THE ELECTRONICS HANDBOOK

Editor-in-Chief

A CRC Handbook Published in Cooperation with IEEE Press

DOCKET ALARM
Find authenticated court documents without watermarks at docketalarm.com.

THE INDUSTRIAL ELECTRONICS HANDBOOK

Editor-in-Chief
J. DAVID IRWIN

CRC PRESS
IEEE PRESS
A CRC Handbook Published in Cooperation with IEEE Press

DOCKET ALARM
Find authenticated court documents without watermarks at docketalarm.com.

Library of Congress Cataloging-in-Publication Data

The industrial electronics handbook/edited by J. David Irwin.
p. cm.--(The electrical engineering handbook series)
Includes bibliographical references and index.
ISBN 0-8493-8343-9 (alk. paper)
1. Industrial electronics—Handbooks, manuals, etc. I. Irwin, J. David, 1939– . II. Series.
TK7881.I52 1996
621.3-dc20

96-3070
CIP

This book contains information obtained from authentic and highly regarded sources. Reprinted material is quoted with permission, and sources are indicated. A wide variety of references are listed. Reasonable efforts have been made to publish reliable data and information, but the editor, authors, and the publisher do not assume responsibility or liability for the validity of any materials or for the consequences of their use.

Neither this book nor any part may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, microfilming, and recording, or by any information storage or retrieval system, without prior permission in writing from the publisher.

All rights reserved. Authorization to photocopy items for internal or personal use, or the personal or internal use of specific clients, may be granted by CRC Press LLC, provided that $.50 per page photocopied is paid directly to Copyright Clearance Center, 27 Congress Street, Salem, MA 01970 USA. The fee code for users of the Transactional Reporting Service is ISBN 0-8493-8343-9/97 $0.00 + $.50. The fee is subject to change without notice. For organizations that have been granted a photocopy license by the CCC, a separate system of payment has been arranged.

The consent of CRC Press does not extend to copying for general distribution, for promotion, for creating new works, or for resale. Specific permission must be obtained in writing from CRC Press for such copying.

Direct all inquiries to CRC Press LLC, 2000 Corporate Blvd., N.W., Boca Raton, Florida 33431.

© 1997 by CRC Press LLC

No claim to original U.S. Government works
International Standard Book Number 0-8493-8343-9
Library of Congress Card Number 96-3070
Printed in the United States of America 1 2 3 4 5 6 7 8 9 0
Printed on acid-free paper

DOCKET ALARM Find authenticated court documents without watermarks at docketalarm.com.

Boolean Operations on Bit Variable b

	OR	XOR
	$b + 0 = b$	$b \oplus 0 = b$
	$b + 1 = 1$	$b \oplus 1 = \bar{b}$

	b_2	b_1	b_0		b_3	b_2	b_1	b_0		b_3	b_2	b_1	b_0
	1	0	1	$\vee$	0	0	1	0	$\oplus$	0	0	1	0
	b_2	0	b_0		b_3	b_2	1	b_0		b_3	b_2	$\bar{b}_1$	b_0

(a) Clear b_1 (b) Set b_1 (c) Toggle b_1

Figure 3.19 Logical operations used to alter a selected bit.

device register. Table 3.4 summarizes the three Boolean ors applied to a one-bit Boolean variable.

AND operator can be used to force selected bits of a word illustrated in Figure 3.19a. The second operand is a bit called a *mask* that contains a 0 in each bit position that be forced to 0, and a 1 in each bit position that is to be changed. Similar masks can be created for the OR operator e selected bits to 1, and for the XOR operator to force ed bits to be complemented. These are illustrated in Figures and 3.19c, respectively.

ny input/output devices contain a status register whose bits t the readiness of the device to perform an operation. The operator can be used to isolate a selected bit of a byte read a status register to determine if that bit is 0 or 1. This is ated in Figure 3.20. Here the mask is used to force all bits except for bit b_1. If the zero flag of the CPU's processor register is set, indicating a result of 0000, then it follows that $b_1 = 0$; if the zero flag is not set, the result is nonzero which es $b_1 = 1$.

