 dividend is in the MQ register.

UDIVIS computes

$F \leftarrow \overline{R} + S + C_n$.

A double precision left shift is performed; bit 7 of the most-significant package of the MQ shifter is transferred through $\overline{SIO7}$-$\overline{SIO0}$ to bit 0 of the least-significant package of the ALU shifter. Bit 7 of the most-significant package of the ALU shifter is lost. The unfixed quotient bit is circulated into the least-significant bit of MQ through $\overline{QIO7}$-$\overline{QIO0}$.

Available R Bus Source Operands

RF (A3-A0)	A3-A0 Immediate	DA Port	A3-A0 : : C3-C0 Mask
•		•	

Recommended S Bus Source Operands

RF (B3-B0)	DB Port	MQ Register
•	•	

Recommended Destination Operands

RF (C3-C0)	RF (B3-B0)	Y Port
•		

Shift Operations

ALU	MQ
Left	Left

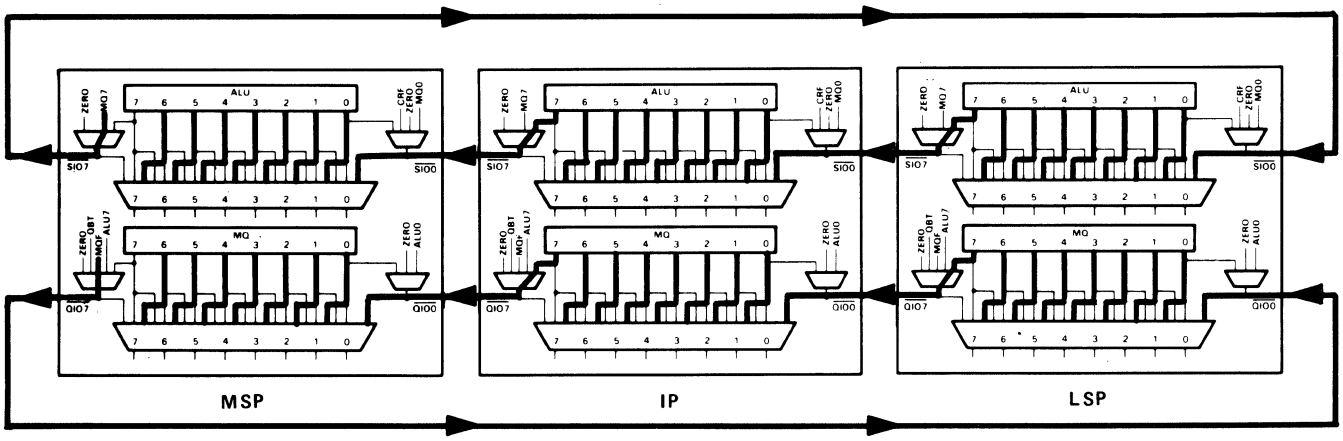
Control/Data Signals

Signal	User Programmable	Use
SSF	No	Preserves result of test subtraction for next instruction.
$\overline{SIO0}$	No	Link cascaded ALU shifters. Output value of MSP's $\overline{SIO7}$ is MSB of MQ shifter (inverted).
$\overline{SIO7}$	No	
$\overline{QIO0}$	No	Link cascaded MQ shifters. Output value of MSP's $\overline{QIO7}$ is unfixed quotient bit (MQF).
$\overline{QIO7}$	No	
C_n	Yes	Should be programmed high.

Status Signals

ZERO	=	1 if intermediate result = 0
N	=	0
OVR	=	1 if divide overflow
C_{n+8}	=	1 if carry-out

DATA FLOW



FUNCTION

Solves the final quotient bit during nonrestoring unsigned division. A description of nonrestoring unsigned division and an algorithm using this instruction are given in section 2.3.1.

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 tests SSF (used to signal pass/fail of previous test) and evaluates

$Y \leftarrow R + S$ if SSF = 0
$Y \leftarrow \overline{R} + S + C_n$ if SSF = 1.

The contents of the MQ register are shifted one bit to the left; the unfixed quotient bit is circulated into the least-significant bit through $\overline{QIO7}$-$\overline{QIO0}$.

SSF is used to indicate to all slices whether the remainder must be corrected in the subsequent instruction.

Available R Bus Source Operands

RF (A3-A0)	A3-A0 Immediate	DA Port	A3-A0 : : C3-C0 Mask
•		•	

Recommended S Bus Source Operands

RF (B3-B0)	DB Port	MQ Register
•	•	

Recommended Destination Operands

RF (C3-C0)	RF (B3-B0)	Y Port
•		

Shift Operations

ALU	MQ
None	Left

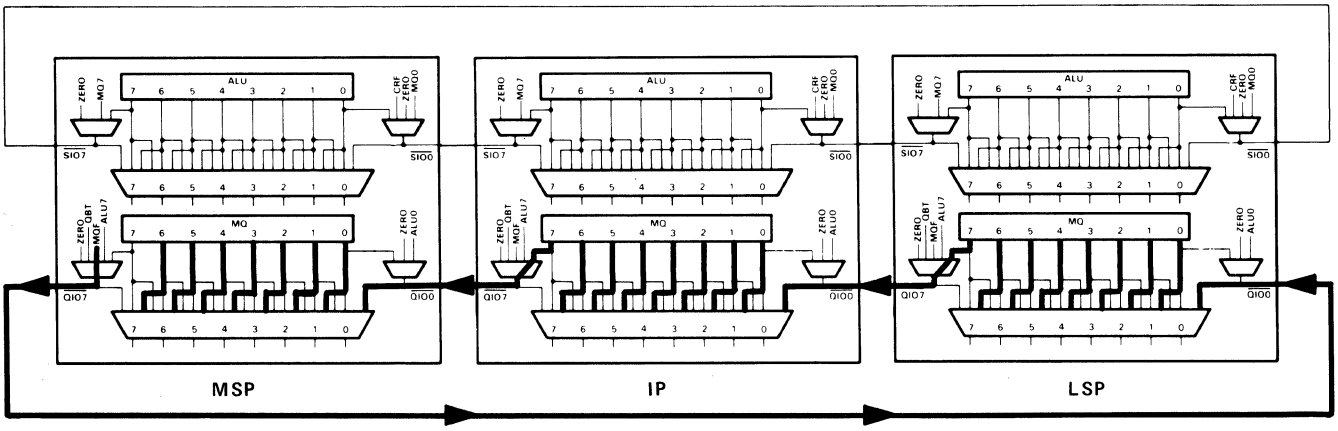
Control/Data Signals

Signal	User Programmable	Use
SSF	No	Carries result of remainder correction test.
$\overline{SIO0}$	No	Inactive
$\overline{SIO7}$	No	Inactive
$\overline{QIO0}$	No	Link cascaded MQ shifters. Output value of MSP's $\overline{QIO7}$ is unfixed quotient bit (MQF).
$\overline{QIO7}$	No	
C_n	Yes	Should be programmed high.

Status Signals

ZERO = 1 if intermediate result = 0
N = 0
OVR = 0
C_{n+8} = 1 if carry-out

DATA FLOW



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 section 2.3.1.

DESCRIPTION

UMULI tests SSF to determine whether the multiplicand should be added with the present partial product. The instruction evaluates

$F \leftarrow R + S + C_n$ if SSF = 1
$F \leftarrow S + C_n$ if SSF = 0.

A double precision right shift is performed. Bit 0 of the least-significant package of the ALU shifter is passed through $\overline{QIO0}$-$\overline{QIO7}$ to bit 7 of the most-significant package 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 (A3-A0)	A3-A0 Immediate	DA Port	A3-A0 : : C3-C0 Mask
•		•	

Recommended S Bus Source Operands

RF (B3-B0)	DB Port	MQ Register
•	•	

Recommended Destination Operands

RF (C3-C0)	RF (B3-B0)	Y Port
•		

Shift Operations

ALU	MQ
Right	Right

Control/Data Signals

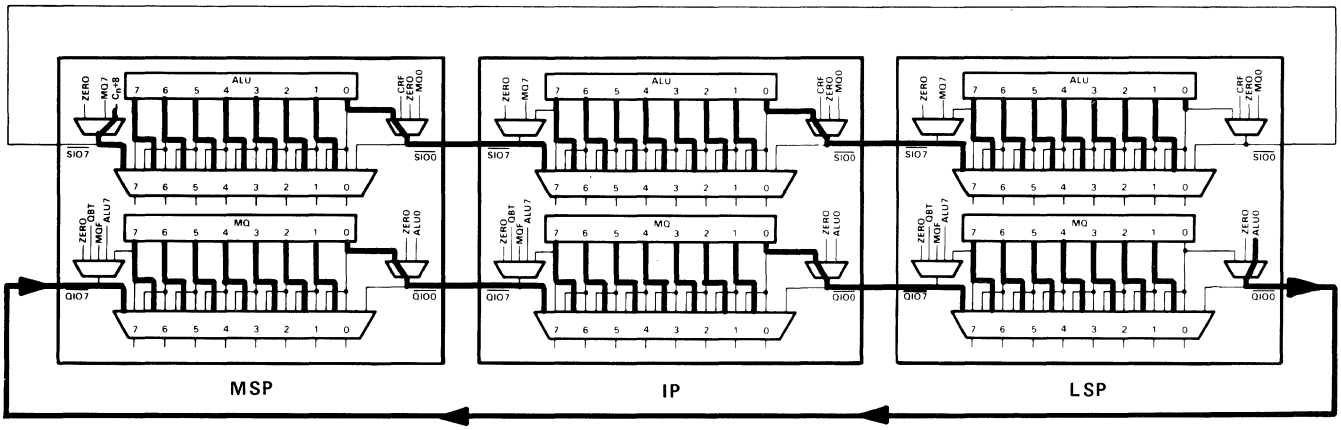
Signal	User Programmable	Use
SSF	No	Holds LSB of MQ.
$\overline{SIO0}$	No	Link cascaded ALU shifters. Input value of MSP's $\overline{SIO7}$ is carry-out.
$\overline{SIO7}$	No	
$\overline{QIO0}$	No	Link cascaded MQ shifters. Output value of MSP's $\overline{QIO7}$ is LSB of ALU shifter (inverted).
$\overline{QIO7}$	No	
C_n	Yes	Should be programmed low.

Status Signals†

ZERO	=	1 if result = 0
N	=	1 if MSB = 1
OVR	=	0
C_{n+8}	=	1 if carry-out

† Valid only on final execution of multiply iteration

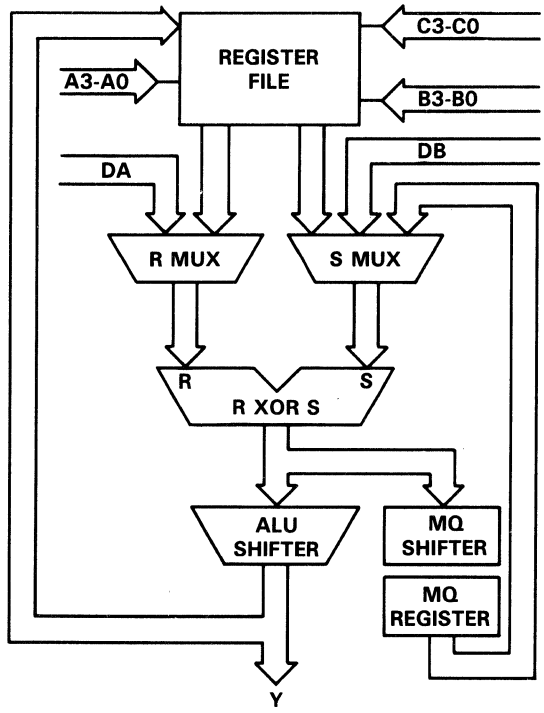
DATA FLOW



FUNCTION

Evaluates the logical expression R XOR S.

DATA FLOW



Available R Bus Source Operands

RF (A3-A0)	A3-A0 Immediate	DA Port	A3-A0 : : C3-C0 Mask
•		•	

Available S Bus Source Operands

RF (B3-B0)	DB Port	MQ Register
•	•	•

Available Destination Operands

RF (C3-C0)	RF (B3-B0)	Y Port
•		•

Shift Operations

ALU	MQ
•	•

Control/Data Signals

Signal	User Programmable	Use
$\overline{SSF}$	No	Affect shift instructions programmed in bits I7-I4 of instruction field.
$\overline{SIO0}$	No	
$\overline{SIO7}$	No	
$\overline{QIO0}$	No	
$\overline{QIO7}$	No	
C_n	No	Inactive

Status Signals†

ZERO = 1 if result = 0
N = 1 if MSB = 1
OVR = 0
C_{n+8} = 0

† C_{n+8} 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.

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 on page 2-9.

EXAMPLE (assumes a 24-bit cascaded system)

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

Instruction Code I7-I0	Operand Address A3-A0	Operand Address B3-B0	Operand Select $\overline{EA}$ EB1-EB0	Destination Address C3-C0	Destination Select $\overline{WE}$ SELY $\overline{OEY}$	Carry-in C_n
1111 1001	0011	0101	0 00	0101	0 0 1	X

Assume register file 3 holds $F6D840_{16}$ and register file 5 holds $F6D842_{16}$.

	Most Significant Package	Next Most Significant Package	Least Significant Package	
Source	1111 0110	1101 1000	0100 0000	R ← RF(3)
Source	1111 0110	1101 1000	0100 0010	S ← RF(5)
Destination	0000 0000	0000 0000	0000 0010	RF(5) ← R XOR S

3 'AS890 Microsequencer

3.1 Overview

The 'AS890 is a high performance microsequencer which is used in fast, low power microprogrammed processors. The bipolar device is fabricated in low voltage Schottky Transistor Logic (STL) with TTL compatible inputs and outputs.

The 14-bit device addresses 16,384 micromemory locations. Short routines which perform complex operations can be realized, especially when the 'AS888 is used as the ALU. The net result for the user is reduced hardware complexity and fewer and shorter execution cycles. Parallel independent control of onboard circuitry allows the user to merge basic operations, such as doubly nested loops, n-way branches, conditional branches and subroutine calls and returns to create complex single instructions such as Decrement and Branch on Non-Zero, Decrement and Return on Non-Zero, Decrement and Branch to A on Non-Zero Else Branch to B, or Exit Loop on Condition Code or at End of Loop.

Figure 3-1 illustrates the architecture of a typical microprogrammmed processor: the micromemory, or control store, in which the user's microprogram resides; the instruction register, which synchronizes the instructions with the system clock; the microsequencer, which computes the next address based on the instruction and the state of the system; the ALU, which processes data based on the microinstruction; and the status register, which samples the status at each instruction cycle. At the beginning of an instruction cycle, the state of the system is as follows:

1. The microinstruction register contains the instruction currently being executed.
2. The ALU has just executed an instruction and has the current status ready at its output pins.
3. The status register contains the status results of the previous instruction.
4. The next microaddress is being generated while the current instruction is being executed.

The system shown allows many arithmetic, logical, conversion and mask/test instructions to be implemented. The 'AS890 reduces overhead in processing loops, iterations, flag tests, subroutines and interrupts. The three-port device lends itself to a number of hardware configurations to support a wide range of applications. The Y-port drives the microaddress bus and can be disabled for operations such as loading interrupt vectors. The DRA and DRB ports can be used to load or save branch addresses or loop counts from the microprogram and load, save or read branch addresses or loop counts to and from other user hardware.

Best hardware configuration is arrived at by test coding the most critical operations of the application in order to determine which of the Figure 3-1 paths are required. For example, a floating-point CPU may need to compute loop counts in the ALU (see section 5) while a real-time digital filter may be concerned with fixed loops and interrupt processing. The former will require a path between hardware and DRA or DRB, while the latter may only require a path between the instruction register and DRA or DRB.

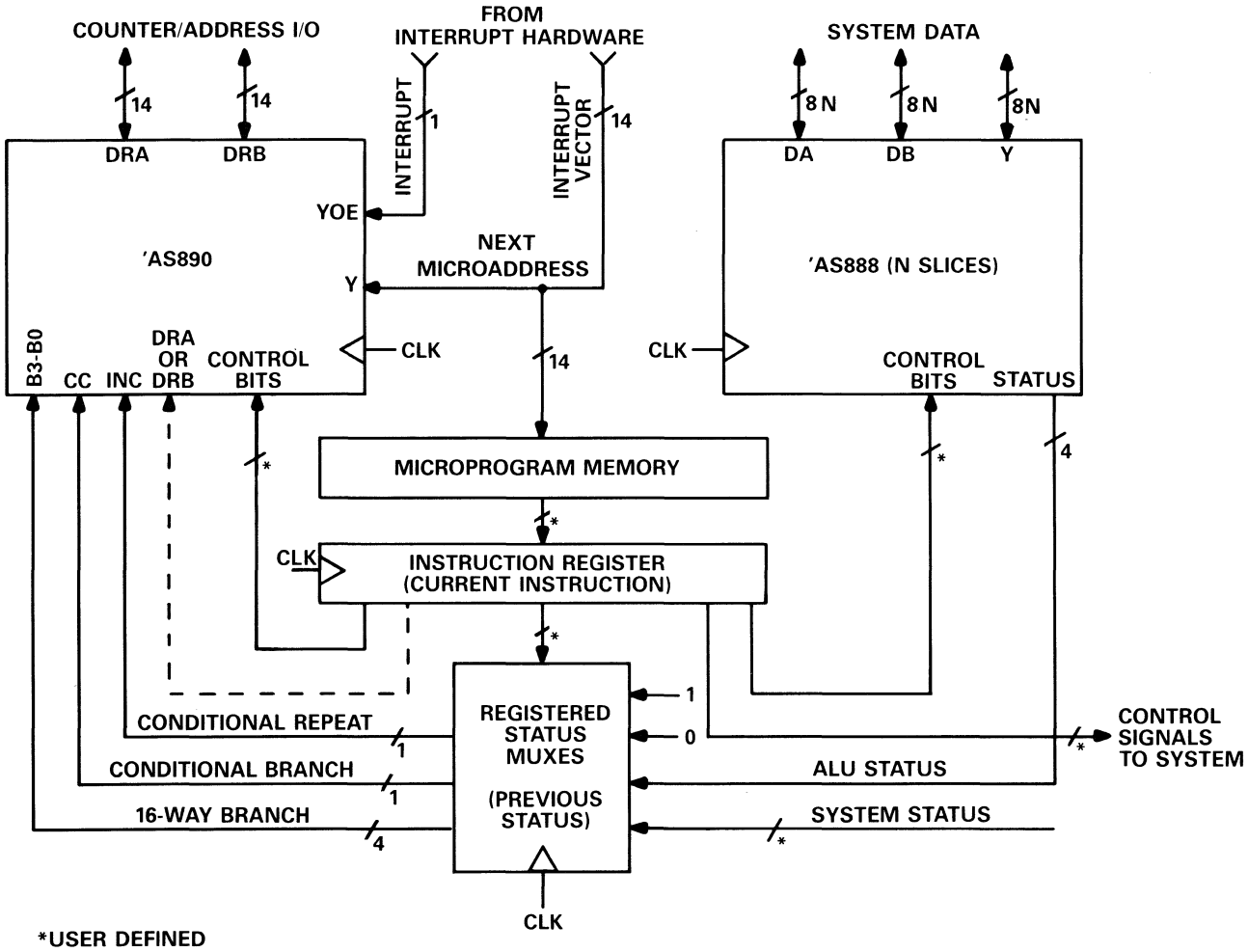


Figure 3-1. Typical Microprogrammed Processor

3.2 Architecture

The 'AS890 block diagram is given in Figure 3-2. The chip is made up of the following components:

1) A 14-bit microprogram counter (MPC) consisting of a register and incrementer which generates the next sequential address
2) Two register/counters (RCA and RCB) for counting loops and iterations, storing branch addresses or driving external devices
3) A 9 × 14 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.

'AS890 control pins are summarized in Table 3-1. Those signals which typically originate from the instruction register are Y output multiplexer controls, MUX2-MUX0, which 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; 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.

Table 3-1. Response to Control Inputs

PIN NAME	LOGIC LEVEL	
	HIGH	LOW
$\overline{\text{RAOE}}$	DRA output in high-Z state	DRA output active
$\overline{\text{RBOE}}$	DRB output in high-Z state	DRB output active
$\overline{\text{YOE}}$	Y output in high-Z state	Y output active
$\overline{\text{INT}}$	MPC to stack	INT RT register to stack
OSEL	Stack to DRA buffer input	RCA to DRA buffer input
INC	Y output plus one to MPC	Y output to MPC
MUX2-MUX0	See Table 3.2	See Table 3.2
S2-S0	See Table 3.3	See Table 3.3
RC2-RC0	See Table 3.4	See Table 3.4

Control and data signals which commonly orginate 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 microsequencer not increment the address if INC is off, allowing wait states and repeat until flag instructions to be implemented. If INC originates from status, repeat until flag instructions are possible.

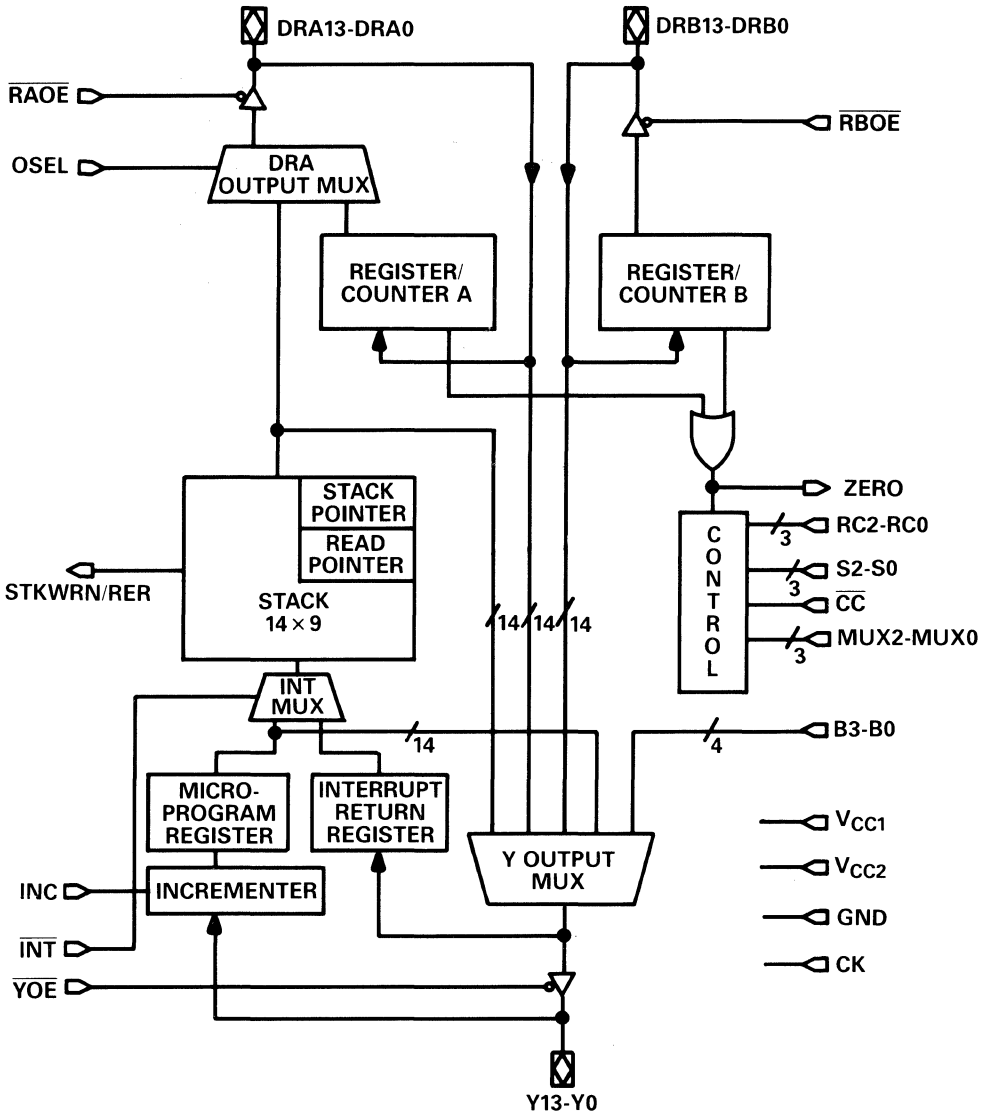


Figure 3-2. Functional Block Diagram for 'AS890

$\overline{CC}$ 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, $\overline{CC}$ will normally be generated by the output of a status multiplexer. In this case, whether $\overline{CC}$ is to be forced high, forced low or taken from ALU status will be determined by a status MUX select field in the microinstruction.

Control signals which generally originate from hardware are B3-B0, which can be used as a 4-bit variable to support 16- and 32-way branches; and $\overline{YOE}$, which allows interrupt hardware to place an address on the microaddress bus.

Status from the 'AS890 is provided by ZERO, 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 eighth of nine locations is written. In the latter case, STKWRN/RER remains high until the stack pointer is decremented to seven.

3.2.1 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 MUX controls to program the 'AS890 is discussed in section 3.3.

3.2.2 Microprogram Counter

The Y bus generates the next 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.

3.2.3 Register/Counters

Addresses or loop counts may be loaded directly into register/counters RCA and RCB through the direct data ports DRA13-DRA0 and DRB13-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.

