COP400 Microcontroller Family

COPS™ Family User's Guide

National Semiconductor

COPS Family User's Guide Table of Contents

Section	Description	Page
	Chapter 1. Introduction to COP400 Microcontrollers	
1.1	Summary of COP400 Microcontroller Features	9-7
	Chapter 2. COP400 Architecture	
2.1	COP420/COP421 Architecture	9-8
2.2	COP420/COP421 Functional Description	9-10
2.3	Initialization	9-11
2.4	COP420/COP421 Mask Programmable Options	9-12
2.5	COP420L/COP421L Description	9-15
2.6	COP420L/COP421L Mask-Programmable Options	9-15
2.7	COP420C Description	9-17
2.8	COP444L Description	9-18
2.9	COP402 and COP402M ROMless Part Description	9-18
2.10	COP404L ROMless Part Description	9-18
2.11	COP410L/COP411L Architecture	9-18
2.12	COP410L/COP411L Functional Description	9-20
2.13	COP410L/COP411L Mask-Programmable Options	9-21
	Chapter 3. COP400 Instruction Sets	
3.1	COP420-Series/COP444L Instruction Set	9-23
3.2	COP420-Series/COP444L Instruction Set Description	9-27
3.3	COP421-Series Instruction Set Differences	9-34
3.4	COP410L/COP411L Instruction Set	9-34
3.5	COP410L/COP411L Instruction Set Differences	9-37
	Chapter 4. COP400 Programming Techniques	
4.1	Program Memory Allocation	9-42
4.2	Data Memory Allocation and Manipulation	9-45
4.3	Subroutine Techniques	9-46
4.4	Utility Routines	9-47
4.5	Timing Considerations	9-48
4.6	BCD Arithmetic Routines	9-49
4.7	Simple Display Loop Routine	9-51
4.8	Interrupt Service Routine	9-53
4.9	Timekeeping Routine	9-53
4.10	String Search Routine	9-56
4.11	Programming Techniques for the COP421-Series, COP410L and 411L	9-57
	Chapter 5. COP400 I/O Techniques	
5.1	Hardware Interfacing Techniques	9-58
5.2	Software I/O Techniques	9-63
5.3	Keyboard/Display Interface	9-64
5.4	SIO (Serial) Input/Output	9-75
5.5	Add-on RAM	9-76
5.6	IN_3/IN_0 Inputs	9-77

COPS Family User's Guide
List of Figures

Figure	Description	Page
2.1	COP420/COP421 Block Diagram	9-8
2.2	COP420/COP421 Connection Diagrams	9-9
2.3	COP420/COP421 Pin Descriptions	9-9
2.4	Power-Clear Circuit	9-9
2.5	COP420/COP421 Clock Oscillator Configurations	9-13
2.6	COP420/COP421 Input/Output Configurations	9-14
2.7	COP420L/COP421L Oscillator Configurations	9-16
2.8	COP410L/COP411L Block Diagram	9-19
2.9	COP410L/COP411L Connection Diagrams	9-20
2.10	COP410L/COP411L Pin Description	9-20
2.11	COP410L/COP411L Oscillator Configurations	9-22
3.1	INIL Hardware Implementation	9-28
3.2	Enable Register Features — Bits EN_3 and EN_0	9-33
4.1	COP420 Data Memory Map	9-45
4.2	Flowchart for Multiply Routine	9-50
4.3	Flowchart for Timekeeping Routine	9-54
5.1	COP420 I/O Lines	9-59
5.2	COP420 I/O Options	9-59
5.3	COP420 Standard Output Characteristics	9-59
5.4	COP420 I/O Interconnect Examples	9-60
5.5	COP420 IN Input Characteristics	9-60
5.6	D and G Port Characteristics	9-61
5.7	COP420L I/O Port Characteristics	9-61
5.8	COP420 SI, SO, SK Characteristics	9-62
5.9	COP420 CKO, CKI, RESET Characteristics	9-62
5.10	COP420 I/O Expansion	9-63
5.11	COP420 LED Display System	9-63
5.12	COP420 VF Display System	9-63
5.13	COP420 MICROBUS™ Interconnect	9-63
5.14	COP420 Add-On RAM	9-63
5.15	Display/Keyboard Interconnect	9-64
5.16	Flowchart for Display/Keyboard Debounce Routine	9-65
5.17	Display Timing Diagram	9-66
5.18	Display/Keyboard Interface Source Code	9-71
5.19	Key-Decode Routine Assembler Output Listing	9-74
5.20	Additional I/O Using SI and SO	9-75
5.21	Multi-COP420 System	9-75
5.22	Add-On RAM Interconnect	9-77

COPS Family User's Guide
List of Tables

Table	Description	Page
3.1	COP420/COP421 Instruction Set Table	9-24
3.2	COP420/COP421 Instruction Set Symbols	9-27
3.3	COP410L/COP411L Instruction Set Table	9-34
3.4	COP410L/COP411L Instruction Set Symbols	9-37
3.5	Alphabetical Mnemonic Index of COP420/COP421 Instructions	9-38
3.6	COP420/COP421 Instructions Listed by Hex Opcodes	9-39
3.7	Alphabetical Mnemonic Index of COP410L/COP411L Instructions	9-40
3.8	COP410L/COP411L Instructions Listed by Hex Opcodes	9-41
4.1	Page to Hexadecimal Address	9-42
5.1	COP400 I/O Comparison Chart	9-58
5.2	Seven-Segment Decode Values	9-68
5.3	JID Pointer Table for Display/Keyboard Routine	9-73

1 Introduction to the COP400 Microcontrollers



This manual provides information on the COP400 series of National's single-chip microcontrollers. The material contained in this manual is intended to assist the reader in understanding the internal architecture, instruction set, programming techniques, and hardware and software I/O techniques pertaining to the COP400 family of microcontroller devices.

The primary focus of this manual is the COP420 — at the time of this printing the most inclusive device, on a hardware and software level, of the COP400 family. Other members of the COP400 family are discussed primarily in terms of the less inclusive features of these other parts (i.e., the COP421, COP410L, COP411L). This approach should not result in a lack of understanding in terms of the operation and programming of these parts since they are "subset" devices of the COP420, distinguished, for the most part, by deleted hardware and software features. For further information on these other devices and on future COP400 devices the reader should consult the data sheets appropriate to particular COP400 devices.

1.1 Summary of COP400 Microcontroller Features

COP400 Microcontrollers are fabricated using CMOS or N-channel, silicon gate MOS technology. They are complete microcomputers containing all system timing, internal logic, ROM, RAM, and I/O necessary to implement dedicated control functions in a variety of applications. Features of the COP400 devices include an instruction set, internal architecture, and I/O scheme designed to facilitate keyboard input, display output, and efficient BCD data manipulation.

The various members of the COP400 family allow the user to specify a microcontroller best suited for use in a particular dedicated application. Specifically, COP400 devices offer a choice among single-chip parts with differing amounts of ROM, RAM, I/O capability, and number of instructions. Additionally, many parts have different versions which allow a choice of electrical characteristics while retaining the basic architecture and instruction set of the basic device. (For example, the COP420L and COP420C are available as low-power and CMOS versions, respectively, of the standard COP420 device.) Finally, each part contains a number of clock, I/O and other options, mask-programmed into the part at the same time as the user's program; this allows even greater flexibility in matching the COP400 Microcontroller to the user's specifications, reducing the need for external interface logic.

All COP400 devices feature single-supply operation and fast, standardized, "in-house" test procedures which verify the internal logic and user program (ROM code) mask-programmed into the device. Several COP400 controllers are available in ROM-less versions for use in prototyping a COP400 system (using the COP400 Development System) or for low-volume applications.

Section 1 provides a list of COP400 devices currently available or in design, together with a summary of the basic features of each device. Refer to this manual and data sheets of particular devices for further information on these parts. Future members of the COP400 family will include more powerful hardware and software capabilities, alternative electrical specification devices (low power, CMOS versions) and peripheral devices suitable for use in many applications.

The flexible I/O configuration of COP400 Microcontrollers allows them to interface with and drive a wide range of devices using minimal external parts. Typical peripheral devices include:

1. Keyboards and displays (direct segment and digit drive possible for several devices).
2. External data memories.
3. Printers.
4. Other COPS™ devices.
5. A/D and D/A converters.
6. Power control devices (SCRs, TRIACs).
7. Mechanical actuators.
8. General purpose microprocessors (communication with host CPUs over National's MICROBUS™ for several COP400 devices).
9. Shift registers.
10. External ROM data storage devices.

2 COP400 Architecture

This chapter provides information on the architecture of the COP400 Microcontrollers. Consistent with the general approach of this manual, the COP420 is primarily discussed with the COP421 treated in terms of differences with respect to the COP420. The COP410L, COP411L and COP444L are similarly treated. The text, therefore, primarily discusses the internal architecture of the COP420, with differences noted for the other devices. Also briefly discussed are different versions of each primary device (e.g., for the COP420, the COP420L and COP420C). As these additional devices, as well as the most inclusive COP400 device, the COP440, become available, further information will be provided in data sheets for each part.

2.1 COP420/COP421 Architecture

Figure 2.1 provides a block diagram of the COP420/COP421. It is intended to acquaint the user with the functions of, and interconnections among, the various logic blocks within the processor. Data paths are illustrated in simplified form to depict how the logic elements communicate with each other in implementing the instruction set of the devices. Note that the IN_3-IN_0 general purpose inputs are not available on the COP421, nor are the two internal IL latches associated with IN_3 and IN_0.

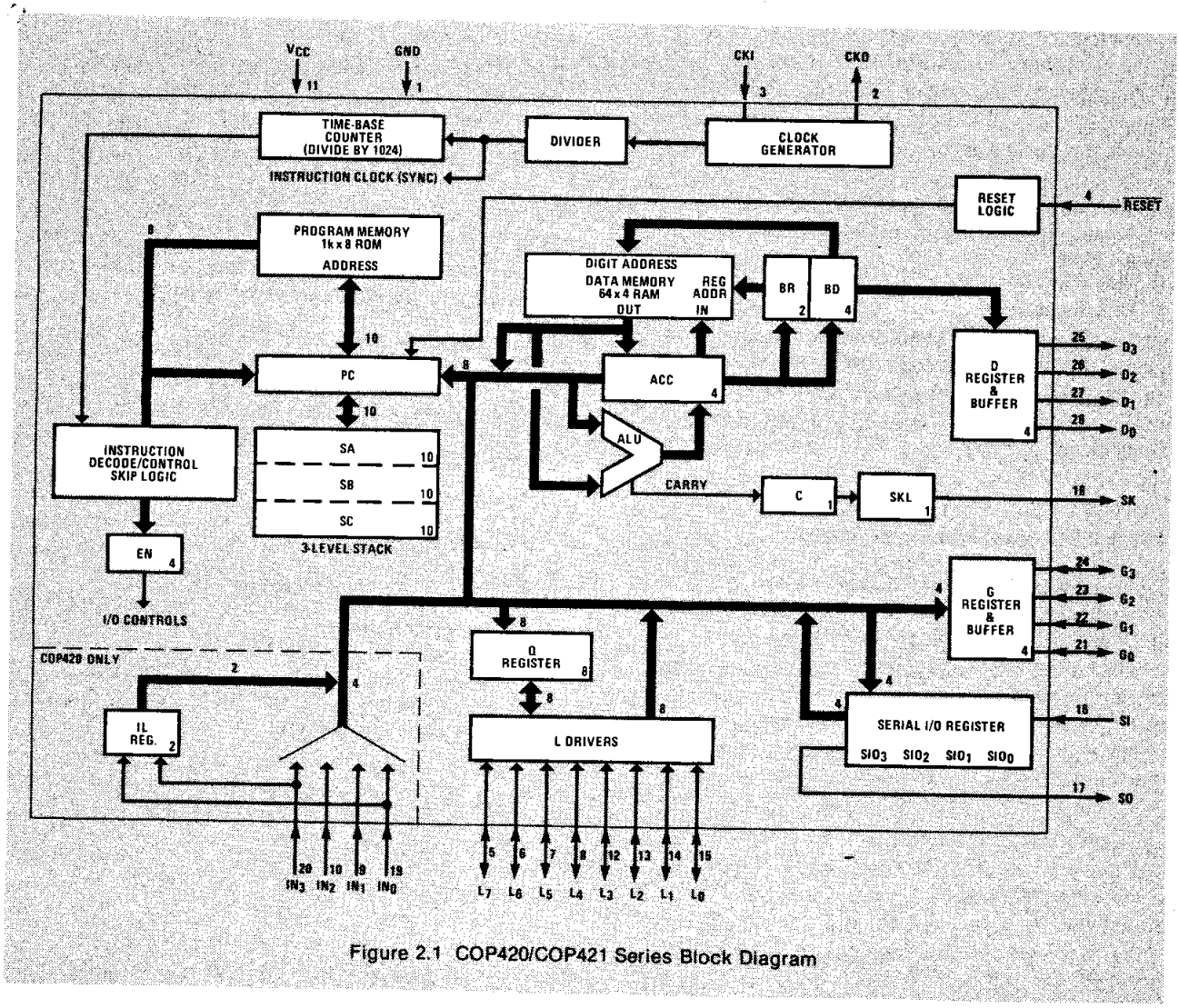


Figure 2.1 COP420/COP421 Series Block Diagram

Figure 2.2 shows the connection diagrams for the 28-pin COP420 and the 24-pin COP421. Figure 2.3 provides a pin description for the COP420/COP421 devices.

One should consult the COP420/COP421 data sheet for maximum ratings, DC and AC electrical characteristics for these devices.

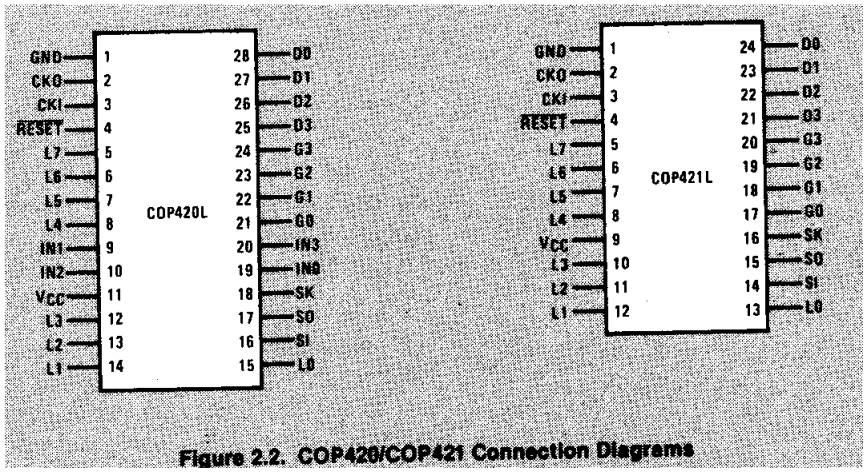


Figure 2.2. COP420/COP421 Connection Diagrams

L_7-L_0	8 bidirectional I/O ports with TRI-STATE®
G_3-G_0	4 bidirectional I/O ports
D_3-D_0	4 general purpose outputs
IN_3-IN_0	4 general purpose inputs (COP420 only)
SI	Serial input (or counter input)
SO	Serial output (or general purpose output)
SK	Logic-controlled clock (or general purpose output)
CKI	System oscillator input
CKO	System oscillator output (or general purpose input or RAM power supply)
$\overline{\text{RESET}}$	System reset input
V_{CC}	Power Supply
GND	Ground

Figure 2.3 COP420/COP421 Pin Description

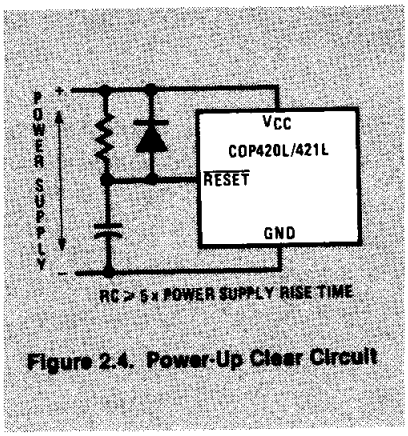


Figure 2.4. Power-Up Clear Circuit

2.2 COP420/COP421 Functional Description

The following text provides a functional description of the logic elements depicted in the COP420/COP421 block diagram.

Program Memory

Program memory consists of a 1,024-byte ROM. ROM words may be program instructions, program data or ROM address pointers. Due to the special characteristics associated with the JP and JSRP instructions, ROM must often be conceived of as organized into 16 pages of 64 words (bytes) each. Also, because of the unique operations performed by the LQID and JID instructions, ROM pages must often be thought of as organized into four consecutive blocks of four ROM pages. (For further information on the paging characteristics of these instructions, see Section 4.1.)

ROM addressing is accomplished by the 10-bit P register. Its binary value selects one of the 1,024 8-bit words (I_7-I_0) contained in ROM. The value of P is automatically incremented by 1 *prior* to the execution of the current instruction to point to the next sequential ROM location, unless the current instruction is a transfer of control instruction. In the latter case, P is loaded with the appropriate non-sequential value to implement the transfer of control operation performed by the instruction. It should be noted that P will automatically "roll-over" to point to the next page of program memory. This feature has particular significance for transfer of control instructions with paging restrictions, i.e., JP, JSRP, JID and LQID. Since P is incremented to roll-over to the next ROM page *prior* to executing these instructions, they will be treated as residing on the *next* ROM page if they reside in the last word of a ROM page. Further information is provided in Section 4.1.

Three levels of subroutine are implemented by the 10-bit subroutine save registers, SA, SB and SC, providing a last-in, first-out (LIFO) hardware subroutine stack.

ROM instruction words are fetched, decoded and executed by the Instruction Decode, Control and Skip Logic circuitry.

Data Memory

Data memory consists of a 256-bit RAM, organized as 4 data registers of 16 4-bit digits. RAM addressing is implemented by a 6-bit B register whose upper 2 bits (Br) select 1 of 4 data registers and lower 4 bits (Bd) select 1 of 16 4-bit digits in the selected data register. While the 4-bit contents of the selected RAM digit (M) are usually loaded into or from, or exchanged with, the A register (accumulator), they may also be loaded into or from the Q latches or loaded from the L ports. RAM addressing may also be performed directly by the LDD and XAD instructions based upon the 6-bit contents of the operand field of these instructions. The Bd register also serves as a source register for 4-bit data sent directly to the D outputs.

Internal Logic

The 4-bit A register (accumulator) is the source and destination register for most I/O, arithmetic, logic and data memory access operations. It can also be used to load the Br and Bd portions of the B register, to load and input 4 bits of the 8-bit Q latch data, to input 4 bits of the 8-bit L I/O port data and to perform data exchanges with the SIO register.

A 4-bit adder performs the arithmetic and logic functions of the COP420, storing results in A. It also outputs a carry bit to the 1-bit C register, most often employed to indicate arithmetic overflow. The C register, in conjunction with the XAS instruction and the EN register, also serves to control the SK output. C can be outputted directly to SKL or can enable SKL to be a SYNC pulse, providing a clock each instruction cycle time. (See XAS instruction, Table 3.1, and EN register description, below.)

Four general-purpose inputs, IN_3-IN_0, are provided for the COP420: IN_1, IN_2 and IN_3 may be selected, by a mask-programmable option, as Read Strobe, Chip Select and Write Strobe inputs, respectively, for use in MICROBUS™ applications.

The COP421 does not contain the IN_3-IN_0 inputs and, therefore, must use the 4 bidirectional G I/O ports or 8 bidirectional L I/O ports as input pins to the device. Use of National's MICROBUS is inappropriate with the COP421.

The D register provides 4 general purpose outputs and is used as the destination register for the 4-bit contents of Bd.

The G register contents are output to 4 general-purpose bidirectional I/O ports. The COP420 G_0 pin may be mask-programmed as a "ready" output for MICROBUS applications.

The Q register is an internal, latched, 8-bit register, used to hold data loaded to or from M and A, as well as 8-bit program data from ROM. Its contents are output to the L I/O ports when the L drivers are enabled under program control (via an LEI instruction). The COP420 may use the MICROBUS option to write L I/O port data into Q upon the occurrence of a $\overline{WR}$ pulse from the host CPU.

The 8 L drivers, when enabled, output the contents of latched Q data to the L I/O ports. Also, the contents of L may be read directly into A and M. As explained above, the COP420 MICROBUS option allows L I/O port data to be latched into the Q

register. L I/O ports can be directly connected to the segments of a multiplexed LED display (using the TRI-STATE® LED Direct Drive output configuration option) with Q data being outputted to the Sa-Sg and decimal point segments of the display.

The SIO register functions as a 4-bit serial-in/serial-out shift register or as a binary counter depending on the contents of the EN register. (See EN register description, below.) Its contents can be exchanged with A, allowing it to input or output a continuous serial data stream. SIO may also be used to provide additional parallel I/O when used as a shift register with its input or output connected to external serial-in/parallel-out shift registers.

The 10-bit time base counter divides the instruction cycle frequency by 1,024, providing a pulse upon overflow. The COP420 SKT instruction tests for the occurrence of this pulse, allowing the programmer to rely on this internal time-base rather than external inputs (e.g., 50/60 Hz signals) to implement "real-time" routines.

The EN register is an internal 4-bit register loaded under program control by the LEI instruction. The state of each bit of this register selects or deselects the particular feature associated with each bit of the EN register (EN_3-EN_0).

1. The least significant bit of the enable register, EN_0, selects the SIO register as either a 4-bit shift register or a 4-bit binary counter. With EN_0 set, SIO is an asynchronous binary counter, decrementing its value by one upon each low-going pulse ("1" to "0") occurring on the SI input (count-down counter). Each pulse must be at least two instruction cycles wide. SK outputs the value of C upon execution of XAS and remains latched until the execution of another XAS instruction. The SO output is equal to the value of EN_3. With EN_0 reset, SIO is a serial shift register shifting left each instruction cycle time. The data present at SI goes into the least significant bit of SIO. SO can be enabled to output the most significant bit of SIO each cycle time. The SK output becomes a logic-controlled clock, providing a SYNC signal each instruction time. It will start outputting a SYNC pulse upon the execution of an XAS instruction with C = 1, stopping upon the execution of a subsequent XAS with C = 0.

2. With EN_1 set, the COP420 IN_1 input is enabled as an interrupt input. Immediately following an interrupt, EN_1 is reset to disable further interrupts. Note that this interrupt feature associated with IN_1 is unavailable on the COP421 since it lacks the IN inputs. Bit 1 (EN_1) of the Enable Register is, therefore, a "don't care" bit for the COP421: setting or resetting this bit via an LEI instruction will have no effect on the operation of the COP421. (For further information on the procedure and protocol of this COP420 interrupt feature, see Section 3.2, LEI instruction description.)

3. With EN_2 set, the L drivers are enabled to output the data in Q to the L I/O ports. Resetting EN_2 disables the L drivers, placing the L I/O ports in a high-impedance input state. If the COP420 MICROBUS™ option is being used, EN_2 does not affect the L drivers.

4. EN_3, in conjunction with EN_0, affects the SO output. With EN_0 set (binary counter option selected), SO will output the value loaded into EN_3. With EN_0 reset (serial shift register option selected), setting EN_3 enables SO as the output of the SIO shift register, outputting serial shifted data each instruction time. Resetting EN_3 with the serial shift register option selected disables SO as the shift register output: data continues to be shifted through SIO and can be exchanged with A via an XAS instruction but SO remains reset to "0." Table 2.1 provides a summary of the options and features associated with EN_3 and EN_0.

2.3 Initialization

Upon initialization of the COP420/COP421 as described below, the P register is cleared to 0 (ROM address 0) and the A, B, C, D, EN, and G registers are cleared. The IN_0 and IN_3 latches are not cleared. The SK output is enabled as a SYNC output, providing a pulse each instruction cycle time. *Data memory (RAM) can only be cleared by the user's program. The first instruction at address 0 must be a CLRA.*

The Reset Logic, internal to the COP420/COP421, will initialize (clear) the device upon power-up if the power supply rise time is less than 1 ms and greater than 1 μs. If the power supply rise time is greater than 1 ms, the user must provide an external RC network and diode to the $\overline{\text{RESET}}$ pin as shown in Figure 2.4 below. The $\overline{\text{RESET}}$ pin is configured as a Schmitt trigger input. If not used, it should be connected to V_{CC}. Initialization will occur whenever a logic "0" is applied to the $\overline{\text{RESET}}$ input, provided it stays low for at least three instruction cycle times. In order to reset the Time Base Counter, a $\overline{\text{RESET}}$ pulse ten instruction cycle times wide must be applied; note that the counter will overflow and generate an output pulse.

2.4 COP420/COP421 Mask Programmable Options

To allow even greater flexibility in specifying a COP400 device appropriate to the user's application, all COP400 microcontrollers have specific clock configuration, I/O and other mask-programmable options associated with them. These options are masked into the part simultaneously with the masking of the user's program in ROM and have been chosen to offer the user a wide range of options which encompasses design options most frequently employed in dedicated, small system applications.

The following text summarizes the COP420/COP421 options according to the various functions (oscillator, I/O, etc.) with which they are associated.

Clock Oscillator Options

There are four basic COP420/COP421 clock oscillator configurations avilable as shown by Figure 2.5 (a-d):

a. Crystal Controlled Oscillator. CKI and CKO are connected to an external crystal. The instruction cycle time equals the crystal frequency (4 MHz maximum) divided by 16 (optional by 8).

b. External Oscillator. CKI is configured as a TTL compatible input accepting an external clock signal. The external frequency (4 MHz maximum) is divided by 16 (optional by 8) to derive the instruction cycle time. CKO is now available to be used as the RAM power supply (V_R) pin, as a general purpose input, or as a synchronizing input.

c. RC Controlled Oscillator. CKI is configured as a single-pin RC controlled Schmitt trigger oscillator. The instruction cycle equals the oscillation frequency divided by 4. CKO is available for non-timing functions as in b above.

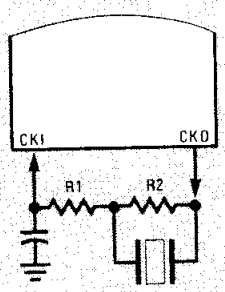
d. Externally Synchronized Oscillator. Intended for use in multi-COP systems, CKO is programmed to function as an input connected to the SK output of another COP420/COP421 with CKI connected as shown. In this configuration, the SK output connected to CKO must provide a SYNC (instruction cycle) signal to CKO, thereby allowing synchronous data transfer between the COPs using only the SI and SO serial I/O pins in conjunction with the XAS instruction. Note that on power-up SK is automatically enabled as a SYNC output. (See Initialization, above.)

The lower portion of Figure 2.5 provides component values for several instruction cycle times and crystal values associated with the RC controlled and Crystal Oscillator options, respectively.

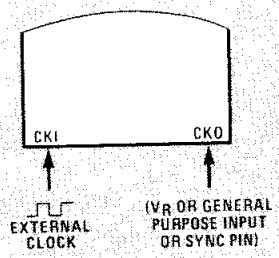
CKO Non-Timing Options

In a crystal controlled or multi-COP oscillator system, CKO is used as an output to the crystal network. In the other two configurations (external clock or RC controlled oscillator), CKO may be mask-programmed to perform one of two available options. Specifically, CKO may be mask-programmed as a general purpose input, read into bit 1 of the accumulator (A_2) upon the execution of an INIL instruction.

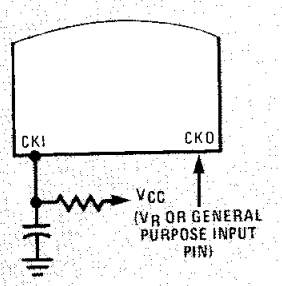
As another option (for both the COP420 and COP421), CKO can be a RAM power supply pin (V_R), allowing its connection to a standby/backup power supply to maintain the integrity of RAM data with minimum power drain when the main supply is inoperative or shut down to conserve power. Use of this options should include external circuitry to detect loss of V_{CC} power and force $\overline{\text{RESET}}$ low before V_{CC} drops below spec.