example, assume that a printer interface contains a status register in which the rightmost bit indicates whether the printer dy to accept another character to print. The following program loop will be continuously executed as long as the nter ready" bit is 0. The CPU will exit the loop and continue oon as the ready bit becomes 1.

```
Check:  IN    AL, PrintStatus   ;read printer status register
        AND   AL,0000 0001      ;isolated "printer ready" bit
        JZ    Check             ;go back to Check if printer not ready
```

Shift and Rotate

Shift and rotate instructions slide bits right or left within a ster or memory location as illustrated in Figure 3.21. These can be used for extracting or combining bit fields within an

	b_3	b_2	b_1	b_0
$\wedge$	0	0	1	0
	0	0	b_1	0

Figure 3.20 Using AND to isolate one bit.

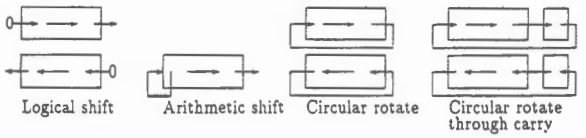

Figure 3.21 Shift and rotate operations.

operand, to convert data between parallel and serial form, and to perform multiplication and division by powers of 2.

In a *logical shift* operation, the bits are shifted right or left by one bit position, with the vacated bit replaced by a 0. For unsigned numbers, this is equivalent to dividing or multiplying the number by 2. An *arithmetic right shift* implements a divide by 2 operation on a two's complement number by preserving the sign bit as the operand is shifted. Some CPUs allow an operand to be shifted by more than one bit position with a single instruction. The following 68000 example packs two BCD digits into a single byte by shifting one digit four bits to the left and then combining the two digits.

```
Check:  SHL.B   #4,D0    ;shift BCD digit to upper nibble of D0
        OR.B    D1,D0    ;combine two BCD digits in D1 and D0
```

Circular rotate instructions perform a shift operation while replacing the vacated bit with the bit shifted out of the other end of the operand. A second rotate operation is often provided that rotates the number through the carry flag of the processor status register. In most CPUs, the bit shifted out of an operand is copied to the carry flag of the processor status register where it can be tested or used to support multi-precision shift operations. A multi-precision number can be shifted by using the carry flag as a link between parts of the number, allowing a bit shifted out of one part to be shifted into the other using a rotate-through-carry operation. The following 8086 example multiplies a 32-bit number by 2 by shifting one byte at a time one bit to the left.

```
SHL   NUMBER      ;shift memory byte 1 bit left
RLC   NUMBER+1    ;shift carry and 2nd byte 1 bit left
RLC   NUMBER+2    ;shift carry and 3rd byte 1 bit left
RLC   NUMBER+3    ;shift carry and 4th byte 1 bit left
```

Control Transfer

The normal flow of a program is to execute instructions in order from sequential memory addresses. To control this flow, the program counter increments automatically after each instruction. Jump, branch, and subroutine call instructions interrupt the normal flow by transferring control of the program to some instruction other than the next one in sequence. This allow looping and decision-making programs to be written, as well as supporting procedure and function calls. The following are examples of instructions that unconditionally transfer control of a program to location X within the current program:

DOCKET ALARM Find authenticated court documents without watermarks at docketalarm.com.

```
8051/8086:    JMP X
6805/68000:   JMP X or BR X
SPARC:        BRA X
```

Decision making and looping require conditional branch instructions that jump only if a given condition is true and continue with the next sequential instruction if the condition is false.