3.2.4 Stack

The positive-edge-triggered 14-bit address stack allows up to nine levels of nested calls or interrupts and can be used to support branching and looping. Six 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 onto the stack and increments the stack pointer
5) read, which makes the address pointed to by the read pointer available at the DRA port
6) hold, which causes the address of the stack and read pointers to remain unchanged.

3.2.4.1 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 'AS890 accomplishes both events within a single cycle.

3.2.4.2 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.

3.2.4.3 Stack Warning/Read Error Pin

The STKWRN/RER pin alerts the system to a potential stack overflow or underflow condition. STKWRN/RER becomes active under two conditions. If seven of the nine stack locations (0-8) are full (the stack pointer is at 7) and a push occurs, the STKWRN/RER pin will produce a high-level signal to warn that the stack is approaching its capacity and will be full after one more push. Knowledge that overflow potential exists allows bit-slice-based systems to continuously process real-time interrupt vectors. This signal will remain high if hold, push or pop instructions occur, until the stack pointer is decremented to seven. Should a push instruction occur when the stack is full, the new address will be written over the address in stack location 8.

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.

3.2.5. 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 simply replaces the pending address.

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

3.3 Microprogramming the 'AS890

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 the ALU is user defined, which makes the microinstruction width even more application dependent. Types of instructions resulting from manipulation of the sequencer's basic controls are discussed below, followed by examples of some commonly used instructions in section 3.4.

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'	DRA13-DRA4::B3-B0
DRB	Bidirectional data port (can be loaded externally or from RCB)
DRB'	DRB13-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

3.3.1 Address Selection

Y-output multiplexer controls, MUX2-MUX0, select one of eight three-source branches as shown in Table 3-2. The state 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 one of the register/counters will decrement to zero.

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 Non-Zero 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 of 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, 32-way branches.

Table 3-2. Y Output Control

MUX CONTROL MUX2-0	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

†DRA13 – DRA4::B3-B0
‡DRB13 – DRB4::B3-B0

3.3.2 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 3-3). 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.

Combining stack and MUX controls with status 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 Non-Zero; Call 16-Way.

Diagnostic stack dumps are possible using Read (S2-S0 = HHH), while asserting OSEL.

Table 3-3. Stack Control

STACK CONTROL 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

3.3.3 Register Controls

Unlike stack and MUX control, register control is not dependent upon $\overline{CC}$ and ZERO. Registers can be independently loaded, decremented or held using register control inputs R2-R0 (see Table 3-4). All combinations are supported with the exception of simultaneous register decrements. The register control inputs can be used to store branch address and loop counts and to decrement loop counts to facilitate the complex branching instructions described above.

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.

Table 3-4. Register Control

REGISTER CONTROL RC2-RC0	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

3.3.4. Continue/Repeat Instructions

The most commonly used instruction is a continue or microprogram counter advance, implemented by selecting MPC at the Y output MUX and forcing INC high. If MPC is selected and INC is off, the 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; these are summarized in Table 3-5.

Table 3-5. Continue/Repeat Encodings

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

3.3.5 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 3-2). 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 require 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 would precipitate 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 3-2). 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 Non-Zero 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 3-6.

3.3.6 Conditional Branch Instructions

Perhaps the most useful of all branches is the conditional branch. The 'AS890 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 ($\overline{CC}$) and four-bit status (using B3-B0) are allowed. Testing single bit status is preferred when the status can be set up and selected through a status MUX prior the conditional branch. Four-bit status allows the 'AS890 to process instructions based on Boolean status expressions, such as Branch if Overflow and Not Carry or 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. Table 3-7 lists conditional branching codes.

Table 3-6. Branch Encodings

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

Table 3-6. Branch Encodings (continued)

MUX2-MUX0	S2-S0	OSEL	INSTRUCTION $\overline{CC}$ = L
LHL	HLL	X	BR S
LHH	HLL	X	BR S
LLL	HHH	L	BR S
LLL	LHL	X	BR S
LLL	HLL	X	BR S
LLH	HHH	L	BR S
LLH	LHL	X	BR S
LLH	HLL	X	BR S
LHL	LLH	X	BR S: CLR SP/RP
LHH	LLH	X	BR S: CLR SP/RP
LLL	LLH	X	BR S: CLR SP/RP
LLH	LLH	X	BR S: CLR SP/RP
LHL	HHH	H	BR S: READ
LHH	HHH	H	BR S: READ
LLL	HHH	H	BR S: READ
LLH	HHH	H	BR S: READ

3.3.7 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 Non-Zero, 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 3-2). 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 Non-Zero else Return, Decrement and Call on Non-Zero; and Decrement and Branch 16-Way on Non-Zero. Possible variations are summarized in Table 3-8.

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 Non-Zero.

Table 3-7. Conditional Branch Encodings

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

Table 3-8. Decrement and Branch on Non-Zero Encodings

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

3.3.8 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 Non-Zero 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; Decrement and Return on Non-Zero Else Branch. Codes that cause subroutine calls are summarized in Table 3-9 (with decrement) and Table 3-10 (without decrement).

Table 3-9. Call Encodings without Register Decrements

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

3.3.9 Subroutine Returns

A return from subroutine can be implemented by coding a branch to stack with a pop. Since pop is also conditional on $\overline{CC}$ and ZERO, the complex forms discussed in section 3.3.7 also apply to return instructions: Decrement and Return on Non-Zero; Return on Condition Code; Branch on Condition Code Else Return. Return encodings are summarized in Table 3-12 (without decrements) and Table 3-13 (with decrements).

3.3.10 Reset

Pulling the S2-S0 pins low clears the stack and read pointers, and zeros the Y output multiplexer (See Table 3-3).

Table 3-10. Call Encodings with Register Decrements

MUX2-MUX0	S2-S0	OSEL	INSTRUCTION		
			$\overline{CC}$ = L		$\overline{CC}$ = H
			ZERO = L	ZERO = H	
LLL	HLH	X	CALL S	CONT/RPT	BR A
LLH	HLH	X	CALL S	CONT/RPT	BR B
LLL	HHL	X	CALL S	CONT/RPT	CALL A
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
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
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)

Table 3-11. Reset Encoding

MUX2-MUX0	S2-S0	OSEL	INSTRUCTION		
			$\overline{CC}$ = L		$\overline{CC}$ = H
			ZERO = L	ZERO = H	
XXX	LLL	X	RESET	RESET	RESET

Table 3-12. Return Encodings without Register Decrements

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

Table 3-13. Return Encodings with Register Decrements

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

3.3.11 Clear Pointers

The stack and read pointers may be cleared without affecting the Y output multiplexer by setting S2-S0 to LLH and forcing $\overline{CC}$ low (See Table 3-3).

3.3.12 Read Stack

Placing a high value on all of the stack inputs (S2-S0) and OSEL places the 'AS890 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 will appear 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.

3.3.13 Interrupts

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

1. Place the bidirectional Y bus into a high-impedance state by forcing $\overline{YOE}$ high.
2. Force the interrupt entry point vector onto the Y bus. INC should be high.

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 forcing $\overline{INT}$ low and coding a push.

3.4 Examples

Representative examples of instructions using the 'AS890 are given on the following pages. The examples assume the system shown in Figure 3-1, in which the address and contents of the next instruction are being fetched while the current instruction is being executed, and the ALU status register contains the status results of the previous instruction.

3.4.1 Required Set-Up

Since the incrementer looks two addresses ahead of the address in the instruction register to set up some instructions such as continue or repeat (see section 3.2.2), 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 (see Figure 3-1).

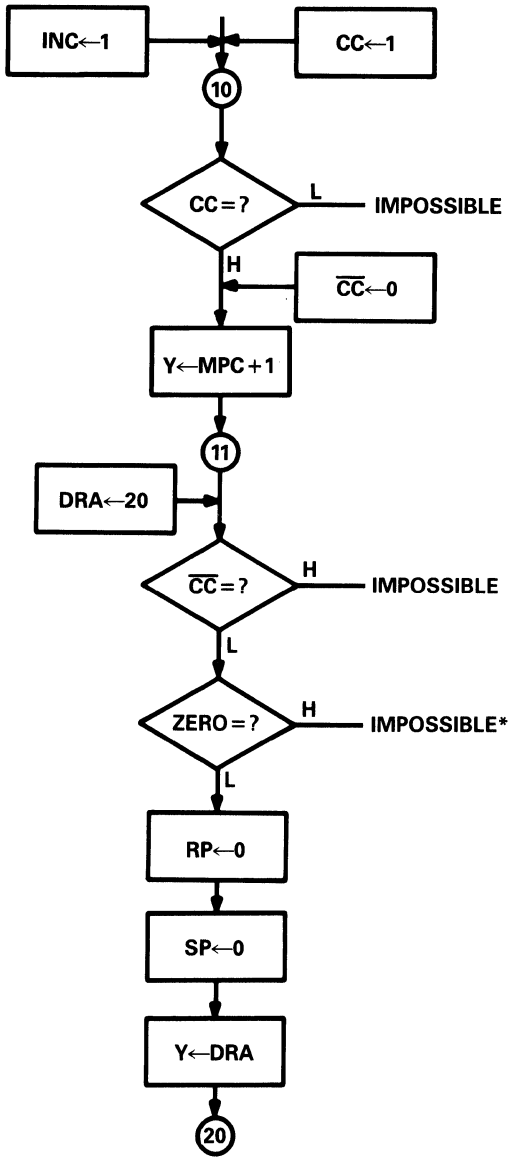
CLEAR POINTERS

To Continue (instruction 10), this example uses the first instruction in Table 3-5 with CONT/RPT in the instruction column. INC must be high; $\overline{CC}$ must be programmed high in the previous instruction (See Section 3.4.1).

To Clear the Stack and Read Pointers and Branch to address 20 (instruction 11), this example uses the first BR A: Clear SP, RP instruction in Table 3-7. $\overline{CC}$ 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

Clear Pointers



*no register decrement

Flow diagrams and suggested code for representative Continue instructions are given below. Numbers inside the circles are microword address locations. For a discussion of sequencing instructions, see Section 3.3.4.

CONTINUE

To Continue (instruction 10), this example uses the first instruction in Table 3-5 with CONT/RPT in the instruction column. INC and $\overline{CC}$ must be programmed high one cycle ahead of instruction 10 (See Section 3.4.1).

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	110	111	XXX	0	X	X	XXXX	XXXX

CONTINUE AND POP

To Continue and decrement the stack pointer (Pop), this example uses the first instruction in Table 3-5 with CONT/RPT: POP in the instruction column. INC and $\overline{CC}$ are forced high in the previous instruction (See Section 3.4.1).

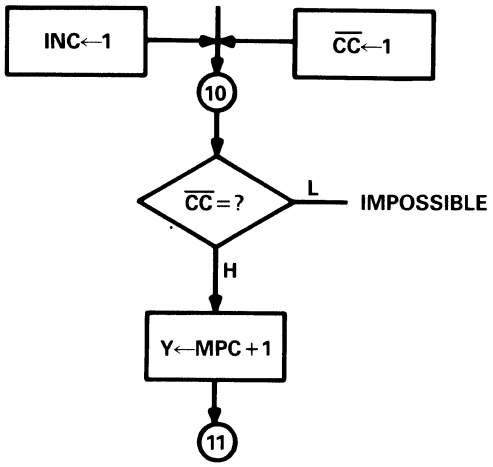
Address	Instruction	MUX2-MUX0	S2-S0	R2-R0	OSEL	$\overline{CC}$	INC	DRA	DRB
(Set-up)		XXX	XXX	XXX	X	1	1	XXXX	XXXX
10	Continue/Pop	110	010	XXX	X	X	X	XXXX	XXXX

CONTINUE AND PUSH

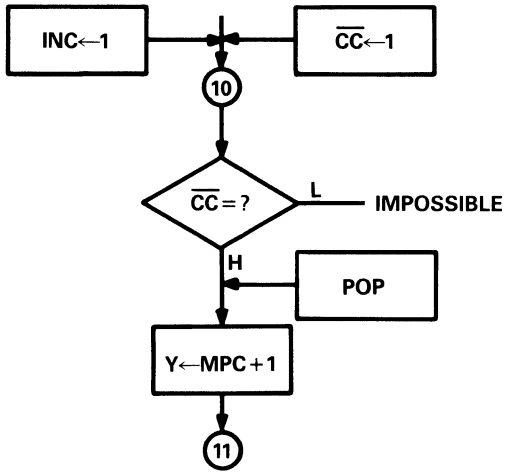
To Continue and push the microprogram counter onto the stack (Push), this example uses the first instruction in Table 3-5 with CONT/RPT: PUSH in the instruction column. INC and $\overline{CC}$ are forced high one cycle ahead of instruction 10 (See Section 3.4.1).

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	110	100	XXX	0	X	X	XXXX	XXXX

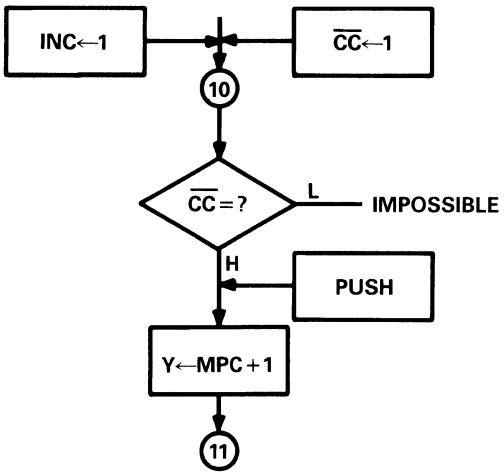
Continue



Continue and Pop



Continue and Push



Flow diagrams and suggested code for representative Branch instructions are shown below. Numbers inside the circles are microaddresses. Coding of branch instructions is discussed in Section 3.3.5.

BRANCH EXAMPLE 1

To Branch from address 10 to address 20, this example uses the first BR A instruction from the $\overline{CC}$ = H column of Table 3-6. $\overline{CC}$ must be programmed high one cycle ahead of instruction 10 (See Section 3.4.1).

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 the first BR A instruction from the $\overline{CC}$ = L column of Table 3-6. $\overline{CC}$ 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.

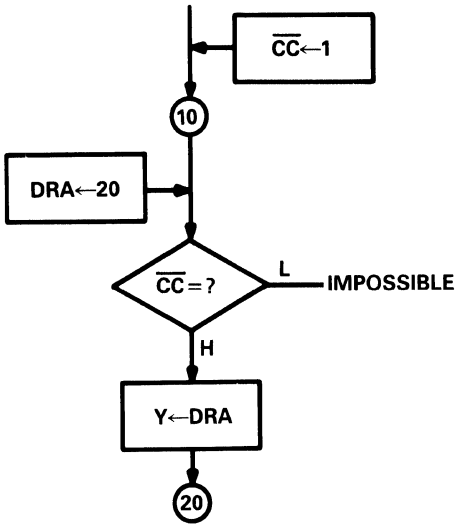
Address	Instruction	MUX2-MUX0	S2-S0	R2-R0	OSEL	$\overline{CC}$	INC	DRA	DRB
(Set-up)		XXX	XXX	XXX	X	0	X	XXXX	XXXX
10	BR A	110	111	000	0	X	X	0020	XXXX

16-WAY BRANCH

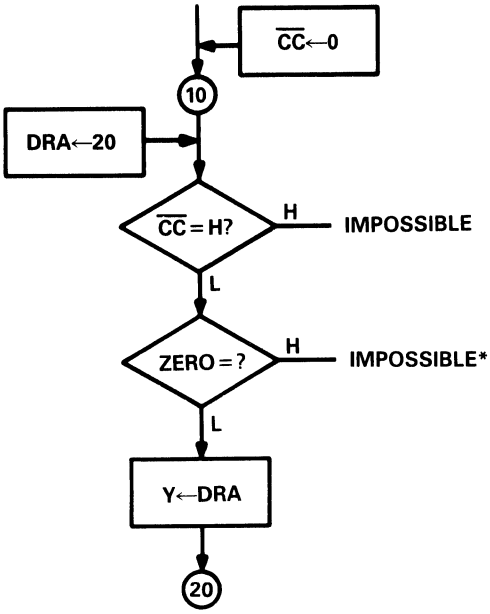
To Branch 16-Way, this example uses the first BR B' instruction in Table 3-6. $\overline{CC}$ is programmed high in the previous instruction. The branch address is derived from the concatenation DRB13-DRB4::B3-B0.

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

Branch Example 1

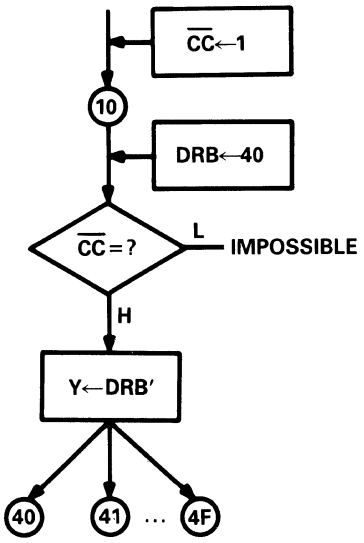


Branch Example 2



*no register decrement

16-Way Branch



Flow diagrams and suggested code for representative Conditional Branch instructions are shown below. Numbers inside the circles are microaddresses. Further information concerning conditional branches can be found in Section 3.3.6.

CONDITIONAL BRANCH

To Branch to address 20 Else Continue to address 11, this example uses the first instruction from Table 3-7 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	1	1	XXXX	XXXX
10	BR A else Continue	110	111	000	0	X	X	0020	XXXX

THREE-WAY BRANCH

To Continue (instruction 10), this example uses the first instruction in Table 3-5 with CONT/RPT in the instruction column. INC and $\overline{CC}$ must be programmed high in the previous instruction. Register A is loaded using Table 3-4.

To Branch 3-Way, this example uses the first instruction from Table 3-8 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 3-4 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

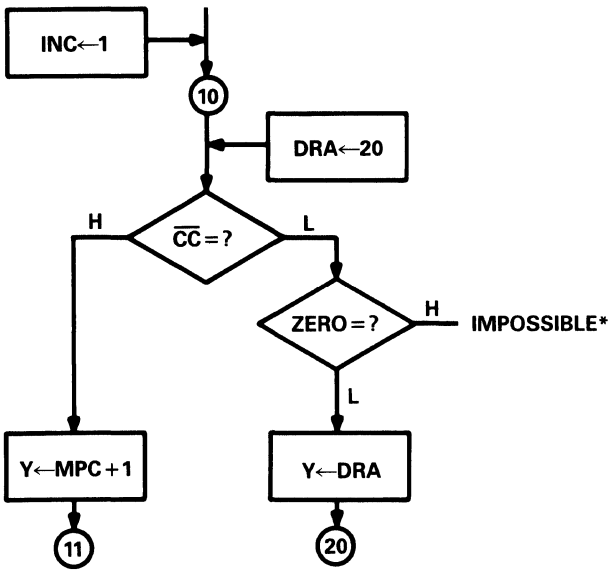
* Selected from external status

THIRTY-TWO-WAY BRANCH

To Branch 32-Way, this example uses the first instruction from Table 3-7 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 ten most-significant bits of DRA and DRB to provide new addresses DRA' (DRA13-DRA4::B3-B0) and DRB' (DRB13-DRB4::B3-B0).

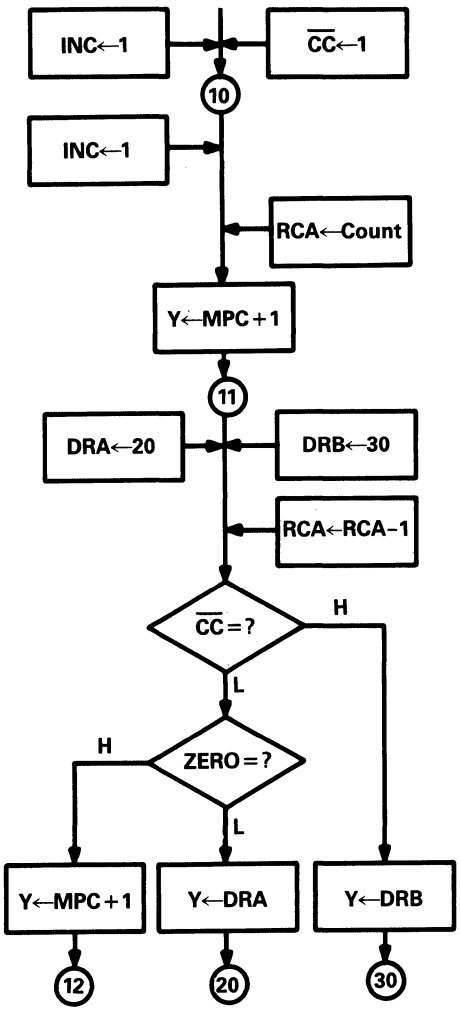
Address	Instruction	MUX2-MUX0	S2-S0	R2-R0	OSEL	$\overline{CC}$	INC	DRA	DRB
10	32-way Branch	101	111	000	0	X	X	0040	0030

Conditional Branch

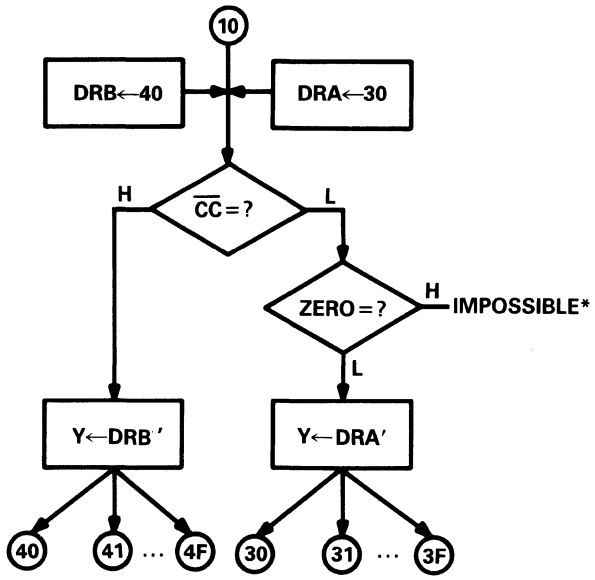


*no register decrement

Three-Way Branch



Thirty-Two-Way Branch



*no register decrement

Flow diagrams and suggested code for representative Loop instructions are shown below. Numbers inside the circles are microaddresses. Further information concerning loop routines can be found in Section 3.3.7.

REPEAT

To Repeat (instruction 10), this example uses the first instruction in Table 3-5 with CONT/RPT in the instruction column. INC must be programmed low and $\overline{CC}$ high one cycle ahead of instruction 10 (See Section 3.4.1).

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

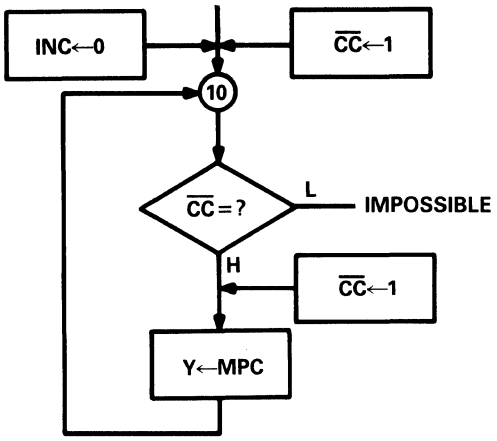
REPEAT ON STACK

To Continue and push the microprogram counter onto the stack (Push), this example uses the first instruction in Table 3-5 with CONT/RPT: PUSH in the instruction column. INC and $\overline{CC}$ must be forced high one cycle ahead (See Section 3.4.1).

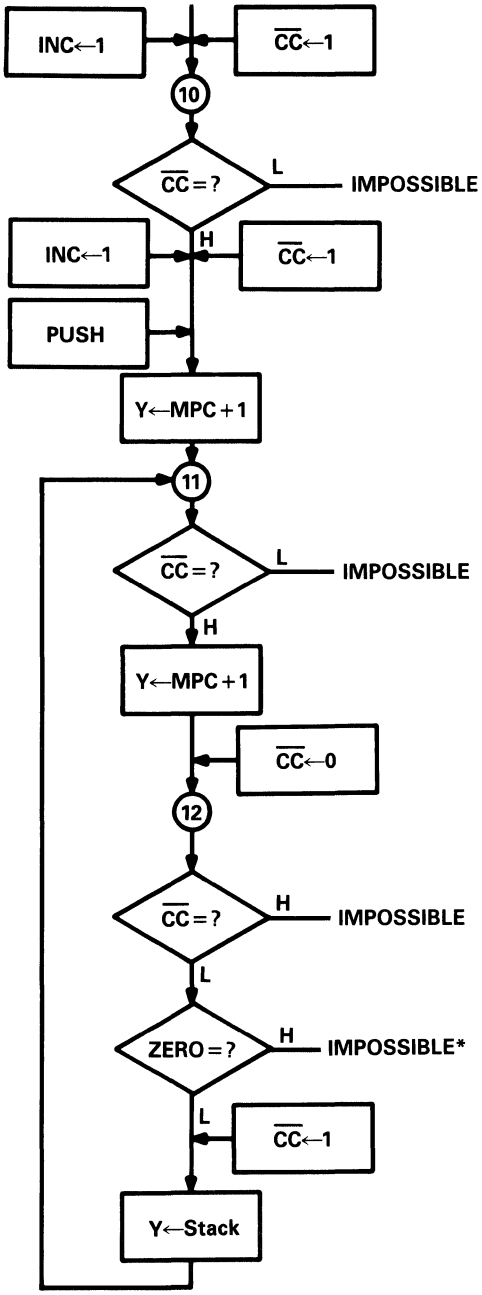
To Repeat (instruction 12), the first BR S instruction from the ZERO = L column of Table 3-6 is used. To avoid a ZERO = H condition, registers are not decremented during this instruction (see Table 3-4 for possible register operations). $\overline{CC}$ and INC are programmed high in instruction 12 to 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

Repeat



Repeat on Stack



*no register decrement

REPEAT UNTIL $\overline{CC}$ = H

To Continue and push the microprogram counter onto the stack (Push), this example uses the first instruction in Table 3-5 with CONT/RPT: PUSH in the instruction column. INC and $\overline{CC}$ must be forced high one cycle ahead (See Section 3.4.1).