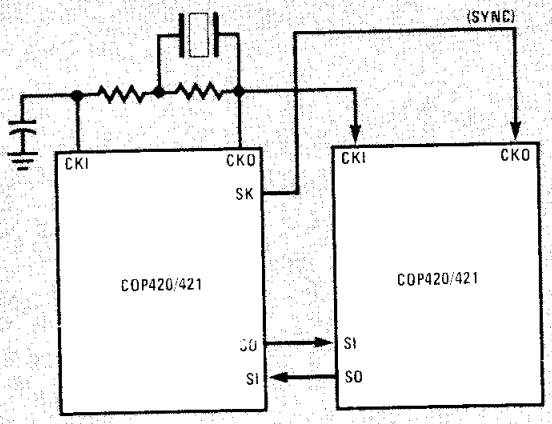
a. Crystal Oscillator



b. External Oscillator



c. RC Controlled Oscillator



d. Externally Synchronized Oscillator

Crystal Oscillator

Crystal Value	Component Values		
	R1	R2	C
4 MHz	1k	1M	27 pF
3.58 MHz	1k	1M	27 pF
2.09 MHz	1k	1M	56 pF

RC Controlled Oscillator

R (kΩ)	C (pF)	Instruction Cycle Time (μs)
12	100	5 ±20%
6.8	220	5.3 ±23%
8.2	300	8 ±29%
22	100	8.6 ±16%

Figure 2.5 COP420/COP421 Oscillator Configurations

MICROBUS™ Option

The COP420 has an option which allows it to be used as a peripheral microprocessor device, inputting and outputting data from and to a host microprocessor (μP). IN_1, IN_2, and IN_3 general purpose inputs become MICROBUS compatible read-strobe, chip-select, and write-strobe lines, respectively. IN_1 becomes $\overline{RD}$ — a logic "0" on this input will cause Q latch data to be enabled to the L ports for input to the μP. IN_2 becomes $\overline{CS}$ — a logic "0" on this line selects the COP420 as the μP peripheral device by enabling the operation of the $\overline{RD}$ and $\overline{WR}$ lines and allows for the selection of one of several peripheral components. IN_3 becomes $\overline{WR}$ — a logic "0" on this line will write bus data from the L ports to the Q latches for input to the COP420. G_0 becomes a "ready" output, reset by a write pulse from the μP on the $\overline{WR}$ line, providing the "handshaking" capability necessary for asynchronous data transfer between the host CPU and the COP420.

This option has been designed for compatibility with National's MICROBUS — a standard interconnect system for 8-bit parallel data transfer between MOS/LSI CPUs and interfacing devices. (See *MICROBUS™*, National Publication.) The functioning and timing relationships between the COP420 signal lines affected by this option are as specified for the MICROBUS interface. Connection of the COP420 to the MICROBUS is shown in Figure 5.13.

I/O Options

COP420/421 outputs have the following optional configurations, illustrated in Figure 2.6:

a. **Standard** — an enhancement mode device to ground in conjunction with a depletion-mode device to V_{CC}, compatible with TTL and CMOS input requirements. Available on SO, SK, and all D and G outputs.

b. **Open-Drain** — an enhancement-mode device to ground only, allowing external pull-up as required by the user's application. Available on SO, SK, and all D and G outputs.

c. **Push-Pull** — An enhancement-mode device to ground in conjunction with a depletion-mode device paralleled by an enhancement-mode device to V_{CC}. This configuration has been provided to allow for fast rise and fall times when driving capacitive loads. Available on SO and SK outputs only.

d. **Standard L** — same as **a.**, but may be disabled. Available on L outputs only.

e. **Open Drain L** — same as **b.**, but may be disabled. Available on L outputs only.

f. **LED Direct Drive** — an enhancement-mode device to ground and to V_{CC}, meeting the typical current sourcing requirements of the segments of an LED display. the sourcing device is clamped to limit current flow. These devices may be turned off under program control (See Functional Description, EN Register), placing the outputs in a high-impedance state to provide required LED segment blanking for a multiplexed display.

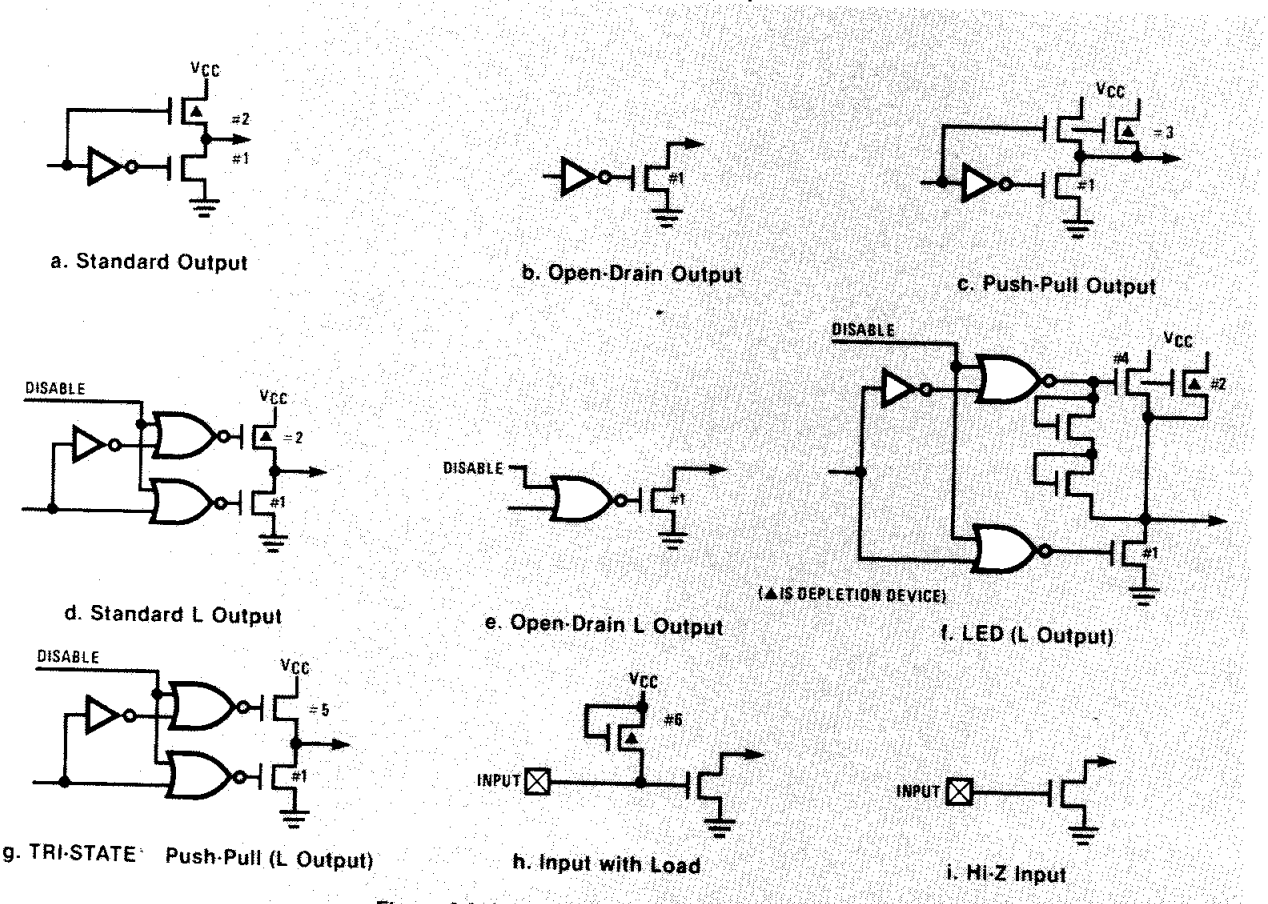


Figure 2.6 Input/Output Configurations

g. **TRI-STATE® Push-Pull** — an enhancement-mode device to ground and V_{CC}. These outputs are TRI-STATE outputs, allowing for connection of these outputs to a data bus shared by other bus drivers.

COP420/COP421 inputs have the following optional configurations:

h. An on-chip depletion load device to V_{CC}.

i. A Hi-Z input which must be driven to a "1" or "0" by external components.

The above input and output configurations share common enhancement-mode and depletion-mode devices. Specifically, all configurations use one or more of six devices (numbered 1-6, respectively).

The SO, SK outputs can be configured as shown in **a.**, **b.**, or **c.** The D and G outputs can be configured as shown in **a.** or **b.** Note that when inputting data to the G ports, the G outputs should be set to "1." The L outputs can be configured as in **d.**, **e.**, **f.** or **g.**

An important point to remember if using configuration **d.** or **f.** with the L drivers is that even when the L drivers are disabled, the depletion load device will source a small amount of current; however, when the L lines are used as inputs, the disabled depletion device can *not* be relied on to source sufficient current to pull an input to logic "1".

All of the L driver options are TRI-STATE® -able. Therefore, the L drivers have TRI-STATE-able Standard and Open-Drain output options as well as the TRI-STATE LED Direct Drive and Push-Pull output options. Since the device to V_{CC} in the Standard output configuration is a depletion-mode device, it will source up to 0.125 mA when this output is "turned off" in the TRI-STATE mode. This is not a worst case input for a logic "1" level on these inputs and will not be sufficient for an input level without previously enabling Q to L with $(Q) = FF_{16}$.

Bonding Option

The COP421 is a bonding option of the COP420: if the COP420 is bonded as a 24-pin device (without the 4 IN inputs), it becomes the COP421. Note that since it lacks the IN inputs, use of the COP421 bonding option precludes use of the IN input options; the MICROBUS™ option which would otherwise affect IN_3-IN_1 and G_0: use of the IN_1 hardware interrupt pin and the use of the IL_3 and IL_0 latches associated with the IN_3 and IN_0 pins. All other options are available. The COP421 is pin-compatible with the COP410L.

2.5 COP420L/COP421L Description

The COP420L/COP421L are low power versions of the COP420/COP421 containing the *same* internal logic elements and instruction set as the COP420/COP421, with *electrical* characteristics which are similar to the COP410L. The major differences between the COP420L/COP421L and COP420/COP421 are the following:

- Wider operating voltage range of 4.5 to 9.5V optionally available.
- Operating supply current less than 8 mA @ $V_{CC} = 5V$.
- Minimum instruction cycle time of 15 μs.
- Divide-by-32 crystal clock option (2 MHz XTAL divided by 32 = 15 μs instruction cycle time).
- D and G outputs have direct LED digit drive option (sink 30 mA).
- Other outputs will drive 1 LSTTL or 2 LPTTL loads ($I_{OL} = 360\,\mu A$ at 0.4V; $I_{OH} = 40\,\mu A$ at 2.4V).
- No MICROBUS™ option available.

The COP421L is simply a COP420L packaged in a 24-pin dual-in-line package. As a result, the IN inputs are not available on the COP421L, so that the COP421L is pin-compatible with the COP410L.

For further information, see the COP420L/COP421L data sheet.

2.6 COP420L/COP421L Mask Programmable Options

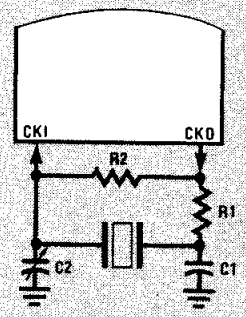
Since the COP420L/COP421L are frequently used in battery-operated and/or hand-held consumer-type products, an even greater array of system-cost-reducing options is available. The following text summarizes these options.

Clock Oscillator Options

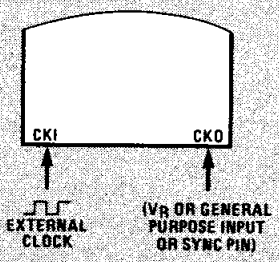
There are four basic COP420L/COP421L clock oscillator configurations available as shown in Figure 2.8 (a-d):

a. Crystal/Resonator Controlled Oscillator. CKI and CKO are connected to an external crystal or ceramic resonator. The instruction cycle time equals the crystal/resonator frequency (2.097 MHz maximum) divided by 32 (optional by 16 or 8).

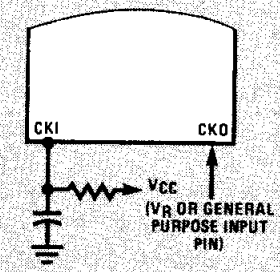
b. External Oscillator. CKI is configured as a CMOS compatible input accepting an external clock signal. The external frequency (2 MHz maximum) is divided by 32 (optional by 16, 8 or 4) to derive the instruction cycle time. CKO is now available to be used as the RAM power supply (V_R) pin, as a COP420L general purpose input, or as a synchronizing input.



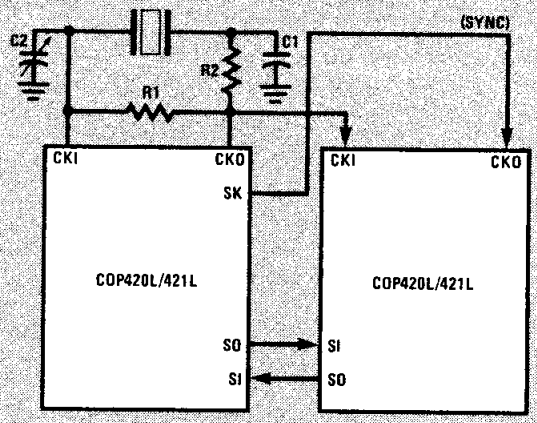
a. Crystal Oscillator



b. External Oscillator



c. RC Controlled Oscillator



d. Externally Synchronized Oscillator

Crystal Oscillator

Crystal Value	Component Values			
	R1	R2	C1	C2
455kHz (Resonator)	16k	1M	80pF	80pF
2.09MHz	1k	1M	56pF	6-36pF

RC Controlled Oscillator

R (kΩ)	C (pF)	Instruction Cycle Time (μs)
51	100	19 ± 15%
82	56	19 ± 15%

Figure 2.7 COP420L/COP421L Oscillator Configurations

c. RC Controlled Oscillator. CKI is configured as a single-pin RC controlled Schmitt trigger oscillator. The instruction cycle equals the oscillation frequency divided by 4. CKO is available for non-timing functions as in b above.

d. Externally Synchronized Oscillator. Intended for use in multi-COP systems, CKO is programmed to function as an input connected to the SK output of another COP420L/COP421L with CKI connected as shown. In this configuration, the SK output connected to CKO must provide a SYNC (instruction cycle) signal to CKO, thereby allowing synchronous data transfer between the COPs using only the SI and SO serial I/O pins in conjunction with the XAS instruction. Note that on power-up SK is automatically enabled as a SYNC output.

The lower portion of Figure 2.7 provides component values for several instruction cycle times and crystal values associated with the RC controlled and crystal controlled oscillator options, respectively.

CKO Non-Timing Options

In a crystal controlled or multi-COP oscillator system, CKO is used as an output to the crystal network. In the other two configurations (external clock or RC controlled oscillator), CKO may be mask-programmed to perform one of two available options. Specifically, CKO may be mask-programmed as a general purpose COP420L input, read into bit 1 of the accumulator (A_2) upon the execution of an INIL instruction.

As another option (for both the COP420L and COP421L), CKO can be a RAM power supply pin (V_R), allowing its connection to a standby/backup power supply to maintain the integrity of RAM data with minimum power drain when the main supply is inoperative or shut down to conserve power.

I/O Options

While the COP420L/COP421L has capabilities to directly drive LED displays through increased voltage and current specs, the circuit configurations are identical to those of the COP420 in Figure 2.6. Increased current sink and source values are a result of changing device sizes (within the bounds of the same circuit configuration). When emulating the COP420L with the COP402, one might use the typical values of the 402 as worst case COP420L drive parameters. An alternative is the use of the COP404L to emulate the drive of the COP420L.

For detailed electrical characteristics, refer to the COP420L/COP421L data sheet.

The SO and SK outputs can be configured as shown in Figure 2.6, a, b, or c. The D and G outputs can be configured as shown in a or b. Note that when inputting data to the G ports, the G outputs should be set to "1." The L outputs can be configured as shown in d, e, f, or g.

An important point to remember is that *all* of the L driver options are TRI-STATE® -able. Therefore, the L drivers have TRI-STATE-able Standard and Open-Drain output options as well as the TRI-STATE LED Direct Drive and Push-Pull output options. Since the device to V_{CC} in the Standard output configuration is a depletion-mode device, it will source up to 0.125mA when this output is "turned off" in the TRI-STATE mode,which is insufficient to guarantee a logic "1" input level.

Bonding Option

The COP421L is a bonding option of the COP420L: if the COP420L is bonded as a 24-pin device (without the 4 IN inputs), it becomes the COP421L. The COP421L is pin-compatible with the COP410L.

2.7 COP420C Description

The COP420C is a CMOS version of the COP420. It differs from the COP420 primarily in electrical specifications; however, it also features a dual clock mode option for operation at low speed (typically 244μs instruction cycle time) with low power consumption (25μA with $V_{CC} = 2.4V$) or high speed (15μs instruction cycle time) when necessary to perform internal data computations at a faster rate. The COP420C has the same output drive characteristics as the COP420 (TTL/CMOS compatible) and retains the MICROBUS™ option. The following are the major differences between the COP420C and the COP420:

- Operating voltage of 2.4V to 6.0V.
- Low power consumption at 244μs instruction cycle time (inexpensive 32kHz XTAL ÷ 8) = 25μA at $V_{CC} = 2.4V$.
- Dual clock mode option allowing operation at 16μs instruction cycle time (using external RC network) for internal data computation operations.
- "Fast" clock mode entered under program control.

For further information, see the COP420C data sheet.

2.8 COP444L/COP445L Description

The COP444L/COP445L are expanded-memory versions of the COP420L containing the same internal logic elements and instruction set as the COP420 and COP420L, but with twice the amounts of ROM and RAM. The major differences between the COP444L/COP445Land the COP420L/COP421L are the following:

- Operating supply current less than 11 mA at $V_{CC} = 5V$.
- 2048 × 8 ROM.
- 128 × 4 RAM.

The COP445L is simply a COP444L in a 24-pin dual-in-line package. As a result, the IN inputs are not available on the COP445L, so that the COP445L is pin-compatible with the COP421L and COP410L.

These devices are emulated using the COP404L.

For further information, see the COP444L/445L and/or COP404L data sheets.

2.9 COP402 and COP402M ROM-Less Parts Description

The COP402 and COP402M are ROM-less versions of the COP420. They are packaged in 40-pin packages and are available for prototyping a COP420 system using the COP400 Development System (PDS) or, in quantity, for small volume applications using external ROM.

The COP402 has been mask programmed with options suitable for use as a general controller. COP402 inputs have load devices to V_{CC}, the various outputs have the fullest drive capability associated with them (L outputs = LED direct drive; G and D outputs = standard; SO, SK outputs = push-pull). The COP402 has been programmed for use with an external crystal network, using CKI and CKO, with an instruction cycle time equal to the crystal frequency divided by 16.

The COP402M is the MICROBUS™ compatible version of the COP402. It features the same options as the COP402 with the single exception that the MICROBUS option has been selected. It is, of course, intended for use in prototyping systems or small volume applications which use the microcontroller as a CPU peripheral component, with communication over National's MICROBUS.

2.10 COP404L ROM-Less Part Description

The COP404L is a ROM-less version of the COP444L. It is packaged in a 40-pin package and may be used to prototype all low-power COP400 devices (COP411L, COP410L, COP420L, COP421L, COP444L).

2.11 COP410L/COP411L Architecture

Figure 2.9 provides a block diagram of the COP410L/COP411L. As with the COP420/COP421 block diagram, it depicts the internal logic and interconnects of the device in simplified form. Note that the COP410L is functionally a subset of the 24-pin COP421L. As with the COP421L, it lacks the COP420L IN inputs and the internal IL latches associated with two of these deleted input pins. These and other architectural differences are discussed in the Functional Description, below.

Figure 2.10 shows the Connection Diagrams for the 24-pin COP410L and the 20-pin COP411L. Figure 2.11 provides a pin description for the COP410L/COP411L devices.

See data sheet for the electrical specifications of the COP410L/COP411L, showing maximum ratings plus DC and AC characteristics for these devices.

The COP401L is available for final program verification for a COP410L/COP411L application.

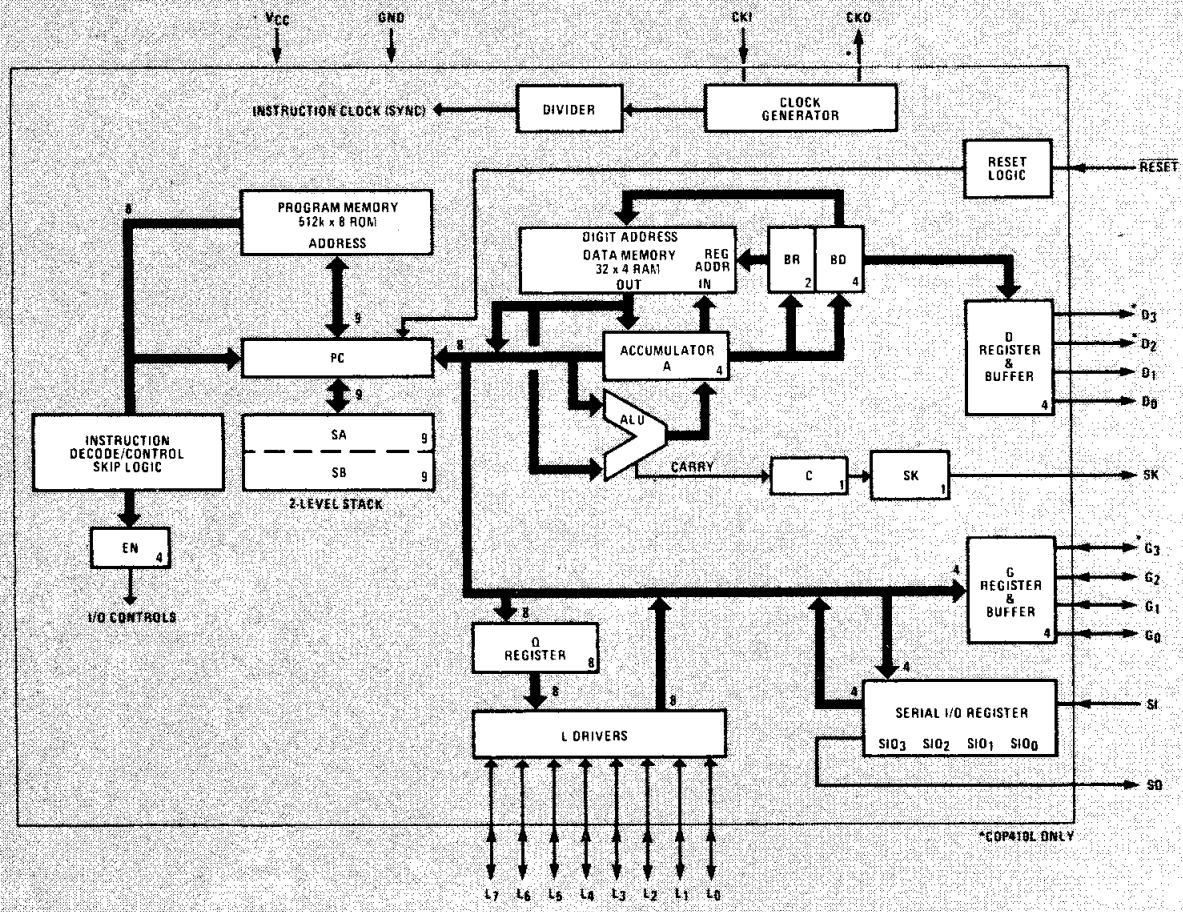


Figure 2.8 COP410L/COP411L Block Diagram

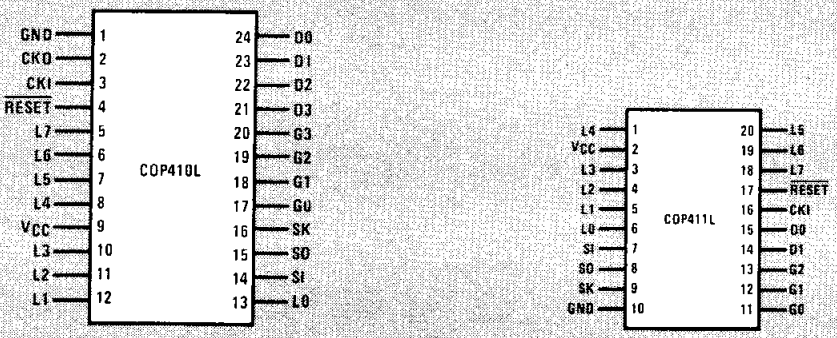


Figure 2.9 COP410L/COP411L Connection Diagrams

L_7-L_0	8 bidirectional I/O ports with TRI-STATE®	CKI	System oscillator input
G_3-G_0	4 bidirectional I/O ports	CKO	System oscillator output (or RAM power supply)
D_3-D_0	4 general purpose outputs	$\overline{\text{RESET}}$	System reset input
SI	Serial input (or counter input)	V_{CC}	Power supply
SO	Serial output (or general purpose output)	GND	Ground
SK	Logic-controlled clock (or general purpose output)		

Figure 2.10 COP410L/COP411L Pin Description

2.12 COP410L/COP411L Functional Description

The following text provides a functional description of the differences which exist between the internal architecture of the COP420, covered in detail in Section 2.2, and that of the COP410L and COP411L. Consequently, for information on logic elements not discussed below which appear in Figure 2.9, COP410L/COP411L Block Diagram, refer to Section 2.2. Where appropriate, differences between the COP410L and its smaller version, the COP411L, are noted in the following text.

Program Memory

Program memory consists of a 512-byte ROM. The same paging characteristics apply to the COP410L/COP411L when allocating program memory instruction code as those which apply to the COP420 (see Section 4.1) except that ROM consists of *8* (0-7) pages of 64 (0-63) words each.

ROM addressing is accomplished by a *9-bit* P register. The auto increment-before-execution and page-rollover features of the COP420 apply to the COP410L/COP411L.

Since the COP410L/COP411L have *2* 9-bit subroutine-save registers, SA and SB, subroutine nesting is allowable to two levels (only one level when executing a LQID instruction since this instruction pushes the stack).

Data Memory

Data memory consists of a 128-bit RAM organized as 4 (0-3) data registers of 8 4-bit digits. Digit addressing is valid only for digits 0, 9-15 in a particular register. (The COP410L/COP411L will, however, treat digit addresses of 1-7 as valid digit values of 9-15, respectively.) As with the COP420, RAM addressing is accomplished by a 6-bit B register whose upper 2 bits (Br) select 1 of 4 data registers and lower 3 bits (Bd) select 1 of *8* 4-bit digits.

A direct access to data memory, without using the B register, is only permissible with respect to M(3, 15) by using an XAD 3, 15 instruction. All other XAD and all LDD instructions have been deleted from the COP410L/COP411L instruction set. Consequently, all other RAM locations must be accessed by loading the B register with the address of data memory to be accessed.

As with the COP420, Bd also may be used as a source register to output its 4-bit contents directly to the D outputs via an OBD instruction.

The Q register functions in a similar manner as the COP420 Q register with the following exceptions:

1. Its contents must be read with the INL instruction, since the CQMA instruction has been deleted.
2. It cannot be loaded with the contents of the L I/O ports since this function is associated with the deleted MICROBUS™ option.

The COP410L/COP411L does not contain the COP420 internal divide-by-1024 time-base counter; hence, the SKT instruction has been deleted. "Real-time" program counters must, therefore, rely on an external time-base input (e.g., 50/60 Hz square wave) to derive a program "clock" for such applications, rather than on the COP410L/COP411L instruction cycle clock itself.

Bit 1 of the EN register (EN_1) is a "don't care" bit, as explained above, due to the lack of a COP410L/COP411L IN_1 input. (The COP420 uses the EN_1 bit to enable IN_1 as an interrupt signal.)

The CASC, ADT and OGI instructions have been deleted. See Section 3.4 for hints on performing these functions.

2.13 COP410L/COP411L Mask Programmable Options

The following text describes the differences which exist between the COP420L mask programmable options and those which are available for the COP410L and COP411L devices.

Available clock oscillator configurations are as follows:

a. Ceramic Resonator Controlled Oscillator. CKI and CKO are connected to an external ceramic resonator. The instruction cycle time equals the resonator frequency (500 kHz maximum) divided by 8. This configuration and its associated options are not available on the 20-pin COP411L since it lacks the CKO pin.

b. External Oscillator. CKI is configured as a Schmitt trigger input (not TTL compatible), accepting an external clock signal. The external frequency (500 kHz maximum) is divided by 8 to derive the instruction cycle time. This option applies to both the COP410L and the COP411L. For the COP410L, moreover, this configuration allows CKO to be used for a RAM power supply (V_R).

c. RC Controlled Oscillator. CKI is configured as a single pin RC controlled Schmitt trigger oscillator. The instruction cycle equals the oscillator (RC time-constant) frequency divided by 4.

d. Externally Synchronized Oscillator. CKO is configured as a synchronizing input from the SK output of another COP400 device. CKI is an external oscillator (divide by 8).