Conditional branch instructions typically test selected bits of the processor status register, which reflect the result of a previous arithmetic or logical operation. The following 8086 program loop adds a list of four numbers in memory, decrementing the SI register at the end of each iteration and repeating the loop as long as SI is greater than or equal to 0.

```
        MOV   SI,3              ;set counter to 3
        MOV   AL,0              ;clear accumulator
Start:  ADD   AL,TABLES[SI]     ;add next element of TABLE
        DEC   SI                ;subtract 1 from SI
        JGE   Start             ;repeat if SI ≥ 0
```

The relationship between two operands can be tested by subtracting them and then testing the resulting condition codes according to Table 3.5. Many CPUs provide a compare instruction (CMP) that performs the subtraction and sets the condition code flags without altering either operand. The following 6805 program branches to location RICK if the unsigned number in accumulator *A* is less than or equal to 10, using the "branch if less or same" instruction to test the result of a compare instruction.

```
Check:  CMP   #10     ;subtract 10 from A
        BLS   RICK    ;go to RICK if A lower than or same as 10
```

Modular programming requires the ability to partition software into separate subroutines, such as procedures and functions, that can be invoked as needed. This is supported by special subroutine call instructions instructions that jump from a main program to the start of a subroutine after saving a pointer to the next instruction in the main program, allowing a return to the main program after completing the subroutine.

A subroutine call (CALL) or jump to subroutine (JSR) instruction typically pushes the current program counter onto the system stack to save the address of the next instruction in the main program. A return (RET) or return from subroutine (RTS) is executed as the last instruction of the subroutine to pop the program counter from the stack and thus return to the main program. The SPARC does not support a system stack; subroutines are called with a jump and link (JMPL) instruction, which saves the program counter in register *r*31 of the current register window, and then slides the window down 16 registers as illustrated in Figure 3.10. the subroutine returns to the main program by retrieving the return address from register *r*7 of the register window, which corresponds to *r*31 of the calling program.

Table 3.5 Condition Codes for Relational Operators

Condition	Symbol	Relation	Number type	Boolean condition
Zero	Z	$A = B$	Both	Z
Not zero	NZ	$A \neq B$	Both	$\overline{Z}$
Greater than	G	$A > B$	Signed	$\overline{(N \oplus V) + Z}$
Greater than or equal	GE	$A \geq B$	Signed	$\overline{N \oplus V}$
Less than	L	$A < B$	Signed	$N \oplus V$
Less than or equal	LE	$A \leq B$	Signed	$(N \oplus V) + Z$
Above	A	$A > B$	Unsigned	$\overline{C + Z}$
Above or equal	AE	$A \geq B$	Unsigned	$\overline{C}$
Below	B	$A < B$	Unsigned	C
Below or equal	BE	$A \leq B$	Unsigned	$C + Z$

Input and Output

Some CPUs utilize separate address spaces for memory and for input/output devices. In these cases, special instructions are provided to read information into the CPU from an input device and to write information from the CPU to an output device. The Intel CPUs support an isolated I/O address space that can be accessed only by the two special instructions IN and OUT as follows:

```
IN    AL,25   ;data from IO address 25 to AL register
OUT   25,AL   ;data from AL register to IO address 25
```

Processor Control

These instructions manipulate various hardware elements within the CPU and are therefore CPU-specific. The reader is referred to *The SPARC Architecture Manual, Ver. 7* (1983, 1987), Motorola Inc. (1990), Brey (1994), and Stewart (1993) for descriptions of processor control instructions for specific CPUs.

3.8 Interrupts and Exceptions

Events often occur that require interruption of normal instruction processing to perform some special action. Such exceptional events, or simply *exceptions*, can be triggered by condition signaled by devices external to the CPU, or by conditions detected within the CPU.

For example, desktop PCs often use a timer to interrupt the CPU once per second to make it update an image of a clock displayed on the screen. PCs used in process control are typically interrupted by sensors that detect various conditions in the plant that require immediate attention. An example of an internally detected condition is an attempt to divide a number by 0, which cannot produce a valid result. This type of exceptional condition should suspend normal processing to abort the operation and send a warning message to the user.

A primary advantage of external interrupt is that a CPU may work in parallel with one or more external processes, such as printing a document, and be interrupted only when the process requires attention. The alternative is to continuously monitor the process by checking a status register in the device to detect when the device requires attention. Such monitoring would prevent the CPU from doing other work while waiting for the device.

DOCKET ALARM Find authenticated court documents without watermarks at docketalarm.com.