To Repeat Until $\overline{CC}$ = H (instruction 12), the first instruction from Table 3-7 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 3-4 for possible register operations). $\overline{CC}$ and INC are programmed high in instruction 12 to set up the Continue in instruction 11. A repercussion 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

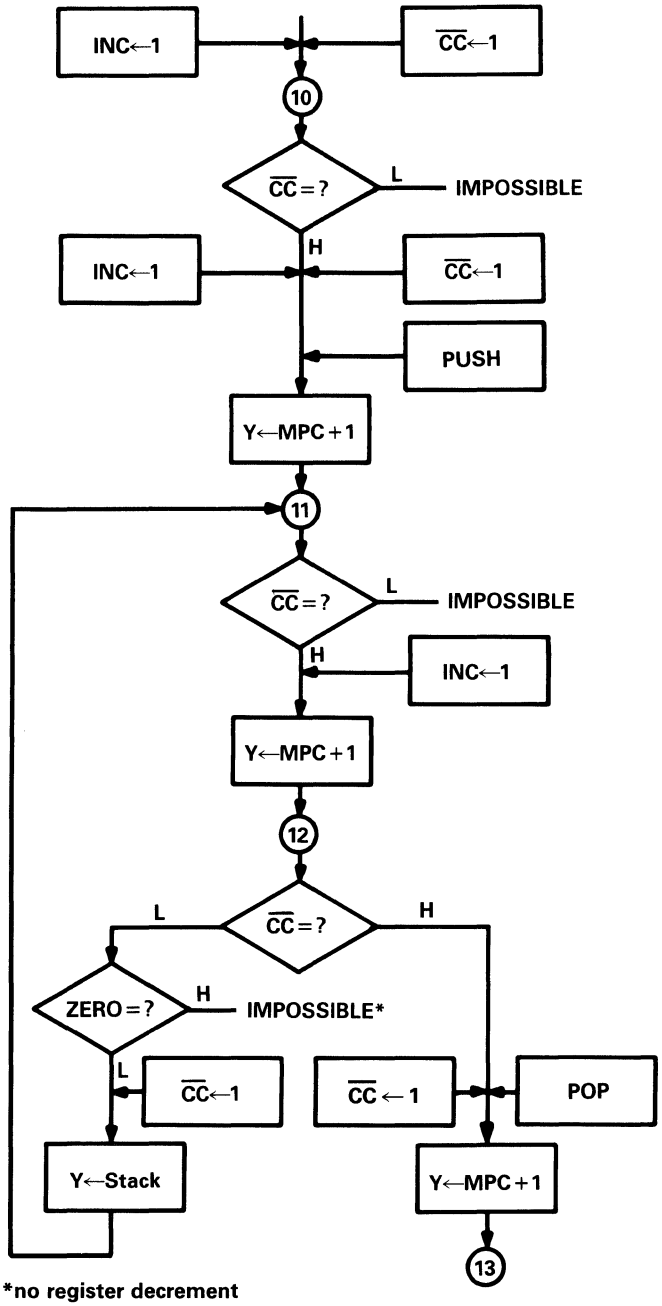
LOOP UNTIL ZERO

To Continue and push the microprogram counter onto the stack (Push), this example uses the first instruction in Table 3-5 with CONT/RPT: PUSH in the instruction column. INC and $\overline{CC}$ are forced high one cycle ahead (See Section 3.4.1). Register A is loaded with the loop counter using a Load A instruction from Table 3-4.

To decrement the loop count, a decrement register A and hold register B instruction from Table 3-4 is used. To Repeat Else Continue and Pop (decrement the stack pointer), the first instruction from Table 3-8 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.

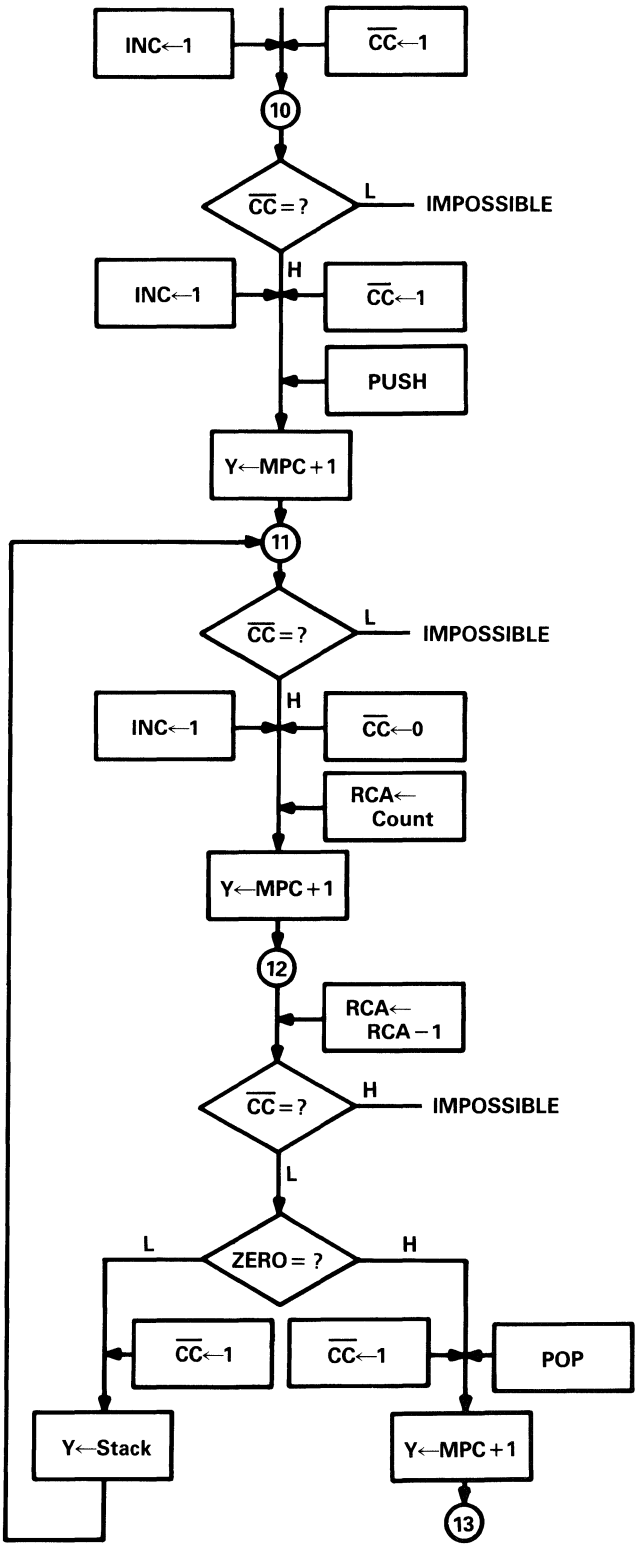
Address	Instruction	MUX2-MUX0	S2-S0	R2-R0	OSEL	$\overline{CC}$	INC	DRA	DRB
(Set-up)		XXX	XXX	XXX	X	1	1	XXXX	XXXX
10	Continue/Push	110	100	XXX	0	1	1	XXXX	XXXX
11	Continue/Load Reg A	110	111	010	0	0	1	XXXX	XXXX
12	Decrement Reg A; BR S else Continue: Pop	000	010	001	1	1	1	XXXX	XXXX

Repeat Until $\overline{CC}$ = H



*no register decrement

Loop Until Zero



CONDITIONAL LOOP UNTIL ZERO

Two examples of a Conditional Loop on Stack with Exit are shown below. Both use the microcode shown below to branch to the stack on non-zero, 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, as shown in the figure opposite.

To Continue and push the microprogram counter onto the stack (Push), these examples use the first instruction in Table 3-5 with CONT/RPT: PUSH in the instruction column. INC must be high. $\overline{CC}$ is forced high in the preceding instruction (See Section 3.4.1).

To Continue (instruction 11), these examples use the first instruction in Table 3-5 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), the first instruction from Table 3-8 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

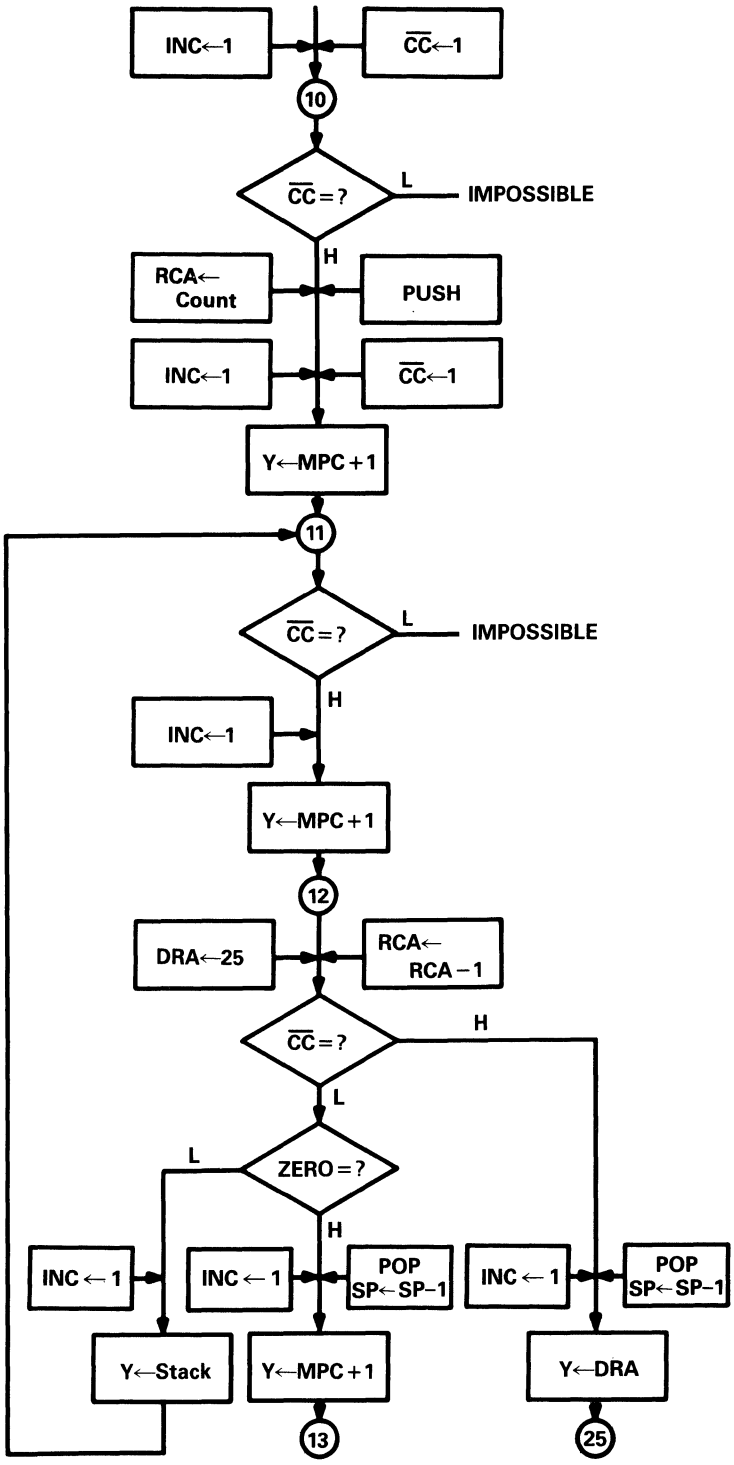
* Selected from external status

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	1	0025	XXXX

* Selected from external status

Conditional Loop Until Zero (Example 2)



Flow diagrams and suggested code for representative jump to subroutine (Call) instructions are given below. Numbers inside the circles are microaddresses. Further information about Call instructions is given in Section 3.3.8.

JUMP TO SUBROUTINE

To Call a Subroutine at address 30, this example uses the first instruction from Table 3-9 with CALL A in the $\overline{CC}$ = H column. $\overline{CC}$ is programmed high in the previous instruction. INC is programmed high to set up the push.

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 the first instruction from Table 3-9 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

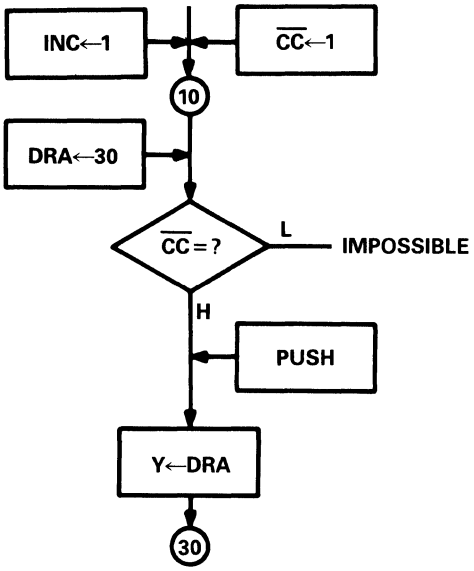
TWO-WAY JUMP TO SUBROUTINE

To perform a Two-Way Call to Subroutine at address 20 or address 30, this example uses the first instruction from Table 3-9 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.

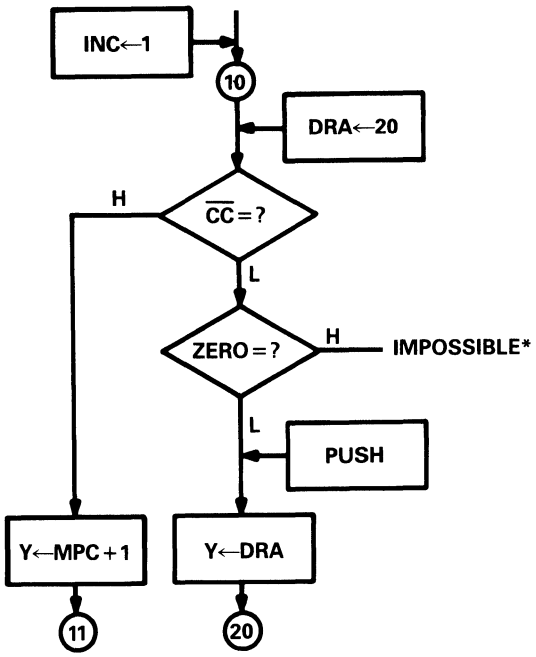
Address	Instruction	MUX2-MUX0	S2-S0	R2-R0	OSEL	$\overline{CC}$	INC	DRA	DRB
(Set-up)		XXX	XXX	XXX	X	*	1	XXXX	XXXX
23	Call A else Call B	100	110	000	X	X	X	0020	0030

* Selected from external status

Jump to Subroutine

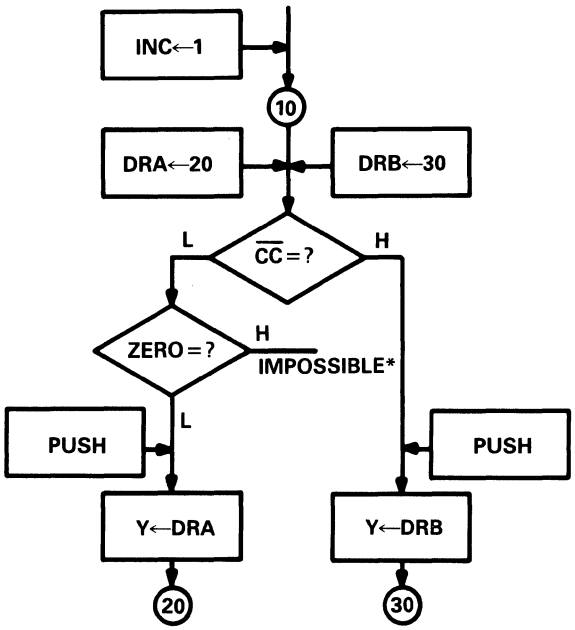


Conditional Jump to Subroutine



*no register decrement

Two-Way Jump to Subroutine



*no register decrement

Flow diagrams and suggested code for representative Return from Subroutine instructions are shown below. Numbers inside the circles are microaddresses. For more information about Return instructions, see Section 3.3.9.

RETURN FROM SUBROUTINE

To Return from a subroutine, this example uses the first instruction from Table 3-12 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

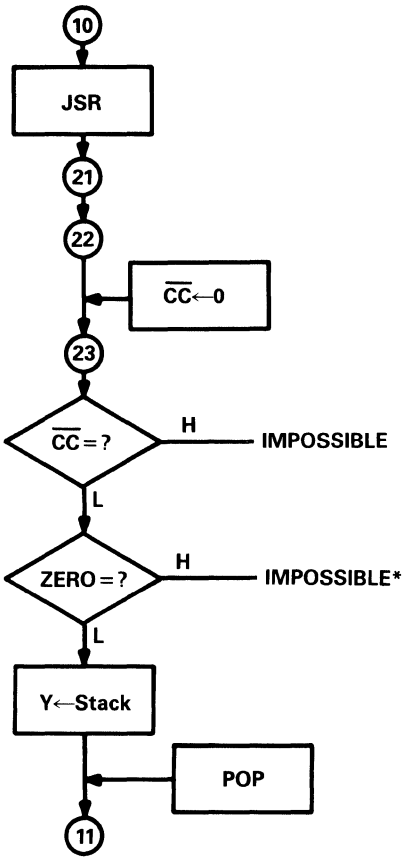
CONDITIONAL RETURN FROM SUBROUTINE

To conditionally Return from a Subroutine, this example uses the first instruction from Table 3-12 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

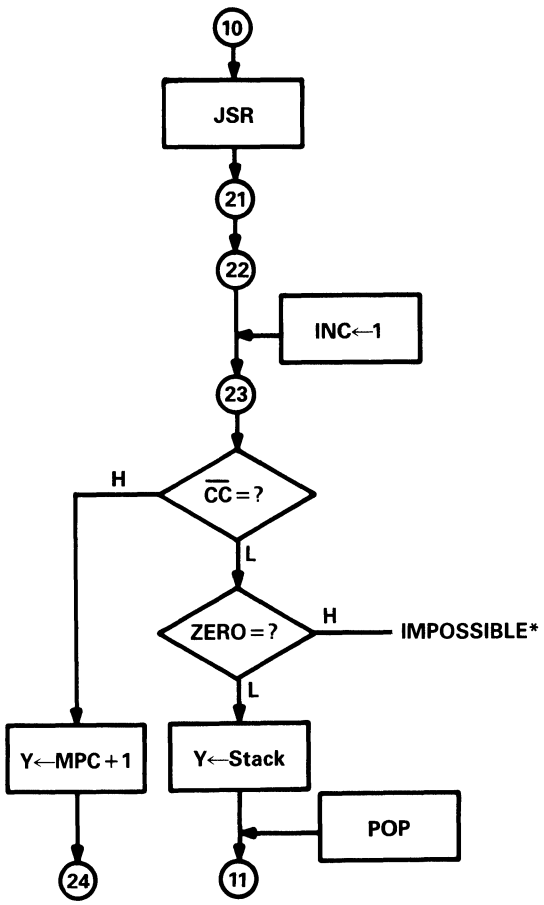
* Selected from external status

Return from Subroutine



* no register decrement

Conditional Return from Subroutine



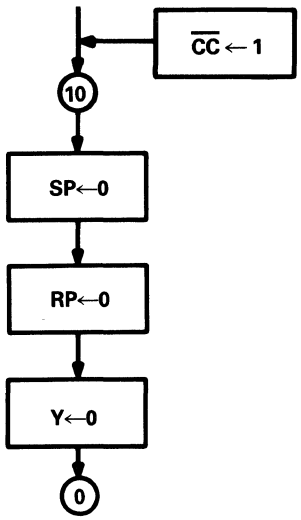
*no register decrement

RESET

To Reset the 'AS890, 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
(Set-up)		XXX	XXX	XXX	X	1	X	XXXX	XXXX
10	Reset	XXX	000	XXX	X	X	X	XXXX	XXXX

Reset



4. 32-Bit CPU Design Methodology

Microprogramming and bit-slice technology have made possible the development of powerful systems using flexible instructions sets and wide address/data buses to access more than one Gigaword of physical main memory. This section discusses one design approach to such a system, using 'AS888 bit-slice and 'AS890 microsequencer components.

A structured approach to system design, such as that illustrated in Figure 4-1, is recommended in developing custom bit-slice designs. The product specification gives a starting point or basis for the project. In this example, four 'AS888 bit slices are used to implement the 32-bit arithmetic portion of the CPU, and an 'AS890 microsequencer is used for ALU and system control. A group of PROMs stores the microinstructions; a writable control store could also be implemented using additional control logic and components to load and modify the microprogram memory. The system is designed to access more than one Gigaword of memory.

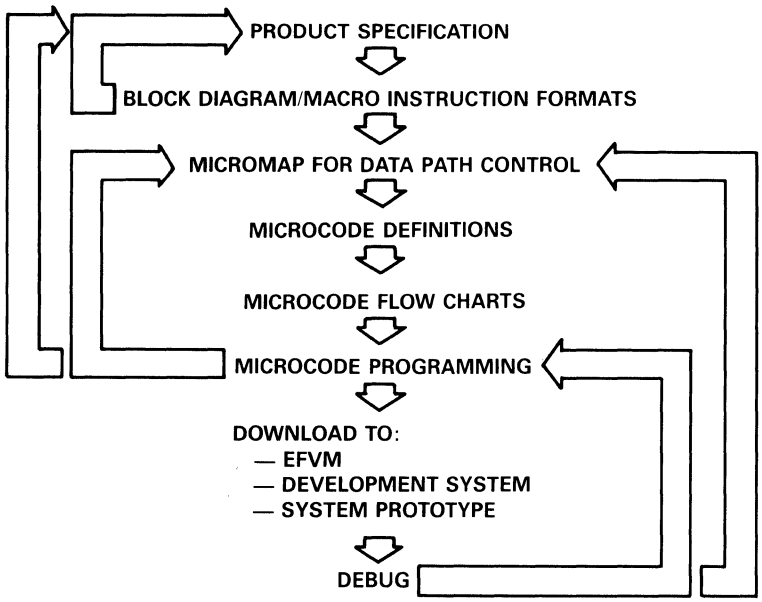


Figure 4-1. System Design Approach

Since speed is a concern, carry look-ahead rather than ripple-through logic is recommended. If ripple-through logic were used, the system clock would need to be slowed down to allow the propagation of the carry bits through the various 'AS888 stages. By using carry look-ahead, the amount of time needed for the data to stabilize is greatly reduced by anticipating the carry across the 'AS888 packages.

So that the scratchpad area can be used for address calculations and mathematical computations, the 'AS888's internal register file is dedicated for system functions. To provide the system user with a macrolevel equivalent of register locations, a 16-word external register file is also included. Access to the external register file will be under microprogram control, allowing address selection to come from the microcode itself or from one of the three operand fields of the instruction register.

PROMs eliminate the use of main memory as a source for constants used in initialization or table look-up functions. Accessing main memory for table values would require time and slow system throughput; by placing fixed values in fast PROMs, access time is kept to a minimum and system throughput is not altered.

Control, data and address buses shared by the system are accessed by three-state registers. The control register, as explained in section 4.1.2, supplies the non-CPU part of a computer system with control signals.The data bus allows the ALU to supply data for the rest of the system and can also be a source of data for the ALU; this is accomplished by using three-state registers to drive the bi-directional data bus, along with registers to sample the bus. The address bus uses one of the external register file locations to maintain a program counter, thus allowing a 32-bit address bus capable of addressing about four Gigawords of main memory. Using three-state drivers for this bus enables other subsystems to take control of the system buses.

A pipeline register supplies the microsequencer and the ALU with both data and instructions. To get macrocode into the system, an instruction register and a mapping PROM are used to convert the opcode to a microprogram routine address. The condition code signal, used for testing various conditions, is supplied by a register-input based PAL. PAL inputs can be fixed values or combinations of the status signals coming from the ALU. The read address select pins for the 'AS888's internal B register can be sourced from the microword itself or from three nibbles of the macroword, to provide offsets for the N-way branches to various microcode routines.

4.1 Designing a 32-Bit System

A typical 32-bit system block diagram using the 'AS888 bit-slice and 'AS890 microsequencer is shown in Figures 4-2 and 4-3. It can be broken down into two sections, the ALU (arithmetic logic unit) and the CCU (computer control unit). The ALU section performs all manipulation of data both to and from main memory, such as arithmetic and logical operations. The CCU section controls instruction (macrocode) flow and any miscellaneous control operations, such as fetching instructions or supplying addresses for main memory access.