The lower portion of Figure 2.11 provides component values associated with the RC controlled oscillator option.

COP410L CKO Non-Timing Options

In the COP410L resonator controlled configuration, CKO is used as an output to the resonator network. In the other two configurations (external clock and RC controlled), CKO may be mask-programmed as a RAM power supply pin (V_R), allowing its connection to a standby battery backup power supply to maintain the integrity of RAM data with minimum power drain when the main supply is inoperative or shut down to conserve power.

COP410L/COP411L I/O Options

COP410L/COP411L *inputs* and *outputs* have the same optional configurations as the COP420L/COP421L; see Section 2.7.

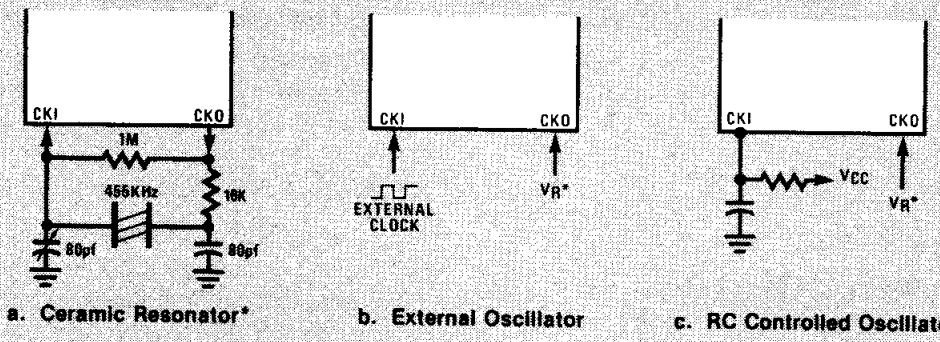
The input and output configurations share common enhancement-mode and depletion-mode devices. For detailed electrical characteristics on these devices, refer to the COP410L and COP421L data sheets.

The SO and SK outputs can be configured as shown in Figure 2.6, a, b, or c. The D and G outputs can be configured as shown in a or b. Note that when inputting data to the G ports, the G outputs should be set to "1." The L outputs can be configured as shown in d, e, f, or g.

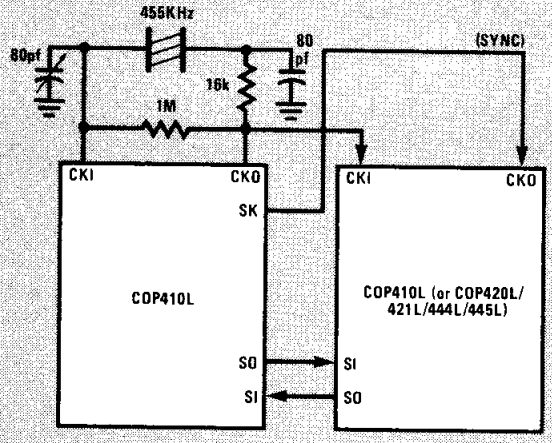
An important point to remember is that *all* of the L driver options are TRI-STATE® -able. Therefore, the L drivers have TRI-STATE-able Standard and Open-Drain output options as well as the TRI-STATE LED Direct Drive and Push-Pull output options. Since the device to V_{CC} in the Standard output configuration is a depletion-mode device, it will source up to 0.125 mA when this output is "turned off" in the TRI-STATE mode, which is insufficient to guarantee a logic "1" input level.

Bonding Option

The COP411L is a bonding option of the COP410L: if the COP410L is bonded as a 20-pin device (without CKO, D_2, D_3, and G_3), it becomes the COP411L. Use of output options associated with these deleted pins are, of course, precluded. All other COP410L options are available.



a. Ceramic Resonator* b. External Oscillator c. RC Controlled Oscillator



*COP410L only.

d. Externally Synchronized Oscillator*

RC Controlled Oscillator

R (kΩ)	C (pF)	Instruction Cycle Time (μs)
51	100	19 ± 15%
82	56	19 ± 15%

Figure 2.11 COP410L/COP411L Oscillator Configurations

3 COP400 Instruction Sets

This chapter provides information on the instruction sets of the COP400 microcontrollers. As with the architecture of the different devices in the COP400 family, the instruction sets of the various devices allow the user to choose among several devices to provide only as much software capability as is needed for a particular application. Specifically, the instruction sets of the various devices are, generally, subsets of the most inclusive instruction set of the COP440. This chapter will discuss the COP420-series (includes COP421, COP421L, COP421C), COP444L, COP410L, and COP411L, respectively. Users of the COP440 should refer to the COP440 data sheet (when the device becomes available) for information on the additional instructions associated with the COP440 instruction set.

This chapter primarily provides information on the machine operations associated with the instruction set of COP400 devices. However, where appropriate, short examples indicating typical usage of particular instructions are provided. For a detailed treatment on using COP400 instructions to write COP400 assembly language programs, see Chapter 4 of this manual.

3.1 COP420-Series/COP444L Instruction Set

Table 3.1 provides the mnemonic, operand, machine code, data flow, skip conditions and description associated with each instruction in the COP420-series/COP444L instruction set. As indicated, an asterisk in the description column signifies a double-byte instruction. Also, notes are provided following this table which describe or refer to additional information relevant to particular instructions. As indicated by Note 3, the INI and INIL instructions are not included in the COP421 instruction set, due to its lack of IN inputs and the IL_3 and IL_0 latches associated with two of the IN inputs (IN_3 and IN_0, respectively).

Note that the COP420-series/COP444L set, as with all COP400 instruction sets, is divided into the following categories: Arithmetic Operations, Input/Output Instructions, Transfer of Control Instructions, Memory Reference Instructions, Register Reference Instructions, and Test Instructions.

Table 3.1 COP420 Series/COP444L Instruction Set

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description
ARITHMETIC INSTRUCTIONS						
ASC		30	0 0 1 1 | 0 0 0 0	A + C + RAM(B) → A Carry → C	Carry	Add with Carry, Skip on Carry
ADD		31	0 0 1 1 | 0 0 0 1	A + RAM(B) → A	None	Add RAM to A
ADT		4A	0 1 0 0 | 1 0 1 0	A + 10_{10} → A	None	Add Ten to A
AISC	y	5–	0 1 0 1 | y	A + y → A	Carry	Add Immediate, Skip on Carry (y ≠ 0)
CASC		10	0 0 0 1 | 0 0 0 0	$\overline{A}$ + RAM(B) + C → A Carry → C	Carry	Complement and Add with Carry, Skip on Carry
CLRA		00	0 0 0 0 | 0 0 0 0	0 → A	None	Clear A
COMP		40	0 1 0 0 | 0 0 0 0	$\overline{A}$ → A	None	Ones complement of A to A
NOP		44	0 1 0 0 | 0 1 0 0	None	None	No Operation
RC		32	0 0 1 1 | 0 0 1 0	"0" → C	None	Reset C
SC		22	0 0 1 0 | 0 0 1 0	"1" → C	None	Set C
XOR		02	0 0 0 0 | 0 0 1 0	A ⊕ RAM(B) → A	None	Exclusive-OR RAM with A
TRANSFER OF CONTROL INSTRUCTIONS						
JID		FF	1 1 1 1 | 1 1 1 1	ROM ($PC_{9:8}$,A,M) → $PC_{7:0}$	None	Jump Indirect (Note 3)
JMP	a	6– ––	0 1 1 0 | 0 | $a_{9:8}$ $a_{7:0}$	a → PC	None	• Jump
JP	a	––	1 | $a_{6:0}$ (pages 2,3 only) or	a → $PC_{6:0}$	None	Jump within Page (Note 4)
		––	1 1 | $a_{5:0}$ (all other pages)	a → $PC_{5:0}$		
JSRP	a	––	1 0 | $a_{5:0}$	PC+1 → SA → SB → SC 0010 → $PC_{9:6}$ a → $PC_{5:0}$	None	Jump to Subroutine Page (Note 5)
JSR	a	6– ––	0 1 1 0 | 1 0 | $a_{9:8}$ $a_{7:0}$	PC+1 → SA → SB → SC a → PC	None	• Jump to Subroutine
RET		48	0 1 0 0 | 1 0 0 0	SC → SB → SA → PC	None	Return from Subroutine
RETSK		49	0 1 0 0 | 1 0 0 1	SC → SB → SA → PC	Always Skip on Return	Return from Subroutine then Skip

Table 3.1 COP420 Series/COP444L Instruction Set (continued)

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description
MEMORY REFERENCE INSTRUCTIONS						
CAMQ		33	0 0 1 1 | 0 0 1 1	A → $Q_{7:4}$	None	* Copy A, RAM to Q
		3C	0 0 1 1 | 1 1 0 0	RAM(B) → $Q_{3:0}$		
CQMA		33	0 0 1 1 | 0 0 1 1	$Q_{7:4}$ → RAM(B)	None	* Copy Q to RAM, A
		2C	0 0 1 0 | 1 1 0 0	$Q_{3:0}$ → A		
LD	r	-5	0 0 | r | 0 1 0 1	RAM(B) → A	None	Load RAM into A,
				Br ⊕ r → Br		Exclusive-OR Br with r
LDD	r,d	23	0 0 1 0 | 0 0 1 1	RAM(r,d) → A	None	* Load A with RAM pointed
		--	0 0 | r | d			to directly by r,d
LQID		BF	1 0 1 1 | 1 1 1 1	ROM($PC_{9:8}$,A,M) → Q	None	Load Q Indirect (Note 3)
				SB → SC		
RMB	0	4C	0 1 0 0 | 1 1 0 0	0 → $RAM(B)_0$	None	Reset RAM Bit
	1	45	0 1 0 0 | 0 1 0 1	0 → $RAM(B)_1$		
	2	42	0 1 0 0 | 0 0 1 0	0 → $RAM(B)_2$		
	3	43	0 1 0 0 | 0 0 1 1	0 → $RAM(B)_3$		
SMB	0	4D	0 1 0 0 | 1 1 0 1	1 → $RAM(B)_0$	None	Set RAM Bit
	1	47	0 1 0 0 | 0 1 1 1	1 → $RAM(B)_1$		
	2	46	0 1 0 0 | 0 1 1 0	1 → $RAM(B)_2$		
	3	4B	0 1 0 0 | 1 0 1 1	1 → $RAM(B)_3$		
STII	y	7-	0 1 1 1 | y	y → RAM(B)	None	Store Memory Immediate
				Bd + 1 → Bd		and Increment Bd
X	r	-6	0 0 | r | 0 1 1 0	RAM(B) ⟷ A	None	Exchange RAM with A,
				Br ⊕ r → Br		Exclusive-OR Br with r
XAD	r,d	23	0 0 1 0 | 0 0 1 1	RAM(r,d) ⟷ A	None	* Exchange A with RAM
		--	1 0 | r | d			pointed to directly by r,d
XDS	r	-7	0 0 | r | 0 1 1 1	RAM(B) ⟷ A	Bd decrements past 0	Exchange RAM with A
				Bd − 1 → Bd		and Decrement Bd,
				Br ⊕ r → Br		Exclusive-OR Br with r
XIS	r	-4	0 0 | r | 0 1 0 0	RAM(B) ⟷ A	Bd increments past 15	Exchange RAM with A
				Bd + 1 → Bd		and Increment Bd,
				Br ⊕ r → Br		Exclusive-OR Br with r
REGISTER REFERENCE INSTRUCTIONS						
CAB		50	0 1 0 1 | 0 0 0 0	A → Bd	None	Copy A to Bd
CBA		4E	0 1 0 0 | 1 1 1 0	Bd → A	None	Copy Bd to A
LBI	r,d	--	0 0 | r | (d − 1)	r,d → B	Skip until not a LBI	Load B Immediate with
			(d = 0, 9:15)			r,d (Note 6)
			or			
		33	0 0 1 1 | 0 0 1 1			
		--	1 0 | r | d			
			(any d)			
LEI	y	33	0 0 1 1 | 0 0 1 1	y → EN	None	* Load EN Immediate
		6-	0 1 1 0 | y			(Note 7)
XABR		12	0 0 0 1 | 0 0 1 0	A ⟷ Br (0,0 → A_3,A_2)	None	Exchange A with Br

Table 3.1 COP420 Series/COP444L Instruction Set (continued)

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description
TEST INSTRUCTIONS						
SKC		20	0 0 1 0 | 0 0 0 0		C = "1"	Skip if C is True
SKE		21	0 0 1 0 | 0 0 0 1		A = RAM(B)	Skip if A Equals RAM
SKGZ		33	0 0 1 1 | 0 0 1 1		$G_{3:0}$ = 0	* Skip if G is Zero (all 4 bits)
		21	0 0 1 0 | 0 0 0 1			
SKGBZ		33	0 0 1 1 | 0 0 1 1	1st byte		* Skip if G Bit is Zero
	0	01	0 0 0 0 | 0 0 0 1	2nd byte	G_0 = 0	
	1	11	0 0 0 1 | 0 0 0 1		G_1 = 0	
	2	03	0 0 0 0 | 0 0 1 1		G_2 = 0	
	3	13	0 0 0 1 | 0 0 1 1		G_3 = 0	
SKMBZ	0	01	0 0 0 0 | 0 0 0 1		$RAM(B)_0$ = 0	Skip if RAM Bit is Zero
	1	11	0 0 0 1 | 0 0 0 1		$RAM(B)_1$ = 0	
	2	03	0 0 0 0 | 0 0 1 1		$RAM(B)_2$ = 0	
	3	13	0 0 0 1 | 0 0 1 1		$RAM(B)_3$ = 0	
SKT		41	0 1 0 0 | 0 0 0 1		A time-base counter carry has occurred since last test	Skip on Timer (Note 3)
INPUT/OUTPUT INSTRUCTIONS						
ING		33	0 0 1 1 | 0 0 1 1	G → A	None	* Input G Ports to A
		2A	0 0 1 0 | 1 0 1 0			
ININ		33	0 0 1 1 | 0 0 1 1	IN → A	None	* Input IN Inputs to A (Note 2)
		28	0 0 1 0 | 1 0 0 0			
INIL		33	0 0 1 1 | 0 0 1 1	IL_3, "1", "0", IL_0 → A	None	* Input IL Latches to A (Note 3)
		29	0 0 1 0 | 1 0 0 1			
INL		33	0 0 1 1 | 0 0 1 1	$L_{7:4}$ → RAM(B)	None	* Input L Ports to RAM, A
		2E	0 0 1 0 | 1 1 1 0	$L_{3:0}$ → A		
OBD		33	0 0 1 1 | 0 0 1 1	Bd → D	None	* Output Bd to D Outputs
		3E	0 0 1 1 | 1 1 1 0			
OGI	y	33	0 0 1 1 | 0 0 1 1	y → G	None	* Output to G Ports Immediate
		5–	0 1 0 1 | y			
OMG		33	0 0 1 1 | 0 0 1 1	RAM(B) → G	None	* Output RAM to G Ports
		3A	0 0 1 1 | 1 0 1 0			
XAS		4F	0 1 0 0 | 1 1 1 1	A ⟷ SIO, C → SK	None	Exchange A with SIO (Note 3)

Note 1: All subscripts for alphabetical symbols indicate bit numbers unless explicitly defined (e.g., Br and Bd are explicitly defined). Bits are numbered 0 to N where 0 signifies the least significant (low-order, right-most bit). For example, A_3 indicates the most significant (left-most) bit of the 4-bit A register.

Note 2: The ININ instruction is not available on the 24-pin COP421 since this device does not contain the IN inputs.

Note 3: For additional information on the operation of the XAS, JID, LQID, INIL, and SKT instructions, see Section 3.2.

Note 4: The JP instruction allows a jump, while in subroutine pages 2 or 3, to any ROM location within the two-page boundary of pages 2 or 3. The JP instruction, otherwise, permits a jump to a ROM location within the current 64-word page. JP may not jump to the last word of a page.

Note 5: A JSRP transfers program control to subroutine page 2 (0010 is loaded into the upper 4 bits of P). A JSRP may not be used when in pages 2 or 3. JSRP may not jump to the last word in page 2.

Note 6: LBI is a single-byte instruction if d = 0, 9, 10, 11, 12, 13, 14, or 15. The machine code for the lower 4 bits equals the binary value of the "d" data *minus 1*, e.g., to load the lower four bits of B (Bd) with the value 9 (1001_2), the lower 4 bits of the LBI instruction equal 8 (1000_2). To load 0, the lower 4 bits of the LBI instruction should equal 15 (1111_2).

Note 7: Machine code for operand field y for LEI instruction should equal the binary value to be latched into EN, where a "1" or "0" in each bit of EN corresponds with the selection or deselection of a particular function associated with each bit. (See Functional Description, EN Register.)

Table 3.2 provides a list of internal architecture, instruction operand and operational symbols used in the COP420-series/COP444L Instruction Set Table. Table 3.5 shows an alphabetical mnemonic index of COP420-series/COP444L instructions, indicating the hexadecimal opcode and description associated with each instruction. Table 3.6 is a list of COP420-series/COP444L instructions arranged in order of their hexadecimal opcodes.

The following text gives a description of each COP420-series/COP444L instruction, explaining the machine operations performed by each instruction and, where appropriate, providing short examples illustrating typical usage of particular instructions.

Table 3.2 COP420-Series/COP444L Instruction Set Table Symbols

Symbol	Definition
INTERNAL ARCHITECTURE SYMBOLS	
A	4-bit Accumulator
B	6-bit RAM Address Register
Br	Upper 2 bits of B (register address)
Bd	Lower 4 bits of B (digit address)
C	1-bit Carry Register
D	4-bit Data Output Port
EN	4-bit Enable Register
G	4-bit Register to latch data for G I/O Port
IL	Two 1-bit Latches associated with the IN_3 or IN_0 Inputs
IN	4-bit Input Port
L	8-bit TRI-STATE I/O Port
M	4-bit contents of RAM Memory pointed to by B Register
PC	10-bit ROM Address Register (program counter)
Q	8-bit Register to latch data for L I/O Port
SA	10-bit Subroutine Save Register A
SB	10-bit Subroutine Save Register B
SC	10-bit Subroutine Save Register C
SIO	4-bit Shift Register and Counter
SK	Logic-Controlled Clock Output

Symbol	Definition
INSTRUCTION OPERAND SYMBOLS	
d	4-bit Operand Field, 0-15 binary (RAM Digit Select)
r	2-bit Operand Field, 0-3 binary (RAM Register Select)
a	10-bit Operand Field, 0-1023 binary (ROM Address)
y	4-bit Operand Field, 0-15 binary (Immediate Data)
RAM(s)	Contents of RAM location addressed by s
ROM(t)	Contents of ROM location addressed by t
OPERATIONAL SYMBOLS	
+	Plus
−	Minus
→	Replaces
⟷	Is exchanged with
=	Is equal to
$\overline{A}$	The ones complement of A
⊕	Exclusive-OR
:	Range of values

3.2 COP420-Series/COP444L Instruction Set Description

Arithmetic Instructions

ASC (Add with carry, Skip on Carry) performs a binary addition of A, C (Carry bit), and M, placing the result in A and C. If a carry occurs, the next program instruction is skipped.

ADD (ADD) performs binary addition. The 4-bit addends are A and M. The 4-bit sum is placed in A. ADD does not affect the carry or skip.

ADT (ADd Ten to A) adds ten (1010_2) to A and, like ADD, does not affect the carry or skip. It is intended to facilitate Binary Coded Decimal (BCD) arithmetic. For example, the following sequence of instructions will perform a single-digit BCD add of the contents of A and M [the carry is assumed set when entering this routine if addition of the previous least significant digits produced an overflow (A > 9)]:

```
AISC 6
ASC
ADT
```

The AISC 6 instruction adds a BCD correction factor (i.e., 6) to the digit in the accumulator. (See AISC instruction.) Since the accumulator contains a BCD digit (≤ 9) no carry will occur and the next instruction, ASC, will always be executed. The ASC instruction adds the carry and memory digit to A, as explained above. If the result does *not* produce a carry, signifying that the previous AISC 6 (correction factor) instruction was unnecessary, the ADT instruction is executed, readjusting the accumulator to the proper BCD result. (Remember: ADT neither affects the carry nor skips.)

If the ASC result does produce a carry, C is set for propagation to the addition of the next most significant digits and, since no readjustment of the result is necessary, the ADT instruction is skipped.

AISC (Add Immediate, Skip on Carry) adds the instruction operand constant "y" (1-15) to A, skipping the next instruction if a carry out occurs (C is *not* changed). This instruction finds frequent use in BCD add and subtract routines (see ADT and CASC descriptions) as well as in testing the value of A. (If A is greater than 12, for instance, an AISC 5 will skip the next instruction.)

CASC (Complement and Add, Skip on Carry) performs a binary subtraction of A from M by summing the complement of A ($\overline{A}$) with C and M, placing the result in A and C. If no carry out occurs, indicating a borrow, C is reset and the next instruction is executed. If a carry occurs, indicating no borrow, C is set and the next instruction is skipped.

A single BCD digit binary subtraction of A from M may be performed as follows. (The carry bit is assumed set upon initial entry to the routine.)

```
CASC
ADT
```

The CASC instruction will set C and skip the ADT instruction if the subtraction does not result in a borrow (A > M). If a borrow occurs, the ADT instruction is executed, readjusting the result to the proper BCD value, leaving C reset for propagation of the borrow in the subtraction of the next most significant BCD digits. CASC is functionally equivalent to a COMP instruction followed by an ASC.

CLRA (CLeaR A) clears the accumulator by placing zeros in each of the 4 bits of A.

This instruction is often required prior to loading A equal to a desired value with an AISC instruction if the previous contents of A are unknown. For instance, to load A = 11, the following sequence may be used:

```
CLRA
AISC  11
```

The skip features associated with AISC need not be considered in this example. (A carry will never occur.)

COMP (COMPlement A) changes the state of each of 4 bits of A with ones becoming zeros and zeros becoming ones. It has the effect of, and may be used to perform, a binary (one's complement) subtraction of A from 15 (1111_2), e.g., complementing A = 6 (0110_2) will yield 9 (1001_2).

NOP (No OPeration) does not perform any operation. It is useful, however, for simple single instruction time delays or to defeat the skip conditions associated with particular instructions.

SC (Set Carry) and **RC** (Reset Carry) set C and reset C, respectively. SC and RC are most often employed to initialize C prior to entering arithmetic routines. They also allow C to be used as a general-purpose (testable) flag, as long as subsequent instructions do not inadvertently affect the C register.

XOR (eXclusive-OR A with M) performs a logical EXCLUSIVE-OR operation of each bit of A with each corresponding bit of M, *placing the result in A*. This operation can be used to change the state of any bit in M, if the corresponding (equally weighted) bit of A is set. This follows from the EXCLUSIVE-OR truth table where a X + "1" = $\overline{X}$, and a X + "0" = X, assuming the "X" bits to be one of the 4 bits in M, and the "1" and "0" to be equally weighted bits in A. This instruction, therefore, allows the selective complementing or toggling of one or more bits of M. Example: to change the state of bit 2 of M, set A = 0100, perform an XOR, then exchange A into M with an X instruction.

Input/Output Instructions

ING (INput G ports to A) transfers the 4-bit contents of the IN ports (IN_3-IN_0) to A.

ININ (INput IN inputs to A) transfers the 4-bit contents of the IN ports (IN_3-IN_0) to A.

INIL (INput IL latches to A) is a special purpose instruction which inputs the two latches IL_3 and IL_0 (see Figure 3.1 below) and, if the appropriate option is selected, a general-purpose input, CKO, to the accumulator — the unused bit/bits of A are reset. Specifically, INIL places $IL_3 \rightarrow A_3$, CKO $\rightarrow A_2$, "0" $\rightarrow A_1$, $IL_0 \rightarrow A_0$. IL_3 and IL_0 are the outputs of latches associated with the IN_3 and IN_0 inputs. (The *general purpose* inputs, IN_3-IN_0, are input to A upon the execution of an ININ instruction. (See ININ Instruction.) The IL_3 and IL_0 latches are set if a low-going pulse ("1" to "0") has occurred on the IN_3 and IN_0 inputs, respectively, since the last INIL instruction, provided the input pulse stays low for at least two instruction times. Execution of an INIL inputs IL_3 and IL_0 into A_3 and A_0 respectively, and resets these latches to allow them to respond to subsequent low-going pulses on the IN_3 and IN_0 lines. These latches are not cleared during a power on reset.

If CKO is mask-programmed as a general-purpose input, an INIL will input the state of CKO into A_2. If CKO has not been so programmed, a "1" will be placed in A_2. A "0" is always placed in A_1 upon the execution of an INIL.

INIL is useful in recognizing and capturing pulses of short duration or which can't be read conveniently by an ININ instruction.

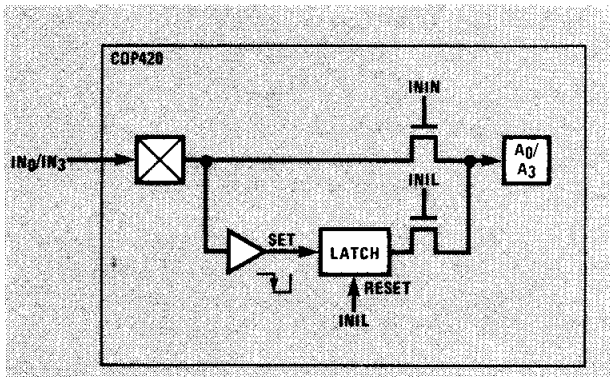


Figure 3.1 INIL Hardware Implementation

INL (INput L ports to M, A) transfers the 8-bit contents of the bidirectional TRI-STATE® I/O ports to M, A. L_7-L_4 are placed in M_3-M_0 (the memory digit pointed to by the B register); L_3-L_0 are placed in A_3-A_0.

OBD (Output Bd to D outputs) transfers the 4-bit contents of Bd (lower 4 bits of the B register) to the D output ports (D_3-D_0). Since, in many applications, the D outputs are connected to a digit decoder, the direct output of Bd allows for a standard interconnect to the binary inputs of the decoder/driver device.

OGI (Output to G ports Immediate) transfers the four bits specified in the "y" operand field of this instruction (0-15, binary) to G_3-G_0.

OMG (Output M to G ports) transfers the 4-bit contents of M (M_3-M_0) to G_3-G_0.

XAS (eXchange A with SIO) exchanges the 4-bit contents of A (A_3-A_0) with the 4-bit contents of the SIO register (SIO_3-SIO_0). SIO will contain serial-in/serial-out shift register or binary counter data, depending on the value of the EN register. An XAS instruction will also affect the SK output. The XAS instruction copies C into the SKL latch. In the counter mode, SK is the output of SKL; in the shift register mode, SK outputs SKL ANDed with the clock.

For further information on the EN register and its relationship to the XAS instruction, see LEI Instruction, below. If SIO is selected as a shift register, an XAS instruction must be performed once every 4 instruction cycle times to effect a continuous serial-in or serial-out data stream.

Transfer of Control Instructions

JID (Jump InDirect) is an indirect addressing instruction, transferring program control to a new ROM location addrssed by the *contents* of the ROM location pointed to by A and M. Specifically, it loads the lower 8 bits of the ROM address register P with the *contents* of ROM pointed to by the 10-bit word $P_9P_8A_3A_2A_1A_0M_3M_2M_1M_0$. The contents of the selected ROM location (I_7-I_0) are, therefore, loaded into P_7-P_0, changing the lower 8 bits of P to transfer program control to the new ROM location.

P_9 and P_8 remain unchanged throughout the execution of the JID instruction. JID, therefore, may only jump to a ROM location within the current 4-page ROM "block" (pages 0-3, 4-7, 8-11 or 12-15). For further information regarding the "paging" restrictions associated with the JID instruction, see Section 4.1.

JID can be useful in keyboard-decode routines when the values associated with the row and column of a particular key closure are placed in A and M for a jump indirect to the contents of ROM which point to the starting address of the appropriate routine associated with that particular key closure. For an example of use of the JID instruction to access a keyboard-decode ROM pointer table, see Display/Keyboard Program, Section 5.3, #16.

JMP (JuMP) transfers program control to any word in the ROM as specified by the "a" field of this instruction. The 10-bit "a" field is placed in P_9-P_0. JMP is used to transfer program control from one page to *another* page (if in page 2 or 3, the more efficient single-byte JP instruction may be used) or to transfer control to the *last* word of the current page — an invalid transfer for the JP instruction.