4.1.1 Construction of the ALU

To cascade the four 'AS888s to obtain the 32-bit arithmetic unit shown in Figure 4-4, the shift multiplex $\overline{SIO0}$ and $\overline{QIO0}$ terminals are connected to the $\overline{SIO7}$ and $\overline{QIO7}$ terminals of adjacent packages, and the least significant package's signals are connected to the most-significant package's. Optionally, SN74ALS240 inverting gates can be connected to the $\overline{SIO0}$-$\overline{SIO7}$ terminals and the byte inputs to implement byte and bit control. Another chip, the SN74AS182 look-ahead carry generator, provides a ripple-carry function, to help system throughput.

The design includes a 16-word register file, the SN74AS870 (see Figure 4-3).This allows the user to access 16 working areas for temporary data storage or address calculations such as indexing. In this design example, the 'AS888's internal register file is not accessible directly by the user; it is reserved for microcode operations, such as address computation and temporary storage for arithmetic operations. Addressing the register files is permitted through the microprogram or from the macrocode instruction register under microcode control.

The transfer register connected to the 'AS888's Y and DB buses allows for feedback into the 'AS888 under microprogram control. Since the constant PROMs and the external register file share the A bus, they cannot be accessed at the same time. The transfer register enables data from the external register file to be transmitted to the B bus, making possible the addition of operands from the constant PROMs and the external register file, for example.

Constant PROMs are also included to simplify the programming and operation of the ALU by supplying fixed data for various operations, such as:

1) Clearing the system register files for initialization. This will bring the system up to a known state.
2) Supplying a correction value to the offset in a branch instruction, i.e., converting a 16-bit offset to a true 32-bit address.
3) Table look-up for fixed mathematical operations, such as computing sines and cosines.

4.1.2 Construction of the CCU

Sequencing and branching operations at speeds compatible with the 'AS888 are supplied by the 'AS890, a microprogrammed controller working as a powerful microsequencer (see Figure 3-1). Features of the 'AS890 include:

1) Stack capability. The 9-word stack can be accessed by using a stack pointer or a read pointer; the latter is designed for non-destructive dumping of the stack contents.
2) Register/counter facility. Two registers, DRA and DRB, can be used for latching data from the external data buses or as counters for loops. A ZERO signal is generated when the decremented counter reaches a zero value.
3) Interrupt control. A register for temporarily holding the return address is supplied; upon entering the interrupt routine, the contents of the return register must be pushed onto the stack for later use.
4) Next address generation. The Y output multiplexer offers a selection of same or incremented address, address from DRA or DRB buses, address from stack, or a concatenation of DRA13-DRA4 and B3-B0.

A microprogram memory/pipeline register supplies the microsequencer and the rest of the system with instructions (see Figure 4-2). The memory might consist of ROMs, or it could be a writable-control store with support logic to allow loading or updating of the control store. For a general purpose machine with a fixed instruction set, ROMs would be more economic.

Some 'AS890 instructions are influenced by the $\overline{CC}$ input. Many are variations of branch and jump instructions. To form and supply $\overline{CC}$, a register can be used to latch the state of the 'AS888 and supply inputs to a PAL for decoding, based upon the microcode's needs. Combinatorial logic in the PAL allows multiple or single events to be selected or provides a fixed value of "1" or "0" for forced conditions.

To supply the microsequencer with the proper address of the microcode-equivalent version of the macrocode instruction, an instruction register and mapping PROM are needed. Under microprogram control, the instruction register samples the data bus to get the macrocode instruction. The opcode portion is passed to the mapping PROM to form an address to the microcode routine. When the microcode is ready to jump to the routine, it turns off the Y bus output of the 'AS890 and enables the output of the mapping PROM. An optional means of altering the address uses B3-B0 inputs of the 'AS890 to implement a N-way branch routine. In this method, the ten most significant address bits of DRA or DRA are concatenated with the B3-B0 bits to supply an address.

Control information is supplied to the rest of the system via the control register and bus. By setting various bits within the control register, information can be passed to other subsystems, such as memory and I/O peripherals. Bit 0 could represent the read/write control line while bit 1 could select memory or I/O for the read/write. Bit 2 might function to enable interrupts and bit 3 to indicate when the system should enter a "wait" state for slow memory. The remaining control bits can be programmed by the system designer to indicate additional condition states of the "macrosystem".

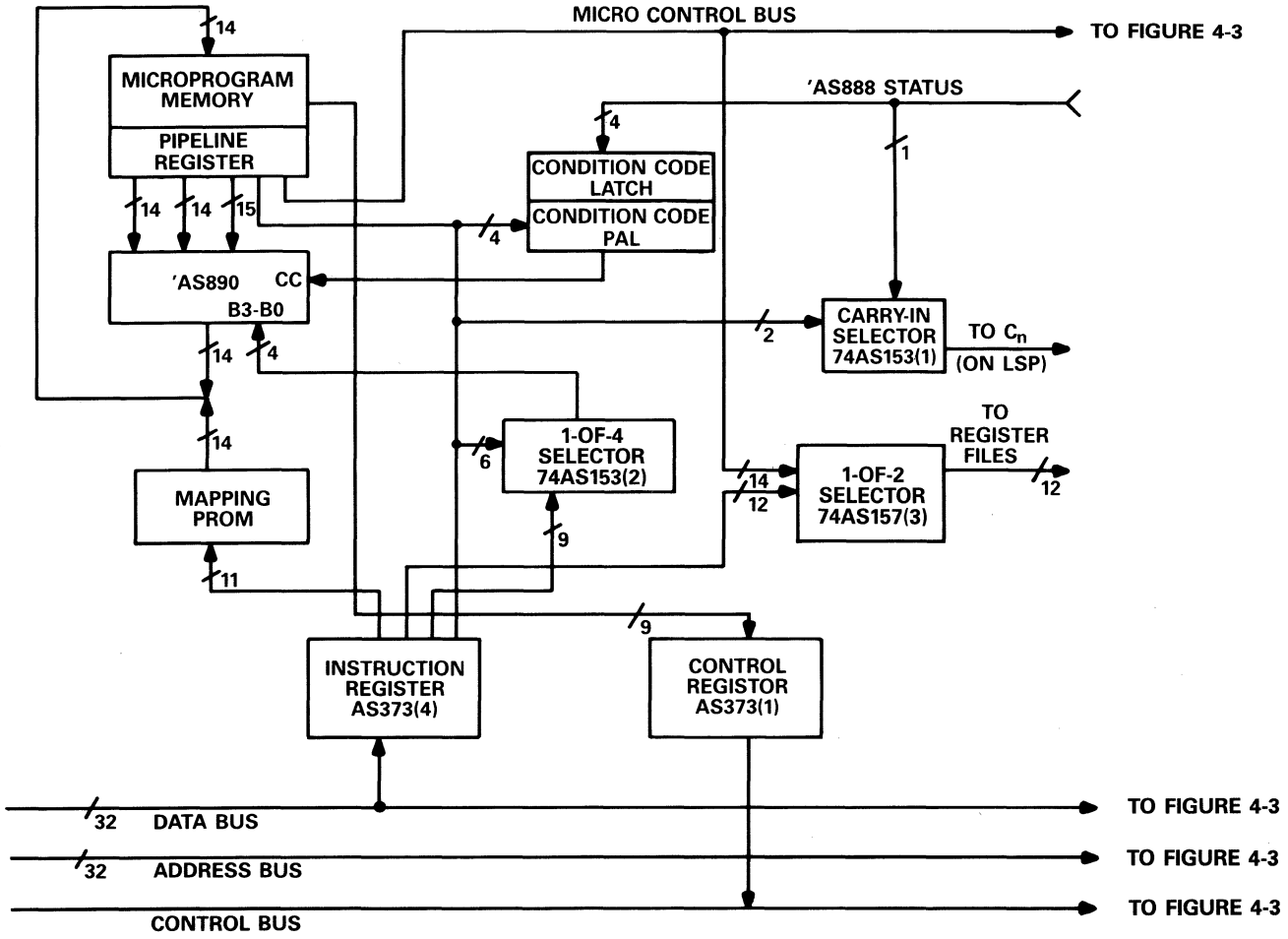


Figure 4-2. CCU Block Diagram

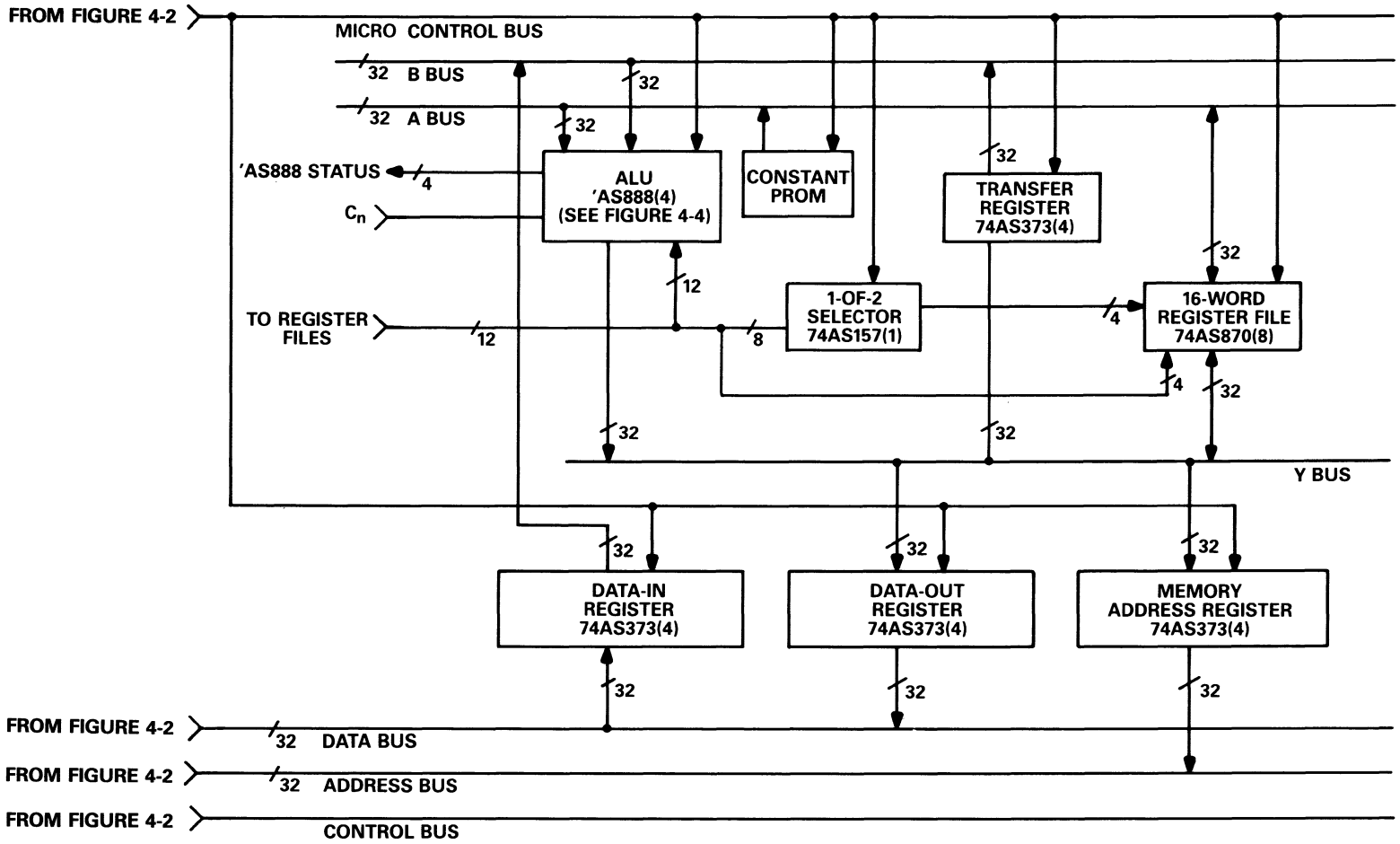


Figure 4-3. ALU Block Diagram

Addressing of the register files, both the 'AS888 internal and the 'AS870 external, is done through the use of two 1-of-2 selector banks. The first bank selects address source; this design offers a choice for operand processing of fixed values from the microcode or values from the macroinstruction latched in the instruction register. The second bank selects the first or second operand as an address source for port 0 of the external register file; port 1 uses the third operand as an address source.

It should be noted that the design presented in Figure 4-2 for the computer control unit is a one-level pipeline that is instruction-data based.The address and contents of the next instruction are being fetched while the current instruction is being executed. Tracing through the data flow, the following can be observed:

1) The pipeline register contains the current instruction being executed;
2) The ALU has just executed its instruction, and has the current status ready at its output pins;
3) The status register that is attached to the ALU contains the previous instruction's resulting status;
4) The contents of the next microprogram word are being fetched at the same time that the current instruction is being executed.

4.2 Tracing through a 32-Bit Computer

With the 'AS888 and 'AS890 as foundation chips, the typical 32-bit supermini of Figures 4-2 and 4-3 can now be functionally traced. First, note that the data of the main program is handled separately from that of the microcode — each on its own bus. The system is initialized by setting the "clear" signal high — this causes a forced jump to the beginning of the microcode memory. Instructions carried out by the microcode at this point might run system diagnostics, clear all registers throughout the 'AS888-based system, and set up the initial macrocode program address. In this design, the first program address to fetch an instruction from main memory comes from a fixed value in the microcode memory; it is possible to allow the address to be retrieved from a permanent location in main memory or from either a front panel or console, by modifying the microcode program slightly.

Table 4-1 illustrates the microcode format for this design. Note that it contains control signals for all chips involved in the design. Some of these, such as $\overline{\text{TRANSLATCH}}$ and $\overline{\text{MARLATCH}}$, are used with the system clock to provide controlled loading of the various holding registers. Others supply necessary addressing information, directing input from either the main data bus or from the microcode word itself.

The FETCH routine is shown in functional, assembler and microcoded forms in Tables 4-2, 4-3 and 4-4. First, the program counter is read from the external register file and stored into the memory address register. After the program counter is placed on the address bus, the program counter is updated and stored while the data from memory is allowed to settle down to a stable condition. The data is then latched in both the instruction register and data-in register.

The opcode field of the instruction register is passed through the mapping PROM to convert the opcode to an equivalent microcode routine address. When $\overline{\text{YOE}}$ is forced high by the microcode, the 'AS890 is tri-stated from the Y bus, and the mapping PROM's output is taken out of the tri-state mode to supply an address to the control store (microprogram memory); a forced jump is made to the microcode routine to perform the instruction.

After the routine is complete, a jump is made back to the FETCH routine using the next-address supplied by the microprogram. It is up to the system designer/programmer to make sure that all system housekeeping is performed so that nothing causes a fatal endless loop.

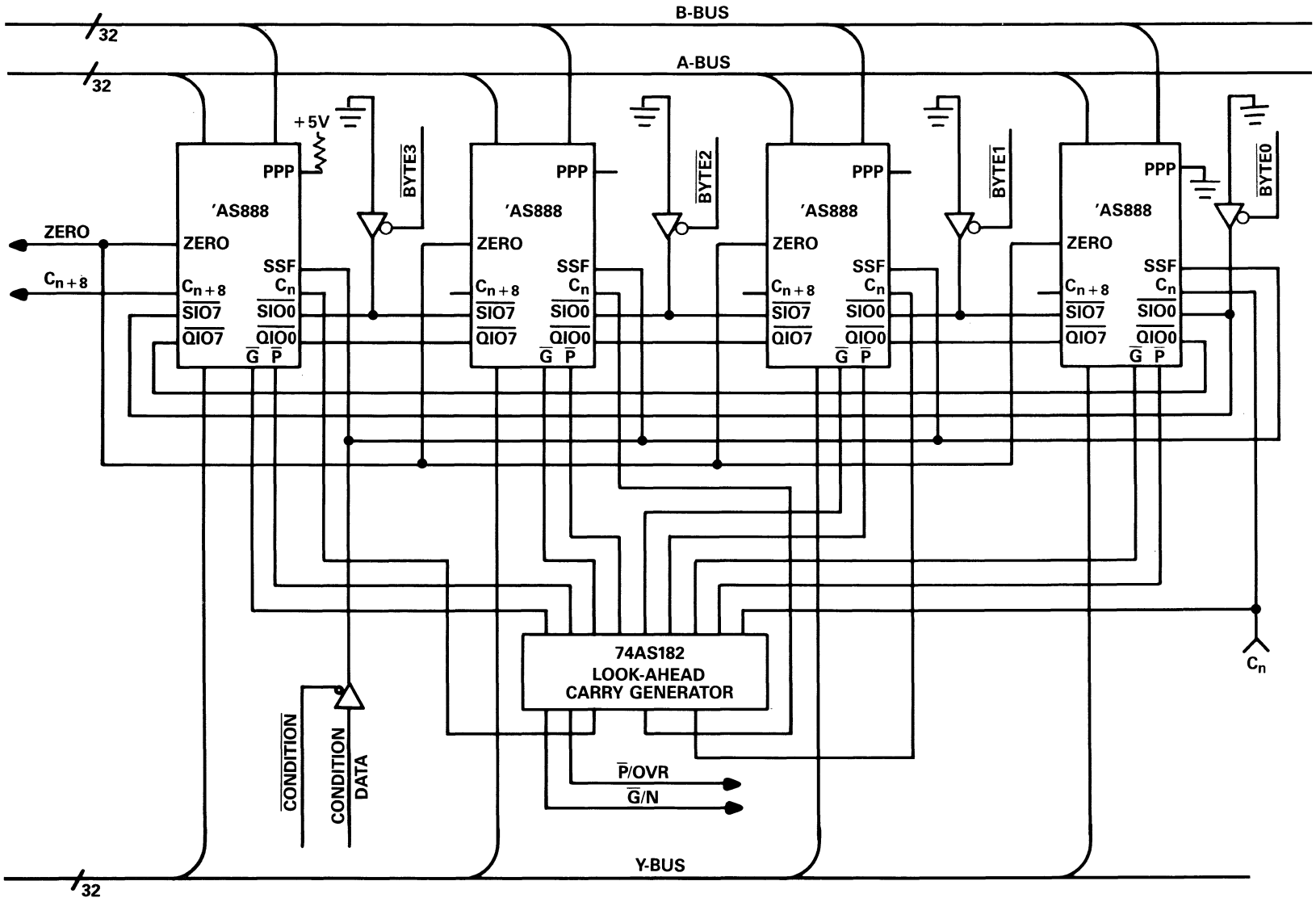


Figure 4-4. Cascaded 'AS888 Packages

Table 4-1. Microcode Definition

MICROCODE FIELD	PIN NAME	INPUT TO	FUNCTION
0–13	DRA13–DRA0	'AS890	Used for next-address branches
14–27	DRB13–DRB0	'AS890	Used for loading counter
28–30	RC2–RC0	'AS890	Register/counter controls
31–33	S2–S0	'AS890	Stack control
34–36	MUX2–MUX0	'AS890	MUX control of Y output bus
37	$\overline{\text{INT}}$	'AS890	Interrupt control
38	$\overline{\text{RAOE}}$	'AS890	Enables DRA output
39	$\overline{\text{RBOE}}$	'AS890	Enables DRB output
40	OSEL	'AS890	Mux control for DRA source
41	INC	'AS890	Incrementer control
42	$\overline{\text{YOE}}$	'AS890	Enables Y output bus
43–50	I7-I0	'AS888	Instruction inputs
51	$\overline{\text{OEA}}$	'AS888	DA bus enable
52	$\overline{\text{EA}}$	'AS888	ALU input operand select
53	$\overline{\text{OEB}}$	'AS888	DB bus enable
54	OEY	'AS888	Y bus output enable
55	SELY	'AS888	Y bus select
56–57	EB1–EB0	'AS888	ALU input operand selects
58	$\overline{\text{WE}}$	'AS888	Register file write enable
59	$\overline{\text{MAP}}$	PROM	Enables mapping PROM to 'AS890 Y bus
60	$\overline{\text{IR}}$	Latch	Latches data bus to instruction register
61	$\overline{\text{CR}}$	Latch	Latches control data to bus
62–69	CTRL7–CTRL0	Latch	Data for control latch
70–71	BSEL1–BSEL0	Multiplexer	Selects data for 'AS890
72–75	B3–B0	Multiplexer	Microcode data to switch
76	$\overline{\text{CONDCD}}$	Latch	Controls latch of 'AS888 status
77–80	SELC3–SELC0	PAL	Selects combination of 'AS888 status
81	$\overline{\text{DTALATCHI}}$	Latch	Controls latching of data-in
82	$\overline{\text{DTAIN}}$	Latch	Enables data-in output to bus
83	$\overline{\text{DTALATCHO}}$	Latch	Controls latching of data-out
84	$\overline{\text{DTAOUT}}$	Latch	Enables data-out output to DB bus
85	$\overline{\text{MARLATCH}}$	Latch	Controls latching of address
86	$\overline{\text{MAR}}$	Latch	Enables MAR output to address bus
87	$\overline{\text{CONSTPROM}}$	PROM	Enables PROM to DA bus
88–99	A11–A0	PROM	Address of constant in PROM
100	SWITCH2	Multiplexer	Selects microcode or Instruction Register data
101	SWITCH1	Multiplexer	Selects microcode or Instruction Register data
102–105	A3–A0	Multiplexer	Register file address ('AS888)
106–109	B3–B0	Multiplexer	Register file address ('AS888)
110–113	C3–C0	Multiplexer	Register file address ('AS888)
114	$\overline{\text{REGUWR}}$	Register File	Port 0 write enable
115	$\overline{\text{REGLWR}}$	Register File	Port 1 write enable
116	$\overline{\text{REGU}}$	Register File	Chip enable on port 0
117	$\overline{\text{REGL}}$	Register File	Chip enable on port 1
118	$\overline{\text{TRANSLATCH}}$	Latch	Controls latch between Y and DB bus
119	$\overline{\text{TRANS}}$	Latch	Enables output to DB bus
120	SELCN2	Multiplexer	Supplies carry input to 'AS888
121	SELCN1	Multiplexer	Supplies carry input to 'AS888
122	REGUB	Multiplexer	Selects address for external register file
123–126	$\overline{\text{BYTE3}}$ – $\overline{\text{BYTE0}}$	Three-state	Enables data for byte/bit operations

Table 4-2. Functional Listing of Fetch

```
FETCH:    MAR = PC, Enable MAR output
          PC = PC + 1
          IR = DIR = data bus, Disable 'AS890 Y bus,
               Enable mapping PROM to Y bus
```

Table 4-3. Assembler Listing of Fetch

```
FETCH: OP890 ,,,111,10;INC;          Set 'AS890 for continue
       OP888 NOP,GROUP5,10,,,1111;   Perform NOP and read external register 15
       OEY;SELY;                     Enable Y bus output
       CR;CTRL 00000011;             Generate external control bus signals
       SELC 01;                      Select fixed CC value to 'AS890
       MARLATCH;MAR;                 Latch value on Y bus and enable output
       SWITCH 00;REGL;               Select address source and enable port
       TRANSLATCH                    Latch Y bus for transfer to B bus
      OP890 ,,,111,10;INC;           Set 'AS890 for continue
       OP888 PASS,INCS,00,,,1111;    Increment program counter
       OEB;OEY;                      Enable Y bus output
       SELC 01;                      Select fixed CC value to 'AS890
       MAR;                          Output address to address bus
       REGLWR;REGL;                  Update program counter in register file
       TRANS;                        Enable transfer latch output to B bus
       SELCN 01                      Select carry input to LSP to be "1"
      OP890 ,,,111,10;               Set 'AS890 for continue
       OP888 NOP,GROUP5,10;          Perform NOP
       MAP;                          Enable mapping PROM to 'AS890 Y bus
       IR;                           Latch data bus to get macrolevel code
       SELC 01                       Select fixed CC value to 'AS890
       DTALATCHI;                    Put data bus also in data register
       MAR                           Output address to address bus
```

Key to Table 4-3

OP888 a,b,c,d,e,f
where:

- a = upper bits of instruction, 17-14
- b = lower bits of instruction, 13-10
- c = value of EB1–EB0
- d = A address of register files
- e = B address of register files
- f = C address of register files

OP890 v,w,x,y,z
where:

- v = DRA value, 14-bits
- w = DRB value, 14-bits
- x = RC2–RC0
- y = S2–S0
- z = MUX2–MUX0

Table 4-4. Microcode Listing of Fetch

DRA13-DRA0	DRB13-DRB0	RC2-RC0	S2-S0	MUX2-MUX0	$\overline{INT}$ $\overline{RAOE}$ RBOE OSEL $\overline{INC}$ $\overline{YOE}$	I7-I0	$\overline{OEA}$ $\overline{EA}$ $\overline{OEB}$ OEY SELY EB1 $\overline{EB0}$ $\overline{WE}$	$\overline{MAP}$ $\overline{IR}$ $\overline{CR}$
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0	1 1 1	0 1 0	1 1 1 0 1 0	1 1 1 1 1 1 1 1	1 1 1 0 1 1 0 1	1 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0	1 1 1	0 1 0	1 1 1 0 1 0	1 1 1 1 0 1 0 0	1 1 0 0 0 0 0 1	1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0	1 1 1	0 1 0	1 1 1 0 0 1	1 1 1 1 1 1 1 1	0 1 1 1 1 1 0 1	0 0 1

Table 4-4. Microcode Listing of Fetch (continued)

CTRL7-CTRL0	BSEL1-BSEL0	B3-B0	$\overline{CONDCO}$	SELC3-SELC0	$\overline{DTALATCHI}$ $\overline{DTAIN}$ $\overline{DTALATCHO}$ $\overline{DTAOUT}$ $\overline{MARLATCH}$ $\overline{MAR}$ $\overline{CONSTPRROM}$	A11-A0	SWITCH2-SWITCH1	A3-A0	B3-B0	C3-C0	$\overline{REGUWR}$ $\overline{REGLWR}$ $\overline{REGU}$ $\overline{REGL}$ TRANSLATCH TRANS SELCN2 SELCN1 REGUB	$\overline{BYTE3}$-$\overline{BYTE0}$
0 0 0 0 0 0 1 1	0 0	0 0 0 0	1	0 0 0 1	1 1 1 1 0 0 1	0 0 0 0 0 0 0 0 0 0 0 0	0 0	0 0 0 0	0 0 0 0	1 1 1 1	1 1 1 0 0 1 0 0 0	1 1 1 1
0 0 0 0 0 0 0 0	0 0	0 0 0 0	1	0 0 0 1	1 1 1 1 1 1 0 1	0 0 0 0 0 0 0 0 0 0 0 0	0 0	0 0 0 0	0 0 0 0	1 1 1 1	1 0 1 0 1 0 0 1 0	1 1 1 1
0 0 0 0 0 0 0 0	0 0	0 0 0 0	1	0 0 0 1	0 1 1 1 1 0 1	0 0 0 0 0 0 0 0 0 0 0 0	0 0	0 0 0 0	0 0 0 0	0 0 0 0	1 1 1 1 1 1 0 0 0	1 1 1 1

4.3 Defining the Macrocode Instruction Format

Since this is a 32-bit design, a variety of instruction formats are available. The size of the opcode, along with the types of addressing used, will affect both system size and performance. The formats shown in Table 4-5 will be used for discussion.