JP (Jump within Page) transfers program control to the ROM address specified in the operand field of this instruction. The machine code and operand field of this instruction have two formats. If program execution is currently within page 2 or 3 (subroutine pages) a 7-bit "a" field is specified, transferring program control to a word within either of the two subroutine pages. Otherwise, only a 6-bit "a" field is specified, transferring program control to a particular word within the *current* 64-word ROM page.

Specifically, this instruction places a_6-a_0 in P_6-P_0 if the program is currently in subroutine page 2 or 3. If in any other page, it places a_5-a_0 in P_5-P_0.

The restrictions associated with the JP instruction, therefore, are that a 7-bit "a" field may be used only when in pages 2 or 3. Otherwise, a JP may be used only to jump within the current page by specifying a 6-bit "a" field in the operand of this instruction. An additional restriction associated with the JP instruction, in either of the above two formats, is that a JP to the last word of any page is invalid, i.e., "a" may not equal all 1s. A transfer of program control to last word on a page may be effected by using a JMP instruction. (See JMP Instruction, above.)

JSRP (Jump to SubRoutine Page) is used to transfer program control from a page other than 2 or 3 to a word within page 2. It accomplishes this by placing a 2 (0010_2) in P_9-P_6, and the word address specified in the 6-bit "a" field of the instruction into P_5-P_0. Designed to transfer control to subroutines, it *pushes the stack* to save the subroutine return address — the address of the next program instruction is saved in SA and the other subroutine-save registers are likewise pushed ($P + 1 \rightarrow SA \rightarrow SB \rightarrow SC$). Any previous contents of SC are lost, since SC is the last of the three subroutine-save registers. Subroutine nesting, therefore, is permitted to three levels. JSRP is used in conjunction with the RET or RETSK instructions which "pop" the stack at the end of subroutine to return program control to the main program. As with the JP instruction, JSRP may not transfer program control to the last word of page 2: "a" may not equal all "1s." A JSR may be used to jump to the last word of a subroutine beginning at the last word of page 2. (See JSR, below.) As mentioned above, a further restriction is that a

JSRP may not be used when in subroutine pages 2 or 3. To transfer program control to a subroutine in page 2 when in pages 2 or 3, the double-byte JSR should be used, or, if it is not necessary to push the stack, a JP instruction may be used.

JSR (Jump to SubRoutine) transfers program control to a subroutine located at a particular word address in *any* ROM page. It modifies the entire P register with the value of the "a" operand of this instruction, as follows: a_9-$a_0 \rightarrow P_9$-P_0. As with the JSRP instruction, JSR pushes the stack ($P+1 \rightarrow SA \rightarrow SB \rightarrow SC$), saving the next program instruction for a return from the subroutine to the main program via a RET or RETSK instruction. JSR may be used to overcome the restrictions associated with the JSRP instruction: to jump to a subroutine and push the stack when in pages 2 or 3, or to jump to a subroutine located at the last word of page 2.

RET (RETurn from subroutine) is used to return program control to the main program following a JSR or JSRP instruction. RET "pops" the stack ($SC \rightarrow SB \rightarrow SA \rightarrow P$): the next main program instruction address ($P+1$) saved in SA is loaded into P, the contents of SB are loaded into SA and the contents of SC are loaded into SB. (The contents of SC are also retained in SC.) Program control, therefore, is returned to the instruction immediately following the previous subroutine call.

RETSK (RETurn from subroutine then SKip), as with the RET instruction above, pops the stack ($SC \rightarrow SB \rightarrow SA \rightarrow P$), restoring program control to the main program following a subroutine call. It, however, *always* skips the first instruction encountered when it returns to the main program. This instruction, therefore, provides the programmer with an alternate return from subroutines, either via a RET or RETSK, based upon tests made within the subroutine itself.

CAMQ (Copy A, M to Q) transfers the 8-bit contents of A and M to the Q latches. A_3-A_0 are output to Q_7-Q_4; M_3-M_0 are output to Q_3-Q_0. Note that CAMQ is the inverse of CQMA (see CQMA Instruction, below) with respect to the 4 bits of Q with which A and M communicate. Therefore, the input and processing of Q must often be followed by an X (Exchange M with A) instruction before final output to Q in order to maintain the proper bit-weights of the Q data. For example, the following instructions read Q to M, A, set Q_7 and perform the necessary exchange before execution of the CAMQ instruction:

```
CQMA      ; Q TO M, A
SMB  3    ; SET Q7 BIT LOCATED IN M3
X         ; EXCHANGE M WITH A
CAMQ      ; A, M TO Q
```

CQMA (Copy Q to M, A) transfers the 8-bit contents of the Q latches to M and A. Q_7-Q_4 are placed in M_3-M_0; Q_3-Q_0 are placed in A_3-A_0. CQMA can be employed after an LQID (Load Q InDirect) instruction to input or alter the value of lookup data. CQMA is also an essential instruction when the COP420 is employed as a MICROBUS™ peripheral component. In such applications, IN_3 is used by the control microprocessor to write bus data from the L ports to the Q latches. (See Section 2.4, MICROBUS™ option.) A CQMA will then input this data to M, A as explained above for processing by the COP420 program.

Memory Reference Instructions

LD (LoaD M into A) loads M (the 4-bit contents of RAM pointed to by the B register: M_3-M_0) into A_3-A_0. After M is loaded into A, the 2-bit "r" operand field is EXCLUSIVE-ORed with the contents of Br (upper 2 bits of B — RAM register select) to point to a new RAM register for successive memory reference operations. Since the properties of the EXCLUSIVE-OR logic operation are such that a $1 \oplus X$ equals the complement of X, use of the "r" field allows the programmer to switch between any one of the 4 RAM registers by complementing the appropriate bit/bits of the current contents of the Br register. Of course, if "r" = 0, the contents of Br will remain unchanged after the execution of a LD instruction.

For example, if the assembly language instruction LD 3 ("r" = 11_2) is executed with Br = 2 (10_2) and Bd = 12 (1100_2), the contents of RAM register 2, digit 12 will be loaded to A and Br will be changed to ($11_2 + 10_2 = 01_2$), with B pointing to RAM register 1, digit 12. For assembly language programming use of an EXCLUSIVE-OR "r" operand field with memory reference instructions which use this field is optional — if not specified, an "0" operand is assumed. For further information on allocating RAM map locations for optimum use of the EXCLUSIVE-OR feature associated with this and other memory reference instructions and for sample routines utilizing this feature, refer to Sections 4.2 and 4.4.

SMB (Set Memory Bit) and **RMB** (Reset Memory Bit) set and reset, respectively, a bit in M as specified by the operand field of these instructions. (Remember: M is the 4-bit RAM digit pointed to by the B register.) The operand field is specified according to the bit number (0-3, left-most to right-most bit) of the particular bit to be set or reset, e.g., an SMB 3 would set the most significant bit of M. These instructions are useful in operating upon program status flags located in RAM.

STII (Store Memory Immediate and Increment Bd) loads the 4-bit contents specified by the "y"

OBD (Output Bd to D outputs) transfers the 4-bit contents of Bd (lower 4 bits of the B register) to the D output ports (D_3-D_0). Since, in many applications, the D outputs are connected to a digit decoder, the direct output of Bd allows for a standard interconnect to the binary inputs of the decoder/driver device.

OGI (Output to G ports Immediate) transfers the four bits specified in the "y" operand field of this instruction (0-15, binary) to G_3-G_0.

OMG (Output M to G ports) transfers the 4-bit contents of M (M_3-M_0) to G_3-G_0.

XAS (eXchange A with SIO) exchanges the 4-bit contents of A (A_3-A_0) with the 4-bit contents of the SIO register (SIO_3-SIO_0). SIO will contain serial-in/serial-out shift register or binary counter data, depending on the value of the EN register. An XAS instruction will also affect the SK output. The XAS instruction copies C into the SKL latch. In the counter mode, SK is the output of SKL; in the shift register mode, SK outputs SKL ANDed with the clock.

For further information on the EN register and its relationship to the XAS instruction, see LEI Instruction, below. If SIO is selected as a shift register, an XAS instruction must be performed once every 4 instruction cycle times to effect a continuous serial-in or serial-out data stream.

Transfer of Control Instructions

JID (Jump InDirect) is an indirect addressing instruction, transferring program control to a new ROM location addrssed by the *contents* of the ROM location pointed to by A and M. Specifically, it loads the lower 8 bits of the ROM address register P with the *contents* of ROM pointed to by the 10-bit word $P_9P_8A_3A_2A_1A_0M_3M_2M_1M_0$. The contents of the selected ROM location (I_7-I_0) are, therefore, loaded into P_7-P_0, changing the lower 8 bits of P to transfer program control to the new ROM location.

P_9 and P_8 remain unchanged throughout the execution of the JID instruction. JID, therefore, may only jump to a ROM location within the current 4-page ROM "block" (pages 0-3, 4-7, 8-11 or 12-15). For further information regarding the "paging" restrictions associated with the JID instruction, see Section 4.1.

JID can be useful in keyboard-decode routines when the values associated with the row and column of a particular key closure are placed in A and M for a jump indirect to the contents of ROM which point to the starting address of the appropriate routine associated with that particular key closure. For an example of use of the JID instruction to access a keyboard-decode ROM pointer table, see Display/Keyboard Program, Section 5.3, #16.

JMP (JuMP) transfers program control to any word in the ROM as specified by the "a" field of this instruction. The 10-bit "a" field is placed in P_9-P_0. JMP is used to transfer program control from one page to *another* page (if in page 2 or 3, the more efficient single-byte JP instruction may be used) or to transfer control to the *last* word of the current page — an invalid transfer for the JP instruction.

JP (Jump within Page) transfers program control to the ROM address specified in the operand field of this instruction. The machine code and operand field of this instruction have two formats. If program execution is currently within page 2 or 3 (subroutine pages) a 7-bit "a" field is specified, transferring program control to a word within either of the two subroutine pages. Otherwise, only a 6-bit "a" field is specified, transferring program control to a particular word within the *current* 64-word ROM page.

Specifically, this instruction places a_6-a_0 in P_6-P_0 if the program is currently in subroutine page 2 or 3. If in any other page, it places a_5-a_0 in P_5-P_0.

The restrictions associated with the JP instruction, therefore, are that a 7-bit "a" field may be used only when in pages 2 or 3. Otherwise, a JP may be used only to jump within the current page by specifying a 6-bit "a" field in the operand of this instruction. An additional restriction associated with the JP instruction, in either of the above two formats, is that a JP to the last word of any page is invalid, i.e., "a" may not equal all 1s. A transfer of program control to last word on a page may be effected by using a JMP instruction. (See JMP Instruction, above.)

JSRP (Jump to SubRoutine Page) is used to transfer program control from a page other than 2 or 3 to a word within page 2. It accomplishes this by placing a 2 (0010_2) in P_9-P_6, and the word address specified in the 6-bit "a" field of the instruction into P_5-P_0. Designed to transfer control to subroutines, it *pushes the stack* to save the subroutine return address — the address of the next program instruction is saved in SA and the other subroutine-save registers are likewise pushed (P + 1 → SA → SB → SC). Any previous contents of SC are lost, since SC is the last of the three subroutine-save registers. Subroutine nesting, therefore, is permitted to three levels. JSRP is used in conjunction with the RET or RETSK instructions which "pop" the stack at the end of subroutine to return program control to the main program. As with the JP instruction, JSRP may not transfer program control to the last word of page 2: "a" may not equal all "1s." A JSR may be used to jump to the last word of a subroutine beginning at the last word of page 2. (See JSR, below.) As mentioned above, a further restriction is that a

JSRP may not be used when in subroutine pages 2 or 3. To transfer program control to a subroutine in page 2 when in pages 2 or 3, the double-byte JSR should be used, or, if it is not necessary to push the stack, a JP instruction may be used.

JSR (Jump to SubRoutine) transfers program control to a subroutine located at a particular word address in *any* ROM page. It modifies the entire P register with the value of the "a" operand of this instruction, as follows: a_9-$a_0 \rightarrow P_9$-P_0. As with the JSRP instruction, JSR pushes the stack ($P+1 \rightarrow SA \rightarrow SB \rightarrow SC$), saving the next program instruction for a return from the subroutine to the main program via a RET or RETSK instruction. JSR may be used to overcome the restrictions associated with the JSRP instruction: to jump to a subroutine and push the stack when in pages 2 or 3, or to jump to a subroutine located at the last word of page 2.

RET (RETurn from subroutine) is used to return program control to the main program following a JSR or JSRP instruction. RET "pops" the stack ($SC \rightarrow SB \rightarrow SA \rightarrow P$): the next main program instruction address ($P+1$) saved in SA is loaded into P, the contents of SB are loaded into SA and the contents of SC are loaded into SB. (The contents of SC are also retained in SC.) Program control, therefore, is returned to the instruction immediately following the previous subroutine call.

RETSK (RETurn from subroutine then SKip), as with the RET instruction above, pops the stack ($SC \rightarrow SB \rightarrow SA \rightarrow P$), restoring program control to the main program following a subroutine call. It, however, *always* skips the first instruction encountered when it returns to the main program. This instruction, therefore, provides the programmer with an alternate return from subroutines, either via a RET or RETSK, based upon tests made within the subroutine itself.

CAMQ (Copy A, M to Q) transfers the 8-bit contents of A and M to the Q latches. A_3-A_0 are output to Q_7-Q_4; M_3-M_0 are output to Q_3-Q_0. Note that CAMQ is the inverse of CQMA (see CQMA Instruction, below) with respect to the 4 bits of Q with which A and M communicate. Therefore, the input and processing of Q must often be followed by an X (Exchange M with A) instruction before final output to Q in order to maintain the proper bit-weights of the Q data. For example, the following instructions read Q to M, A, set Q_7 and perform the necessary exchange before execution of the CAMQ instruction:

```
CQMA      ; Q TO M, A
SMB  3    ; SET Q7 BIT LOCATED IN M3
X         ; EXCHANGE M WITH A
CAMQ      ; A, M TO Q
```

CQMA (Copy Q to M, A) transfers the 8-bit contents of the Q latches to M and A. Q_7-Q_4 are placed in M_3-M_0; Q_3-Q_0 are placed in A_3-A_0. CQMA can be employed after an LQID (Load Q InDirect) instruction to input or alter the value of lookup data. CQMA is also an essential instruction when the COP420 is employed as a MICROBUS™ peripheral component. In such applications, IN_3 is used by the control microprocessor to write bus data from the L ports to the Q latches. (See Section 2.4, MICROBUS™ option.) A CQMA will then input this data to M, A as explained above for processing by the COP420 program.

Memory Reference Instructions

LD (LoaD M into A) loads M (the 4-bit contents of RAM pointed to by the B register: M_3-M_0) into A_3-A_0. After M is loaded into A, the 2-bit "r" operand field is EXCLUSIVE-ORed with the contents of Br (upper 2 bits of B — RAM register select) to point to a new RAM register for successive memory reference operations. Since the properties of the EXCLUSIVE-OR logic operation are such that a $1 \oplus X$ equals the complement of X, use of the "r" field allows the programmer to switch between any one of the 4 RAM registers by complementing the appropriate bit/bits of the current contents of the Br register. Of course, if "r" = 0, the contents of Br will remain unchanged after the execution of a LD instruction.

For example, if the assembly language instruction LD 3 ("r" = 11_2) is executed with Br = 2 (10_2) and Bd = 12 (1100_2), the contents of RAM register 2, digit 12 will be loaded to A and Br will be changed to ($11_2 + 10_2 = 01_2$), with B pointing to RAM register 1, digit 12. For assembly language programming use of an EXCLUSIVE-OR "r" operand field with memory reference instructions which use this field is optional — if not specified, an "0" operand is assumed. For further information on allocating RAM map locations for optimum use of the EXCLUSIVE-OR feature associated with this and other memory reference instructions and for sample routines utilizing this feature, refer to Sections 4.2 and 4.4.

SMB (Set Memory Bit) and **RMB** (Reset Memory Bit) set and reset, respectively, a bit in M as specified by the operand field of these instructions. (Remember: M is the 4-bit RAM digit pointed to by the B register.) The operand field is specified according to the bit number (0-3, left-most to right-most bit) of the particular bit to be set or reset, e.g., an SMB 3 would set the most significant bit of M. These instructions are useful in operating upon program status flags located in RAM.

STII (Store Memory Immediate and Increment Bd) loads the 4-bit contents specified by the "y"

operand field of the instruction into the RAM memory digit pointed to by the B register, M_3-M_0. It is important to note that the value of Bd (RAM digit-select) is *incremented* (as with the XIS instruction) after the "y" data is stored in M.

LDD (LoaD A with M Directly) loads the 4-bit contents of the RAM memory location pointed to directly by the "r" and "d" operand fields (register and digit select, respectively) of the instruction, M_3-M_0, into A_3-A_0. Note that this instruction and the XAD instruction differ from other memory reference instructions in that the operand of the instruction, not the B register, is used to point to the appropriate RAM digit location to be accessed — the B register is unaffected by these instructions. This instruction is useful in accessing RAM counters, status and flag digits, etc., within routines or loops without destroying the previous value of B, allowing the latter to be used for sequential memory access operations and for other reiterative purposes.

LQID (Load Q InDirect) is, in effect, a ROM data "lookup" instruction. It transfers the 8-bit contents of ROM, I_7-I_0, pointed to by the 10-bit word P_9P_8AM to Q_7-Q_0, respectively. It does this by pushing the stack (P + 1 → SA → SB → SC) and replacing the least significant 8 bits of P as follows: A_3-A_0 → P_7-P_4; M_3-M_0 → P_3-P_0, leaving the two most significant bits of P unchanged. The ROM data pointed to by the new P address is fetched and loaded into the Q latches, Q_7-Q_0. Next, the stack is popped (SC → SB → SA → P), restoring the previous pushed value of P (P + 1) to continue sequential program execution. Since LQID pushes SB → SC, the previous contents of SC are lost. Also, when LQID pops the stack, the previously pushed contents of SB are left in SC as well as loaded back into SB. The net result, therefore, of an LQID instruction upon the subroutine-save stack is that the contents of SB are placed in SC (SB → SC). Since it pushes the stack, a LQID should not be executed when *three* levels of subroutine nesting are currently in effect. (The last return address in SC will be lost.)

Since, as with the JID instruction, LQID affects only the lower 8 bits of P (P_9 and P_8 are unchanged), it may only access ROM data located within the current 4-page ROM "block" (pages 0-3, 4-7, 8-11 or 12-15). For further information on the use of the LQID instruction, see Section 4.1.

X (eXchange M with A) exchanges the 4-bit contents of RAM pointed to by the B register, M_3-M_0, with A_3-A_0. The "r" operand field of the instruction is EXCLUSIVE-ORed with the contents of Br after the exchange to provide a new Br RAM register select value as explained in the LD instruction above.

XAD (eXchange A with M Directly) exchanges the 4-bit contents of the RAM memory location pointed to directly by the "r" and "d" operand fields of the instruction, M_3-M_0, with A_3-A_0. It has the same characteristics and utility as the LDD instruction above, e.g., the B register is not affected.

XDS (eXchange M with A, Decrement Bd and Skip on borrow) performs the same operation as the X instruction above, and also decrements the value of the Bd register (RAM digit-select) *after* the exchange. Use of an "r" operand field will, therefore, result in both an altered RAM *digit*-select value and a new RAM *register* select value in B. XDS skips the next program instruction when Bd is decremented *past* 0 (after the contents of RAM digit 0 have been exchanged with A and XDS decrements Bd to 15). Repeated XDSs will "walk down" through the digits of a RAM register before skipping. XDS together with X instructions can be used to operate upon the corresponding digits of different RAM registers in successive fashion. (See Section 4.2.)

XIS (eXchange M with A, Increment Bd, and Skip on carry) performs the same operation as the XDS instruction except that it *increments* Bd *after* the exchange and skips the next program instruction after Bd increments *past* 15 (after the contents of RAM digit 15 have been exchanged with A and XIS increments Bd to 0). Consequently, successive XISs "walk up" through the digits of a RAM register before skipping.

Register Reference Instructions

CAB (Copy A to Bd) transfers the 4-bit contents of A, A_3-A_0, to Bd (the RAM digit-select register). This instruction allows the loading of a new RAM digit-select value via the accumulator, a useful operation in many memory-digit access loops.

CBA (Copy Bd to A) transfers the 4-bit contents of Bd (RAM digit select) to A_3-A_0. It is the functional complement of the CAB instruction and finds similar use in memory-digit access loops.

LBI (Load B Immediate) loads the B register with the 6-bit value specified by the "r" (2-bit) and "d" (4-bit) fields of the instruction. Its purpose is to directly load a new RAM register and digit select value into B and, unlike CAB, CBA or XABR, does not require use of the accumulator. A further distinction with respect to CAB and CBA is its ability to alter the Br register (RAM register-select).

The LBI instruction is coded or assembled into machine language as *either* a single- or a double-byte instruction, depending on the value of the "d" field. If the "d" field value equals 0 or 9 through 15, the instruction is coded as a single-byte instruction with the lower 6 bits equal to the value of "d" *minus* 1. If the "d" field equals 1 through 8 (1-8), the instruction is coded as a double-byte instruction, with the lower 6 bits of the second byte equal to the value of "d." (See LBI Instruction, Table 3.1, and Note 6 of Table 3.1.)

To take advantage of the more efficient single-byte LBI format, frequently used program data (counters, flags, etc.) should be placed within RAM digit locations accessible by the LBI single-byte "d" field (d = 0, 9-15). (See Section 4.2 for further information.)

An important characteristic of the LBI instruction is that it will skip all subsequent LBI instructions until it encounters an instruction which is not an LBI. This feature accommodates it for use in multiple-entry subroutines. (For example, see Adjacent Memory Move Routine, Section 4.4.)

LEI (Load EN Immediate) loads the enable register with the value contained in the "y" operand field of this instruction (0-15, binary). Its function is to select or deselect a particular software selectable feature associated with each of the four bits of the enable register (EN_3-EN_0). These features and the corresponding bit-weights and values associated with each feature are as follows:

1. The least significant bit of the enable register, EN_0, selects the SIO register as either a 4-bit shift register or a 4-bit binary counter.

   With EN_0 set, SIO is an asynchronous binary counter, decrementing its value by one upon each low-going pulse ("1" to "0") occurring on the SI input. Each pulse must remain at each logic level at least two instruction cycles. SK outputs the value of the C upon the execution of an XAS and remains latched until the execution of another XAS instruction. The SO output is equal to the value of EN_3.

   With EN_0 reset, SIO is a serial shift register, shifting continuously left each instruction cycle time. The data present at SI goes into the least significant bit of SIO; SO can be enabled to output the most significant bit of SIO each cycle time. SK output becomes a logic-controlled clock, providing a SYNC signal each instruction time. It will start outputting a SYNC pulse upon the execution of an XAS instruction with C = "1," stopping upon the execution of a subsequent XAS with C = "0."

   If EN_0 is changed from "1" to "0" ("0" to "1"), the SK output will change from "1" to SYNC (SYNC to "1") *without* the execution of an XAS instruction.

2. With EN_1 set, the IN_1 input is enabled as an interrupt input. Upon the occurrence of a negative pulse on IN_1, program control is transferred to the last word of page 3 (address $0FF_{16}$). Immediately following an interrupt, EN_1 is reset to disable further interrupts until later set by an LEI instruction (usually at the end of the interrupt service routine or later within the main program).

   The following features are associated with the IN_1 interrupt procedure and protocol and must be considered by the programmer when utilizing this software-selectable feature of the COP420-series. (Interrupt is unavailable on the COP421-series since it does not have the IN_3-IN_0 inputs.)

   a. The interrupt, once acknowledged as explained below, pushes the next sequential program counter address (P + 1) onto the stack, pushing in turn the contents of the other subroutine-save registers to the next lower level (P + 1 → SA → SB → SC). Any previous contents of SC are lost. The program counter is set to address $0FF_{16}$ (the last word of page 3) and EN_1 is reset.

   b. An interrupt will be acknowledged only after the following conditions are met:

      1) EN_1 has been set;

      2) A low-going pulse ("1" to "0") at least two instruction cycles in width has occurred on the IN_1 input;

      3) A currently executing instruction has been completed;

      4) All successive transfer of control instructions and successive LBIs have been completed (e.g., if the main program is executing a JP instruction which transfers program control to another JP instruction, the interrupt will not be acknowledged until the second JP instruction has been executed).

   c. Upon acknowledgement of an interrupt, the skip logic status is saved and implemented upon the execution of a subsequent RET instruction. For example, if an interrupt occurs during the execution of ASC (Add with carry, Skip on Carry) instruction which results in a carry, the next instruction (which would normally be skipped) is *not* skipped; instead, its address is pushed onto the stack, the skip logic status is saved and program control is transferred to the interrupt servicing routine at location $0FF_{16}$. At the *end* of the interrupt routine, a RET instruction is executed to pop the stack and return program control to the instruction following the original ACS. *At this time,* the skip logic is enabled and skips this instruction because of the previous ASC carry. Since, as explained above, it is the RET instruction which enables the previously saved status of the skip logic, subroutines should not be nested within the interrupt service routine since their RET instruction will enable any previously saved main program skips, interfering with the orderly execution of the interrupt routine.

   d. The first instruction of the interrupt routine at address $0FF_{16}$ must be NOP.

3. With EN_2 set, the L drivers are enabled, loading data previously latched into Q to the L I/O ports. Resetting EN_2 disables the L drivers, placing the L I/O ports in a high-impedance state. When the L I/O ports are used as segment drivers to an LED display, the setting and resetting of EN_2 results in the outputting and blanking, respectively, of segment data to the display. When using the MICROBUS™ option EN_2 does not affect the L drivers.

4. EN_3, in conjunction with EN_0, affects the SO output. With EN_0 set (binary counter option selected) SO will output the value loaded into EN_3. With EN_0 reset (serial shift register feature selected), setting EN_3 enables SO as the output of the SIO shift register, outputting serial shifted data (the most significant bit of SIO) each instruction time as explained above. Resetting EN_3 with the serial shift register feature selected disables SO as the shift register output: data continues to be shifted through SIO and can be exchanged with A via an XAS instruction but SO remains reset to "0." Figure 3.2 below provides a summary of the features associated with EN_3 and EN_0.

EN_3	EN_0	SIO	SI	SO	SK after XAS
0	0	Shift Register	Input to Shift Register	0	If SKL = 1, SK = SYNC If SKL = 0, SK = 0
1	0	Shift Register	Input to Shift Register	Serial Out	If SKL = 1, SK = SYNC If SKL = 0, SK = 0
0	1	Binary Counter	Negative Edge Sensitive Input to Binary Counter	0	If SKL = 1, SK = 1 If SKL = 0, SK = 0
1	1	Binary Counter	Negative Edge Sensitive Input to Binary Counter	1	If SKL = 1, SK = 1 If SKL = 0, SK = 0

Figure 3.2 Enable Register Features — Bits EN_3 and EN_0

XABR (eXchange A with Br) exchanges Br (upper 2 bits of B: RAM register-select) with A. Since Br contains only 2 bits, only the lower two bits of A, A_1-A_0, are placed in Br. Similarly, the 2 bits of Br are placed in A_1-A_0 with "0s" being loaded into the upper 2 bits of A, A_3-A_2. XABR is an efficient means of loading the Br register via the accumulator — a direct load of the Br register must otherwise be accomplished by an LBI instruction which also affects the Bd portion of the B register.

Test Instructions

SKC (SKip on Carry) skips the next program instruction if the carry bit is equal to "1." When used in conjunction with the RC and SC instructions, it allows C to be used as a 1-bit testable flag.

SKE (SKip if A Equals M) compares all 4 bits of A with M, skipping the next instruction if the value of A is equal to the value of M. SKE can be used to compare A with a status or counter digit in M, skipping to an instruction which transfers program control to another routine if equality exists.

SKGBZ (SKip if G Bit is Zero) is a double-byte instruction. It tests the state of *one* of the four G lines (G_3-G_0) as specified by the "n" operand of the instruction, skipping the next program instruction if the specified G line is equal to "0."

SKGZ (SKip if G is Zero) is a double-byte instruction. It tests the state of all *four* of the G lines, skipping the next program instruction if G_3-G_0 are all equal to "0."