All Table 4-5 formats have an opcode field of 11 bits and source/destination fields of 7 bits; the first three bits of the latter designate the address type, and the remaining four bits are used for register access. The opcode length allows 2,048 macrocoded instructions to be mapped to equivalent microcoded routines. The address fields can specify any of the following modes: register, relative, autoincrement/autodecrement, indexed, absolute, and deferred. The offset used in the Type 0 instruction can be used for branch-based instructions, for an offset range of ±32727.

Table 4-5. Possible Instruction Formats

TYPE 0 — OPCODE + 16-BIT OFFSET

0 — 10	11 — 15	16 — 31
Opcode	Not Used	Offset

TYPE 1 — OPCODE + DESTINATION

0 — 10	11 — 24	25 — 31
Opcode	Not used	Destination

TYPE 2 — OPCODE + SOURCE + DESTINATION

0 — 10	11 — 17	18 — 24	25 — 31
Opcode	Not used	Source	Destination

TYPE 3 — OPCODE + SOURCE1 + SOURCE2 + DESTINATION

0 — 10	11 — 17	18 — 24	25 — 31
Opcode	Source	Source	Destination

4.4 Tracing a Macrocode Instruction

Microcode for a Type 3 multiplication instruction is shown in Table 4-6, using the following assumptions:

1) Code for retrieving the operands will not be shown. Jumps will be made to routines that will place the temporary operands into internal register locations 2 and 3 of the 'AS888, after being fetched from main memory.
2) A jump to a routine to store the product in the destination will be handled similarly.
3) Multiplication will be unsigned; the result will be placed in two temporary locations of the 'AS888.
4) An update to the program status word, which the user can access at the macrocode level must also be performed, but is not shown.

Assembler code is shown in Table 4-7; a microcode listing is given in Table 4-8. The first two lines of microcode are subroutine jumps to opcode fetching routines, which store the operands in register files 2 and 3 in the 'AS888. The next two instructions load up the 'AS890 with a counter constant for performing the multiply loop, load the MQ register of the 'AS888 with the multiplier and clear the register that is temporarily used for the accumulator.

Table 4-6. Functional Listing of Multiply

```
UMULI3:
  JUMPSUB SOURCE1              Get first operand
  JUMPSUB SOURCE2,             Get second operand
    BCOUNT=32                  Load DB counter register
  REG 9=0                      Clear temporary accumulator
  MQ=REG 2                     Load multiplier
LOOP:
  UMULI WITH REG 3             Issue the multiply
    DECREMENT BCOUNT,          Decrement the DB counter
    BRANCH TO LOOP IF NOT ZERO, Loop back until done
    LATCH 'AS888 STATUS,       Store 'AS888 flags
    REG 9=ALU                  Store intermediate result
  REG 8=MQ                     Store intermediate result
  JUMPSUB STORPSW              Update macro program status
  JUMPSUB MDEST                Store result at destination
  JUMP FETCH                   Get next instruction
```

A loop is then entered to perform the multiply instruction 32 times to form the product, with the multiplicand coming from the internal register file of the 'AS888. Upon exiting the loop, the MQ register is stored in a temporary register location in the 'AS888. The MQ register now contains the least-significant bits of the result and the temporary accumulator the most significant bits. A subroutine jump is made to the program status word update routine; this will take the status flags of the last multiplication iteration and change the macrolevel status word. The next subroutine jump is to a destination routine, which is followed by a branch to the FETCH routine to get the next macro instruction to be executed.

4.5 System Enhancements

The above example provides a broad overview of 32-bit system design using the 'AS888 and 'AS890. Certain additional options may enhance system performance. These include:

1) Status latching. The design does not take into account changes that need to be examined at the microlevel while retaining macrolevel status information. One solution would be to include another register in parallel to the status latch and provide control to choose between the two to form the condition code value.
2) Interrupts. To efficiently use a computer system, interrupts are used to alter program flow in the case of I/O programming and real-time applications (involving hardware timers). To include this capability, external hardware must be included and the microcode modified accordingly. Information on interrupt implementation is given in section 3.
3) Control store. One way of implementing microprogram memory is to use a ROM-based design. It is becoming more common to design a writable control store, a completely RAM-based or part RAM, part ROM storage system, that can be altered by system operation, such as initialization from a floppy disk subsystem, or by the user to optimize or implement new macrolevel instructions. The cost of implementation must be weighed with the risks involved in changing instructions which may not be supported by other sites.
4) Instruction word definitions. Changing the instruction word definitions will have an effect on both system design and performance. Removing Type 3 instructions from the design, for example, will have an effect on both hardware and software: the external register file addressing must be changed and the 1-of-2 selector

Table 4-7. Assembler Code of Multiply

```
UMULI3:
    OP890 SOURCE1,,,110,110;                    Perform a subroutine branch
     INC;YOE;                                   Increment address and enable Y bus
     OP888 NOP;GROUP5;                          Tell 'AS888 to do nothing during jump
     SELC 0001;                                 Set CC to "1" to set up 'AS890 continue
     MAR                                        Maintain address on main address buss
    OP890 SOURCE2,00000000100000,110,110,110;   Perform subroutine branch and load B
                                                counter
     INC;YOE;                                   Increment microaddress and enable Y bus
     OP888 NOP,GROUP5;                          Tell 'AS888 to do nothing during jump
     SELC 0001;                                 Set CC to "1" to set up 'AS890 continue
     MAR                                        Maintain address on main address bus
    OP890 ,,,111,110;                           Perform a continue instruction
     INC;YOE;                                   Increment microaddress and enable Y bus
     OP888 CLEAR,GROUP5,,,,1001;                Zero out register file accumulator
     WE;                                        Enable writing to register file
     SELC 0001;                                 Set CC to "1" to set up 'AS890 continue
     MAR                                        Maintain address on main address buss
    OP890 LOOP,,,111,110;                       Perform a continue instruction
     INC;YOE;                                   Increment microaddress and enable Y bus
     OP888 LOADMQ,INCS,,,0010;                  Load MQ register with S + Cn, from external
                                                register file
     MAR                                        Maintain address on main address bus
LOOP:
    OP890 LOOP,,101,111,100;                    Decrement B and loop til ZERO = 1
     INC;YOE;                                   Increment microaddress and enable Y bus
     OP888 UMULI,GROUP4,01,0011,,1001;          Perform unsigned multiply on accumulator
     WE;                                        Update register file accumulator
     MAR                                        Maintain address on main address bus
    OP890 ,,,111,110;                           Perform a continue instruction
     INC;YOE;                                   Increment microaddress and enable Y bus
     OP888 PASS,INCS,,,,1000;                   Put S + Cn in temporary register file
     WE;                                        Allow updating of register file
     MAR                                        Maintain address on main address bus
    OP890 STORPSW,,,110,110;                    Perform a subroutine branch
     INC;YOE;                                   Increment microaddress and enable Y bus
     OP888 NOP,GROUP5;                          Tell 'AS888 to do nothing during jump
     SELC 0001;                                 Set CC to "1" for set up 'AS890 continue
     MAR                                        Maintain address on main address bus
    OP890 FETCH,,,111;                          Perform a branch to FETCH routine
     INC;YOE;                                   Increment microaddress and enable Y bus
     OP888 NOP,GROUP5;                          Tell 'AS888 to do nothing during jump
     SELC 0001                                  Set CC to "1" for 'AS890 continue
```

Key to Table 4-7.

OP888 a,b,c,d,e,f
where:
- a = upper bits of instruction, 17-14
- b = lower bits of instruction, 13-10
- c = value of EB1-EB0
- d = A address of register files
- e = B address of register files
- f = C address of register files

OP890 v,w,x,y,z
where:
- v = DRA value, 14-bits
- w = DRB value, 14-bits
- x = RC2-RC0
- y = S2-S0
- z = MUX2-MUX0

removed. Likewise, changing the opcode length may restrict the instruction address capability and also cause either an increase or decrease in the microcode size.

5) Dynamic memory access (DMA). The above system does not support dynamic memory access. To include this function requires a change in the address output control, along with support circuitry for the type of DMA selected. Some error detection and correction logic for main memory might also be included.
6) Computer control unit. The design presented here shows a one-level pipeline architecture that is instruction-data based. System throughput may be increased by converting to a pipeline of greater depth, or using another variety of one-level pipeline, such as instruction-address based or address-data based. Care must be taken when increasing the size of the pipeline, especially when handling branch/jump situations. The reader is advised to carefully research this area before implementing any design.

4.6 Timing and System Throughput

A critical path analysis was undertaken to determine the maximum clock rate for the proposed system. The longest delay path is the multiplication data path, which involves the internal register file and the shift function of the 'AS888. Table 4-9 contains the critical delay calculations for both the ALU and CCU. Since both portions of the system must be satisfied, a clock rate of 90 ns was selected for the following comparisons.

4.6.1 Fetch Analysis

Most microprocessors perform an instruction fetch in a pipeline mode; the next instruction is fetched while the current instruction is executing. The fetch code shown earlier requires a minimum of four cycles: three to issue the code and one to break the pipeline for processing the branch. This results in a total time of 360 ns, based on a 90 ns cycle time. Fetch times for the representative microprocessors have been estimated from data books and are shown in Table 4-10; wait states for slow memory are not included. As can be seen from the table, the 'AS888 design example is estimated to run from 1.1 to 2.1 times faster than the 16-bit microprocessors.

4.6.2 Multiplication Analysis

This analysis assumes that multiplication is unsigned integer and register to register based. No account is taken of time needed for instruction fetch or operand fetch or store.

The basic loop for the multiply takes 35 cycles: 2 for accumulator and multiplier set up, 32 for actual multiply loop and 1 to store the least-significant bits in an internal register file. Given a cycle time of 90 ns, a 32 by 32 bit multiplication can be implemented in 2.275 microseconds. A 16-bit multiply requires 16 iterations of the inner loop; both timings are included in Table 4-11 for comparison. Values for the 16-bit multiplies of the representative microprocessors have been estimated from data books.

As shown in Table 4-11, the 16 by 16 multiply can be performed with the 'AS888 at a faster rate than the 16-bit microprocessors. Even comparing the 32 by 32 multiply of the application design, one can see that the 'AS888 based system has a better macroinstruction execution speed. Using the 'AS888 and 'AS890 in a system design will allow high throughput and permit a flexible architecture.

Table 4-8. Microcode Listing of Multiply

DRA13-DRA0	DRB13-DRB0	RC2-RC0	S2-S0	MUX2-MUX0	$\overline{INT}$ $\overline{RAOE}$ $\overline{RBOE}$ OSEL $\overline{INC}$ $\overline{YOE}$	I7-I0	$\overline{OEA}$ $\overline{EA}$ $\overline{OEB}$ OEY SELY EB1 $\overline{EB0}$ $\overline{WE}$	$\overline{MAP}$ $\overline{IR}$ $\overline{CR}$
0 0 0 0 0 0 0 0 0 0 1 1 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0	1 1 0	1 1 0	1 1 1 0 1 0	1 1 1 1 1 1 1 1	1 1 1 1 0 0 0 1	1 1 1
0 0 0 0 0 0 0 0 0 1 0 0 0 0	0 0 0 0 0 0 0 0 1 0 0 0 0 0	1 1 0	1 1 0	1 1 0	1 1 1 0 1 0	1 1 1 1 1 1 1 1	1 1 1 1 0 0 0 1	1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0	1 1 1	1 1 0	1 1 1 0 1 0	1 1 1 1 0 0 0 0	1 1 1 0 0 0 0 0	1 1 1
0 0 0 0 1 0 0 0 0 0 1 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0	1 1 1	1 1 0	1 1 1 0 1 0	1 1 1 0 0 1 0 0	1 1 1 0 0 0 0 1	1 1 1
0 0 0 0 1 0 0 0 0 0 1 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0	1 0 1	1 1 1	1 0 0	1 1 1 0 1 0	1 1 0 1 0 0 0 0	1 1 1 0 0 0 1 0	1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0	1 1 1	1 0 1	1 1 1 0 1 0	1 1 1 1 1 1 1 1	1 1 1 0 0 0 1 0	1 1 1
0 0 0 0 0 0 0 0 1 0 1 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0	1 1 0	1 1 0	1 1 1 0 1 0	1 1 1 1 1 1 1 1	1 1 1 1 0 0 0 1	1 1 1
0 0 0 0 0 0 0 0 1 1 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0	1 1 0	1 1 0	1 1 1 0 1 0	1 1 1 1 1 1 1 1	1 1 1 1 0 0 0 1	1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 1 1	0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0	1 1 1	0 0 0	1 1 1 0 1 0	1 1 1 1 1 1 1 1	1 1 1 1 0 0 0 1	1 1 1

Table 4-8. Microcode Listing of Multiply (continued)

CTRL7-CTRL0	BSEL1-BSEL0	B3-B0	$\overline{CONDCO}$	SELC3-SELC0	$\overline{DTALATCHI}$ $\overline{DTAIN}$ $\overline{DTALATCHO}$ $\overline{DTAOUT}$ $\overline{MARLATCH}$ $\overline{MAR}$ $\overline{CONSTPRROM}$	A11-A0	SWITCH2-SWITCH1	A3-A0	B3-B0	C3-C0	$\overline{REGUWR}$ $\overline{REGLWR}$ $\overline{REGU}$ $\overline{REGL}$ TRANSLATCH TRANS SELCN2 SELCN1 REGUB	$\overline{BYTE3}$-$\overline{BYTE0}$
0 0 0 0 0 0 0 0	0 0	0 0 0 0	1	0 0 0 1	1 1 1 1 1 1 0 1	0 0 0 0 0 0 0 0 0 0 0 0	0 0	0 0 0 0	0 0 0 0	0 0 0 0	1 1 1 1 1 1 0 0 0	1 1 1 1
0 0 0 0 0 0 0 0	0 0	0 0 0 0	1	0 0 0 1	1 1 1 1 1 1 0 1	0 0 0 0 0 0 0 0 0 0 0 0	0 0	0 0 0 0	0 0 0 0	0 0 0 0	1 1 1 1 1 1 0 0 0	1 1 1 1
0 0 0 0 0 0 0 0	0 0	0 0 0 0	1	0 0 0 1	1 1 1 1 1 1 0 1	0 0 0 0 0 0 0 0 0 0 0 0	0 0	0 0 0 0	0 0 0 0	1 0 0 1	1 1 1 1 1 1 0 0 0	1 1 1 1
0 0 0 0 0 0 0 0	0 0	0 0 0 0	0	0 0 0 1	1 1 1 1 1 1 0 1	0 0 0 0 0 0 0 0 0 0 0 0	0 0	0 0 0 0	0 0 1 0	0 0 0 0	1 1 0 1 1 1 0 0 0	1 1 1 1
0 0 0 0 0 0 0 0	0 0	0 0 0 0	0	0 0 0 1	1 1 1 1 1 1 0 1	0 0 0 0 0 0 0 0 0 0 0 0	0 0	0 0 1 1	0 0 0 0	1 0 0 1	1 0 1 0 1 1 0 0 0	1 1 1 1
0 0 0 0 0 0 0 0	0 0	0 0 0 0	0	0 0 0 1	1 1 1 1 1 1 0 1	0 0 0 0 0 0 0 0 0 0 0 0	0 0	0 0 0 0	0 0 0 0	1 0 0 0	1 0 1 0 1 1 0 0 0	1 1 1 1
0 0 0 0 0 0 0 0	0 0	0 0 0 0	0	0 0 0 1	1 1 1 1 1 1 0 1	0 0 0 0 0 0 0 0 0 0 0 0	0 0	0 0 0 0	0 0 0 0	0 0 0 0	1 1 1 1 1 1 0 0 0	1 1 1 1
0 0 0 0 0 0 0 0	0 0	0 0 0 0	1	0 0 0 1	1 1 1 1 1 1 0 1	0 0 0 0 0 0 0 0 0 0 0 0	0 0	0 0 0 0	0 0 0 0	0 0 0 0	1 1 1 1 1 1 0 0 0	1 1 1 1
0 0 0 0 0 0 0 0	0 0	0 0 0 0	1	0 0 0 1	1 1 1 1 1 1 1 1	0 0 0 0 0 0 0 0 0 0 0 0	0 0	0 0 0 0	0 0 0 0	0 0 0 0	1 1 1 1 1 1 0 0 0	1 1 1 1

Table 4-9. Critical Delay Path Analysis

CONTROL PATH			DATA PATH		
Pipeline Reg.	Clock to Output	9	'AS888-1	Clock to C_n	46
MUX	Select to Output	13	'AS182	C_n to C_{n+z}	5
'AS890-1	CC to Output	25	'AS888-1	C_n to SIO	25
PROM	Access Time	20	'AS888-1	SIO to Y	14
Pipeline Reg.	Setup Time	2			90 ns
		69 ns			

Table 4-10. Fetch Timing Comparison

FETCH	'AS888 32-BIT	Z8001	8086-1	80286	68000L
Data width	32	16	16	16	16
No. of cycles	4	3	4	4	4
Clock rate	11.11 MHz	4 MHz	10 MHz	10 MHz	8 MHz
Total time	360 ns	750 ns	400 ns	400 ns	600 ns

Table 4-11. Multiply Timing Comparison

MULTIPLY	'AS888 32-BIT	'AS888 16-BIT	Z8001	8086-1	80286	68000L
Size	32 × 32	16 × 16	16 × 16	16 × 16	16 × 16	16 × 16
No. of cycles	35	19	70	128	21	≤74
Clock rate	11.11 MHz	10.98 MHz	4 MHz	10 MHz	10 MHz	8 MHz
Total time	3.150 μs	1.729 μs	17.5 μs	12.8 μs	2.1 μs	≤9.25 μs

5. Floating-Point System Design

Bit-slice processor architecture addresses the problem of optimizing system performance while allowing the user to balance hardware complexity against software flexibility. Bit-slice systems usually operate at or near the speed of the most primitive of programmable processors, the PROM state sequencer. Of course, bit-slice architecture incorporates circuitry dedicated not only to sequencing, but also data processing (ALU) operations. In keeping with the trend of these programmable devices to track the speed of fast discrete hardware, the 'AS888 8-bit slice ALU and 'AS890 microsequencer have been produced in Advanced Schottky bipolar technology. In addition to sheer speed, the components feature greater density (2 micron geometry) for greater functionality (more special purpose circuitry on board). The impact will be faster, more powerful systems in applications which previously pushed the limits of bit-slice processors.

Consider an application in which bit-slice architecture has dominated for years: CPU design. The microprogrammed CPU itself spans a spectrum of uses ranging from general purpose minicomputers to compact airborne computers. A specific example which illustrates various facets of design using the 'AS888 and 'AS890 is a CPU with a floating-point utility to compute sin(x).

The design process can be subject to many influences, including personal preference, available development tools, peculiarities of the application, and constraints from the user, customer or manufacturing environment. No hard and fast design rules could be applied universally, but most designers will start with a specific plan in mind.

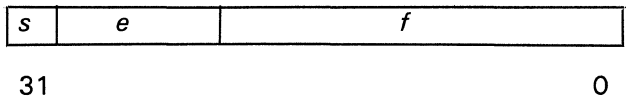
The goal of this example is to produce the hardware and microprogram which will implement the sin(x) function in floating-point arithmetic. Before the microprogram can be assembled, the hardware must be defined since the fields of the microinstruction are dedicated to specific hardware once the microinstruction register is hardwired to the devices it controls. Since the final architecture chosen depends on tradeoffs between implementing certain operations in hardware or software, critical applications will require that a cursory analysis of the software be made before the hardware is cast in concrete. Attempting to develop microcode for a tentative architecture will force the issue on which operations are better suited for hardware. Before the architecture or the microprogram requirements can be known, the algorithms which describe the application processes must be defined. Once an algorithm is formulated it can be broken down into operations involving variable and constant quantities. The variables can be assigned to registers and then the algorithm can be translated into a microprogram. The following steps illustrate the plan for this CPU design example incorporating a floating-point sin(x) utility:

Step 1: Choose a floating-point number system
Step 2: Choose an algorithm for approximating sin(x)
Step 3: Make 'AS888 register assignments
Step 4: Substitute registers for variables in the algorithm
Step 5: Decompose steps of the algorithm into simple operations
Step 6: Translate into 'AS888/890 operations; identify subroutines
Step 7: Translate subroutines into 'AS888/890 operations
Step 8: Evaluate tradeoffs and block diagram the hardware
Step 9: Define microinstruction fields during detailed hardware design
Step 10: Assemble the microprogram

5.1 Choose a Floating-Point Number System

An IEEE floating-point format will be chosen for this example for portability of data and software. It is important to note that the IEEE defines many standards in arithmetic processing, but for simplicity this example will encompass only number format. Furthermore, while several formats are IEEE compatible, only the basic single-precision format will be considered.

The IEEE basic single-precision format is defined as a 32-bit representation in which the component fields are a 1-bit sign *s*, an 8-bit biased exponent *e* and a 23-bit fraction *f* which are assembled in the following order:



The quantity is evaluated as $(-1)^s\ 2^{e-127}\ (1.f)$. Not-a-number, zero and infinity have special representations. The one preceding the binary point is implied and is called the implicit one or implicit bit. It coincides with the fact that the digits are normalized (left justified).

5.2 Choose an Algorithm for Sin(x)

Many algorithms are discussed in the literature for approximating useful quantities like sin(x). Literature research is a good place to start to familiarize oneself with various algorithms and tradeoffs for a particlar application. Computer simulation is also useful to compare algorithms for speed and accuracy. R.F. Ruckdeschel in *BASIC Scientific Subroutines*, Vol. 1 (BYTE, McGraw-Hill Publications Co. New York, N.Y., 1981, pp. 159–191 discusses tradeoffs and provides a simulation in BASIC for a sin(x) algorithm. An adaptation of this material has been chosen for this example:

A) Reduce angle range to first quadrant. $(0 \leq x \leq \pi/2)$

B) Compute $\sin(x) \simeq \sum_{n=0}^{6} A_n x^{2n-1}$. The coefficients are:

Coefficient	Decimal	IEEE hex
A_0	1.000000	3F80 0000
A_1	−0.1666667	BE2A AAAD
A_2	0.008333333	3C08 8888
A_3	−0.0001984127	B950 0D01
A_4	0.000002755760	3638 EF99
A_5	−0.00000002507060	B2D7 5AD5
A_6	0.0000000001641060	2F34 6FBC

The algorithm can be implemented in the following steps:

A) Reduce angle range to first quadrant. ($0 \leq x \leq \pi/2$)

1) SIGN = SGN(x)
2) ABSX = $\|x\|$
3) XNEW = ABSX $- 2\pi \times$ INT(ABSX/2π)
4) If XNEW $> \pi$ then SIGN = $-$SIGN and XNEW = XNEW $- \pi$
5) If XNEW $> \pi/2$ then XNEW = π $-$ XNEW

where

$$SGN(x) = \begin{cases} +1 \text{ if } x \geq 0 \\ -1 \text{ if } x < 0 \end{cases}$$

INT(x) = integer function

B) Compute $\sin(x) \simeq \sum_{n=0}^{6} A_n x^{2n-1}$.

1) Let XSQR = XNEW^2; INITIALIZE SINX = 0
2) Do i = 6 to 1 step -1
   SINX = XSQR $\times$ SINX + A(i)
   Enddo
3) SINX = SIGN $\times$ XNEW $\times$ SINX

Step B-2 computes the summation in a geometric series for economy. The major difference between steps A and B is that A requires more diverse ALU operations while B uses only multiplication and addition recursively.