SKMBZ (SKip on Memory Bit Zero) skips the next program instruction if the RAM memory bit specified by the "n" field of the instruction (0-3, right-most to left-most M bit) is equal to "0." This instruction, together with the SMB and RMB instructions, allow for the testing and manipulation of single-bit flags contained within RAM digit locations.

SKT (SKip on Timer) instruction tests the state of an internal 10-bit time-base counter. This counter divides the instruction cycle clock frequency by 1024 and provides a latched indication of counter overflow. The SKT instruction tests this latch, executing the next program instruction if the latch is not set. If the latch has been set since the previous test, the next program instruction is skipped and the latch is reset. The features associated with this instruction, therefore, allow the controller to generate its own time-base for real-time processing rather than relying on an external input signal.

For example, using a 2.097 MHz crystal as the time-base to the clock generator, the instruction cycle clock frequency will be 131 kHz (crystal frequency ÷ 16) and the binary counter output pulse frequency will be 128 Hz. For time-of-day or similar real-time processing, the SKT instruction can call a routine which increments a "seconds" counter every 128 ticks.

3.3 COP421-Series Instruction Set Differences

The **ININ** instruction has been deleted. This is due to the lack of the IN inputs.

The **INIL** instruction has been substantially modified due to the lack of IN inputs and IL_3/IL_0 latches. If an INIL instruction is executed on a COP421-series device, it will input only the state of CKO, providing CKO has been programmed as a general-purpose input ($0 \rightarrow A_3$, A_1, A_0; $CKO \rightarrow A_2$). If CKO has not been programmed as a general-purpose input, the INIL instruction is non-functional on the COP421-series.

3.4 COP410L/COP411L Instruction Set

The COP410L and COP411L instruction sets are subsets of the COP421-series instruction set.

Table 3.3 provides the mnemonic, operand, machine code, data flow, skip conditions and description associated with each instruction in the COP410L and COP411L instruction sets. An asterisk in the description column indicates the double-byte instruction. Notes are provided, following this table, which include additional information relevant to particular instructions.

Table 3.4 provides a list of internal architecture, instruction operand and operational symbols used in the COP410L/COP411L Instruction Set Table. Table 3.7 provides an alphabetical mnemonic index of COP410L/COP411L instructions, indicating the hexadecimal opcode and description associated with each instruction. Table 3.8 is a list of COP410L/COP411L instructions arranged in order of their hexadecimal opcodes.

The following text discusses the differences which exist between the COP410L and COP411L instruction sets and that of the COP420-series. The COP410L is specifically discussed with differences between it and the COP411L noted. All other instructions perform the same machine operations and have the same typical usage as discussed in Section 3.2. For a treatment of the significance of those differences when writing programs for the COP410L and COP411L, see Section 3.5, COP410L/COP411L Instruction Set Differences, and Section 4.11, COP410L/COP411L Programming.

Table 3.3 COP410L/COP411L Instruction Set

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description
ARITHMETIC INSTRUCTIONS						
ASC		30	0 0 1 1 | 0 0 0 0	A + C + RAM(B) → A Carry → C	Carry	Add with Carry, Skip on Carry
ADD		31	0 0 1 1 | 0 0 0 1	A + RAM(B) → A	None	Add RAM to A
AISC	y	5-	0 1 0 1 | y	A + y → A	Carry	Add Immediate, Skip on Carry (y ≠ 0)
CLRA		00	0 0 0 0 | 0 0 0 0	0 → A	None	Clear A
COMP		40	0 1 0 0 | 0 0 0 0	$\overline{A} \rightarrow A$	None	Ones complement of A to A
NOP		44	0 1 0 0 | 0 1 0 0	None	None	No Operation
RC		32	0 0 1 1 | 0 0 1 0	"0" → C	None	Reset C
SC		22	0 0 1 0 | 0 0 1 0	"1" → C	None	Set C
XOR		02	0 0 0 0 | 0 0 1 0	A ⊕ RAM(B) → A	None	Exclusive-OR RAM with A

Table 3.3 COP410L/COP411L Instruction Set (continued)

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description
TRANSFER OF CONTROL INSTRUCTIONS						
JID		FF	1 1 1 1 | 1 1 1 1	ROM (PC_{8},A,M) → $PC_{7:0}$	None	Jump Indirect (Note 2)
JMP	a	6-	0 1 1 0 | 0 0 0 | a_8	a → PC	None	Jump
		--	$a_{7:0}$			
JP	a	--	1 | $a_{6:0}$ (pages 2,3 only)	a → $PC_{6:0}$	None	Jump within Page (Note 3)
			or			
		--	1 1 | $a_{5:0}$ (all other pages)	a → $PC_{5:0}$		
JSRP	a	--	1 0 | $a_{5:0}$	PC + 1 → SA → SB 010 → $PC_{8:6}$ a → $PC_{5:0}$	None	Jump to Subroutine Page (Note 4)
JSR	a	6-	0 1 1 0 | 1 0 0 | a_8	PC + 1 → SA → SB a → PC	None	Jump to Subroutine
		--	$a_{7:0}$			
RET		48	0 1 0 0 | 1 0 0 0	SB → SA → PC	None	Return from Subroutine
RETSK		49	0 1 0 0 | 1 0 0 1	SB → SA → PC	Always Skip on Return	Return from Subroutine then Skip
MEMORY REFERENCE INSTRUCTIONS						
CAMQ		33	0 0 1 1 | 0 0 1 1	A → $Q_{7:4}$ RAM(B) → $Q_{3:0}$	None	Copy A, RAM to Q
		3C	0 0 1 1 | 1 1 0 0			
LD	r	-5	0 0 | r | 0 1 0 1	RAM(B) → A Br ⊕ r → Br	None	Load RAM into A, Exclusive-OR Br with r
LQID		BF	1 0 1 1 | 1 1 1 1	ROM(PC_{8},A,M) → Q SA → SB	None	Load Q Indirect (Note 2)
RMB	0	4C	0 1 0 0 | 1 1 0 0	0 → $RAM(B)_0$	None	Reset RAM Bit
	1	45	0 1 0 0 | 0 1 0 1	0 → $RAM(B)_1$		
	2	42	0 1 0 0 | 0 0 1 0	0 → $RAM(B)_2$		
	3	43	0 1 0 0 | 0 0 1 1	0 → $RAM(B)_3$		
SMB	0	4D	0 1 0 0 | 1 1 0 1	1 → $RAM(B)_0$	None	Set RAM Bit
	1	47	0 1 0 0 | 0 1 1 1	1 → $RAM(B)_1$		
	2	46	0 1 0 0 | 0 1 1 0	1 → $RAM(B)_2$		
	3	4B	0 1 0 0 | 1 0 1 1	1 → $RAM(B)_3$		
STII	y	7-	0 1 1 1 | y	y → RAM(B) Bd + 1 → Bd	None	Store Memory Immediate and Increment Bd
X	r	-6	0 0 | r | 0 1 1 0	RAM(B) ⟷ A Br ⊕ r → Br	None	Exchange RAM with A, Exclusive-OR Br with r
XAD	3,15	23	0 0 1 0 | 0 0 1 1	RAM(3,15) ⟷ A	None	* Exchange A with RAM (3,15)
		BF	1 0 1 1 | 1 1 1 1			
XDS	r	-7	0 0 | r | 0 1 1 1	RAM(B) ⟷ A Bd − 1 → Bd Br ⊕ r → Br	Bd decrements past 0	Exchange RAM with A and Decrement Bd, Exclusive-OR Br with r
XIS	r	-4	0 0 | r | 0 1 0 0	RAM(B) ⟷ A Bd + 1 → Bd Br ⊕ r → Br	Bd increments past 15	Exchange RAM with A and Increment Bd, Exclusive-OR Br with r

Table 3.3 COP410L/COP411L Instruction Set (continued)

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description
REGISTER REFERENCE INSTRUCTIONS						
CAB		50	0 1 0 1 | 0 0 0 0	A → Bd	None	Copy A to Bd
CBA		4E	0 1 0 0 | 1 1 1 0	Bd → A	None	Copy Bd to A
LBI	r,d	--	0 0 | r | (d – 1) (d = 0, 9:15)	r,d → B	None	Load B Immediate with r,d (Note 5)
LEI	y	33 6-	0 0 1 1 | 0 0 1 1 0 1 1 0 | y	y → EN	None	* Load EN Immediate (Note 6)
TEST INSTRUCTIONS						
SKC		20	0 0 1 0 | 0 0 0 0		C = "1"	Skip if C is True
SKE		21	0 0 1 0 | 0 0 0 1		A = RAM(B)	Skip if A Equals RAM
SKGZ		33 21	0 0 1 1 | 0 0 1 1 0 0 1 0 | 0 0 0 1		$G_{3:0}$ = 0	Skip if G is Zero (all 4 bits)
SKGBZ		33	0 0 1 1 | 0 0 1 1	1st byte		* Skip if G Bit is Zero
	0	01	0 0 0 0 | 0 0 0 1		G_0 = 0	
	1	11	0 0 0 1 | 0 0 0 1		G_1 = 0	
	2	03	0 0 0 0 | 0 0 1 1	2nd byte	G_2 = 0	
	3	13	0 0 0 1 | 0 0 1 1		G_3 = 0	
SKMBZ	0	01	0 0 0 0 | 0 0 0 1		$RAM(B)_0$ = 0	Skip if RAM Bit is Zero *
	1	11	0 0 0 1 | 0 0 0 1		$RAM(B)_1$ = 0	
	2	03	0 0 0 0 | 0 0 1 1		$RAM(B)_2$ = 0	
	3	13	0 0 0 1 | 0 0 1 1		$RAM(B)_3$ = 0	
INPUT/OUTPUT INSTRUCTIONS						
ING		33 2A	0 0 1 1 | 0 0 1 1 0 0 1 0 | 1 0 1 0	G → A	None	* Input G Ports to A
INL		33 2E	0 0 1 1 | 0 0 1 1 0 0 1 0 | 1 1 1 0	$L_{7:4}$ → RAM(B) $L_{3:0}$ → A	None	* Input L Ports to RAM, A
OBD		33 3E	0 0 1 1 | 0 0 1 1 0 0 1 1 | 1 1 1 0	Bd → D	None	* Output Bd to D Outputs
OMG		33 3A	0 0 1 1 | 0 0 1 1 0 0 1 1 | 1 0 1 0	RAM(B) → G	None	* Output RAM to G Ports
XAS		4F	0 1 0 0 | 1 1 1 1	A ⟷ SIO, C → SK	None	Exchange A with SIO (Note 2)

Note 1: All subscripts for alphabetical symbols indicate bit numbers unless explicitly defined (e.g., Br and Bd are explicitly defined). Bits are numbered 0 to N where 0 signifies the least significant (low-order, right-most bit). For example, A_3 indicates the most significant (left-most) bit of the 4-bit A register.

Note 2: For additional information on the operation of the XAS, JID, and LQID instructions, see Section 3.2.

Note 3: The JP instruction allows a jump, while in subroutine pages 2 or 3, to any ROM location within the two-page boundary of pages 2 or 3. The JP instruction, otherwise, permits a jump to a ROM location within the current 64-word page. JP may not jump to the last word of a page.

Note 4: A JSRP transfers program control to subroutine page 2 (0010 is loaded into the upper 4 bits of P). A JSRP may not be used when in pages 2 or 3. JSRP may not jump to the last word in page 2.

Note 5: LBI is a single-byte instruction if d = 0, 9, 10, 11, 12, 13, 14, or 15. The machine code for the lower 4 bits equals the binary value of the "d" data *minus 1*, e.g., to load the lower four bits of B (Bd) with the value 9 (1001_2), the lower 4 bits of the LBI instruction equal 8 (1000_2). To load 0, the lower 4 bits of the LBI instruction should equal 15 (1111_2).

Note 6: Machine code for operand field y for LEI instruction should equal the binary value to be latched into EN, where a "1" or "0" in each bit of EN corre sponds with the selection or deselection of a particular function associated with each bit. (See Functional Description, EN Register.)

Table 3.4 COP410L/411L Instruction Set Table Symbols

Symbol	Definition
INTERNAL ARCHITECTURE SYMBOLS	
A	4-bit Accumulator
B	6-bit RAM Address Register
Br	Upper 2 bits of B (register address)
Bd	Lower 4 bits of B (digit address)
C	1-bit Carry Register
D	4-bit Data Output Port
EN	4-bit Enable Register
G	4-bit Register to latch data for G I/O Port
L	8-bit TRI-STATE I/O Port
M	4-bit contents of RAM Memory pointed to by B Register
PC	9-bit ROM Address Register (program counter)
Q	8-bit Register to latch data for L I/O Port
SA	9-bit Subroutine Save Register A
SB	9-bit Subroutine Save Register B
SIO	4-bit Shift Register and Counter
SK	Logic-Controlled Clock Output

Symbol	Definition
INSTRUCTION OPERAND SYMBOLS	
d	4-bit Operand Field, 0-15 binary (RAM Digit Select)
r	2-bit Operand Field, 0-3 binary (RAM Register Select)
a	9-bit Operand Field, 0-511 binary (ROM Address)
y	4-bit Operand Field, 0-15 binary (Immediate Data)
RAM(s)	Contents of RAM location addressed by s
ROM(t)	Contents of ROM location addressed by t
OPERATIONAL SYMBOLS	
+	Plus
−	Minus
←	Replaces
←→	Is exchanged with
=	Is equal to
$\overline{A}$	The ones complement of A
⊕	Exclusive-OR
:	Range of values

3.5 COP410L/COP411L Instruction Set Differences

Arithmetic Instructions

ADT has been deleted. To perform a similar operation an AISC 10 followed by a NOP to defeat the skip condition (carry) may be used.

CASC has been deleted. A COMP instruction followed by an ASC will achieve the same result (subtraction of A from M).

Input/Output Instructions

ININ has been deleted due to the COP410L's lack of IN inputs.

OGI has been deleted. A loading of data to the G ports must be accomplished via M by first loading M and then outputting its contents to G via an OMG instruction.

Memory Reference Instructions

CQMA has been deleted. Since no MICROBUS™ option is provided for the COP410L, Q is used in the COP410L primarily for output operations. An input of the L I/O ports, therefore, will effectively function as the equivalent of a CQMA; this is accomplished by the execution of an INL instruction.

LDD has been deleted. To load the contents of a data memory digit location into A, the usual procedure of loading B via an LBI to point to a particular RAM location followed by an LD instruction must be used.

XAD has been altered to reference *one* data memory location only; specifically, M(3,15). "Scratch-pad" data to be exchanged with A without affecting the B register should be placed, therefore, in M(3,15) and accessed by the XAD 3,15 instruction.

Register Reference Instructions

LBI has been altered to correspond to the data memory configuration of the COP410L. Specifically, it may only be used to access valid RAM locations, namely digits 9 through 15 and 0 in registers 0-3. The LBI "d" field, therefore, is limited to "d" values of 9-15 and 0, resulting in *all LBIs* being coded as *single-byte* instructions. Remember, the *machine code* for the "d" operand field is the binary value of "d" minus 1.

XABR has been deleted. To load Br, the entire B register must be loaded via an LBI. Altering Br may also be accomplished by using the EXCLUSIVE-OR "r" field associated with the memory reference instructions LD, X, XDS, and XIS.

Test Instructions

SKT has been deleted since the COP410L does not contain an internal divide-by-1024 time-base counter.

Table 3.5 Alphabetical Mnemonic Index of COP420/COP421-Series Instructions

Instruction	Hexadecimal Opcode	Description
ADD	31	ADD RAM to A
ADT	4A	ADd Ten to A
AISC 1-15	51-5F	Add Immediate, Skip on Carry
ASC	30	Add with carry, Skip on Carry
CAB	50	Copy A to Bd
CAMQ*	33/3C	Copy A, RAM to Q
CASC	10	Complement and Add with carry, Skip on Carry
CBA	4E	Copy Bd to A
CLRA	00	CLeaR A
COMP	40	COMPlement A
CQMA*	33/2C	Copy Q to RAM, A
ING*	33/2A	INput G ports to A
INIL*	33/29	INput IL latches to A**
ININ*	33/28	INput IN inputs to A**
INL*	33/2E	INput L ports to RAM, A
JID	FF	Jump InDirect
JMP*	60-63/00-FF	JuMP
JP	80-BE,C0-CE	Jump within Page
JSR*	68-6B/00-FF	Jump to SubRoutine
JSRP	80-BE	Jump to SubRoutine Page
LBI 0;9-15,0	08-0F	Load Bd Immediate (single-byte)
LBI 1;9-15,0	18-1F	
LBI 2;9-15,0	28-2F	
LBI 3;9-15,0	38-3F	
LBI* 0;1-8	33/81-88	Load Bd Immediate (double-byte)
LBI* 1;1-8	33/91-98	
LBI* 2;1-8	33/A1-A8	
LBI* 3;1-8	33/B1-B8	
LD 0,1,2,3	05,15,25,35	LoaD RAM into A
LDD* 0-3,0-15	23/00-3F	LoaD A with RAM, Directly
LEI* 0-15	33/60-6F	Load EN Immediate
LQID	BF	Load Q InDirect
NOP	44	No OPeration
OBD*	33/3E	Output Bd to D outputs
OGI*	33/50-5F	Output to G ports Immediate
OMG*	33/3A	Output RAM to G ports
RC	32	Reset Carry
RET	48	RETurn
RETSK	49	RETurn then SKip
RMB 0,1,2,3	4C,45,42,43	Reset Memory Bit
SC	22	Set Carry
SMB 0,1,2,3	4D,47,46,4B	Set Memory Bit
SKC	20	SKip if Carry is true
SKE	21	SKip if A Equals RAM
SKGBZ* 0,1,2,3	33/01,11,03,13	SKip if G Bit is Zero
SKGZ*	33/21	SKip if G equals Zero (all 4 bits)
SKMBZ 0,1,2,3	01,11,03,13	SKip if Memory Bit is Zero
SKT	41	SKip on Timer
STII	70-7F	STore memory Immediate and Increment Bd
X 0,1,2,3	6,16,26,36	eXchange RAM with A, exclusive-OR r with Br
XABR	12	eXchange A with Br
XAD* 0-3,0-15	23/80-BF	eXchange A with RAM Directly
XAS	4F	eXchange A with SIO (serial I/O)
XDS 0,1,2,3	07,17,27,37	eXchange RAM with A and Decrement Bd
XIS 0,1,2,3	04,14,24,34	eXchange RAM with A and Increment Bd
XOR	02	eXclusive-OR RAM with A

*Double-Byte Instruction: first byte/second byte (or first byte range/second byte range).

**Instruction not available or has different features on COP421-series.

Table 3.6 Table of COP420/COP421-Series Instructions Listed by Opcodes (Hexadecimal)

00	CLRA	26	X 2
01	SKMBZ 0	27	XDS 2
02	XOR	28	LBI 2,9
03	SKMBZ 2	29	LBI 2,10
04	XIS 0	2A	LBI 2,11
05	LD 0	2B	LBI 2,12
06	X 0	2C	LBI 2,13
07	XDS 0	2D	LBI 2,14
08	LBI 0,9	2E	LBI 2,15
09	LBI 0,10	2F	LBI 2,0
0A	LBI 0,11	30	ASC
0B	LBI 0,12	31	ADD
0C	LBI 0,13	32	RC
0D	LBI 0,14	33	TWO WORD* (except LDD, XAD, JMP, JSR)
0E	LBI 0,15		
0F	LBI 0,0		
10	CASC	34	XIS 3
11	SKMBZ 1	35	LD 3
12	XABR	36	X 3
13	SKMBZ 3	37	XDS 3
14	XIS 0	38	LBI 3,9
15	LD 1	39	LBI 3,10
16	X 1	3A	LBI 3,11
17	XDS 1	3B	LBI 3,12
18	LBI 1,9	3C	LBI 3,13
19	LBI 1,10	3D	LBI 3,14
1A	LBI 1,11	3E	LBI 3,15
1B	LBI 1,12	3F	LBI 3,0
1C	LBI 1,13	40	COMP
1D	LBI 1,14	41	SKT
1E	LBI 1,15	42	RMB 2
1F	LBI 1,0	43	RMB 3
20	SKC	44	NOP
21	SKE	45	RMB 1
22	SC	46	SMB 2
23	LDD/XAD**	47	SMB 1
24	XIS 2	48	RET
25	LD 2	49	RETSK

Table 3.6 Table of COP420/COP421-Series Instructions Listed by Opcodes (Hexadecimal) (continued)

4A	ADT
4B	SMB 3
4C	RMB 0
4D	SMB 0
4E	CBA
4F	XAS
50	CAB
51	AISC 1
52	AISC 2
53	AISC 3
54	AISC 4
55	AISC 5
56	AISC 6
57	AISC 7
58	AISC 8
59	AISC 9
5A	AISC 10
5B	AISC 11
5C	AISC 12
5D	AISC 13
5E	AISC 14
5F	AISC 15
60	JMP*** to Page 0, 1, 2, or 3
61	JMP*** to Page 4, 5, 6, or 7
62	JMP*** to Page 8, 9, 10, or 11
63	JMP*** to Page 12, 13, 14, or 15
64	invalid
65	invalid
66	invalid
67	invalid
68	JSR*** to Page 0, 1, 2, or 3
69	JSR*** to Page 4, 5, 6, or 7
6A	JSR*** to Page 8, 9, 10, or 11
6B	JSR*** to Page 12, 13, 14, or 15
6C	invalid
6D	invalid
6E	invalid
6F	invalid
70	STII 0
71	STII 1
72	STII 2
73	STII 3
74	STII 4
75	STII 5
76	STII 6
77	STII 7
78	STII 8
79	STII 9
7A	STII 10
7B	STII 11
7C	STII 12
7D	STII 13
7E	STII 14
7F	STII 15
80-BE	JP to word XX ($0\text{-}3F_{16}$) or JSRP to page 2, word XX ($0\text{-}3F_{16}$): opcode = 80 + XX
BF	LQID
C0-CE	JP to word XX ($0\text{-}3F_{16}$): opcode = C0 + XX
FF	JID
	Two Word Instructions, Second Word:
*00	INIL (different features for COP421)
01	SKGBZ 0
03	SKGBZ 2
11	SKGBZ 1
13	SKGBZ 3
21	SKGZ
28	ININ (invalid for COP421)
2A	ING
2C	COMA
2E	INL
3A	OMG
3C	CAMQ
3E	OBD
50	OGI 0
51	OGI 1
52	OGI 2
53	OGI 3
54	OGI 4
55	OGI 5
56	OGI 6
57	OGI 7
58	OGI 8
59	OGI 9
5A	OGI 10
5B	OGI 11
5C	OGI 12
5D	OGI 13
5E	OGI 14
5F	OGI 15
60	LEI 0
61	LEI 1
62	LEI 2
63	LEI 3
64	LEI 4
65	LEI 5
66	LEI 6
67	LEI 7
68	LEI 8
69	LEI 9
6A	LEI 10
6B	LEI 11
6C	LEI 12
6D	LEI 13
6E	LEI 14
6F	LEI 15
81	LBI 0,1
82	LBI 0,2
83	LBI 0,3
84	LBI 0,4
85	LBI 0,5
86	LBI 0,6
87	LBI 0,7
88	LBI 0,8
91	LBI 1,1
92	LBI 1,2
93	LBI 1,3
94	LBI 1,4
95	LBI 1,5
96	LBI 1,6
97	LBI 1,7
98	LBI 1,8
A1	LBI 2,1
A2	LBI 2,2
A3	LBI 2,3
A4	LBI 2,4
A5	LBI 2,5
A6	LBI 2,6
A7	LBI 2,7
A8	LBI 2,8
B1	LBI 3,1
B2	LBI 3,2
B3	LBI 3,3
B4	LBI 3,4
B5	LBI 3,5
B6	LBI 3,6
B7	LBI 3,7
B8	LBI 3,8
**00	LDD 0,0
01	LDD 0,1
02	LDD 0,2
03	LDD 0,3
04	LDD 0,4
05	LDD 0,5
06	LDD 0,6
07	LDD 0,7
08	LDD 0,8
09	LDD 0,9
0A	LDD 0,10
0B	LDD 0,11
0C	LDD 0,12
0D	LDD 0,13
0E	LDD 0,14
0F	LDD 0,15
10	LDD 1,0
11	LDD 1,1
12	LDD 1,2
13	LDD 1,3
14	LDD 1,4
15	LDD 1,5
16	LDD 1,6
17	LDD 1,7
18	LDD 1,8
19	LDD 1,9
1A	LDD 1,10
1B	LDD 1,11
1C	LDD 1,12
1D	LDD 1,13
1E	LDD 1,14
1F	LDD 1,15
20	LDD 2,0
21	LDD 2,1
22	LDD 2,2
23	LDD 2,3
24	LDD 2,4
25	LDD 2,5
26	LDD 2,6
27	LDD 2,7
28	LDD 2,8
29	LDD 2,9
2A	LDD 2,10
2B	LDD 2,11
2C	LDD 2,12
2D	LDD 2,13
2E	LDD 2,14
2F	LDD 2,15
30	LDD 3,0
31	LDD 3,1
32	LDD 3,2
33	LDD 3,3
34	LDD 3,4
35	LDD 3,5
36	LDD 3,6
37	LDD 3,7
38	LDD 3,8
39	LDD 3,9
3A	LDD 3,10
3B	LDD 3,11
3C	LDD 3,12
3D	LDD 3,13
3E	LDD 3,14
3F	LDD 3,15
80	XAD 0,0
81	XAD 0,1
82	XAD 0,2
83	XAD 0,3
84	XAD 0,4
85	XAD 0,5
86	XAD 0,6
87	XAD 0,7
88	XAD 0,8
89	XAD 0,9
8A	XAD 0,10
8B	XAD 0,11

Table 3.6 Table of COP420/COP421-Series Instructions Listed by Opcodes (Hexadecimal) (continued)

8C	XAD 0,12
8D	XAD 0,13
8E	XAD 0,14
8F	XAD 0,15
90	XAD 1,0
91	XAD 1,1
92	XAD 1,2
93	XAD 1,3
94	XAD 1,4
95	XAD 1,5
96	XAD 1,6
97	XAD 1,7
98	XAD 1,8
99	XAD 1,9
9A	XAD 1,10
9B	XAD 1,11
9C	XAD 1,12
9D	XAD 1,13
9E	XAD 1,14
9F	XAD 1,15
A0	XAD 2,0
A1	XAD 2,1
A2	XAD 2,2
A3	XAD 2,3
A4	XAD 2,4
A5	XAD 2,5
A6	XAD 2,6
A7	XAD 2,7
A8	XAD 2,8
A9	XAD 2,9
AA	XAD 2,10
AB	XAD 2,11
AC	XAD 2,12
AD	XAD 2,13
AE	XAD 2,14
AF	XAD 2,15
B0	XAD 3,0
B1	XAD 3,1
B2	XAD 3,2
B3	XAD 3,3
B4	XAD 3,4
B5	XAD 3,5
B6	XAD 3,6
B7	XAD 3,7
B8	XAD 3,8
B9	XAD 3,9
BA	XAD 3,10
BB	XAD 3,11
BC	XAD 3,12
BD	XAD 3,13
BE	XAD 3,14
BF	XAD 3,15

***00 + XX JSR or JMP to page 0, 4, 10, or 14, word XX ($03F_{16}$): 0-3F
40 + XX JSR or JMP to page 1, 5, 11, or 15, word XX ($0-3F_{16}$):40-7F
80 + XX JSR or JMP to page 2, 6, 12, or 16, word XX ($0-3F_{16}$):80-BF
C0 + XX JSR or JMP to page 3, 7, 13, or 17, word XX ($0-3F_{16}$):C0-FF

Table 3.7 Alphabetical Mnemonic Index of COP410L/COP411L-Series Instructions

Instruction	Hexadecimal Opcode	Description
ADD	31	ADD RAM to A
AISC 1-15	51-5F	Add Immediate, Skip on Carry
ASC	30	Add with carry, Skip on Carry
CAB	50	Copy A to Bd
CAMQ	33/3C	Copy A, RAM to Q
CBA	4E	Copy Bd to A
CLRA	00	CLeaR A
COMP	40	COMPlement A
ING*	33/2A	INput G ports to A
INL*	33/2E	INput L ports to RAM, A
JID	FF	Jump InDirect
JMP*	60-61/00-FF	JuMP
JP	80-BE,C0-CE	Jump within Page
JSR*	68-69/00-FF	Jump to SubRoutine
JSRP	80-BE	Jump to SubRoutine Page
LBI 0;9-15,0	08-0F	Load Bd Immediate (single-byte)
LBI 1;9-15,0	18-1F	
LBI 2;9-15,0	28-2F	
LBI 3;9-15,0	38-3F	
LD 0,1,2,3	05,15,25,35	LoaD RAM into A
LEI* 0-15	33/60-6F	Load EN Immediate
LQID	BF	Load Q InDirect
NOP	44	No OPeration
OBD*	33/3E	Output Bd to D outputs
OMG*	33/3A	Output RAM to G ports
RC	32	Reset Carry
RET	48	RETurn
RETSK	49	RETurn then SKip
RMB 0,1,2,3	4C,45,42,43	Reset Memory Bit
SC	22	Set Carry
SMB 0,1,2,3	4D,47,46,4B	Set Memory Bit
SKC	20	SKip if Carry is true
SKE	21	SKip if A Equals RAM
SKGBZ* 0,1,2,3	33/01,11,03,13	SKip if G Bit is Zero
SKGZ*	33/21	SKip if G equals Zero (all 4 bits)
SKMBZ 0,1,2,3	01,11,03,13	SKip if Memory Bit is Zero
STII	70-7F	STore memory Immediate and Increment Bd
X 0,1,2,3	6,16,26,36	eXchange RAM with A
XAD* 3,15	23-BF	eXchange A with RAM Directly
XAS	4F	eXchange A with SIO (serial I/O)
XDS 0,1,2,3	07,17,27,37	eXchange RAM with A and Decrement Bd
XIS 0,1,2,3	04,14,24,34	eXchange RAM with A and Increment Bd
XOR	02	eXclusive-OR RAM with A

*Double-Byte Instruction: first byte/second byte (or first byte range/second byte range).
**Instruction not available or has different features on COP421-series.