5.3 Make 'AS888 Register Assignments

Just as in assembly language programming, registers must be allocated for variables. Using Rn to denote the 'AS888 register whose address is n, where $0 \leq n \leq$ F (hex), the following register assignments can be made:

R0 = X
R1 = SIGN
R2 = ABSX
R3 = XNEW
R4 = XSQR
R5 = SINX

The following constants can also be defined:

Constant	Decimal	IEEE hex
PI = π	3.141593	4059 0FDB
PIOVR2 = $\pi/2$	1.570797	3FC9 0FDB
2PI = 2π	6.283185	40C9 0FDB
1OVR2PI = $1/2\pi$	0.159155	3E22 F981

5.4 Substitute Registers for Variables in the Algorithm

Now the algorithm can be rewritten with registers replacing variables:

A) Reduce angle range to first quadrant ($0 \leq x \leq \pi/2$).
1) R1 = SGN(R0)
2) R2 = ‖R0‖
3) R3 = R2 − 2π × INT(R2/2π)
4) If R3 > π then R1 = −R1; R3 = R3 − π
5) If R3 > π/2 then R3 = π − R3

B) Compute $\sin(x) \simeq \sum_{n=0}^{6} A_n x^{2n-1}$.
1) Let R4 = R0²; INITIALIZE R5 = 0
2) Do i = 6 to 1 step −1
R5 = R4 × R5 + A(i)
Enddo
3) R5 = R1 × R0 × R5

Since various references to constants are made, it is probably best to load constants as needed rather than attempt to allocate registers for them. Constants can be loaded from a constant field in the microinstruction or from ROM. The tradeoff is 32 bits by 16K of micromemory versus 32 bits by the number of constants (typically less than 16K). For this example, it will be assumed that a constant field in the microinstruction is acceptable.

5.5 Decompose Steps in the Algorithm into Simple Operations

The sin(x) function can be microprogrammed as a subroutine; let FSIN be its entry address. R0 would be loaded with x before FSIN were called. Upon return, R5 would contain sin(x). Now decompose the steps in the algorithm into simple arithmetic and logical operations. Other operations can be left as functions to be defined later.

```
FSIN: SUBROUTINE

;  A) Reduce angle range to first quadrant. (0 ≤ x ≤ π/2)

   R1 = SGN(R0)                    ;  1) Let R1 = Sign of R0

   R2 = ABS(R0)                    ;  2) R2 = ‖R0‖

   R3 = R2 * 1OVR2PI               ;  3) R3 = R2 − 2π * INT(R2/2π)
   R3 = INT(R3)                    ;
   R3 = R3 * 2PI                   ;
   R3 = R2 − R3                    ;

   Y = R3 − PI                     ;  4) If R3 > π,
   Jump if Negative to Step A-5    ;
   R1 = −R1                        ;     then R1 = −R1;
   R3 = R3 − PI                    ;     R3 = R3 − π

   Y = PIOVR2 − R3                 ;
   Jump if Negative to Step B-1    ;  5) If R3 > π/2
   R3 = PI − R3                    ;     then R3 = π − R3
```

```
; B) Compute sin(x) ≃ Σ (n=0 to 6) An x^(2n-1)

    R4 = R0 * R0                ; 1) Let R4 = R0^2. Let R5 = 0
    R5 = 0                      ;

    R5 = R4 * R5                ; 2) Do i=6 to 1 step -1
    R5 = R5 + A6                ;      R5 = R4 × R5 + A(i)
    R5 = R4 * R5                ;    Enddo
    R5 = R5 + A5                ;
    R5 = R4 * R5                ;    (To implement a loop,
    R5 = R5 + A4                ;    use an 'AS890 counter
    R5 = R4 * R5                ;    to index a memory containing
    R5 = R5 + A3                ;    the constants.)
    R5 = R4 * R5                ;
    R5 = R5 + A2                ;
    R5 = R4 * R5                ;
    R5 = R5 + A1                ;
    R5 = R4 * R5                ;
    R5 = R5 + A0                ;

    R5 = R0 * R5                ; 3) R5 = R1 × R0 × R5
    R5 = R5 * R1 : RETURN       ;

  END SUBROUTINE
```

5.6 Translate into 'AS888/890 Instructions; Identify Subroutines

The simplified steps of the algorithm can be represented fairly easily as 'AS888/890 instructions. Necessary functions (and suggested names) can be identified by inspection as:

1) FMUL — Floating-point multiplication
2) FADD — Floating-point addition
3) FINT — Floating-point integer conversion
4) FINV — Floating-point additive inverse (to subtract using FADD)
5) FABS — Floating-point absolute value
6) FSGN — Floating-point sign test
7) FCHS — Floating-point change of sign (to multiply by SIGN)

''Function'' in this context refers to a special operation regardless of how it is coded. In fact, FMUL and FADD are fairly complex and require detailed explanation. FINV, FABS, FSGN and FCHS are single instruction operations that mask or mask and test. FINT requires several inline instructions or a subroutine and will be left to the interested reader as an exercise. Now the steps of the algorithm can be translated into 'AS888/890 operations which include references to these functions.

```
FSIN: SUBROUTINE

;  A) Reduce angle range to first quadrant. (0 ≤ x ≤ π/2)

       R1 = FSGN(R0)                      ; Get sign bit (MSB)
       R2 = FABS(R0)                      ; Take absolute value (clear MSB)
       R3 = FMUL(R2,1OVR2PI)              ; Multiply register and constant
       R3 = FINT(R3)                      ; Floating-point integer conversion
       R3 = FMUL(R3,2PI)                  ; Multiply register and constant
       R3 = FADD(R2,INV(R3))              ; Subtract registers by adding inverse
       Y = FADD(R3,NEGPI) : TEST NEG      ; Subtract by adding negative constant
       JT SIN1                            ; Jump if true (jump if negative)
       R1 = FINV(R1)                      ; Complement sign of R1
       R3 = FADD(R3,NEGPI)                ; Subtract by adding negative constant
SIN1:  Y = PIOVR2 - R3 : TEST NEG         ; Subtract to compare (don't store)
       JT SIN2                            ; Jump if true (jump if negative)
       R3 = FADD(PI,FINV(R3))             ; Subtract by adding negative register

;  B) Compute sin(x) ≃ Σ (n=0 to 6) An X^(2n-1)

SIN2:  R4 = FMUL(R0,R0)                   ; Square by multiplying
       R5 = A6                            ; Initialize series
       R5 = FMUL(R4,R5)                   ; Multiply registers
       R5 = FADD(R5,A5)                   ; Add coefficient
       R5 = FMUL(R4,R5)                   ; Multiply registers
       R5 = FADD(R5,A4)                   ; Add coefficient
       R5 = FMUL(R4,R5)                   ; Multiply registers
       R5 = FADD(R5,A3)                   ; Add coefficient
       R5 = FMUL(R4,R5)                   ; Multiply registers
       R5 = FADD(R5,A2)                   ; Add coefficient
       R5 = FMUL(R4,R5)                   ; Multiply registers
       R5 = FADD(R5,A1)                   ; Add coefficient
       R5 = FMUL(R4,R5)                   ; Multiply registers
       R5 = FADD(R5,A0)                   ; Add coefficient
       R5 = FMUL(R0,R5)                   ; Multiply registers
       R5 = FCHS(R5,R1) : RETURN          ; Change MSB of R5 to MSB of R1

END SUBROUTINE
```

This contrived language has a syntax which may be suitable for a source program. For the sake of illustration, it can be assumed that the microassembler recognizes this particular syntax. The series was computed inline instead of using a loop since it is relatively short. If a loop were used, a means of indexing the constants would be required.

5.7 Expand Subroutines into 'AS888/890 Operations

FMUL and FADD algorithms can now be expanded. Since they are called extensively from FSIN, they are more critical to the efficiency of the final design. Wherever possible, it is desirable to reduce the execution time of both in order to maintain efficiency.

5.7.1 Floating-Point Multiplication

Let M1 be the multiplier and M2 be the multiplicand whose product is P. Let the sign, exponent and fraction fields of their IEEE representation be:

```
M1 : |S1|E1|F1|
M2 : |S2|E2|F2|
 P : |S3|E3|F3|
```

P is found by multiplying mantissas (fraction plus implicit one) and adding exponents. Since M1 and M2 are normalized, the range of 1.F1 × 1.F2 is

$$1.00...0 \leq 1.F1 \times 1.F2 \leq 11.1...10$$

The implicit bit may ''overflow'' into bit position 24. This type of overflow must be detected so that the result can be normalized. Normalization requires right shifting the result of 1.F1 × 1.F2 and incrementing E3. The implicit bit is then cleared when S3, E3 and M3 are packed to form P. The floating-point multiplication algorithm may then be defined as follows:

1) Unpack M1 into signed fraction (SF1) and exponent (E1)
2) Set the implicit bit in SF1
3) Unpack M2 into signed fraction (SF2) and exponent (E2)
4) Set the implicit bit in SF2
5) Perform SF3 = SF1 × SF2 using signed integer multiplication
6) Perform E3 = E1 + E2
7) Test SF3 for overflow into bit 24
8) If true, then increment E3 and right shift SF3
9) Clear the implicit bit in SF3
10) Pack E3 and SF3 to get P

As before, the steps of this algorithm can be broken down into simpler operations:

1) Unpack M1 into signed fraction (SF1) and exponent (E1)
   E1 = FEXP(M1)
   SF1 = FRAC(M1)

2) Set the implicit bit in SF1
   SF1 = SF1 OR BIT23

3) Unpack M2 into signed fraction (SF2) and exponent (E2)
   E2 = FEXP (M2)
   SF2 = FRAC (M2)

4) Set the implicit bit in SF2
   SF2 = SF2 OR BIT23

5) Perform SF3 = SF1 × SF2 using signed integer multiplication
   SF3 = IMUL (SF1, SF2)

6) Perform E3 = E1 + E2
   E3 = E1 + E2

7) Test SF3 for overflow into bit 24
   TEST (SF3 AND BIT24)
   JUMP IF FALSE to step 9

8) If true, then increment E3 and right shift SF3
   INC E3
   SF3 = RSHFT (SF3)

9) Clear the implicit bit in SF3.
   SF3 = SF3 AND NOT_BIT23

10) Pack E3 and SF3 to get P
   P = SF3 OR E3

FEXP, FRAC, testing bit 24 and setting/clearing bit 23 are all mask operations that translate into single 'AS888 instructions. The integer multiplication (IMUL) is simply the multiplication algorithm supported by the 'AS888 instruction set. No significant hardware features are required to do floating-point multiplication, nor are any subroutines required to support it.

Register assignments can now be made as before. Since FSIN uses registers in the lower half of the register file, it might be preferable to restrict FMUL to the upper registers. For example:

```
RF = P
RE = M1, F1, SF1
RD = M2, F2, SF2
RC = E1
RB = E2
```

RE and RD can share variables that need not be preserved. Using this assignment, FMUL computes RF = FMUL(RE,RD). RE and RD must be loaded prior to calling FMUL and RF must be stored upon return. By substituting registers for variables and reorganizing operations in the FMUL algorithm to better fit 'AS888/890 operations the following source program may be created:

```
FMUL: SUBROUTINE

      RC = FEXP(RE)                    ; Unpack M1 into exponent
      RE = FRAC(RE)                    ;  and fraction
      RE = RE OR BIT23                 ; Set implicit bit
      MQ = SMTC(RE)                    ; Prepare to multiply

      RB = FEXP(RD)                    ; Unpack M2 into exponent
      RD = FMAG(RD)                    ;  and fraction
      RD = RD OR BIT23                 ; Set implicit bit
      RD = SMTC(RD)                    ; Prepare to multiply

      RE = 0 : RCA = #22d              ; Initialize to multiply
      RE = SMULI RD : LOOP RCA         ; Integer multiplication iteration
      RE = SMULT RD                    ; Final step in signed multiply
      Y= TB0(RE,BIT1):BYTE= #0100b:TEST Z ; Test ''overflow''
      JF FMUL1                         ; Jump if false (exponent ok)

      INEX(RC)                         ; Increment exponent: add 00800000
      RE = SRA(RE)                     ; Shift fraction to normalize

FMUL1:RC = RC + RB : TEST CARRY        ; Add exponents and test carry
      JT ERROR                         ; Jump if carry true to handler

      RE = SMTC(RE)                    ; Get sign magnitude fraction
      RE = RE AND #807F_FFFFh          ; Clear implicit bit
      RF = RE OR RC : RETURN           ; Pack fraction and exponent
```

5.7.2 Floating-Point Addition

The floating-point addition algorithm (FADD) is slightly more complex than FMUL, since the two addends will usually not have the same exponent. Therefore the smaller (absolute value) addend must first be chosen by comparing exponents. Then it must be denormalized to align its digits with the digits of the larger addend. In other words, the two addends must have the same exponent before their fractions can be added. This process can be described by the following algorithm:

1) Unpack A1 to get SF1 and E1
2) Set implicit bit in SF1
3) Unpack A2 to get SF2 and E2
4) Set implicit bit in SF2
5) If E2 > E1 then go to step 9
   ($\|A1\| \leq \|A2\|$)
6) Let DIFF = E1 − E2
7) Do i = 1 to DIFF
     SF2 = RSHFT(SF2) (Arithmetic right shift)
   Enddo
8) Let E3 = E1 , go to step 12
   ($\|A2\| > \|A1\|$)
9) Let DIFF = E2 − E1
10) Do i = 1 to DIFF
     SF1 = RSHFT(SF1) (Arithmetic right shift)
    Enddo
11) Let E3 = E2
12) SF3 = SF1 + SF2
13) Test "overflow" into bit 24
14) Jump if false to step 17
15) Increment exponent E3
16) Normalize signed fraction with right arithmetic shift
17) Clear implicit bit
18) Pack: SUM = SF3 or E3
19) Return

Register assignments for variables must now be made. Since FSIN uses registers in the lower half of the 'AS888 register file, it is necessary to use the upper registers:

RF = SUM
RE = A1, F1, SF1
RD = A2, F2, SF2
RC = E1
RB = E2

By slightly reorganizing the sequence to better fit 'AS888/890 operations, the following microprogram to perform FADD can be created:

```
FADD: SUBROUTINE

; 1) Unpack A1 to get SF1 and E1
        RC = FEXP(RE)              ; Get exponent (E1)
        RE = FRAC(RE)              ; Get signed fraction (SF1)

; 2) Set implicit bit in SF1
        MQ = RE OR BIT23           ; Set implicit bit
        RE = SMTC(RE)              ; Convert to two's complement
```

```
; 3) Unpack A2 to get SF2 and A2
        RB = FEXP(RD)                    ; Get exponent (E2)
        RD = FRAC(RD)                    ; Get signed fraction (SF2)

; 4) Set implicit bit in SF2
        RD = RD OR BIT23                 ; Set implicit bit
        RD = SMTC(RD)                    ; Convert to two's complement

; 5) If E2 > E1 then go to step 9
        RF = RC - RB : TEST NEGATIVE     ; Compare A2 from A1
        JT FADD1 : RCA = #8              ; Jump if E2 > E1; set up loop count

; 6) Let DIFF = E1 - E2.
        Y/RF = SLC(RF) : LOOP RCA        ; Rotate 8 times to get difference
        RCA = Y/RF                       ; Load difference in loop counter

; 7) Do i = 1 to DIFF
        SF2 = RSHFT(SF2)
    Enddo
        RD = SRA(RD) : LOOP RCA          ; Orient digits of smaller addend

; 8) Let E3 = E1, go to step 12
        RB = RC : JUMP FADD2             ; Swap registers and branch

; 9) Let DIFF = E2 - E1

FADD1: RF = NOT(RF)                      ; Complement result of E1 - E2
        Y/RF = SLC(RF) : LOOP RCA        ; Shift 8 times to get DIFF
        RCA = Y/RF                       ; Load DIFF in loop counter

;10) Do i = 1 TO DIFF
        SF1 = RSHFT(SF1)
    Enddo
        RE = SRA(RE) : LOOP RCA          ; Align SF1 with SF2

;11) Let E3 = E2 (no instruction required — RB already has E2 in it)

;12) SF3 = SF1 + SF2

FADD2: RF = RD + RE                      ; Add
        RF = SMTC(RF)                    ; Convert to sign-magnitude

;13) Test "overflow" into bit 24
        RF = TBO (RF, BIT24)             ; Check for normalization

;14) Jump if false to step 17
        JF FADD3                         ; If so, finish and exit

;15) Else increment exponent
        INC RB : TEST NEG                ; Test for exponent overflow

;16) Normalize signed fraction
        RF = SRA(RF) : JT ERROR          ; Jump to error handler if overflow

;17) Clear implicit bit

FADD3: RF = SETO (RF, BIT23)             ; Reset bit 23 of RF

;18) Pack: SUM = SF3 OR E3
        RF = RF OR RB : RETURN           ; Or signed fraction and exponent
```

There is an important consequence of FADD which impacts the hardware. Since the number of shifts required to denormalize the small addend is data dependent (computed in the ALU) it is necessary to provide a path between the ALU Y bus and the 'AS890 DRA bus. All the other operations are simple 'AS888/890 instructions, including the FRAC and FEXP mask operations discussed during the development of FMUL. ERROR is a floating-point overflow error handler.

5.8 Evaluate Tradeoffs and Block Diagram the Hardware

A rough estimate of the FSIN worst case execution time can be arrived at by making the following observations about FSIN, FMUL and FADD:

FMUL
  integer recursion $\simeq$ 22 cycles
  other instructions $\simeq$ 18 cycles
  total $\simeq$ 40 cycles

FADD
  denormalization $\simeq$ 23 cycles
  other instructions $\simeq$ 25 cycles
  total $\simeq$ 50 cycles

FSIN
  number of calls to FMUL = 12
  number of calls to FADD = 11
  number of other cycles $\simeq$ 10

Approximate worst case total = 10 + (12 × 40) + (11 × 50) = 1040 cycles. At 50 nanoseconds per cycle, this requires approximately 52 microseconds. There are few improvements that could be made in hardware to speed this time, except perhaps the addition of a flash multiplier which would reduce the integer computation by about 20 cycles (an overall reduction of about two percent). A barrel shifter could have the same benefit during floating-point addition for a total reduction of about 4 percent. For the sake of simplicity, it will be assumed that 52 microseconds is acceptable for the sin(x) computation.

Another issue which must be considered is the problem of loading the 'AS888 and 'AS890 with constants. A slight materials cost reduction might be realized by storing constants in table PROMs rather than in control store memory. An interesting use of the DRA and DRB ports on the 'AS890 would be to use the output of RCA or RCB to index data in the constant PROM. This would allow long series to be implemented in loop form rather than the inline method used in FSIN. Once again, the constant PROM will not be implemented for the sake of simplicity.

Now the architecture can be designed to meet the requirements identified throughout this analysis:

1) A path between the 'AS888 Y bus and the 'AS890 DRA bus.
2) A path between the microinstruction register and the 'AS890 DRA bus for loading loop counts and branch addresses.
3) A path between the microinstruction register and the 'AS888 Y bus for loading constants.
4) Independent control of $\overline{\text{SIO0}}$ in each 'AS888 slice to allow bit/byte instructions.
5) A status register to store 'AS888 status for testing.
6) A status mux to test the 'AS888 status, bit 23 of the 'AS888 Y bus, bit 24 of the 'AS888 Y bus and hardwired 0 and 1.

A system having these features is illustrated in Figure 5.1.

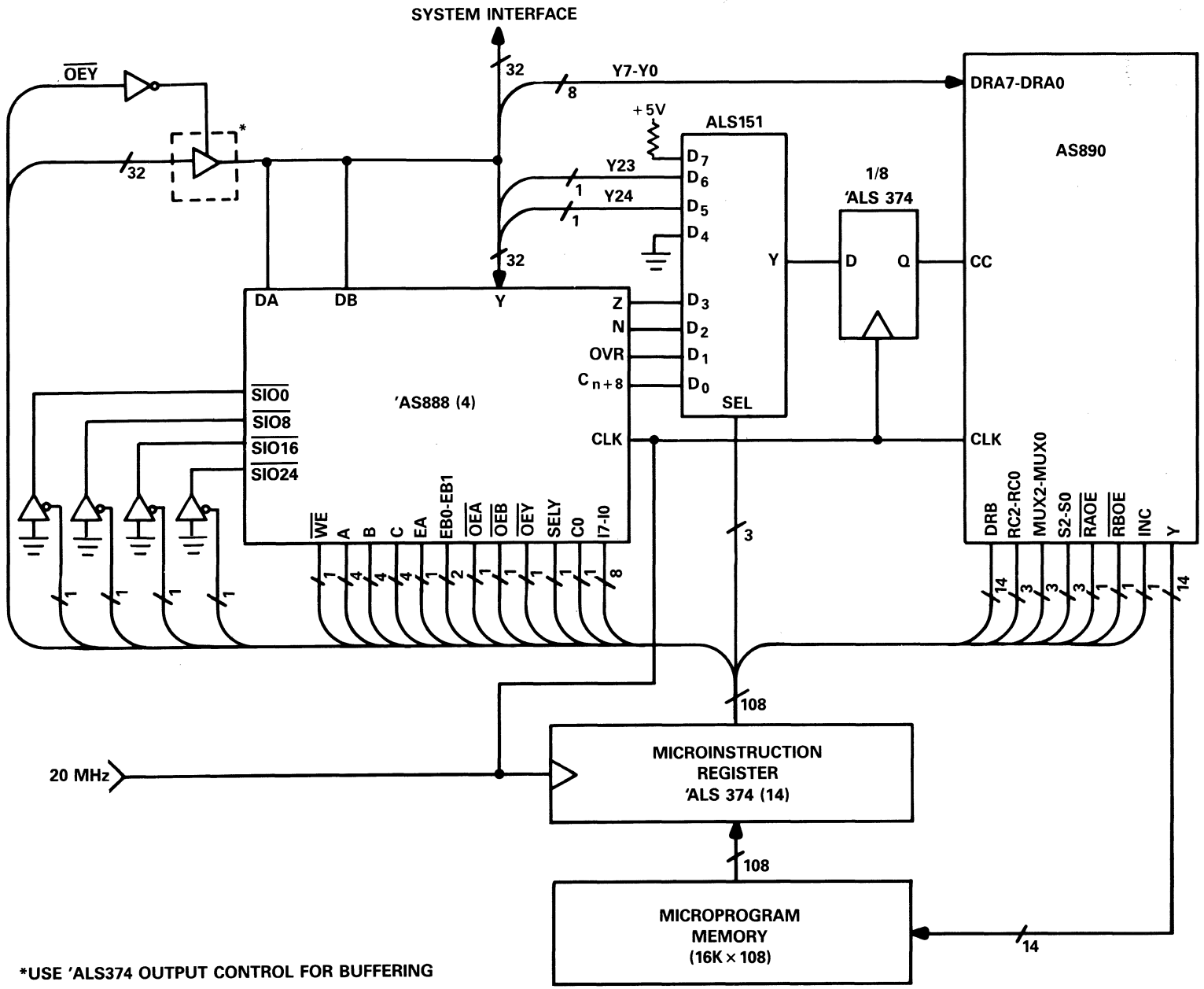


Figure 5-1. Block Diagram of Floating-Point Processor

5.9 Define Microinstruction Fields During Detailed Hardware Design

The detailed hardware design will produce a wiring diagram that fixes the position within the microinstruction of each of the various control signals that are connected from the microinstruction register to the 'AS888, 'AS890, status mux and any other special hardware. Once this design is complete it is possible for the assembler to sort the control bits of each instruction so that they will be properly oriented when the microprogram is installed in the target system.

5.10 Assemble the Microprogram

TI is currently developing an 'AS888/890 microassembler. Several microassemblers are commercially available, and many users prefer to write their own. The microprogram shown in Table 5-1 was hand-assembled, but has a syntax that is suitable for interpretation by a user-written assembler.

Table 5.1. Floating Point Sin(x) Microprogram

		$\overline{WE}$	A3-A0	B3-B0	C3-C0	$\overline{EA}$	EB1-EB0	$\overline{OEA}$	$\overline{OEB}$	OEY	SELY	Cn	I7-I0	32-bit Constant	$\overline{SIO0}$	$\overline{SIO8}$	$\overline{SIO16}$	$\overline{SIO24}$	RC2-RC0	MUX2-MUX0	S2-S0	DRB13-DRB0	$\overline{RAOE}$	$\overline{RBOE}$	$\overline{INC}$	SEL
	* R1 = FSGN(R0)																									
0000 SIN:	R1 = R0 AND #8000 0000h	0	0	X	1	0	2	1	1	1	0	0	F A	8 0 0 0 0 0 0 0	1	1	1	1	0	2	7	X X X X	1	1	1	7
	* R2 = FABS(R0)																									
0001	R2 = R0	0	0	X	2	0	X	1	1	1	0	0	F 6	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	7
0002	R2 = R0 SET0 #80h : BYTE=#1000b	0	0	2	8	0	0	1	1	1	0	0	1 8	X X X X X X X X	1	1	1	0	0	2	7	X X X X	1	1	1	7
	* R3 = FMUL(R2,1OVR2PI)																									
0003	RE = R2	0	2	X	E	0	X	1	1	1	0	0	F 6	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	7
0004	RD = #3EA2 F984h	0	X	X	D	X	X	1	1	1	1	X	F F	3 E A 2 F 9 8 4	1	1	1	1	0	2	7	X X X X	1	1	1	7
0005	JSR FMUL	1	X	X	X	X	X	1	1	1	X	X	F F	X X X X X X X X	1	1	1	1	0	1	4	0 0 6 0	1	1	1	7
0006	R3 = RF	0	F	X	3	0	X	1	1	1	0	0	F 6	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	7
	* R3 = FINT(R3)																									
0007	RF = R3	0	3	X	F	0	X	1	1	1	0	0	F 6	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	7
0008	JSR FINT [EXERCISE FOR READER]	1	X	X	X	X	X	1	1	1	X	X	F F	X X X X X X X X	1	1	1	1	0	1	4	. . . .	1	1	1	7
0009	R3 = RF	0	F	X	3	0	X	1	1	1	0	0	F 6	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	7
	* R3 = FMUL(R3,2PI)																									
000A	RE = R3	0	3	X	E	0	X	1	1	1	0	0	F 6	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	7
000B	RD = #40C9 0FDBh	0	X	X	D	X	X	1	1	1	0	X	F F	4 0 C 9 0 F D B	1	1	1	1	0	2	7	X X X X	1	1	1	7
000C	JSR FMUL	1	X	X	X	X	X	1	1	1	X	X	F F	X X X X X X X X	1	1	1	1	0	1	4	0 0 6 0	1	1	1	7
000D	R3 = RF	0	F	X	3	0	X	1	1	1	0	0	F 6	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	7
	* R3 = FADD(R2,INV(R3))																									
000E	RE = R2	0	2	X	E	0	X	1	1	1	0	0	F 6	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	7
000F	RD = R5 XOR #8000 0000h	0	5	X	D	0	2	1	1	1	0	0	F 9	8 0 0 0 0 0 0 0	1	1	1	1	0	2	7	X X X X	1	1	1	7
0010	JSR FADD	1	X	X	X	X	X	1	1	1	X	X	F F	X X X X X X X X	1	1	1	1	0	1	4	0 0 7 4	1	1	1	7
0011	R3 = RF	0	F	X	3	0	X	1	1	1	0	0	F 6	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	7

Table 5.1. Floating Point Sin(x) Microprogram (continued)

			$\overline{WE}$	A3-A0	B3-B0	C3-C0	$\overline{EA}$	EB1-EB0	$\overline{OEA}$	$\overline{OEB}$	OEY	SELY	Cn	I7-I0	32-bit Constant	$\overline{SIO0}$	$\overline{SIO8}$	$\overline{SIO16}$	$\overline{SIO24}$	RC2-RC0	MUX2-MUX0	S2-S0	DRB13-DRB0	$\overline{RAOE}$	$\overline{RBOE}$	$\overline{INC}$	SEL
		* Y = FADD(R2,NEGPI)																									
0012		RE = R2	0	2	X	E	0	X	1	1	1	0	0	F	6	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	7
0013		RD = #C059 0FDBh	0	X	X	D	X	X	1	1	1	0	X	F	F	C 0 5 9 0 F D B	1	1	1	1	0	2	7	X X X X	1	1	1	7
0014		JSR FADD	1	X	X	X	X	X	1	1	1	X	X	F	F	X X X X X X X X	1	1	1	1	0	1	4	0 0 7 4	1	1	1	7
0015		Y = RF : TEST NEG	1	F	X	X	0	X	1	1	0	0	0	F	6	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	2
0016		JT SIN1	1	X	X	X	X	X	1	1	1	X	X	F	F	X X X X X X X X	1	1	1	1	0	1	7	0 0 0 0	1	1	1	7
		* R1 = FINV(R1)																									
0017		R1 = R1 XOR #8000 0000h	0	1	X	1	0	2	1	1	1	0	0	F	9	8 0 0 0 0 0 0 0	1	1	1	1	0	2	7	X X X X	1	1	1	7
		* R3 = FADD(R3,NEGPI)																									
0018		RE = R3	0	3	X	E	0	X	1	1	1	0	0	F	6	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	7
0019		RD = #C059 0FDBh	0	3	X	D	X	X	1	1	1	1	X	F	F	C 0 5 9 0 F D B	1	1	1	1	0	2	7	X X X X	1	1	1	7
001A		JSR FADD	1	X	X	X	X	X	1	1	1	X	X	F	F	X X X X X X X X	1	1	1	1	0	1	4	0 0 7 4	1	1	1	7
001B		R3 = RF	0	F	X	3	0	X	1	1	1	0	0	F	6	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	7
	* SIN1:	Y = FADD (PIOVR2,INV(R3)) : TEST NEG																									
001C	SIN1:	RE = #3FC9 0FDBh	0	X	X	E	X	X	1	1	1	1	0	F	6	3 5 C 9 0 F D B	1	1	1	1	0	2	7	X X X X	1	1	1	7
001D		RD = R3 XOR #8000 0000h	0	3	X	D	0	2	1	1	1	0	0	F	9	8 0 0 0 0 0 0 0	1	1	1	1	0	2	7	X X X X	1	1	1	7
001E		JSR FADD	1	X	X	X	X	X	1	1	1	X	X	F	F	X X X X X X X X	1	1	1	1	0	1	4	0 0 7 4	1	1	1	7
001F		Y = RF : TEST NEG	1	F	X	X	0	X	1	1	0	0	0	F	6	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	2
0020		JT SIN2	1	X	X	X	X	X	1	1	1	X	X	F	F	X X X X X X X X	1	1	1	1	0	1	7	0 0 0 0	1	1	1	7
		* R3 = FADD(PI,FINV(R3))																									
0021		RE = #4059 0FDBh	0	X	X	E	X	X	1	1	1	1	0	F	6	4 0 5 9 0 F D B	1	1	1	1	0	2	7	X X X X	1	1	1	7
0022		RD = R3 XOR #8000 0000h	0	3	X	D	0	2	1	1	1	0	0	F	9	8 0 0 0 0 0 0 0	1	1	1	1	0	2	7	X X X X	1	1	1	7
0023		JSR FADD	1	X	X	X	X	X	1	1	1	X	X	F	F	X X X X X X X X	1	1	1	1	0	1	4	0 0 7 4	1	1	1	7
0024		R3 = RF	0	F	X	3	0	X	1	1	1	0	0	F	6	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	7
	* SIN2:	R4 = FMUL(R0,R0)																									
0025	SIN2:	RE = R0	0	0	X	E	0	X	1	1	1	0	0	F	6	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	7
0026		RD = R0	0	0	X	D	0	X	1	1	1	0	0	F	6	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	7
0027		JSR FMUL	1	X	X	X	X	X	1	1	1	X	X	F	F	X X X X X X X X	1	1	1	1	0	1	4	0 0 6 0	1	1	1	7
0028		R4 = RF	0	F	X	4	0	X	1	1	1	0	0	F	6	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	7

Table 5.1. Floating Point Sin(x) Microprogram (continued)

		WE	A3-A0	B3-B0	C3-C0	EA	EB1-EB0	OEA	OEB	OEY	SELY	Cn	I7-I0	32-bit Constant	SIO0	SIO8	SIO16	SIO24	RC2-RC0	MUX2-MUX0	S2-S0	DRB13-DRB0	RAOE	RBOE	INC	SEL
	* R5 = A6																									
0029	R5 = #2F34 6FBCh	0	X	X	5	0	X	1	1	1	1	0	F	6	2 F 3 4 6 F B C	1	1	1	1	0	2	7	X X X X	1	1	1	7
	* R5 = FMUL(R4,R5)																									
002A	RE = R4	0	4	X	E	0	X	1	1	1	0	0	F	6	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	7
002B	RD = R5	0	5	X	D	0	X	1	1	1	0	0	F	6	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	7
002C	JSR FMUL	1	X	X	X	X	X	1	1	1	X	X	F	F	X X X X X X X X	1	1	1	1	0	1	4	0 0 6 0	1	1	1	7
002D	R5 = RF	0	F	X	5	0	X	1	1	1	0	0	F	6	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	7
	* R5 = FADD(R5,A5)																									
002E	RE = R5	0	5	X	E	0	X	1	1	1	0	0	F	6	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	7
002F	RD = #B2D7 5AD5h	0	X	X	D	X	X	1	1	1	1	X	F	F	B 2 D 7 5 A D 5	1	1	1	1	0	2	7	X X X X	1	1	1	7
0030	JSR FADD	1	X	X	X	X	X	1	1	1	X	X	F	F	X X X X X X X X	1	1	1	1	0	1	4	0 0 7 4	1	1	1	7
0031	R5 = RF	0	F	X	5	0	X	1	1	1	0	0	F	6	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	7
	* R5 = FMUL(R4,R5)																									
0032	RE = R4	0	4	X	E	0	X	1	1	1	0	0	F	6	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	7
0033	RD = R5	0	5	X	D	0	X	1	1	1	0	0	F	6	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	7
0034	JSR FMUL	1	X	X	X	X	X	1	1	1	X	X	F	F	X X X X X X X X	1	1	1	1	0	1	4	0 0 6 0	1	1	1	7
0035	R5 = RF	0	F	X	5	0	X	1	1	1	0	0	F	6	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	7
	* R5 = FADD(R5,A4)																									
0036	RE = R5	0	5	X	E	0	X	1	1	1	0	0	F	6	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	7
0037	RD = #3638 EF99h	0	X	X	D	X	X	1	1	1	1	X	F	F	3 6 3 8 E F 9 9	1	1	1	1	0	2	7	X X X X	1	1	1	7
0038	JSR FADD	1	X	X	X	X	X	1	1	1	X	X	F	F	X X X X X X X X	1	1	1	1	0	1	4	0 0 7 4	1	1	1	7
0039	R5 = RF	0	F	X	5	0	X	1	1	1	0	0	F	6	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	7
	* R5 = FMUL(R4,R5)																									
003A	RE = R4	0	4	X	E	0	X	1	1	1	0	0	F	6	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	7
003B	RD = R5	0	5	X	D	0	X	1	1	1	0	0	F	6	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	7
003C	JSR FMUL	1	X	X	X	X	X	1	1	1	X	X	F	F	X X X X X X X X	1	1	1	1	0	1	4	0 0 6 0	1	1	1	7
003D	R5 = RF	0	F	X	5	0	X	1	1	1	0	0	F	6	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	7

Table 5.1. Floating Point Sin(x) Microprogram (continued)

		$\overline{WE}$	A3-A0	B3-B0	C3-C0	$\overline{EA}$	EB1-EB0	$\overline{OEA}$	$\overline{OEB}$	OEY	SELY	Cn	I7-I0	32-bit Constant	$\overline{SIO0}$	$\overline{SIO8}$	$\overline{SIO16}$	$\overline{SIO24}$	RC2-RC0	MUX2-MUX0	S2-S0	DRB13-DRB0	$\overline{RAOE}$	$\overline{RBOE}$	$\overline{INC}$	SEL
	* R5 = FADD(R5,A3)																									
003E	RE = R5	0	5	X	E	0	X	1	1	1	0	0	F	6	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	7
003F	RD = #B950 0D01h	0	X	X	D	X	X	1	1	1	1	X	F	F	B 9 5 0 0 D 0 1	1	1	1	1	0	2	7	X X X X	1	1	1	7
0040	JSR FADD	1	X	X	X	X	X	1	1	1	X	X	F	F	X X X X X X X X	1	1	1	1	0	1	4	0 0 7 4	1	1	1	7
0041	R5 = RF	0	F	X	5	0	X	1	1	1	0	0	F	6	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	7
	* R5 = FMUL(R4,R5)																										
0042	RE = R4	0	4	X	E	0	X	1	1	1	0	0	F	6	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	7
0043	RD = R5	0	5	X	D	0	X	1	1	1	0	0	F	6	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	7
0044	JSR FMUL	1	X	X	X	X	X	1	1	1	X	X	F	F	X X X X X X X X	1	1	1	1	0	1	4	0 0 6 0	1	1	1	7
0045	R5 = RF	0	F	X	5	0	X	1	1	1	0	0	F	6	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	7
	* R5 = FADD(R5,A2)																										
0046	RE = R5	0	5	X	E	0	X	1	1	1	0	0	F	6	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	7
0047	RD = #3C08 8888h	0	X	X	D	X	X	1	1	1	1	X	F	F	3 C 0 8 8 8 8 8	1	1	1	1	0	2	7	X X X X	1	1	1	7
0048	JSR FADD	1	X	X	X	X	X	1	1	1	X	X	F	F	X X X X X X X X	1	1	1	1	0	1	4	0 0 7 4	1	1	1	7
0049	R5 = RF	0	F	X	5	0	X	1	1	1	0	0	F	6	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	7
	* R5 = FMUL(R4,R5)																										
004A	RE = R4	0	4	X	E	0	X	1	1	1	0	0	F	6	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	7
004B	RD = R5	0	5	X	D	0	X	1	1	1	0	0	F	6	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	7
004C	JSR FMUL	1	X	X	X	X	X	1	1	1	X	X	F	F	X X X X X X X X	1	1	1	1	0	1	4	0 0 6 0	1	1	1	7
004D	R5 = RF	0	F	X	5	0	X	1	1	1	0	0	F	6	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	7
	* R5 = FADD(R5,A1)																										
004E	RE = R5	0	5	X	E	0	X	1	1	1	0	0	F	6	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	7
004F	RD = #BE2A AAADh	0	X	X	D	X	X	1	1	1	1	1	F	F	B E 2 A A A A D	1	1	1	1	0	2	7	X X X X	1	1	1	7
0050	JSR FADD	1	X	X	X	X	X	1	1	1	X	X	F	F	X X X X X X X X	1	1	1	1	0	1	4	0 0 7 4	1	1	1	7
0051	R5 = RF	0	F	X	5	0	X	1	1	1	0	0	F	6	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	7
	* R5 = FMUL(R4,R5)																										
0052	RE = R4	0	4	X	E	0	X	1	1	1	0	0	F	6	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	7
0053	RD = R5	0	5	X	D	0	X	1	1	1	0	0	F	6	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	7
0054	JSR FMUL	1	X	X	X	X	X	1	1	1	X	X	F	F	X X X X X X X X	1	1	1	1	0	1	4	0 0 6 0	1	1	1	7
0055	R5 = RF	0	F	X	5	0	X	1	1	1	0	0	F	6	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	7

Table 5.1. Floating Point Sin(x) Microprogram (continued)

		$\overline{WE}$	A3-A0	B3-B0	C3-C0	$\overline{EA}$	EB1-EB0	$\overline{OEA}$	$\overline{OEB}$	OEY	SELY	Cn	I7-I0	32-bit Constant	$\overline{SIO0}$	$\overline{SIO8}$	$\overline{SIO16}$	$\overline{SIO24}$	RC2-RC0	MUX2-MUX0	S2-S0	DRB13-DRB0	$\overline{RAOE}$	$\overline{RBOE}$	$\overline{INC}$	SEL
	* R5 = FADD(R5,A0)																									
0056	RE = R5	0	5	X	E	0	X	1	1	1	0	0	F	6	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	7
0057	RD = #3F80 0000h	0	X	X	D	X	X	1	1	1	1	X	F	F	3 F 8 0 0 0 0 0	1	1	1	1	0	2	7	X X X X	1	1	1	7
0058	JSR FADD	1	X	X	X	X	X	1	1	1	X	X	F	F	X X X X X X X X	1	1	1	1	0	1	4	0 0 7 4	1	1	1	7
0059	R5 = RF	0	F	X	5	0	X	1	1	1	0	0	F	6	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	7
	* R5 = FMUL(R0,R5)																										
005A	RE = R0	0	0	X	E	0	X	1	1	1	0	0	F	6	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	7
005B	RD = R5	0	5	X	D	0	X	1	1	1	0	0	F	6	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	7
005C	JSR FMUL	1	X	X	X	X	X	1	1	1	X	X	F	F	X X X X X X X X	1	1	1	1	0	1	4	0 0 6 0	1	1	1	7
005D	R5 = RF	0	F	X	5	0	X	1	1	1	0	0	F	6	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	7
	* R5 = FCHS(R5,R1) : RETURN																										
005E	R1 = R1 OR #7FFF FFFFh	0	1	X	1	0	2	1	1	1	0	0	F	B	B 7 F F F F F F	1	1	1	1	0	2	7	X X X X	1	1	1	7
005F	R5 = R5 XOR R1 : RETURN	0	5	1	5	0	0	1	1	1	0	0	F	0	X X X X X X X X	1	1	1	1	0	2	2	X X X X	1	1	1	7
	* RC = FEXP(RE)																										
0060 FMUL:	RC = RE AND #7F80 0000h	0	E	X	C	0	2	1	1	1	0	0	F	A	7 F 8 0 0 0 0 0	1	1	1	1	0	2	7	X X X X	1	1	1	7
	* RE = FRAC(RE)																										
0061	RE = RE AND #807F FFFFh	0	E	X	E	0	2	1	1	1	0	0	F	A	8 0 7 F F F F F	1	1	1	1	0	2	7	X X X X	1	1	1	7
	* RE = RE OR bit23																										
0062	RE = RE OR #0080 0000h	0	E	X	F	0	2	1	1	1	0	0	F	B	0 0 8 0 0 0 0 0	1	1	1	1	0	2	7	X X X X	1	1	1	7
	* MQ = SMTC(RE)																										
0063	RE = SMTC(RE)	0	X	E	E	X	0	1	1	1	1	0	5	8	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	7
0064	LOADMQ : PASS	1	E	X	X	0	X	1	1	1	1	0	E	6	X X X X X X X X	1	1	1	1	0	2	7	X X X X	1	1	1	7
	* RB = FEXP(RD)																										
0065	RB = RD AND #7F80 0000h	0	D	X	B	0	2	1	1	1	0	0	F	A	7 F 8 0 0 0 0 0	1	1	1	1	0	2	7	X X X X	1	1	1	7

Table 5.1. Floating Point Sin(x) Microprogram (continued)

		WE	A3-A0	B3-B0	C3-C0	EA	EB1-EB0	OEA	OEB	OEY	SELY	Cn	I7-I0	32-bit Constant	SIO0	SIO8	SIO16	SIO24	RC2-RC0	MUX2-MUX0	S2-S0	DRB13-DRB0	RAOE	RBOE	INC	SEL
	* RD = FRAC(RD)																									
0066	RD = RD AND #807F FFFFh	0	D	X	D	0	2	1	1	1	0	0	F A	8 0 7 F F F F F	1	1	1	1	0	2	7	X X X	1	1	1	7
0067	RD = RD OR bit23	0	D	X	D	0	2	1	1	1	0	0	F B	0 0 8 0 0 0 0 0	1	1	1	1	0	2	7	X X X	1	1	1	7
0068	RD = SMTC(RD)	0	X	D	D	X	0	1	1	1	1	D	5 8	X X X X X X X X	1	1	1	1	0	2	7	X X X	1	1	1	7
0069	RE = 0 : RCB = #22D	0	E	E	E	0	0	1	1	1	1	0	F 9	0 0 0 0 0 0 1 6	1	1	1	1	6	1	7	X X X	1	1	1	4
006A	RE = SMULI RD : LOOP RCB	0	D	E	E	0	0	1	1	1	1	0	6 0	X X X X X X X X	1	1	1	1	5	6	7	0 0 6 A	1	1	1	4
006B	RE = SMULT RD	0	D	E	E	0	0	1	1	1	0	0	7 0	X X X X X X X X	1	1	1	1	0	6	7	X X X	1	1	1	7
006C	TB0(RE,bit1) : BYTE=#0100b : TEST Z	0	0	F	0	0	0	1	1	1	1	0	3 8	X X X X X X X X	1	0	1	1	0	2	7	X X X	1	1	1	4
006D	JT FMUL1	1	X	X	X	X	X	X	1	1	1	X	F F	X X X X X X X X	1	1	1	1	0	1	7	X X X	1	1	1	7
	* INEX RC																									
006E	RC = RC ADD #0080 0000h	0	C	X	C	0	2	1	1	1	0	0	F 1	0 0 8 0 0 0 0 0	1	1	1	1	0	2	7	X X X	1	1	1	7
006F	RE = SRA(RE)	0	E	X	E	0	X	1	1	1	0	0	0 6	X X X X X X X X	1	1	1	1	0	2	7	X X X	1	1	1	7
0070 FMUL1:	RC = RC ADD RB : TEST CARRY	0	C	B	C	0	0	1	1	1	0	0	F 1	X X X X X X X X	1	1	1	1	0	2	7	X X X	1	1	1	0
0071	JT ERROR	1	X	X	X	X	X	1	1	1	X	X	F F	X X X X X X X X	1	1	1	1	0	1	7	X X X	1	1	1	7
0072	RE = SMTC(RE)	0	X	E	E	X	0	1	1	1	1	0	5 8	X X X X X X X X	1	1	1	1	0	2	7	X X X	1	1	1	7
0073	RE = RE AND #807F FFFFh	0	E	X	E	0	2	1	1	1	0	0	F A	8 0 7 F F F F F	1	1	1	1	0	2	7	X X X	1	1	1	7
* FADD:	RC = FEXP(RE)																									
0074 FADD:	RC = RC AND #7F80 0000	0	C	X	C	0	2	1	1	1	0	0	F A	7 F 8 0 0 0 0 0	1	1	1	1	0	2	7	X X X	1	1	1	7
	* RE = FRAC(RE)																									
0075	RE = RE AND #807F FFFFh	0	E	X	E	0	2	1	1	1	0	0	F A	8 0 7 F F F F F	1	1	1	1	0	2	7	X X X	1	1	1	7
0076	MQ = RE OR bit23	1	E	X	X	0	1	1	1	1	0	0	E B	0 0 8 0 0 0 0 0	1	1	1	1	0	2	7	X X X	1	1	1	7
0077	RE = SMTC(RE)	0	E	X	E	0	2	1	1	1	0	0	F A	8 0 7 F F F F F	1	1	1	1	0	2	7	X X X	1	1	1	7
	* RB = FEXP(RD)																									
0078	RB = RD AND #7F80 0000	0	D	X	B	0	2	1	1	1	0	0	F A	7 F 8 0 0 0 0 0	1	1	1	1	0	2	7	X X X	1	1	1	7

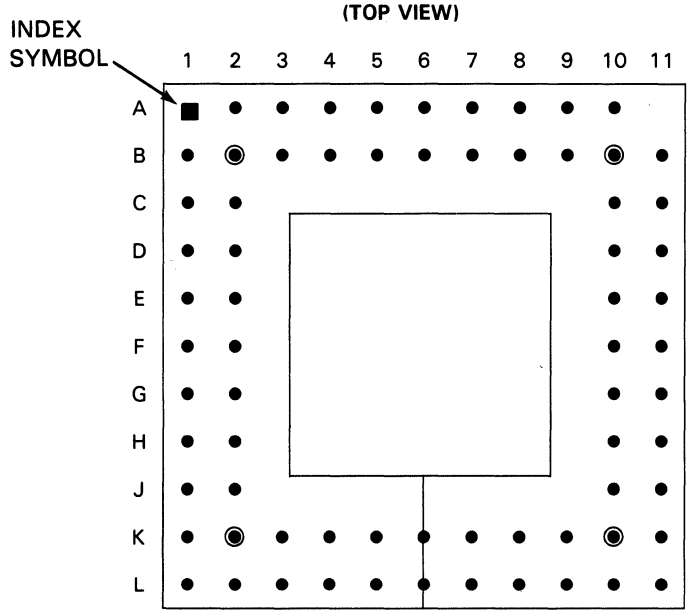
Table 5.1. Floating Point Sin(x) Microprogram (continued)