Table 3.8 Table of COP410L/COP411L-Series Instructions Listed by Opcodes (Hexadecimal) (continued)

00	CLRA
01	SKMBZ 0
02	XOR
03	SKMBZ 2
04	XIS 0
05	LD 0
06	X 0
07	XDS 0
08	LBI 0,9
09	LBI 0,10
0A	LBI 0,11
0B	LBI 0,12
0C	LBI 0,13
0D	LBI 0,14
0E	LBI 0,15
0F	LBI 0,0
10	invalid
11	SKMBZ 1
12	invalid
13	SKMBZ 3
14	XIS 0
15	LD 1
16	X 1
17	XDS 1
18	LBI 1,9
19	LBI 1,10
1A	LBI 1,11
1B	LBI 1,12
1C	LBI 1,13
1D	LBI 1,14
1E	LBI 1,15
1F	LBI 1,0
20	SKC
21	SKE
22	SC
23	XAD**
24	XIS 2
25	LD 2
26	X 2
27	XDS 2
28	LBI 2,9
29	LBI 2,10
2A	LBI 2,11
2B	LBI 2,12
2C	LBI 2,13
2D	LBI 2,14
2E	LBI 2,15
2F	LBI 2,0
30	ASC
31	ADD
32	RC
33	TWO WORD* (except XAD, JMP, JSR)
34	XIS 3
35	LD 3
36	X 3
37	XDS 3
38	LBI 3,9
39	LBI 3,10
3A	LBI 3,11
3B	LBI 3,12
3C	LBI 3,13
3D	LBI 3,14
3E	LBI 3,15
3F	LBI 3,0
40	COMP
41	invalid
42	RMB 2
43	RMB 3
44	NOP
45	RMB 1
46	SMB 2
47	SMB 1
48	RET
49	RETSK
4A	invalid
4B	SMB 3
4C	RMB 0
4D	SMB 0
4E	CBA
4F	XAS
50	CAB
51	AISC 1
52	AISC 2
53	AISC 3
54	AISC 4
55	AISC 5
56	AISC 6
57	AISC 7
58	AISC 8
59	AISC 9
5A	AISC 10
5B	AISC 11
5C	AISC 12
5D	AISC 13
5E	AISC 14
5F	AISC 15
60	JMP*** to Page 0, 1, 2, or 3
61	JMP*** to Page 4, 5, 6, or 7
64	invalid
65	invalid
66	invalid
67	invalid
68	JSR*** to Page 0, 1, 2, or 3
69	JSR*** to Page 4, 5, 6, or 7
6C	invalid
6D	invalid
6E	invalid
6F	invalid
70	STII 0
71	STII 1
72	STII 2
73	STII 3
74	STII 4
75	STII 5
76	STII 6
77	STII 7
78	STII 8
79	STII 9
7A	STII 10
7B	STII 11
7C	STII 12
7D	STII 13
7E	STII 14
7F	STII 15
80-BE	JP to word XX ($0\text{-}3F_{16}$) or JSRP to page 2, word XX ($0\text{-}3F_{16}$): opcode = 80 + XX
BF	LQID
C0-CE	JP to word XX ($0\text{-}3F_{16}$): opcode = C0 + XX
FF	JID

Two Word Instructions, Second Word:

*00	invalid
01	SKGBZ 0
03	SKGBZ 2
11	SKGBZ 1
13	SKGBZ 3
21	SKGZ
28	invalid
2A	ING
2C	invalid
2E	INL
3A	OMG
3C	CAMQ
3E	OBD
50-5F	invalid
60	LEI 0
61	LEI 1
62	LEI 2
63	LEI 3
64	LEI 4
65	LEI 5
66	LEI 6
67	LEI 7
68	LEI 8
69	LEI 9
6A	LEI 10
6B	LEI 11
6C	LEI 12
6D	LEI 13
6E	LEI 14
6F	LEI 15
81-88	invalid
91-98	invalid
A1-A8	invalid
B1-B8	invalid

**00-BE Invalid BF → XAD 3,15

***00 + XX JSR or JMP to page 0 or 4 word XX ($03F_{16}$): 0-3F
40 + XX JSR or JMP to page 1 or 5 word XX ($0\text{-}3F_{16}$):40-7F
80 + XX JSR or JMP to page 2 or 6 word XX ($0\text{-}3F_{16}$):80-BF
C0 + XX JSR or JMP to page 3 or 7 word XX ($0\text{-}3F_{16}$):C0-FF

4 COP400 Programming Techniques



This chapter provides several examples of programming techniques for COP400 devices. The COP420-series/COP444L instruction set is assumed since it falls between the smaller and larger instruction sets, respectively, of the COP410L and the COP440. For users of the COP410L/COP411L, Section 3.5 provides information on use of multiple COP410L instructions to simulate the function of COP420 instructions not provided for the COP410L. Users of the COP440 will find all examples relevant since this device contains all COP420 instructions as well as several additional instructions.

All examples are given in COPS™ Cross Assembler language, using COP400 assembler instruction mnemonics and operand statements. Although, in the following examples, instruction operands and ROM page numbers are written using decimal notation, the programmer may specify these expressions in hexadecimal notation — the assembler accepts either format (e.g., AISC 13 = AISC X'C, Page X'A = Page 10). On occasion, source code examples contain non-instruction statements, such as assembler directives which convey information to the assembler necessary for proper program address allocation and similar assembler related tasks. For further information on the COPS Cross Assembler and its use see *PDS User's Manual*, Chapter 8.

4.1 Program Memory Allocation

Generally, COP420-series program memory may be thought of as one area of 1024 bytes of ROM with an address range of 0 to 3FF (hexadecimal). However, while this concept is convenient in writing, assembling and debugging major portions of COP420-series programs, it is necessary, with respect to a few instructions, to conceptualize program memory on a 64-word "page" basis.

Specifically, because of the characteristics and restrictions associated with the JP, JSRP, JID, and LQID instructions, the programmer must conceive of program memory as 1024 bytes or words, organized as sixteen pages, numbered 0-15 respectively. The following discussion provides information and examples relating to the "page" characteristics of each of these unique instructions. For information on the machine code and operations performed by these instructions, see Section 3.2. Table 4.1 provides a conversion chart indicating the hexadecimal address equivalents for each of the 16 "pages" of ROM. Note — each page consists of 0 through $3F_{16}$ words.

Table 4.1 Page to Hexadecimal Address Table

Page	Hexadecimal Address Range
0	000-03F
1	040-07F
2	080-0BF
3	0C0-0FF
4	100-13F
5	140-17F
6	180-1BF
7	1C0-1FF
8	200-23F
9	240-27F
10	280-2BF
11	2C0-2FF
12	300-33F
13	340-37F
14	380-3BF
15	3C0-3FF

JP Instruction

The JP instruction is used to transfer program control to a ROM location within a page or within a two-page boundary consisting of "subroutine pages" 2 or 3.

The following page restrictions apply to the JP instruction:

- When used in any page other than page 2 or 3, it can only jump to a word within the *current* page.
- When used in page 2 or 3, it may jump to a word within page 2 *or* 3.
- In all cases, it cannot jump to the last word of a page (word $03F_{16}$).

The JP instruction assembly operand normally consists of a program label or expression specifying the address of the word to be jumped to. To specify page boundaries and to ensure correct placement of the JP and other page-oriented

instructions, the assembler .PAGE directive is used to specify the beginning of new page boundaries for program code placement. (See *PDS User's Manual*, Chapter 8.) The following are examples of use of the JP instruction when used outside subroutine pages 2 and 3:

```
        .PAGE 0             ; PLACE FOLLOWING CODE IN
                            ;  PAGE 0
LABEL1:  .
         JP      LABEL2     ; LEGAL JUMP WITHIN PAGE
         .
         .
LABEL2:  .
         JP      LABEL3     ; ILLEGAL JUMP TO LAST
         .                  ;  WORD OF PAGE
         JP      LABEL4     ; ILLEGAL JUMP TO ANOTHER
         .                  ;  PAGE
         .
LABEL3:  ; THIS INSTRUCTION IN LAST
         ;  WORD OF PAGE 0
         ; PLACE FOLLOWING CODE
         ;  ON PAGE 1*
         .
         PAGE 1
         .
         .
         .
LABEL4:  .
         .
         .
         .
```

*Note: The .PAGE 1 directive is not necessary — the PDS Assembler automatically places code in successive memory locations. After a particular page is full, code is automatically placed in successive locations on the following page.

The following examples illustrate use of the JP instruction when in subroutine pages 2 and 3:

```
        .PAGE 2             ; START OF "SUBROUTINE"
LABEL1:  .                  ;  PAGE 2 CODE
         JP      LABEL3     ; LEGAL JUMP TO PAGE 3
         .                  ;  LOCATION
         JP      LABEL2     ; ILLEGAL JUMP TO LAST
         .                  ;  WORD OF PAGE
         .
LABEL2:  .                  ; LAST WORD OF PAGE 2
         .PAGE 3            ; START OF PAGE 3 CODE
         .
         .
         JP      LABEL4     ; ILLEGAL JUMP TO PAGE
         .                  ;  OUTSIDE PAGE 2 OR 3
         .
LABEL3:  .
         JP      LABEL1     ; LEGAL JUMP TO PAGE 2
         .                  ;  LOCATION
         JP      LABEL3     ; LEGAL JUMP WITHIN PAGE
         .
         .
         .PAGE 4            ; START OF PAGE 4 CODE
         .
         .
LABEL4:  .
         .
         .
         JP      LABEL1     ; ILLEGAL JUMP TO PAGE 2
         .                  ;  (MAY ONLY BE DONE WHEN
         .                  ;  IN PAGE 2 OR 3)
```

JSRP Instruction

The JSRP instruction is another page-oriented instruction which transfers program control to a word located within "subroutine" page 2 *only*. Its primary purpose is to allow a single-byte jump to a subroutine in page 2 from any program location other than from page 2 or 3. As explained in Section 3.2, JSRP pushes the subroutine-save stack to allow a return to the next program instruction following the subroutine call. The restrictions with the JSRP instruction are as follows:

- JSRP cannot be used to jump to a subroutine *when in* pages 2 or 3. (The double-byte JSR instruction can be used for this purpose.)
- JSRP cannot be used to jump to a subroutine located at the last word of page 2. (A JSR can also be used for this purpose.)

Examples of use of the JSRP instruction:

```
        .PAGE 0
         .
         .
LABEL1:  .                  ; PAGE 0 SUBROUTINE
         .
         RET                ; RETURN FROM SUBROUTINE
         .
         JSRP    ADD        ; LEGAL CALL TO PAGE 2
         .
         JSRP    SUB        ; ILLEGAL CALL TO PAGE 3
         .
         .PAGE 2            ; START OF PAGE 2 CODE
         .
         .
ADD:     .                  ; START OF ADD SUBROUTINE
         .
         RET
         .
         JSRP    LABEL1     ; ILLEGAL CALL FROM PAGE 2
         .
         .PAGE 3            ; START OF PAGE 3 CODE
         .
         .
SUB:     .                  ; SUBTRACT SUBROUTINE
         .
         RET
         .
         .
```

Subroutine Pages 2 and 3

The special characteristics of the JP and JSRP instructions facilitate the use of pages 2 and 3 as subroutine pages. Programmers should consider dedicating these pages to the recursive program subroutine for the following reasons:

- A single-byte JSRP can be used to transfer program control to a page 2 subroutine.
- When in pages 2 or 3, a single-byte JP can be used to jump to either of these pages.

The following code exemplifies the use of the JP and JSRP instructions to transfer program control to and within pages 2 and 3 as follows. Note that in this example the ADD subroutine jumps to MEMOVE (Memory Move) routine before returning.

Thus, subroutines may share a common "return" subroutine, jumped to from page 2 or 3 with a single-byte JP instruction.

```
                .PAGE 0
                .
                .
                JSRP    ADD         ; CALL ADD SUBROUTINE
                .
                .
                .PAGE 2             ; START OF PAGE 2 CODE
                .
                .
ADD:            .                   ; ADD SUBROUTINE
                .
                .
                JP      MEMOVE      ; JUMP TO MEMOVE
                .                   ; "RETURN" ROUTINE (NO
                .                   ; "PUSH" OF STACK)
                .PAGE 3             ; START OF PAGE 3 CODE
                .
                .
MEMOVE:         .                   ; MEMORY MOVE ROUTINE
                .
                .
                RET                 ; RETURN TO MAIN PROGRAM
                .                   ; (POP STACK)
```

JID Instruction

The JID (Jump Indirect) instruction is another page-oriented instruction. For a machine operation description, see Section 3.2. JID is an *indirect* ROM addressing instruction which transfers program control to a new ROM location based upon the contents of a ROM "pointer." The paging features and restrictions associated with the JID instruction are as follows:

- JID first jumps to a *ROM pointer* based upon the contents of A and RAM.
- JID then transfers program control to the ROM word specified by the *contents* of the ROM pointer.
- The ROM pointer *and* the indirect address jumped to must be within the same 4-page ROM "block" as the JID instruction. Specifically, for purposes of this instruction, the sixteen pages of ROM are divided into 4 blocks as follows:

Block	Pages
1	0-3
2	4-7
3	8-11
4	12-15

For example, if the JID instruction is located in page 5, the ROM pointer and the indirect address to which program control is transferred must be within block 2 (pages 4-7). For an example of the use of the JID instruction in a simple keyboard decode routine, see Section 5.3.

LQID Instruction

The LQID instruction is an *indirect* data output instruction. It loads the 8-bit Q register with the 8-bit contents of a particular ROM location pointed to by A and RAM. For an explanation of the machine operations associated with this instruction, see Section 3.2. The paging restrictions associated with this instruction are similar to those associated with the JID instruction, as follows:

- For purposes of the LQID instruction as with the JID instruction, ROM is divided into 4-page ROM "blocks" (pages 0-3, 4-7, 8-11 and 12-15).
- The ROM location containing the LQID "lookup" data must be within the same ROM block as the LQID instruction.

For example, a LQID instruction located in page 9 must access ROM data located in pages 8 through 11.

Additional Restrictions Associated with JP, JSRP, JID and LQID Instructions

As already mentioned, the ROM address register (P) increments its value when executing an instruction to point to the next memory instruction, automatically "rolling over" to the next page after executing an instruction located in the last word of a page. It is important to realize, however, that P is incremented *prior to the execution of the current instruction.* This characteristic has important consequences for JP, JSR, JID and LQID instructions *which are located in the last word of a page.* Specifically, these instructions will operate on the incremented value of P which, because of the increment-before-execution COP feature, will point to the first word of the next page. Consequently, if any of these instructions are placed in the last word of a page, the program will treat them as residing on the first word of the following page. Given the paging restrictions associated with these instructions, the following operations and restrictions are associated with the following placements of these instructions:

- A JP in the last word of a page will go to any location in the following page (except the last word). A JP in the last word of page 1 will be able to go to any location (except the last word) of page 2 or 3 since it is treated as a JP in page 2. Furthermore, a JP in the last word of page 3 *will not go* to a location within page 2 or 3, but, instead, will go to a location within page 4.
- A JSRP instruction is not allowed to reside in the last word of page 1, since it will be treated as an illegal use of JSRP in page 2. A JSRP in the last word of page 3, however, is allowed, since it will be treated as a JSRP outside of pages 2 or 3, namely in page 4.
- A LQID or JID instruction located in the last word of the last page of a particular ROM block (last word of page 3, 7, 11 or 15) will lookup data or transfer program control, respectively, to a location within the *next* 4 page ROM block.

As is evident from the above, these characteristics are not necessarily restrictions, provided the programmer intentionally uses these instructions to operate in the above manner. For example, a JP on the last word of page 1, unlike other page 1 JP instructions, will be able to transfer program control to the two-page subroutine pages 2 or 3, provided the operand specifies a location within page 2 or 3. Similarly, a LQID or JID located in the last word of the last page of a ROM block will allow data lookups on or indirect program control transfers to locations within the next ROM block, provided the lookup data or address pointers are placed in the appropriate locations within the *next* ROM block.

Use of Assembler .PAGE Directive

Because of the above paging restrictions, programmers are advised to place .PAGE assembler directives at the beginning of each page of code. Although portions of the program may not contain page-related instructions, this practice will facilitate placement of program "patches" or other modifications required during the program debug phase, these often involving page-related instructions. This practice is also a convenient, if not necessary, documentation tool, dividing the assembler output listing into a COPS™ page format. Finally, since the COPS Cross Assembler places program memory words into successive locations without regard to COPS pages, the use of a .PAGE directive is a simple means of reserving program memory space at the end of a page during initial program code generation, often used later for program additions. An alternative means of reserving program memory space anywhere within a page is by use of an assembler *assignment* statement which references the assembler location pointer — the pointer is referenced by a period ("."). For more information on the assignment statement, see *PDS User's Manual*, Section 8.4. An example and explanation of its use in referencing the assembler location counter (".") is contained in Section 4.5 of this manual

4.2 Data Memory Allocation and Manipulation

An important step which should occur prior to writing a COPS™ program is the allocation of program data (registers, flags, counters, etc.) to specific areas of program memory (RAM). This process is referred to as "creating a RAM map" and, although the map will undoubtedly change as programming continues, construction of an initial RAM map will make the ensuing programming process significantly easier.

As explained in Section 2.8, the COP420-series has 4 data memory registers, numbered 0 through 3, consisting of 16 4-bit digits. Frequently accessed data should be stored in locations which are able to be pointed to by loading the B register with a single-byte LBI instruction. These locations consist of digit numbers 0 and 9 through 15 in any data memory register. These areas are indicated by the diagonal-lined areas of Figure 4.1. It requires a double-byte LBI instruction to load the B register to access the other digits in data memory registers, thus requiring an extra program memory word. Single-bit flags and digit counters should be located in these diagonal-lined regions since they tend to be frequently accessed in most programs.

The memory reference instructions LD, X, XDS, and XIS allow the programmer to modify the data memory *register* address without using an LBI instruction. All of these instructions may modify the upper two bits of B (Br — RAM register-select) by specifying an "r" operand field which is exclusive-ORed with the current value of Br. This feature allows the programmer to toggle back and forth between any of the four COP420 data memory registers. For example, data located within the data memory locations marked with shaded boxes in Figure 4.1 can be easily swapped back and forth using the LD and X instructions. They can also be added to or subtracted from each other easily.

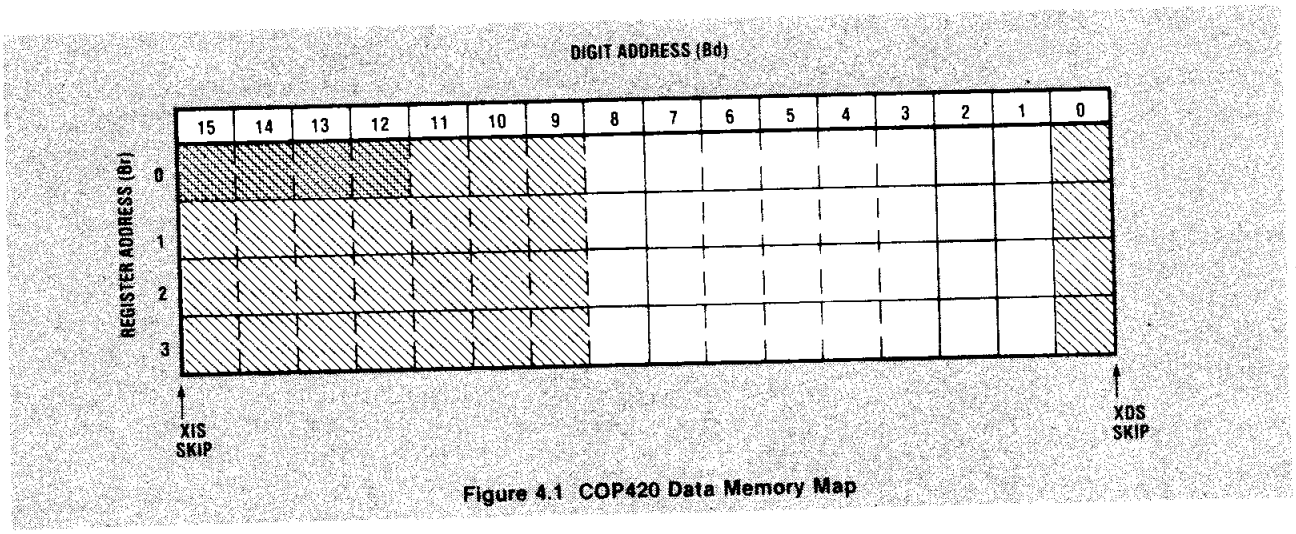


Figure 4.1 COP420 Data Memory Map

The automatic data memory *digit* address increment and decrement features associated with the XIS and XDS instructions and their skip condition features facilitate the shifting, adding, and subtracting of the contents of data memory. Data that needs to be shifted should be located in adjacent digit locations (for example, the dotted-box locations in Figure 4.1). Data that needs to be added, subtracted, or shifted should be located in areas adjacent to the XIS or XDS skip boundaries. The dotted locations in Figure 4.1 are against the XIS boundary at digit 15. This allows the programmer to take advantage of the skip feature of the XIS instruction.

The following examples illustrate several of the principles discussed above. The notation $M(N_1,N_2)$ indicates a particular data memory digit M, where N_1 = register number and N_2 = digit number.

```
; MOVE M(3,0) TO M(1,0)

        LBI     3,0     ; 3 TO BR; 0 TO BD (SINGLE-BYTE
                        ;  LBI; D = 0)
        LD      2       ; M(3,0) TO A; 1 TO BR 3 ÷ 2 = 1)
        X               ; A TO M(1,0)

; MOVE MEMORY REGISTER 1 TO MEMORY REGISTER 0
; M(1,15) - M(1,0) TO M(0,15) - M(0,0)

        LBI     1,15    ; 1 TO BR, 15 TO BD (SINGLE-BYTE
                        ;  LBI)
MV1:    LD      1       ; M(1,15) TO A; 0 TO BR
        XDS     1       ; A TO M(0,15); 1 TO BR; BD – 1 TO
                        ;  BD; CONTINUE TO MOVE NEXT
                        ;  LOWER DIGIT UNTIL BD GOES
                        ;  PAST 0 AND SKIPS
        JP      MV1     ; HERE IF NO SKIP

; LEFT SHIFT DOTTED AREAS OF FIGURE 4.1
; 0 TO M(0,12) → M(0,12) → M(0,13) → M(0,14) → M(0,15) TO A

        CLRA            ; 0 TO A
        LBI     0,12    ; 0 TO BR; 12 TO BD
LSHFT   XIS             ; M(0,12) TO A; 0 TO M(0,12)
        JP      LSHFT   ; EXCHANGE A INTO BD, LEFT
                        ;  SHIFT NEXT HIGHER DIGIT UNTIL
                        ;  "BD" GOES PAST 15 AND SKIPS
```

4.3 Subroutine Techniques

Any section of program code used repeatedly within the main program should be coded as a subroutine, preferably on "subroutine pages" 2 or 3 for the reasons discussed above. Subroutines are jumped to or "called" by the JSRP or JSR (double-byte) instruction, both of which "push" the stack, saving the next memory location address after the subroutine call in the SA subroutine-save *register.* The other subroutine-save registers are correspondingly pushed. Subroutine nesting on the COP420-series is permitted to 3 levels, since this device contains 3 subroutine-save registers.

Subroutines should terminate with a RET or RETSK instruction, both of which "pop" the subroutine stack, with the program return address in SA being placed in the program counter register. The other subroutine-save registers are also popped. The contents of SC, which is the bottom-most subroutine-save register, are retained in SC in addition to being placed in SB.

It is convenient to think of a subroutine as a program module. The programmer should make its interface to the calling program as clearly defined and as simple as possible. *The interface* (including data memory registers, entry points, etc., used by the subroutine) *should be documented fully by comments to the code.*

Subroutine examples presented in this chapter often use the double-byte JSR instruction to call subroutines since no restrictions are associated directly with its use. When writing an actual program, programmers should use the more efficient single-byte JSRP instruction as well as use the double-page boundaries of subroutine pages 2 and 3 for placement of subroutine code (as discussed above) for efficient single-byte jumps while in these pages using the JP instruction.

It is often useful to define multiple-entry points for a single subroutine. The successive-skip feature of the LBI instruction often facilitates this technique. For example, see Register Move Routines, Section 4.4.

The RETSK instruction allows the programmer to use an alternate return to the main program (skipping the first program instruction encountered upon return) based upon tests or computations made within the subroutine itself. Example:

```
        .PAGE 0
        .
        .
        .
        JSRP    ADD     ; CALL ADD SUBROUTINE
        .               ; RETURN HERE IF RESULT ≤ 9
        .               ; RETURN HERE IF RESULT > 9
        .
        .PAGE 2         ; START PAGE 2 CODE
        .
        .
        .
ADD:    ADD             ; ADD SUBROUTINE — ADDS TWO
        .               ;  BCD DIGITS; RESULT TO A
        .
        AISC    7       ; OVERFLOW AND SKIP IF RESULT
        .               ;  > 9
        RET             ; RETURN WITHOUT SKIP (RESULT
        .               ;  ≤ 9)
        RETSK           ; RETURN THEN SKIP (RESULT > 9)
```

4.4 Utility Routines

Programmers often build a library of basic routines which are useful in numerous applications. This and the following sections provide examples of several such "utility" routines.

Register Move Routine

It is often necessary to move data from one memory register to another. The following are examples of this type of routine. Note that the routines may be easily modified to perform moves in the opposite direction (e.g., from register 1 to 0) or to include a move of register 1 to 2.