		$\overline{WE}$	A3-A0	B3-B0	C3-C0	$\overline{EA}$	EB1-EB0	$\overline{OEA}$	$\overline{OEB}$	OEY	SELY	Cn	I7-I0	32-bit Constant	$\overline{SIO0}$	$\overline{SIO8}$	$\overline{SIO16}$	$\overline{SIO24}$	RC2-RC0	MUX2-MUX0	S2-S0	DRB13-DRB0	$\overline{RAOE}$	$\overline{RBOE}$	$\overline{INC}$	SEL
	* RD = FRAC(RD)																									
0079	RE = RE AND #807F FFFFh	0	E	X	E	0	2	1	1	1	0	0	F	A	8	0	7	F	F	F	F	F	1	1	1	1	0	2	7	X	X	X	X	1	1	1	7
007A	RD = RD OR bit23	0	D	X	D	0	2	1	1	1	0	0	F	B	0	0	8	0	0	0	0	0	1	1	1	1	0	2	7	X	X	X	X	1	1	1	7
007B	RD = SMTC(RD)	0	X	D	D	X	0	1	1	1	1	0	5	8	X	X	X	X	X	X	X	X	1	1	1	1	0	2	7	X	X	X	X	1	1	1	7
007C	RF = RC – RB : C0=0: TEST NEG	0	C	B	F	0	0	1	1	1	0	0	F	3	X	X	X	X	X	X	X	X	1	1	1	1	0	2	7	X	X	X	X	1	1	1	2
007D	JT FADD1 : RCB = #8	1	X	X	X	X	X	1	1	1	X	X	F	F	X	X	X	X	X	X	X	8	1	1	1	1	6	1	7	X	X	X	X	1	1	1	4
007E	Y/RF = SLC(RF) : LOOP RCB	0	F	X	F	0	X	1	1	0	X	0	6	6	X	X	X	X	X	X	X	X	1	1	1	1	5	6	7	0	0	7	E	1	1	1	4
007F	Y = RF : RCA = Y	1	X	X	X	X	X	1	1	0	0	X	F	F	X	X	X	X	X	X	X	X	1	1	1	1	2	7	7	0	0	8	0	1	1	1	7
0080	RD = SRA(RD) : LOOP RCA	0	D	X	D	0	X	1	1	1	0	0	0	6	X	X	X	X	X	X	X	X	1	1	1	1	1	6	7	0	0	7	E	1	1	1	4
0081	RB = RC : JUMP FADD2	0	C	X	B	0	X	1	1	1	0	X	F	F	X	X	X	X	X	X	X	X	1	1	1	1	0	2	7	X	X	X	X	1	1	1	7
0082 FADD1:	RF = NOT RF	0	F	X	F	0	X	1	1	1	0	0	F	7	X	X	X	X	X	X	X	X	1	1	1	1	0	2	7	X	X	X	X	1	1	1	7
0083	Y/RF = SLC(RF) : LOOP RCB	0	F	X	F	0	X	1	1	0	0	0	6	6	X	X	X	X	X	X	X	X	1	1	1	1	6	1	7	X	X	X	X	1	1	1	4
0084	Y = RF : RCA = Y	1	X	X	X	X	X	1	1	0	0	X	F	F	X	X	X	X	X	X	X	X	1	1	1	1	5	6	7	0	0	8	4	1	1	1	4
0085	RE = SRA(RE) : LOOP RCA	0	E	X	E	0	X	1	1	1	0	0	0	6	X	X	X	X	X	X	X	X	1	1	1	1	2	7	7	0	0	8	6	1	1	1	7
0086 FADD2:	RF = RD + RE	0	D	E	F	0	0	1	1	1	0	0	F	1	X	X	X	X	X	X	X	X	1	1	1	1	0	2	7	X	X	X	X	1	1	1	7
0087	RF = SMTC(RF)	0	X	F	F	X	0	1	1	1	1	0	5	8	X	X	X	X	X	X	X	X	1	1	1	1	0	2	7	X	X	X	X	1	1	1	7
0088	RF = TB0 (RF, bit24) : TEST Z	0	0	F	0	0	0	1	1	1	1	0	3	8	X	X	X	X	X	X	X	X	0	1	1	1	0	2	7	X	X	X	X	1	1	1	4
0089	JF FADD3	1	X	X	X	X	X	1	1	1	X	X	F	F	X	X	X	X	X	X	X	X	1	1	1	1	0	4	7	X	X	X	X	1	1	1	7
008A	INC RB : TEST NEG	0	B	X	B	0	X	1	1	1	1	1	F	6	X	X	X	X	X	X	X	X	1	1	1	1	0	2	7	X	X	X	X	1	1	1	3
008B	RF = SRA(RF) : JT ERROR	0	F	7	F	0	X	1	1	1	0	0	0	6	X	X	X	X	X	X	X	X	1	1	1	1	0	1	7	X	X	X	X	1	1	1	7
008C FADD3:	RF = SET0 (RF, bit23)	0	7	F	0	0	0	1	1	1	1	0	1	8	X	X	X	X	X	X	X	X	1	0	1	1	0	2	7	X	X	X	X	1	1	1	7
008D	RF = RF OR RB : RETURN	0	F	B	F	0	0	1	1	1	0	0	F	B	X	X	X	X	X	X	X	X	1	1	1	1	0	2	2	X	X	X	X	1	1	1	7

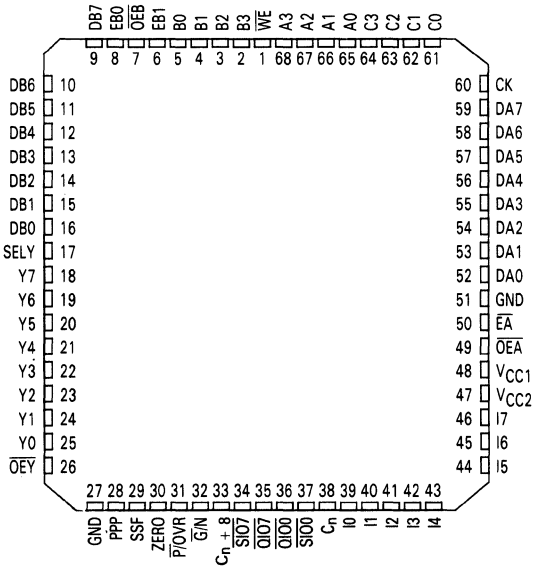
A 'AS888 and 'AS890 Pin Descriptions

Pin descriptions and assignments for the 'AS888 bit-slice processor and 'AS890 microsequencer are given on the following pages.

SN54AS888, SN74AS888 . . . GB PACKAGE
(TOP VIEW)



SN54AS888 . . . FK PACKAGE
SN74AS888 . . . FN PACKAGE
(TOP VIEW)



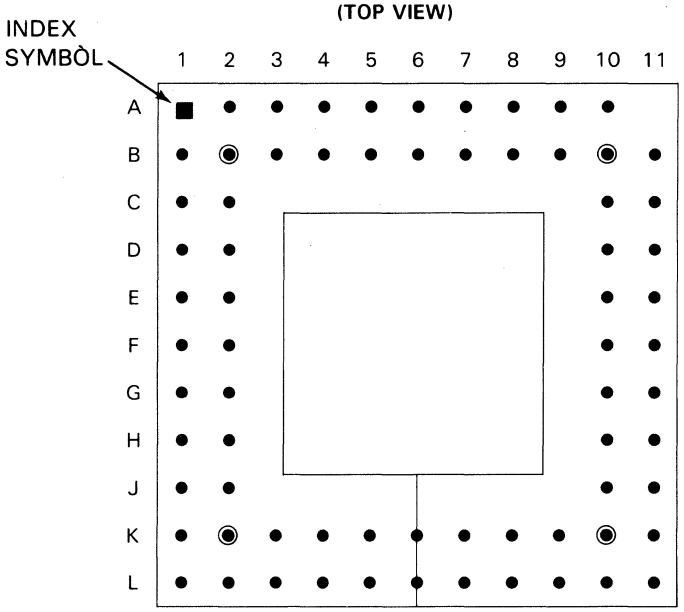
PIN NO.	PIN NAME	PIN NO.	PIN NAME	PIN NO.	PIN NAME	PIN NO.	PIN NAME
A-2	C_n	B-9	$\overline{OEY}$	F-10	Y3	K-4	C2
A-3	$\overline{SIO0}$	B-10	Y0	F-11	DB2	K-5	A0
A-4	$\overline{QIO0}$	B-11	Y1	G-1	DA2	K-6	A3
A-5	$\overline{QIO7}$	C-1	I5	G-2	DA0	K-7	$\overline{WE}$
A-6	C_{n+8}	C-2	V_{CC2}	G-10	DB0	K-8	DB7
A-7	$\overline{G}$/N	C-10	Y4	G-11	DB3	K-9	$\overline{OEB}$
A-8	$\overline{P}$/OVR	C-11	Y6	H-1	DA3	K-10	EB0
A-9	ZERO	D-1	I6	H-2	DA1	K-11	EB1
A-10	PPP	D-2	V_{CC1}	H-10	DB6	L-2	CK
B-1	I2	D-10	Y5	H-11	DB4	L-3	C1
B-2	I3	D-11	Y7	J-1	DA4	L-4	C3
B-3	I1	E-1	I7	J-2	DA5	L-5	A1
B-4	I0	E-2	$\overline{OEA}$	J-10	SELY	L-6	A2
B-5	I4	E-10	Y2	J-11	DB5	L-7	B3
B-6	$\overline{SIO7}$	E-11	DB1	K-1	DA6	L-8	B2
B-7	SSF	F-1	$\overline{EA}$	K-2	DA7	L-9	B1
B-8		F-2	GND	K-3	C0	L-10	B0

Figure A1. 'AS888 Pin Assignments for GB Package

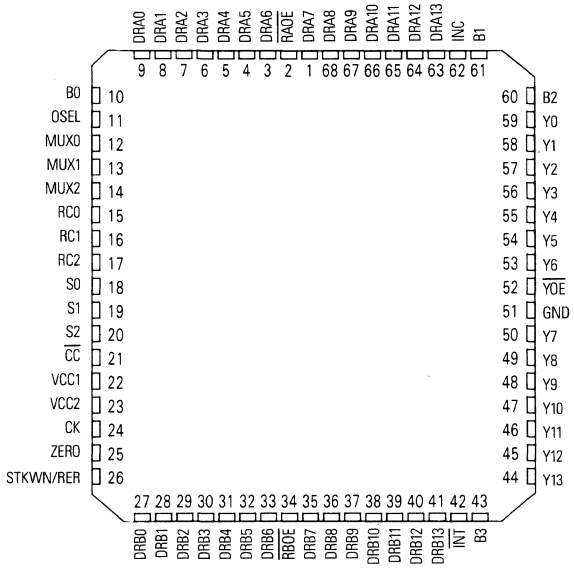
Table A1. 'AS888 Pin Descriptions

NAME	INPUT/OUTPUT	DESCRIPTION
$\overline{WE}$	Input	Register file (RF) write enable. Data is written into RF when $\overline{WE}$ is low and a low-to-high clock transition occurs. RF write is inhibited when $\overline{WE}$ is high.
B3-B0	Input	Register file B port read address select. (0 = LSB).
$\overline{OEB}$	Input	DB bus enable, low active.
DB7-DB0	Input/Output	B port data bus. Outputs register data ($\overline{OEB}$ = 0) or used to input external data ($\overline{OEB}$ = 1). (0 = LSB).
Y7-Y0	Input/Output	Y port data bus. Outputs instruction results ($\overline{OEY}$ = 0) or used to input external data into register file. (OEY = 1).
$\overline{OEY}$	Input	Y bus output enable, low active.
PPP	Input	Package position pin. Tri-level input used to define package significance during instruction execution. Leave open for intermediate positions, tie to V_{CC} for most significant package, and tie to GND for least significant package.
SSF	Input/Output	Special shift function. Used to transfer required information between packages during special instruction execution.
ZERO	Input/Output	Device zero detection, open collector. Input during certain special instructions.
$\overline{P}$/OVR	Output	ALU active low propagate/instruction overflow for most significant package.
$\overline{G}$/N	Output	ALU active low generate/negative result for most significant package.
C_{n+8}	Output	ALU ripple carry output.
$\overline{SIO7}$	Input/Output	Bidirectional shift pin, low active.
$\overline{QIO7}$	Input/Output	Bidirectional shift pin, low active.
$\overline{QIO0}$	Input/Output	Bidirectional shift pin, low active.
$\overline{SIO0}$	Input/Output	Bidirectional shift pin, low active.
C_n	Input	ALU carry input.
I7-I0	Input	Instruction inputs.
V_{CC2}		Low voltage power supply (2 V).
V_{CC1}		I/O interface supply voltage (5 V).
$\overline{EA}$	Input	ALU input operand select. High state selects external DA bus and low state selects register file.
GND		Ground pin.
DA7-DA0	Input/Output	A port data bus. Outputs register file data ($\overline{EA}$ = 0) or inputs external data ($\overline{EA}$ = 1).
CK	Input	Clocks all synchronous registers on positive edge.
C3-C0	Input	Register file write address select.
A3-A0	Input	Register file A port read address select.
$\overline{OEA}$	Input	DA bus enable, low active.
SELY	Input	Y bus select, high active.
EB1, EB0	Input	ALU input operand selects. These inputs select the source of data that the S multiplexer provides for the S bus. Independent control of the DB bus and data path selection allow the user to isolate the DB bus while the R-ALU continues to process data.
GND		Ground pin.

SN54AS890, SN74AS890 . . . GB PACKAGE
(TOP VIEW)



SN54AS890 . . . FK PACKAGE
SN74AS890 . . . FN PACKAGE
(TOP VIEW)



PIN NO.	PIN NAME	PIN NO.	PIN NAME	PIN NO.	PIN NAME	PIN NO.	PIN NAME
A-2	DRB10	B-9	STKWRN/RER	F-10	V_{CC1}	K-4	DRA13
A-3	DRB9	B-10	ZERO	F-11	MUX2	K-5	DRA11
A-4	DRB8	B-11	CK	G-1	Y5	K-6	DRA8
A-5	DRB7	C-1	Y13	G-2	$\overline{YOE}$	K-7	DRA7
A-6	DRB6	C-2	Y10	G-10	RC1	K-8	DRA0
A-7	DRB5	C-10	$\overline{CC}$	G-11	MUX1	K-9	DRA1
A-8	DRB4	C-11	S1	H-1	Y4	K-10	DRA3
A-9	DRB3	D-1	Y12	H-2	Y6	K-11	DRA2
A-10	DRB1	D-2	Y9	H-10	B0	L-2	B2
B-1	DRB13	D-10	S2	H-11	MUX0	L-3	INC
B-2	$\overline{INT}$	D-11	S0	J-1	Y3	L-4	DRA12
B-3	DRB12	E-1	Y11	J-2	Y2	L-5	DRA10
B-4	DRB11	E-2	Y8	J-10	RC2	L-6	DRA9
B-5	B3	E-10	V_{CC2}	J-11	OSEL	L-7	$\overline{RAOE}$
B-6	$\overline{RBOE}$	E-11	RC0	K-1	Y1	L-8	DRA6
B-7	DRB2	F-1	Y7	K-2	Y0	L-9	DRA5
B-8	DRB0	F-2	GND	K-3	B1	L-10	DRA4

Figure A2. 'AS890 Pin Assignments for GB Package

Table A2. 'AS890 Pin Descriptions

PIN NAME	I/O	PIN FUNCTION
$\overline{RAOE}$	In	Enables DRA output, active low
DRA6-DRA0	In/Out	Seven LSBs of the A direct data I/O port
OSEL	In	MUX control for the source to DRA. Low selects RA, high selects stack.
MUX0-MUX2	In	MUX control for Y output bus (see Table 1)
RC0-RC2	In	Register/counter controls (see Table 3)
S0-S2	In	Stack control (see Table 2)
$\overline{CC}$	In	Condition code
V_{CC1}		5-volt supply for TTL compatible I/O
V_{CC2}		2-volt supply for internal STL
CK	In	Clock
ZERO	Out	Zero detect flag for register A and B
STKWRN/RER	Out	Stack overflow, underflow/read error flag
DRB0-DRB6	In/Out	Seven LSBs of the B direct data I/O port (O=LSB)
$\overline{RBOE}$	In	Enables DRB output, active low
DRB7-DRB13	In/Out	Seven MSBs of the B direct data I/O port
$\overline{INT}$	In	Active low selects INT RT register to stack
Y13-Y8	In/Out	Six MSBs of bidirectional Y port
GND		Ground
Y7	In/Out	Seventh bit of bidirectional Y port
$\overline{YOE}$	In	Enables Y output bus, active low
Y6-Y0	In/Out	Seven LSBs of bidirectional Y port (O=LSD)
INC	In	Incrementer control
DRA13-DRA7	In/Out	Seven MSBs of direct B data I/O
B0-B3	In	16-way branch inputs on

TI Sales Offices

ALABAMA: Huntsville (205) 837-7530.

ARIZONA: Phoenix (602) 995-1007.

CALIFORNIA: Irvine (714) 660-8187;
Sacramento (916) 929-1521;
San Diego (619) 278-9601;
Santa Clara (408) 980-9000;
Torrance (213) 217-7010;
Woodland Hills (818) 704-7759.

COLORADO: Aurora (303) 368-8000.

CONNECTICUT: Wallingford (203) 269-0074.

FLORIDA: Ft. Lauderdale (305) 973-8502;
Maitland (305) 660-4600; **Tampa** (813) 870-6420.

GEORGIA: Norcross (404) 662-7900.

ILLINOIS: Arlington Heights (312) 640-2925.

INDIANA: Ft. Wayne (219) 424-5174;
Indianapolis (317) 248-8555.

IOWA: Cedar Rapids (319) 395-9550.

MARYLAND: Baltimore (301) 944-8600.

MASSACHUSETTS: Waltham (617) 895-9100.

MICHIGAN: Farmington Hills (313) 553-1500.

MINNESOTA: Eden Prairie (612) 828-9300.

MISSOURI: Kansas City (816) 523-2500;
St. Louis (314) 569-7600.

NEW JERSEY: Iselin (201) 750-1050.

NEW MEXICO: Albuquerque (505) 345-2555.

NEW YORK: East Syracuse (315) 463-9291;
Endicott (607) 754-3900; **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.

OKLAHOMA: Tulsa (918) 250-0633.

OREGON: Beaverton (503) 643-6758.

PENNSYLVANIA: Ft. Washington (215) 643-6450;
Coraopolis (412) 771-8550.

PUERTO RICO: Hato Rey (809) 753-8700

TEXAS: Austin (512) 250-7655;
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) 785-7140.

CANADA: Nepean, Ontario (613) 726-1970;
Richmond Hill, Ontario (416) 884-9181;
St. Laurent, Quebec (514) 334-3635.

TI Regional Technology Centers

CALIFORNIA: Irvine (714) 660-8140,
Santa Clara (408) 748-2220.

GEORGIA: Norcross (404) 662-7945.

ILLINOIS: Arlington Heights (312) 640-2909.

MASSACHUSETTS: Waltham (617) 890-6671.

TEXAS: Richardson (214) 680-5066.

CANADA: Nepean, Ontario (613) 726-1970

Technical Support Center

TOLL FREE: (800) 232-3200

TI Distributors

TI AUTHORIZED DISTRIBUTORS IN USA

- **Arrow Electronics**
- **Diplomat Electronics**
- **General Radio Supply Company**
- **Graham Electronics**
- **Harrison Equipment Co.**
- **International Electronics**
- **JACO Electronics**
- **Kierulff Electronics**
- **LCOMP, Incorporated**
- **Marshall Industries**
- **Milgray Electronics**
- **Newark Electronics**
- **Rochester Radio Supply**
- **Time Electronics**
- **R.V. Weatherford Co.**
- **Wyle Laboratories**

TI AUTHORIZED DISTRIBUTORS IN CANADA

- **Arrow/CESCO Electronics, Inc.**
- **Future Electronics**
- **ITT Components**
- **L.A. Varah, Ltd.**

ALABAMA: Arrow (205) 882-2730;
Kierulff (205) 883-6070; Marshall (205) 881-9235.

ARIZONA: Arrow (602) 968-4800;
Kierulff (602) 243-4101; Marshall (602) 968-6181;
Wyle (602) 866-2888.

CALIFORNIA: Los Angeles/Orange County:
Arrow (818) 701-7500, (714) 838-5422;
Kierulff (213) 725-0325, (714) 731-5711, (714) 220-6300;
Marshall (818) 999-5001, (818) 442-7204,
(714) 660-0951; R.V. Weatherford (714) 634-9600,
(213) 849-3451, (714) 623-1261; Wyle (213) 322-8100,
(818) 880-9001, (714) 863-9953; **Sacramento:** Arrow
(916) 925-7456; Wyle (916) 638-5282; **San Diego:**
Arrow (619) 565-4800; Kierulff (619) 278-2112;
Marshall (619) 578-9600; Wyle (619) 565-9171;
San Francisco Bay Area: Arrow (408) 745-6600;
(415) 487-4600; Kierulff (408) 971-2600;
Marshall (408) 732-1100; Wyle (408) 727-2500;
Santa Barbara: R.V. Weatherford (805) 965-8551.

COLORADO: Arrow (303) 696-1111;
Kierulff (303) 790-4444; Wyle (303) 457-9953.

CONNECTICUT: Arrow (203) 265-7741;
Diplomat (203) 797-9674; Kierulff (203) 265-1115;
Marshall (203) 265-3822; Milgray (203) 795-0714.

FLORIDA: Ft. Lauderdale: Arrow (305) 429-8200;
Diplomat (305) 974-8700; Kierulff (305) 486-4004;
Orlando: Arrow (305) 725-1480;
Milgray (305) 647-5747; **Tampa:**
Arrow (813) 576-8995; Diplomat (813) 443-4514;
Kierulff (813) 576-1966.

GEORGIA: Arrow (404) 449-8252;
Kierulff (404) 447-5252; Marshall (404) 923-5750.

TEXAS INSTRUMENTS

Creating useful products
and services for you.

ILLINOIS: Arrow (312) 397-3440;
Diplomat (312) 595-1000; Kierulff (312) 250-0500;
Marshall (312) 490-0155; Newark (312) 784-5100.

INDIANA: Indianapolis: Arrow (317) 243-9353;
Graham (317) 634-8202; Marshall (317) 297-0483;
Ft. Wayne: Graham (219) 423-3422.

IOWA: Arrow (319) 395-7230.

KANSAS: Kansas City: Marshall (913) 492-3121;
Wichita: LCOMP (316) 265-9507.

MARYLAND: Arrow (301) 995-0003;
Diplomat (301) 995-1226; Kierulff (301) 636-5800;
Milgray (301) 793-3993.

MASSACHUSETTS: Arrow (617) 933-8130;
Diplomat (617) 935-6611; Kierulff (617) 667-8331;
Marshall (617) 272-8200; Time (617) 935-8080.

MICHIGAN: Detroit: Arrow (313) 971-8220;
Marshall (313) 525-5850; Newark (313) 967-0600;
Grand Rapids: Arrow (616) 243-0912.

MINNESOTA: Arrow (612) 830-1800;
Kierulff (612) 941-7500; Marshall (612) 559-2211.

MISSOURI: Kansas City: LCOMP (816) 221-2400;
St. Louis: Arrow (314) 567-6888;
Kierulff (314) 739-0855.

NEW HAMPSHIRE: Arrow (603) 668-6968.

NEW JERSEY: Arrow (201) 575-5300, (609) 596-8000;
Diplomat (201) 785-1830;
General Radio (609) 964-8560; Kierulff (201) 575-6750;
(609) 235-1444; Marshall (201) 882-0320,
(609) 234-9100; Milgray (609) 983-5010.

NEW MEXICO: Arrow (505) 243-4566;
International Electronics (505) 345-8127.

NEW YORK: Long Island: Arrow (516) 231-1000;
Diplomat (516) 454-6400; JACO (516) 273-5500;
Marshall (516) 273-2053; Milgray (516) 420-9800;
Rochester: Arrow (716) 427-0300;
Marshall (716) 235-7620;
Rochester Radio Supply (716) 454-7800;
Syracuse: Arrow (315) 652-1000;
Diplomat (315) 652-5000; Marshall (607) 798-1611.

NORTH CAROLINA: Arrow (919) 876-3132,
(919) 725-8711; Kierulff (919) 872-8410.

OHIO: Cincinnati: Graham (513) 772-1661;
Cleveland: Arrow (216) 248-3990;
Kierulff (216) 587-6558; Marshall (216) 248-1788.
Columbus: Graham (614) 895-1590;
Dayton: Arrow (513) 435-5563; Kierulff (513) 439-0045;
Marshall (513) 236-8088.

OKLAHOMA: Kierulff (918) 252-7537.

OREGON: Arrow (503) 684-1690; Kierulff
(503) 641-9153; Wyle (503) 640-6000; Marshall
(503) 644-5050.

PENNSYLVANIA: Arrow (412) 856-7000,
(215) 928-1800; General Radio (215) 922-7037.

RHODE ISLAND: Arrow (401) 431-0980

TEXAS: Austin: Arrow (512) 835-4180;
Kierulff (512) 835-2090; Marshall (512) 837-1991;
Wyle (512) 834-9957; **Dallas:** Arrow (214) 380-6464;
International Electronics (214) 233-9323;
Kierulff (214) 343-2400; Marshall (214) 233-5200;
Wyle (214) 235-9953;
El Paso: International Electronics (915) 598-3406;
Houston: Arrow (713) 530-4700;
Marshall (713) 789-6600;
Harrison Equipment (713) 879-2600;
Kierulff (713) 530-7030; Wyle (713) 879-9953.

UTAH: Diplomat (801) 486-4134;
Kierulff (801) 973-6913; Wyle (801) 974-9953.

VIRGINIA: Arrow (804) 282-0413.

WASHINGTON: Arrow (206) 643-4800;
Kierulff (206) 575-4420; Wyle (206) 453-8300; Marshall
(206) 747-9100.

WISCONSIN: Arrow (414) 764-6600; Kierulff
(414) 784-8160.

CANADA: Calgary: Future (403) 235-5325; Varah
(403) 255-9550; **Edmonton:** Future (403) 486-0974;
Varah (403) 437-2755; **Montreal:** Arrow/CESCO
(514) 735-5511; Future (514) 694-7710; ITT
Components (514) 735-1177; **Ottawa:** Arrow/CESCO
(613) 226-6903; Future (613) 820-8313; ITT
Components (613) 226-7406; Varah (613) 726-8884;
Quebec City: Arrow/CESCO (418) 687-4231; **Toronto:**
CESCO (416) 661-0220;
Future (416) 638-4771; ITT Components
(416) 736-1144; Varah (416) 842-8484;
Vancouver: Future (604) 438-5545; Varah
(604) 873-3211; **Winnipeg:** Varah (204) 633-6190 BL

TEXAS INSTRUMENTS

Creating useful products
and services for you.

Printed in U.S.A.

SDBU001A