ADJACENT MEMORY MOVE ROUTINE

```
; ADJACENT MEMORY REGISTER MOVE, MULTIPLE ENTRY POINT SUBROUTINE
; MOV0T1: MOVE MEMORY REGISTER 0 TO REGISTER 1 ENTRY POINT
; MOV2T3: MOVE MEMORY REGISTER 2 TO REGISTER 3 ENTRY POINT
; ROUTINE MOVES DIGITS 15 THROUGH 0
; PREVIOUS CONTENTS OF A AND B ARE LOST

MOV0T1:  LBI   0,15   ; POINT TO M(0,15)
MOV2T3:  LBI   2,15   ; NOTE LBI SUCCESSIVE SKIP FEATURE
MOV:     LD    1      ; TRANSFER M TO A; EXCLUSIVE-OR 1 WITH BR
         XDS   1      ; EXCHANGE A WITH M; EXCLUSIVE-OR 1 WITH BR; DECREMENT BD
         JP    MOV    ; JUMP TO "MOV" IF MORE DIGITS TO MOVE
         RET          ; RETURN WHEN XDS SKIPS (LAST DIGIT MOVED)
```

DATA MEMORY SHIFT AND ROTATE ROUTINES

```
; MULTIPLE ENTRY POINT SUBROUTINE TO RIGHT SHIFT MEMORY REGISTER 0, 1, 2, OR 3 ONE DIGIT POSITION
; ZEROS ARE SHIFTED INTO DIGIT 15
; PREVIOUS CONTENTS OF A AND B ARE LOST
; RSH0: RIGHT SHIFT REGISTER 0 ENTRY POINT
; RSH1: RIGHT SHIFT REGISTER 1 ENTRY POINT
; RSH2: RIGHT SHIFT REGISTER 2 ENTRY POINT
; RSH3: RIGHT SHIFT REGISTER 3 ENTRY POINT

RSH0:    LBI   0,15   ; POINT TO DIGIT 15 IN APPROPRIATE REGISTER
RSH1:    LBI   1,15   ; NOTE LBI SUCCESSIVE SKIP FEATURE
RSH2:    LBI   2,15
RSH3:    LBI   3,15
         CLRA         ; ZEROS IN FIRST DIGIT (DIGIT 15)
SHFTR:   XDS          ; SHIFT RIGHT*
         JP    SHFTR  ; CONTINUE UNTIL ENTIRE REGISTER SHIFTED
         RET          ; RETURN WHEN FINISHED ("XDS" SKIPS)
```

*NOTE THAT THE ABOVE ROUTINE CAN SHIFT THE REGISTERS ONE DIGIT TO THE *LEFT* USING THE "XIS" INSTRUCTION IN PLACE OF "XDS" AND STARTING AT DIGIT 0.

```
; MULTIPLE ENTRY POINT SUBROUTINE TO LEFT SHIFT THE BITS OF A MEMORY DIGIT
; UPON ENTRY, B MUST POINT TO THE DIGIT TO BE SHIFTED
; ZEROS ARE SHIFTED IN FROM THE RIGHT
; PREVIOUS CONTENTS OF A ARE LOST
; LEF1: SHIFT DIGIT LEFT 1 BIT ENTRY POINT
; LEF2: SHIFT DIGIT LEFT 2 BITS ENTRY POINT
; LEF3: SHIFT DIGIT LEFT 3 BITS ENTRY POINT

LEF3:    LD    ; DIGIT TO A
         ADD   ; ADD DIGIT TO ITSELF
         X     ; SHIFTED DIGIT TO MEMORY
LEF2:    LD
         ADD
         X
LEF1:    LD
         ADD
         X
         RET
```

```
; MULTIPLE ENTRY POINT SUBROUTINE TO LEFT ROTATE THE BITS OF A MEMORY DIGIT
; UPON ENTRY, B MUST POINT TO THE DIGIT TO BE ROTATED
; PREVIOUS CONTENTS OF A ARE LOST
; LR01: ROTATE DIGIT LEFT 1 BIT ENTRY POINT
; LR02: ROTATE DIGIT LEFT 2 BITS ENTRY POINT
; LR03: ROTATE DIGIT LEFT 3 BITS ENTRY POINT (SAME AS RIGHT ROTATE 1)

L0R3:   JSR     LR01    ; ROTATE 1, THEN 2 MORE
L0R2:   JSR     LR01
L0R1:   LD              ; DIGIT TO A
        ADD             ; ADD DIGIT TO ITSELF
        X               ; EXCHANGE M WITH A
        AISC    8       ; WAS MEMORY BIT3 ON?
        RET             ; NO, RETURN
        SMB     0       ; YES, WRAP AROUND BIT0
        RET
```

ACCUMULATOR SHIFT ROUTINE:

```
; SUBROUTINE TO LEFT SHIFT BITS OF A BY USING THE SIO REGISTER (SIO MUST BE ENABLED AS A SERIAL SHIFT REGISTER)
; SI MUST BE CONNECTED TO LOGIC "0" (GROUND)
; ZEROS ARE SHIFTED IN FROM THE RIGHT
; LFTA1: LEFT SHIFT A 1 BIT ENTRY POINT
; LFTA2: LEFT SHIFT A 2 BITS ENTRY POINT
; LFTA3: LEFT SHIFT A 3 BITS ENTRY POINT

LFTA1:  XAS             ; A TO SIO
LFT2:   XAS             ; SIO TO A (SIO SHIFT RIGHT 1 BIT)
        RET
LFTA2:  XAS             ; A TO SIO
LFT3:   JP      LFT2    ; DELAY 1 INSTRUCTION CYCLE TIME — SIO SHIFT RIGHT 1 MORE BIT
LFTA3:  XAS             ; A TO SIO
        JP      LFT3    ; DELAY 1 INSTRUCTION CYCLE TIME — SI SHIFT RIGHT 2 MORE BITS
```

CLEAR DATA MEMORY ROUTINE:

```
; SUBROUTINE TO CLEAR ALL RAM
; CLEAR REGISTERS 3 THROUGH 0 IN SUCCESSION, THEN RETURN

CLRAM:  LBI     3,15    ; START BY CLEARING REGISTER 3
CLR:    CLRA            ; 0 TO A
        XDS             ; EXCHANGE WITH DIGIT 15, DECREMENT DIGIT
        JP      CLR     ; CONTINUE UNTIL DIGIT 0 CLEARED
        XABR            ; BR TO A
        AISC    15      ; REGISTER 0 CLEARED?
        RET             ; YES, RETURN
        XABR            ; NO, REPLACE BR - 1 INTO BR
        JP      CLR     ; CLEAR NEXT REGISTER
```

4.5 Timing Considerations

Programmers must often synchronize programs with external events ("real-time" programming). Such programs must be balanced with respect to the execution times of the various branches taken by the program. To ensure equal execution times, program timing delays are added. There are numerous ways of introducing timing delays, the simplest but least efficient involving the use of NOPs. Obviously these are appropriate for only the shortest delays.

A counting loop, such as:

```
        CLRA
        AISC    1
        JP      .-1     ; ADD 1 TO A UNTIL A
CONTINUE: .             ; OVERFLOWS*
          .
          .
```

is more efficient for longer delays, but destroys the previous contents of A. Another method is to use a "scratch-pad" counter in data memory using the XAD instruction. For example, assuming the use of a counter in M(3,15):

```
        XAD     3,15    ; COUNTER TO A; A TO M(3,15)
        AISC    1       ; ADD 1 TO COUNTER UNTIL IT
        JP      .-1     ; OVERFLOWS*
        XAD             ; RESTORE A THEN CONTINUE
        .
        .
```

*Note: The above timing code example shows the use of a special assembler symbol in the operand of the JP instruction. Namely, the operand of the JP instruction, rather than using a program label, references the

assembler location counter (which equals the address of the current program address). The "." signifies the assembler location counter and the value of the operand equals the location counter minus the number of memory bytes to the right of the "." sign. Use of the "." location pointer symbol for transfer of control instructions facilitates coding in avoiding the need to create unique program labels to reference memory addresses.

Larger delays may be implemented by using multi-digit RAM counters. Another technique is calling unrelated subroutines which change registers or memory locations not currently in use or whose net effect on memory is null. An example of the latter technique is illustrated below.

```
JSR     LR03    ; LEFT ROTATE 3 BITS
JSR     LR01    ; LEFT ROTATE 1 MORE BIT
```

This combination of subroutines only affects A, while maintaining the integrity of data in the rotated memory digit.

4.6 BCD Arithmetic Routines

BCD data manipulation routines are essential in applications which interface with human operators of a microcomputer system. They are easily translated to and from codes used by decimal displays and keyboards. The COP400 series instruction set and internal architecture has been designed to perform BCD routines efficiently. The following routines are examples of simple BCD data manipulation routines.

Unsigned BCD Integer Add and Subtract Routines

The following programs present unsigned BCD integer add and subtract subroutines. Data is stored in data memory registers 0 and 1 and is 13 digits long, occupying memory digits 0 through 12, respectively. The most significant BCD digit is in memory digit 12. The techniques used to manipulate the contents of memory address register B are common to many arithmetic routines. The LD and XIS instructions transfer data between memory and A. After the transfer they modify B. LD 1 causes a "1" to be exclusive-ORed with Br. Since, in these routines, Br is always equal to 1 when the LD 1 instruction operates upon it, Br is always changed to 0. (LD 1 causes Br to point to memory register 0.) Similarly, XIS 1 also changes Br to point to memory register 0, *as well as* incrementing the value of Bd to point to the next higher memory *digit*. Thus, Br "flip-flops" between registers 1 and 0 while Bd "walks-up" the digits of the registers.

```
; SUBROUTINE TO DO UNSIGNED BCD INTEGER ADD OF R1 AND R0, RESULT TO R0
; EACH INTEGER OCCUPIES MEMORY DIGITS 0 (LOW ORDER) THROUGH 12 (HIGH ORDER)
; ON RETURN, C = 1 INDICATES OVERFLOW
; PREVIOUS CONTENTS OF A AND B ARE LOST
; ENTRY POINT: BCDADD

BCDADD:  LBI     1,0     ; POINT TO LOW ORDER DIGIT, REGISTER 1
         RC              ; INITIALIZE C TO "0" (NO CARRY)
ADDL:    LD      1       ; MOVE R1 DIGIT TO A, POINT TO SAME DIGIT IN R0
         AISC    6       ; ADD BCD CORRECTION FACTOR OF 6 TO A
         ASC             ; ADD R0 DIGIT TO R1 DIGIT
         ADT             ; RESTORE BCD VALUE IF BCD CORRECTION NOT NECESSARY
         XIS     1       ; MOVE SUM DIGIT TO R0: POINT TO R1, NEXT HIGHER DIGIT
         CBA             ; BD TO A
         AISC    3       ; LAST DIGITS ADDED?
         JP      ADDL    ; NO, ADD NEXT HIGHER DIGITS
         RET             ; YES, RETURN

; SUBROUTINE TO DO UNSIGNED BCD INTEGER SUBTRACT
; MINUEND IS IN R0, SUBTRAHEND IS IN R1
; DIFFERENCE IS PLACED IN R0
; MINUEND, SUBTRAHEND AND DIFFERENCE DIGITS EACH OCCUPY MEMORY DIGITS 0 (LOW ORDER) THROUGH 12 (HIGH ORDER)
; ON RETURN: C = 1 INDICATES NO BORROW, C = 0 INDICATES BORROW
; PREVIOUS CONTENTS OF A AND B ARE LOST
; ENTRY POINT: BCDSUB

BCDSUB:  LBI     1,0     ; POINT TO LOW ORDER DIGIT IN R1
         SC              ; INITIALIZE C TO "1" (NO BORROW)
SUB:     LD      1       ; LOAD R1 DIGIT TO A, POINT TO SAME DIGIT IN R0
         CASC            ; SUBTRACT R1 DIGIT FROM R0 DIGIT
         ADT             ; BCD ADJUST IF BORROW (C = 0)
         XIS     1       ; PLACE DIFFERENCE DIGIT IN R0, POINT TO NEXT HIGHER DIGIT IN R1
         CBA             ; BD TO A
         AISC    3       ; HIGH ORDER DIGITS (12) SUBTRACTED?
         JP      SUB     ; NO, SUBTRACT NEXT HIGHER DIGITS
         RET             ; YES, RETURN
```

BCD Integer Multiply Routine

This routine will multiply the contents of data memory register 2 with register 1, placing the result in register 2 (digits 0-12). It also calls the BCD add routine ("BCDADD") given above. Note that a loop-counter is contained in M(0,13) which causes the program to return after all 12 digits have been multiplied. Also note the alternate-return feature of page 3 subroutine TMZERO (Test Memory Digit = 0). A flowchart for the routine is given in Figure 4.2.

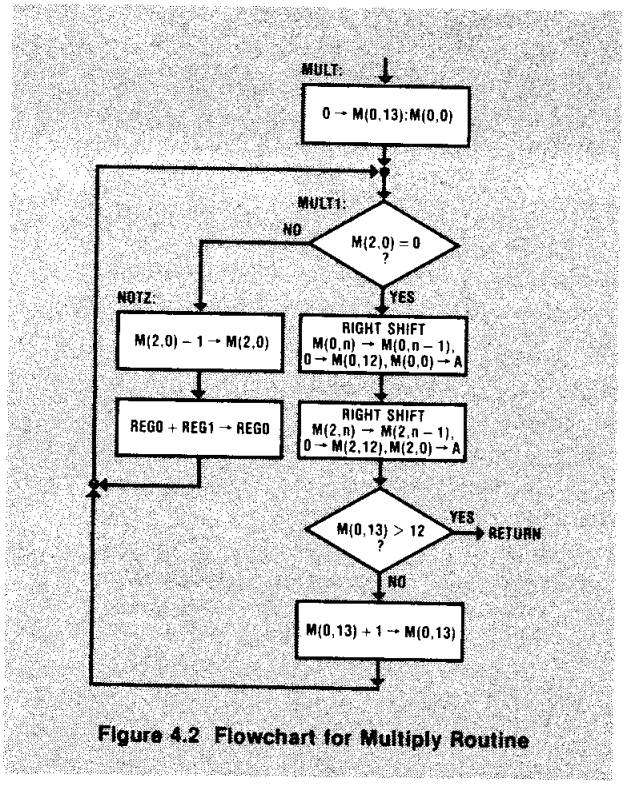


Figure 4.2 Flowchart for Multiply Routine

```
; TWO-LEVEL BCD INTEGER MULTIPLY SUBROUTINE
; 12 DIGIT BCD INTEGER CONTAINED IN REGISTER 1, DIGITS 0 - 12 (LOW ORDER TO HIGH ORDER) MULTIPLIED BY 12 DIGIT BCD
;  INTEGER CONTAINED IN REGISTER 2, DIGITS 0 - 12 (LOW ORDER TO HIGH ORDER), RESULT TO REGISTER 2
; MULTIPLICATION OF DIGITS PERFORMED BY MULTIPLE ADDITIONS OF REGISTER 1 ACCORDING TO VALUE OF REGISTER 2
;  DIGITS
; DIGIT ADDITION RESULTS TEMPORARILY STORED IN R0 AND CONSECUTIVELY RIGHT SHIFTED INTO RESULT REGISTER 2, HIGH
;  ORDER DIGIT
; ENTRY POINT: MULT
; SUBROUTINES CALLED: RSHR0, RSHR2, CLR, DEC 1, INC 1, TMZERO, BCDADD

MULT:   LBI     0,13        ; POINT TO M(0,13)
        JSR     CLR         ; CLEAR REGISTER 0, DIGITS 13 - 0
MULT1:  LBI     2,0         ; POINT TO M(2,0)
        JSR     TMZERO      ; IS M(2,0) = 0?
        JP      NOTZ        ; NO, JUMP TO NOTZ
        JSR     RSHR0       ; YES, RIGHT SHIFT REGISTER 0, DIGITS 12 - 0
        JSR     RSHR2       ; RIGHT SHIFT REGISTER 2, DIGITS 12 - 0
        LBI     0,13        ; POINT TO LOOP COUNTER
        LD                  ; LOOP COUNTER TO A
        AISC    3           ; IS COUNTER > 12
        JP      .+2         ; NO, CONTINUE
        RET                 ; YES, ALL DIGITS MULTIPLIED, RETURN
        JSR     INC1        ; CONTINUE, INCREMENT LOOP COUNTER DIGIT
        JP      MULT1       ; MULTIPLY NEXT HIGHER ORDER DIGITS
NOTZ:   JSR     DEC1        ; DECREMENT M(2,0)
        JSR     BCDADD      ; ADD R0, DIGITS 0 - 12, TO R1, DIGITS 0 - 12, RESULT TO R0
        JP      MULT1       ; JUMP BACK TO MULT 1

; MULTIPLE ENTRY POINT SUBROUTINE TO RIGHT SHIFT DIGITS 12 - 0 OF REGISTER 0 OR 2
; ON RETURN A CONTAINS LOW ORDER REGISTER DIGIT
; RSHR0: RIGHT SHIFT DIGITS OF REGISTER 0 ENTRY POINT
; RSHR2: RIGHT SHIFT DIGITS OF REGISTER 2 ENTRY POINT

RSHR0:  LBI     0,12        ; POINT TO HIGH ORDER DIGIT, REGISTER 0
RSHR2:  LBI     2,12        ; POINT TO HIGH ORDER DIGIT, REGISTER 2
RSH:    XDS                 ; SHIFT RIGHT DIGITS 12 - 0 IN REGISTER
        JP      RSH
        RET
```

```
; SUBROUTINE TO CLEAR ALL DIGITS TO THE RIGHT AND INCLUSIVE OF A HIGH-ORDER DIGIT OF A REGISTER
; ON ENTRY, B MUST POINT TO THE REGISTER AND HIGH ORDER DIGIT NUMBER

CLR:    CLRA
        XDS                     ; CLEAR REGISTER, STARTING WITH HIGH ORDER DIGIT
        JP          CLR
        RET                     ; RETURN WHEN DIGIT 0 CLEARED

; MULTIPLE ENTRY SUBROUTINE TO EITHER DECREMENT OR INCREMENT BY 1 THE VALUE OF A MEMORY DIGIT
; ON ENTRY, B MUST POINT TO THE DIGIT TO BE OPERATED UPON
; DEC1: ENTRY POINT TO DECREMENT A DIGIT
; INC1: ENTRY POINT TO INCREMENT A DIGIT

DEC1:   CLRA
        COMP                    ; 15 TO A
ADEX:   ADD                     ; ADD MEMORY DIGIT TO A
        X                       ; EXCHANGE BACK TO MEMORY
        RET                     ; RETURN
INC1:   CLRA
        AISC        1           ; 1 TO A
        JP          ADEX        ; ADD AND EXCHANGE WITH MEMORY DIGIT

; SUBROUTINE TO TEST MEMORY DIGIT EQUAL TO ZERO
; ON ENTRY, B MUST POINT TO MEMORY DIGIT TO BE TESTED
; ON RETURN, SKIP FIRST INSTRUCTION IF MEMORY DIGIT EQUAL TO ZERO
; NORMAL RETURN IF MEMORY DIGIT NOT EQUAL TO ZERO

TMZERO: CLRA                    ; 0 TO A
        SKE                     ; DIGIT = ZERO?
        RET                     ; NO, NORMAL RETURN
        RETSK                   ; YES, RETURN THEN SKIP
```

4.7 Simple Display Loop Routine

The following routine is a simple LED display loop routine. It illustrates the use of LEI and LQID instructions, both designed to facilitate the outputting of segment data to a multiplexed display. As explained in Section 3.2, LEI Instruction description, setting bit 2 of the EN register enables Q latch (segment) data to the L I/O ports; resetting EN_2 disables the L I/O ports, providing segment blanking for the LED display. EN_2 is set and reset, respectively, by the LEI 4 and LEI 0 instructions.

As explained in Sections 3.2 and 4.1, LQID loads the 8-bit Q register with the contents of a ROM location pointed to by A and M (ROM "lookup" data must be within the same 4-page ROM block as the LQID instruction). In this example, since A is always equal to 0 at the time of the LQID instruction, the ROM data accessed by this instruction must be within the first 16 words of the first page of the ROM block in which the LQID instruction is located as pointed to by the 4-bit contents of M (P_9 and P_8 remain the same, P_7-P_4 equal "0"). For example, if, as is the case for the following routine, LQID is in page 5, it will lookup data within one of the first 16 locations of page 4. The value of the contents of the memory digit pointed to by the B register at the time of the LQID instruction determines which one of the 16 words is accessed (e.g., if M = 2, word 2 is loaded into Q).

Due to these considerations, page 4, words 0-9 should equal the 8-bit, seven-segment decode lookup data for the BCD digits 0-9 respectively. (In this example the low-order bit — decimal point — of each lookup data word is reset, signifying that the decimal point is off.) ROM seven-segment decode lookup data is placed in ROM memory locations by the Assembler .WORD directive. (See *PDS User's Manual*, Section 8.4.)

Another feature of this routine is the dual function of Bd. Its value may be output directly to the D outputs to select one of 16 digits of the multiplexed display (assuming the D outputs are connected to a 1-of-16 decoder/driver device). Also, its value is used to select one of 16 RAM digits whose contents are used by the LQID instruction to access the segment data to be output to the selected digit. To facilitate coding (by avoiding the need to change the value of Bd after its contents are output to D to select or display digit), RAM digit locations should correspond to the digit of the display. In other words, RAM digits 0-15 should contain, respectively, the LQID pointers to segment data for display digits 0-15. This technique, used below, allows Bd to first enable the appropriate display digit and then, without its value being changed, to point to the RAM digit used to access the segment data for the same display digit.

```
; SEVEN-SEGMENT DECODE DATA TABLE:
; ROM BITS I7 - I0 = SA - SG, D.P. (DECIMAL POINT) BITS, RESPECTIVELY

            .PAGE     4          ; PLACE LOOKUP DATA IN WORDS 0 - 9, PAGE 4
LOOKUP:     .WORD     X'FC       ; = 0 (SEVEN-SEGMENT DECODE HEX VALUES)
            .WORD     X'60       ; = 1
            .WORD     X'DA       ; = 2
            .WORD     X'F2       ; = 3
            .WORD     X'66       ; = 4
            .WORD     X'B6       ; = 5
            .WORD     X'BE       ; = 6
            .WORD     X'E0       ; = 7
            .WORD     X'F4       ; = 8
            .WORD     X'F6       ; = 9
            .                    ; NEXT FIVE LOCATIONS CAN BE USED FOR SPECIAL ALPHABETICAL DISPLAY
            .                    ;  CHARACTER DATA
            .

; BEGIN CODE FOR DISPLAY LOOP

            .PAGE     5          ; PLACE FOLLOWING CODE ON PAGE 5
DSPLY:      LBI       0,15       ; POINT TO HIGH ORDER RAM DIGIT, BD = 15
LOOP:       CLRA                 ; A = 0 FOR LOOKUP
            LEI       0          ; BLANK SEGMENTS (EN2 = 0)
            OBD                  ; OUTPUT DIGIT VALUE
            LQID                 ; LOOKUP DATA TO Q
            LEI       4          ; OUTPUT SEGMENT DATA (EN2 = 1)
            CBA                  ; BD TO A
            AISC      15         ; DECREMENT A
            JP        .+3        ; JUMP 3 WORDS WHEN FINISHED
            CAB                  ; A(BD - 1) TO BD
            JP        LOOP       ; DISPLAY NEXT LOWER DIGIT
            .                    ; CONTINUE WHEN FINISHED
            .
            .
```

4.8 Interrupt Service Routine

As explained in Section 3.2, LEI Instruction description, setting bit 1 of the EN register enables the COP420-series and COP444L IN_1 input as an interrupt input, responding to low going pulses. Upon the occurrence of an interrupt signal, the subroutine stack is pushed and program control is transferred to the last word of page 3 (address $0FF_{16}$). The following routine contains code which may be placed at the beginning and end of the interrupt service routine to save the contents of A, C and B, freeing them for use by the interrupt routine. At the end of the routine the previous contents of A, C and B are restored for use by the main program. It should be noted that the main program need only enable IN_1 as an interrupt input once; thereafter, the interrupt service routine, itself, re-enables interrupt servicing (LEI 1 instruction before return).

```
; INTERRUPT SERVICE ROUTINE TO SAVE AND RESTORE THE CONTENTS OF A, C AND B (BR AND BD) IN MEMORY REGISTER 0,
;  DIGITS 0 - 2.
; AUTOMATIC ENTRY TO LAST WORD OF PAGE 3
; ON RETURN, IN1 INPUT RE-ENABLED AS INTERRUPT INPUT

INTSER:  NOP              ; FIRST INTERRUPT ROUTINE INSTRUCTION MUST BE A NOP (LOCATION X'FF)
         XAD     0,0      ; SAVE A IN M(0,0)
         CBA              ; BD TO A
         XAD     0,1      ; SAVE BD IN M(0,1)
         XABR             ; BR TO A
         SKC              ; CARRY = 1?
         AISC    8        ; NO, SET A3
         XAD     0,2      ; SAVE C AND BR IN M(0,2)
         .                ; PERFORM INTERRUPT ROUTINE
         .
         .
         LDD     0,2      ; M(0,2) (C AND BR) TO A
         RC               ; RESET CARRY
         AISC    8        ; A3 SET (SAVED CARRY = 0)?
         SC               ; NO, RESTORE CARRY = 1
         XABR             ; RESTORE BR
         LDD     0,1      ; M(0,1) (BD) TO A
         CAB              ; RESTORE BD
         LDD     0,0      ; M(0,0) TO A, RESTORE A
         LEI     1        ; ENABLE INTERRUPT (SET IN1)
         RET              ; RETURN FROM INTERRUPT SERVICE ROUTINE
```

4.9 Timekeeping Routine

The following multilevel subroutine counts time in a 12-hour format. It relies on the COP420 system oscillator, itself (controlled by an inexpensive 3.58 MHz color TV crystal), and the COP420 internal time-base counter for a real-time base, rather than on a 60 Hz external input. The subroutine is entered each time the SKT instruction skips, indicating time-base counter overflow. As explained in Section 3.2, SKT Instruction description, overflow frequency is dependent upon the frequency of the COPS™system oscillator. This frequency equals the oscillator frequency, first divided by 16 by the instruction cycle divider, then by 1024 by the internal 10-bit time-base counter. In this case the SKT overflow frequency will equal a fractional number: 218.478 Hz (3.58 MHz divided by 16, divided by 1024). Consequently, the timekeeping *calling* routine must execute a SKT instruction at least once approximately each 218 Hz to ensure that each SKT overflow is detected.

As indicated above, using an inexpensive TV crystal results in a fractional SKT frequency. Program compensation techniques, therefore, must be employed to derive an integer which may be used by the program in counting seconds, the basic timekeeping units.

This routine derives this integer and utilizes it to keep accurate time in the following manner:

- A 2-digit binary "SKT" counter in RAM is initialized to different values at different times during the course of an hour so that the total counts for the hour equal an integer which corresponds to the 218.478 Hz SKT frequency.
- Every odd second in the range of 0-59 seconds, the SKT counter is set to 218, decremented by 1 each time the SKT instruction skips. When decremented to 0, a 2-digit BCD "seconds" counter in RAM is incremented by 1. (The seconds counter overflows every 60 counts to a 2-digit BCD "minute" counter. The minutes counter overflows every 60 counts to a 1-digit "hours" counter.)
- Every even second in the range of 0-59 seconds, the SKT counter is set to 219 and decremented by 1, as above, each time the SKT pulse occurs.
- Every minute in the range of 0-59 minutes, the SKT counter is set to 218 and decremented as above.
- Every hour, the SKT counter is set to 199 and decremented as above.

The above compensation techniques result in a timekeeping routine which is accurate at the end of each hour. (During the hour, inaccuracy is extremely small.) The basis for the above compensation scheme is as follows:

- Using a 3.58 MHz crystal resulting in a 218.478 Hz SKT frequency, an SKT integer count of 786,521 is obtained each hour (218.478 × 3600 seconds/hour).
- Using the above compensation scheme, the same number of "SKT" counts (786,521) is required to increment the time by 1 hour. This follows since 392,400 counts are required by the "odd" seconds compensation (30 × 60 × 218 counts); 381,060 by the "even" seconds compensation (29 × 60 × 219 counts); 12,862 by the "minutes" compensation (59 × 218 counts) and 199 by the "hours" compensation — resulting in a total hours count of 786,521.

A flowchart and a RAM map for this routine are provided in Figure 4.3. Note that an assembler assignment statement is used in the assembler source code to equate the address of low order digits of the RAM SKT counter and seconds counter with the symbols "COUNT" and "SECS," respectively. This provides clearer documentation of the program since an instruction referencing the seconds counter, for instance, can use the word "SECS" instead of a numerical value in the operand field (i.e., LBI SECS). For further information on the assignment statement, see *PDS User's Manual,* Section 8.4. Also note that the program initializes the SKT counter to 218, 219 and 199, respectively, by loading its two digits with the following binary equivalent pairs (high-order value, low-order value): 13, 10; 13, 11; and 12, 7.

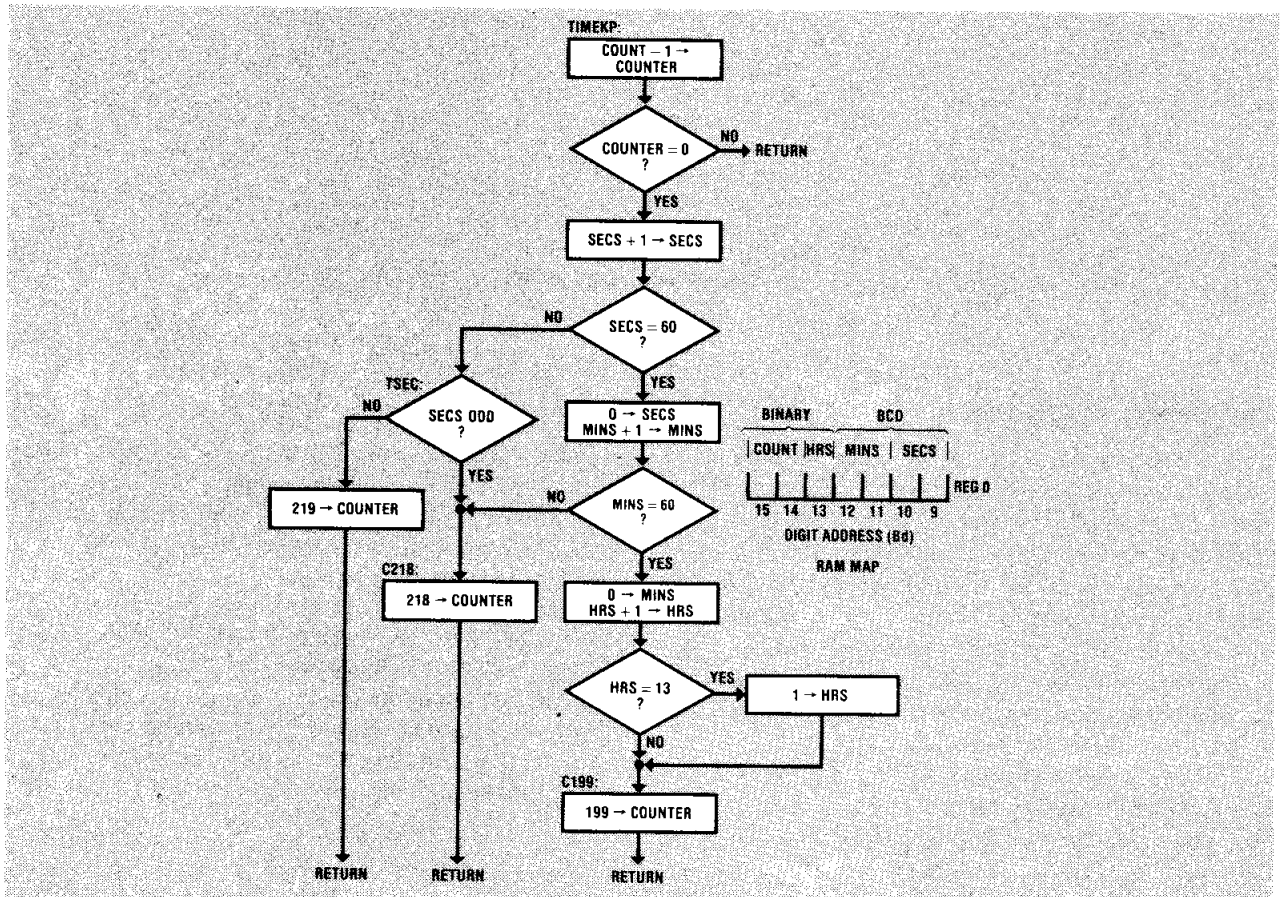


Figure 4.3 Flowchart for Timekeeping Routine

This subroutine is coded to reside on subroutine page 2. The source code provided below also illustrates the use of the PDS Assembler .LOCAL directive and local symbol labels. Specifically, the program begins and ends with a .LOCAL directive, making the memory addresses between them a local region. Within this local region, local symbols (labels whose first character is a "$") will be defined only within the local region — they will not conflict with labels appearing in other portions of program source code. This relieves the programmer from worry about duplicate label definitions, allowing the subroutine or other utility program to be included or added to different programs, regardless of the labels used by these other programs.

In effect, therefore, utility programs or commonly used subroutines may be coded in this manner and placed in separate "utility" files on a disk. They can then be added or included, when needed, to main programs at a later date. For an example of a program which includes this "TIMEKP" subroutine (using the assembler .INCLD directive), see Figure 5.18.

Local symbols must begin with a "$" and be unique within the particular local region in the first *4* characters following the "$." The programmer may, as is done in this example, use local labels with more than four characters for convenience and, although not "recognized" by the assembler, these extra characters will be printed out on the assembler output listing. Note: The label of the starting address of a local utility routine must be a *long* (regular) label, since it will be referenced by a portion of the program outside of the local region (e.g., "TIMEKP" is not a local label).

```
; PAGE 2 SUBROUTINE TO KEEP TIME IN A 12-HOUR FORMAT USING A 3.58 MHZ TV CRYSTAL
; 2-DIGIT "SKT" COUNTER CONTAINED IN M(2,15) - M(2,14): HIGH- TO LOW-ORDER
; 1-DIGIT BINARY HOURS COUNTER IN M(2,13)
; 2-DIGIT BCD MINUTES COUNTER IN M(2,12) - M(2,11): HIGH- TO LOW-ORDER
; 2-DIGIT BCD SECONDS COUNTER IN M(2,10) - M(2,9): HIGH- TO LOW-ORDER
; ENTRY POINT: TIMEKP; ENTRY UPON SKT INSTRUCTION OVERFLOW
; SUBROUTINES CALLED: INC2

            .PAGE     2          ; PAGE 2 SUBROUTINE
            .LOCAL               ; CREATE LOCAL REGION FOR LOCAL SYMBOLS
            $COUNT    =2,14      ; ASSIGN "COUNT" = ADDRESS OF LOW-ORDER SKT COUNTER DIGIT
            $SECS     =2,9       ; ASSIGN "SECS" = ADDRESS OF LOW-ORDER SECONDS COUNTER DIGIT
TIMEKP:
            LBI       $COUNT     ; POINT TO LOW-ORDER DIGIT OF SKT COUNTER
            LD                   ; LOAD DIGIT TO A
            AISC      15         ; DIGIT = 0? (A = DIGIT - 1)
            JP        $HIGHST    ; YES, TEST HIGH-ORDER DIGIT
            X                    ; NO, EXCHANGE DIGIT - 1 INTO M
            RET                  ; RETURN UNTIL NEXT SKT OVERFLOW
$HIGHTST:   XIS                  ; REPLACE DIGIT IN COUNTER, INCREMENT BD
            JP        TIMEKP+1   ; JUMP BACK AND TEST HIGH-ORDER DIGIT — IF ALREADY TESTED AND =0,
                                 ;  SKIP AND CONTINUE
            LBI       $SECS      ; POINT TO LOW-ORDER SECS DIGIT
            JSR       $INC2      ; INCREMENT SECS COUNTER
            JP        $TSEC      ; SECS < 60, TEST SECS FOR ODD OR EVEN
            STII      0          ; SECS = 60, 0 TO HIGH-ORDER DIGIT, POINT TO LOW-ORDER MINS DIGIT
            JSR       $INC2      ; INCREMENT MINS COUNTER
            JP        $C218      ; MINS < 60, SET COUNTER = 218
            STII      0          ; MINS = 60, 0 TO HIGH-ORDER DIGIT, POINT TO HOURS DIGIT
            LD                   ; LOAD HOURS DIGIT TO A
            AISC      1          ; INCREMENT HOURS
            X                    ; PLACE IN M, PREVIOUS HRS TO A
            AISC      4          ; HOURS > 12?
            JP        $C199      ; NO, SET COUNTER = 199
            STII      1          ; YES, SET HOURS = 1
$C199:      LBI       $COUNT     ; POINT TO LOW-ORDER COUNTER DIGIT
            STII      7          ; SET COUNTER = 199 (BINARY 12,7)
            STII      12
            RET                  ; RETURN UNTIL NEXT SKT OVERFLOW
$TSEC:      LBI       $SECS      ; POINT TO LOW-ORDER SECS DIGIT
            SKMBZ     0          ; SECS ODD?
            JP        $C218      ; YES, SET COUNTER = 218 (BINARY 13,10)
$C219:      LBI       $COUNT     ; NO, POINT TO LOW-ORDER COUNTER DIGIT
            STII      11         ; SET COUNTER = 219 (BINARY 13,11)
$C21X       STII13
            RET
$C218:      LBI       COUNT      ; POINT TO LOW-ORDER COUNTER DIGIT
            STII      10         ; SET COUNTER = 218
            JP        $C21X      ; JUMP TO "C21X" THEN RETURN
```

```
; SUBROUTINE TO INCREMENT A 2-DIGIT BCD RAM COUNTER
; ON ENTRY, B MUST POINT TO LOW-ORDER DIGIT OF COUNTER
; ENTRY POINT: INC2
; NORMAL RETURN IF 2-DIGIT VALUE LESS THAN 60
; RETURN THEN SKIP IF 2-DIGIT VALUE EQUAL TO 60
; BOTH RETURNS EXIT WITH B POINTING TO HIGH-ORDER DIGIT

$INC2:
        SC                      ; INITIALIZE C TO 1 TO ADD TO LOW-ORDER DIGIT
        CLRA                    ; ZERO TO A
        AISC    6               ; BCD ADJUST RESULT IF NECESSARY
        ASC                     ; IF RESULT > 9, LOW ORDER DIGIT = 0
        ADT
        XIS                     ; PLACE INCREMENTED DIGIT IN M, POINT TO HIGH-ORDER DIGIT
        CLRA                    ; ZERO TO A
        AISC    6               ; ADD CARRY, IF PROPAGATED FROM LOW-ORDER DIGIT TO HIGH-ORDER DIGIT
        ASC
        ADT                     ; BCD RESULT IF NECESSARY
        X                       ; REPLACE DIGIT IN M
        LD                      ; LOAD HIGH-ORDER DIGIT INTO A
        AISC    10              ; HIGH-ORDER DIGIT = 6 (COUNT = 60)?
        RET                     ; NO, NORMAL RETURN
        RETSK                   ; YES, RETURN THEN SKIP
        .LOCAL                  ; END LOCAL REGION
```

4.10 String Search Routine

It is often necessary to search data memory for a string of characters. The following routine searches register 0 for a match with three contiguous 4-bit characters, "X," "Y," and "Z." Note that a match with more than three characters is easily accommodated by providing for additional character tests, using the simple character test instructions provided below containing modified LDD instructions whose operands specify the additional characters to be matched. Also, the code may be easily modified to search through more than one RAM register for a match.

```
; SUBROUTINE TO SEARCH STRING OF DATA MEMORY CHARACTERS FOR A MATCH WITH "X," "Y," AND "Z" CONTIGUOUS
;  CHARACTERS
; 16 4-BIT CHARACTERS ASSUMED STORED IN M(0,15) THROUGH M(0,0)
; "X," "Y," AND "Z" CHARACTERS ASSUMED STORED IN AND ASSIGNED VALUES OF M(1,15) THROUGH M(1,13), RESPECTIVELY
; NORMAL RETURN IF NO MATCH
; RETURN THEN SKIP IF MATCH OCCURS WITH THE ACCUMULATOR CONTAINING THE DIGIT NUMBER OF "X"

        X = 1,15
        Y = 1,14
        Z = 1,13

SEARCH:
        LBI     0,15            ; POINT TO M(0,15)
LOOKX:
        LDD     X               ; X TO A
        SKE                     ; X FOUND?
        JP      NOX             ; NO, JUMP TO X
        XDS                     ; YES, POINT TO NEXT LOWER DIGIT
        JP      LOOKY           ; LOOK FOR Y MATCH, IF AT M(0,0) SKIP AND NORMAL RETURN — NO MATCH
NOX:
        LD
        XDS                     ; DECREMENT DIGIT POINTER
        JP      LOOKX           ; LOOK AGAIN FOR X MATCH, IF AT M(0,0), SKIP AND NORMAL RETURN — NO
        RET                     ;  MATCH
LOOKY:
        LDD     Y               ; Y TO A
        SKE                     ; Y FOUND?
        JP      LOOKX           ; NO, TRY AGAIN
        XDS                     ; YES, POINT TO NEXT LOWER DIGIT
        JP      LOOKX           ; LOOK FOR Z MATCH, IF AT M(0,0), SKIP AND NORMAL RETURN — NO MATCH
        RET
LOOKZ:
        LDD     Z               ; Z TO A
        SKE                     ; Z FOUND?
        JP      LOOKX           ; NO, TRY AGAIN
        OBA                     ; YES, MATCH COMPLETE, COPY Z DIGIT ADDRESS TO A
        AISC    2               ; ADD 2 TO A TO EQUAL X DIGIT ADDRESS
        RETSK                   ; RETURN THEN SKIP — MATCH FOUND
```

4.11 Programming Techniques for the COP421-Series, COP410L and COP411L

COP421-Series Programming

Since the COP421-series differs from the COP420-series only in not having the IN_3-IN_0 inputs, the foregoing programming considerations and examples for the COP420-series are, for the most part, relevant to COP421-series programming. However, due to its lack of IN inputs, the COP421-series does not include the ININ instruction, and its INIL instruction inputs only CKO into A (when CKO is programmed as a general-purpose input). The following are the results of these COP421 differences:

1. MICROBUS™ interface programming is not available since IN_3-IN_0 cannot be mask-programmed as WR, CS, and RD, respectively. Also, G_0 cannot be mask-programmed as a "ready" output to facilitate "handshaking" with a host CPU over the MICROBUS™ bus. The COP421 may still, however, function as a CPU peripheral component, relying on more general, programmed I/O techniques.
2. Due to the lack of IN inputs, other bidirectional I/O pins must be used as general purpose input pins when implementing a programmed input operation.
3. A hardware interrupt utilizing IN_1 is not possible. (Setting EN_1 has no effect on the operation of any COP421.) Any interrupt servicing must be accomplished using software interrupt techniques. (The routine provided in Section 4.8 is inapplicable to the COP421-series.)
4. A software interrupt cannot rely on the inputting and testing of the IL_3 or IL_0 latches associated with IN_3 and IN_0 inputs. Software interrupts, therefore, require that the interrupt signal be tied to one of the non-latched input pins. As a result, the input interrupt signal must be input and tested at least once during each "low" and "high" pulse occurring during each period of the signal. For example, if the interrupt signal is a 50% duty cycle, 60 Hz square wave, it must be tested at least twice every 1/60 second.

COP410L/COP411L Programming

Since the COP410L/COP411L, as with the COP421-series, does not have IN inputs, the above programming considerations relating to the COP421 apply as well as to COP410L/COP411L programming. Also, since, as discussed below, other hardware logic elements are not included in the architecture of the COP410L, the following additional considerations apply to COP410L programming:

1. The COP410L/COP411L has one-half the ROM and RAM of the COP420-series and COP421-series. ROM, therefore, consists of 512 × 8-bit words, limiting program code to eight pages (pages 0-7). RAM consists of a 32 × 4-bit RAM, organized as four RAM registers (0-3) consisting of *8* 4-bit digits (9-15,0). The LBI register reference instruction should, therefore, contain a "d" field equal to 9-15 or 0. Since all LBIs will reference RAM digits 9-15 or 0, all LBIs are single-byte instructions, occupying one word in program memory. A field restriction occurs with respect to the memory reference XAD instruction: only an XAD 3,15 instruction is valid, limiting its use to reference a RAM "scratch-pad" digit *contained in M(3,15) only.*
2. The COP410L/COP411L has 2 subroutine save registers, SA and SB. Only two levels of subroutine nesting, therefore, are allowed. The programmer should also realize that since LQID pushes and pops the stack in performing the operation associated with this instruction, only *1* level of subroutine nesting should be in effect at the time of the execution of this instruction. (Otherwise the second level of previous subroutine nesting will be disrupted — the previous contents of SB will be lost.)
3. Since the COP410L/COP411L does not have an internal divide-by-1024 time-base counter, the SKT instruction is not available. "Real-time" routines, such as 12-hour timekeeping and the like, must rely on *external* time-base inputs in order to derive a time-base for such routines (e.g., external 50/60 Hz input for time-of-day routines).
4. Certain deleted or altered instructions have already been mentioned: INIL, ININ, and SKT are not available; LBIs must have a "d" field equal to 9-15 or 0, and XAD's operand must equal 3,15. The following instructions have also been deleted from the COP410L/COP411L instruction set. To the right of each of the following deleted instructions, where appropriate, alternative COP410L/COP411L instructions are shown which, when executed in succession, will perform the same or similar operation as the deleted instruction:

Deleted Instructions	Alternative COP410L/COP411L Instructions
LDD	LBI, LD
CASC	COMP, ASC
ADT	AISC 10, NOP
CQMA	INL
OGI	OMG
XABR	
SKT	
ININ	
INIL	

For further information on deleted or altered COP410L/COP411L instructions and the operations performed by the alternative instructions given above, see Section 3.4.

5 COP400 I/O Techniques

This chapter provides information and examples pertaining to hardware and software interfacing techniques for the COP400 Microcontrollers. The information contained in this chapter is derived, in large part, from material already provided in previous chapters, particularly Chapter 2. The reader should refer to this chapter when reading the following material to obtain a complete picture of the COP400 series I/O characteristics and capability.

The following text provides I/O examples for the COP420 specifically. The I/O capability of the other members of the COP420-series (e.g., COP420L and COP420C), the COP444L and other, less inclusive devices, the COP410L and COP411L, are summarized in Table 5.1.

5.1 Hardware Interfacing Techniques

COP420 I/O

Figure 5.1 depicts the I/O lines associated with the COP420. As indicated, there are 24 I/O lines. The following discussion provides information on the capabilities of the mask-programmable I/O options associated with the COP420. These optional configurations are shown in Figure 5.2.

COP420 Inputs

COP420 inputs may be programmed either with a depletion-load device to V_{CC} or floating (Hi-Z input). All inputs are TTL/CMOS compatible. Hi-Z inputs should not be left floating; they should be connected to the output of a "high" and "low" driving device if active or to V_{CC} or ground if unused. Inputs may also be optionally programmed for higher trip levels for interfacing to non-TTL sources (e.g., keyboards, switches).

Table 5.1 COP400 Comparison Chart

I/O Pins	Bits	COP420	COP420C	COP420L	COP410L
D_{OUT}	4	TTL	LSTTL	20mA Sink	20mA Sink
G_{OUT}	4	TTL	LSTTL	20mA Sink	LS TTL
L_{OUT}	8	TTL or LED	LSTTL or LED	LS or LED	LS or LED
SO, SK	2	TTL	LSTTL	LS	LS
IN	1	4 Inputs	4 Inputs	4 Inputs	No
SI	1	Shift Register or Counter Input	Shift Register	Shift Register or Counter Input	Shift Register or Counter Input
CKI	1	Oscillator Input	Oscillator Input	Oscillator Input	Oscillator Input
CKO	1	Oscillator Out or SYNC In or General In or RAM Supply	Oscillator Out or General In	Oscillator Out or SYNC In or General In or RAM Supply	Oscillator Out or SYNC In or RAM Supply
RESET	1	RESET Input	RESET Input	RESET Input	RESET Input
V_{CC}, GND	2	Power Supply	Power Supply	Power Supply	Power Supply
Oscillator Frequency Range		0.4 to 4MHz	32kHz to 2MHz	0.2 to 2MHz	200 to 500kHz
Cycle Time		4 to 10μs	15 to 250μs	15 to 40μs	15 to 40μs
V_{CC} Supply		4.5 to 6.3V	2.4 to 6.3V	4.5 to 9.5V	4.5 to 9.5V
V_{CC} Current (max)		25mA	1mA (25μA)	8mA	5mA

COP420 Outputs

Standard Output: The N-channel device to ground is good at sinking current and is compatible with the sinking requirements of 1 TTL load (1.6mA at 0.4V); it will meet the "low" voltage requirements of CMOS logic. All output options use this device (device #1), as illustrated in Figure 5.2, for current sinking. The depletion-load device to V_{CC} provides low sourcing capability (100μA at 2.4V). While this device meets the sourcing requirements of TTL logic and will go to V_{CC} to meet the "high" voltage requirements of CMOS logic, an external resistor to V_{CC} may be required to interface to other external devices requiring higher sourcing capability. A standard output may be connected directly to the base of an external transistor for current sourcing since the depletion-load device's current capability is limited to a safe operating area. Figure 5.3 provides a summary of the characteristics of the COP420 Standard Output.

Open-Drain Output: The COP420 open-drain output uses the same enhancement mode device to ground as the standard output with the same current sinking capability. As its name implies, this output configuration does not contain a load device to V_{CC}, allowing various external pullup techniques as required by the user's application.

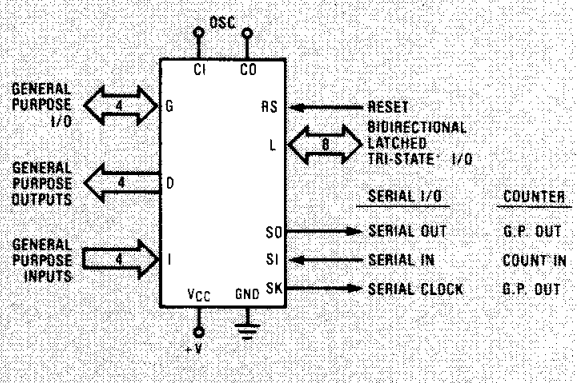
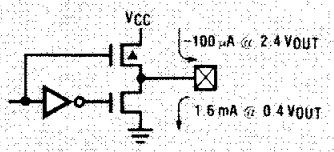


Figure 5.1 COP420 I/O Lines



"Standard" Output

N-channel Si-gate
Good at sinking current
Low sourcing — external resistors may be needed
Direct interface to: standard TTL — fanout of one transistor
CMOS
Depletion load output will go to V_{CC}

Figure 5.3 COP420 Standard Output Characteristics

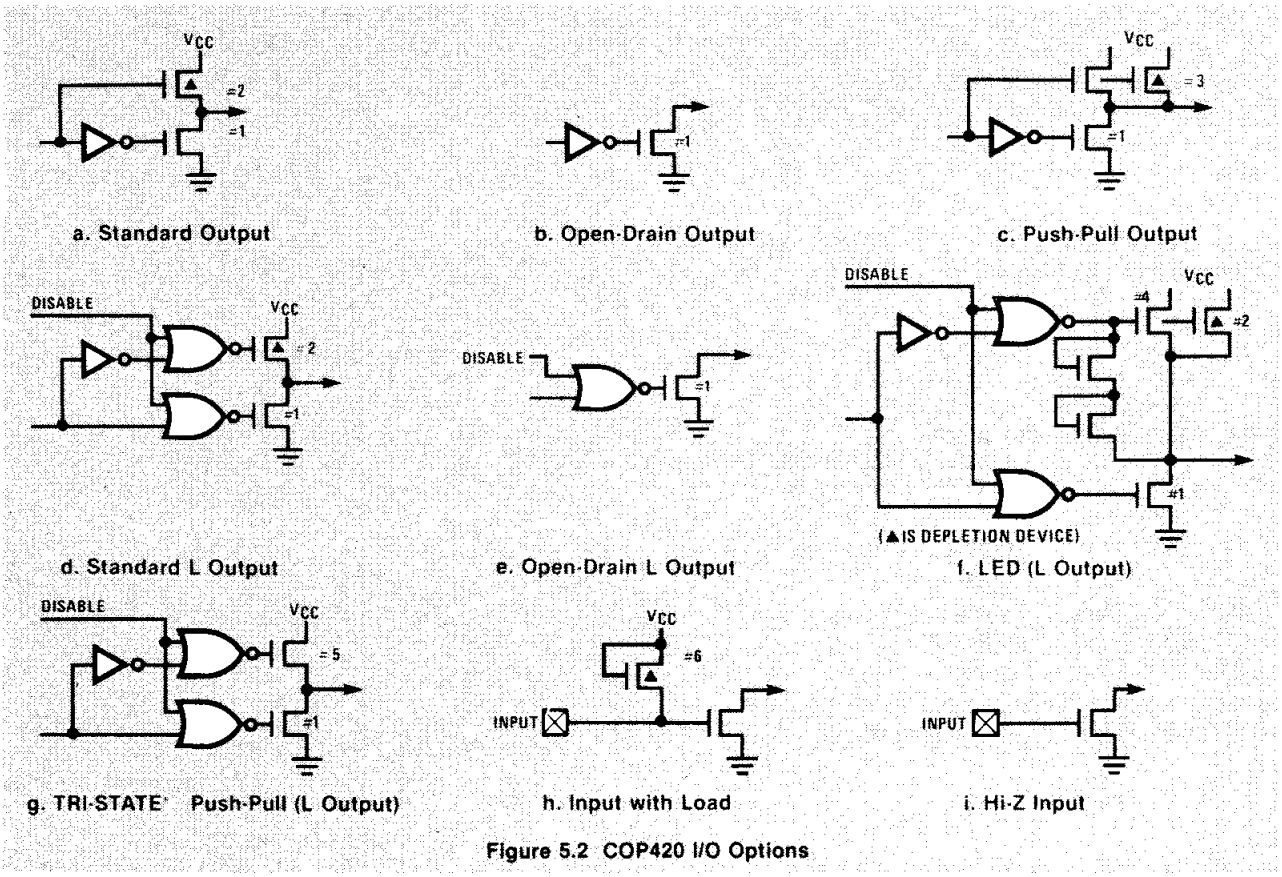


Figure 5.2 COP420 I/O Options

Push-Pull Output: The COP420 push-pull output differs from the standard output configuration in having an enhancement mode device in parallel with the depletion-load device to V_{CC}, providing greater current sourcing capability and faster rise and fall times when driving capacitive loads. This option is available for the COP420 SO and SK outputs, often tied to the highly capacitive clock lines of external shift registers to provide additional external I/O for the COP420. (For an example, see Figure 5.20.) If a push-pull output is interfaced to an external transistor, a limiting resistor must be placed in series with the base of the transistor to avoid excessive source current flow out of the push-pull output.

Figure 5.4 summarizes, in interconnect form, the information provided above relevant to the capabilities of the push-pull, open drain and standard outputs, as well as the Hi-Z and load device input configurations.

For an example of use of the SK output, configured as a push-pull output to drive the clock lines of an external shift register, see Figure 5.10.

LED Direct Drive Output: The COP420 LED direct drive output differs from the standard output configuration in two basic ways:

1. Its depletion-load device to V_{CC} is paralleled by an enhancement mode device to V_{CC} to allow for the greater current sourcing capacity required by the segments of an LED display. Source current is clamped to prevent excessive source current flow.
2. This configuration can be disabled under program control by resetting bit 2 (EN_2) of the enable register to provide simplified display segment blanking. However, while both enhancement mode devices are turned off in the disabled mode, the depletion-load device to V_{CC} will still source up to 0.125mA when this output is turned off. (This is not a worst case pull-up for keyboard input loads).

For an example of use of the L I/O ports, using this option, to directly drive the segments of a LED and VF display, respectively, see Figures 5.11 and 5.12.

TRI-STATE® Push-Pull Output

This COP420 output was designed to meet the specifications of National's MICROBUS™, outputting data over the data bus to a host CPU. It has TRI-STATE® logic to disable both enhancement mode devices to free the MICROBUS™ data lines for COP420 input operation. Figure 5.13 shows an interconnect between a host CPU and the COP420 over the MICROBUS™ using this L output option.

COP420 I/O Summary

Figures 5.5 through 5.9 provide diagrams of the internal logic and a summary of the hardware and software features associated with the COP420 I/O ports.

Interconnect Examples

Figures 5.10 through 5.14 provide interconnect diagrams illustrating several schemes for interconnecting the COP420 to external devices. Several of these interconnect diagrams, with minor variations, are used in providing software I/O techniques in the final sections of this chapter.

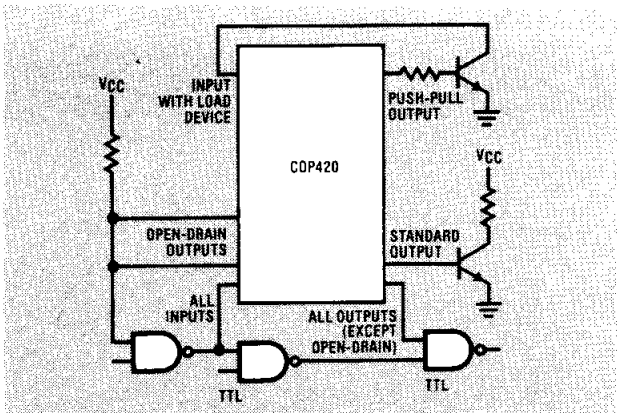
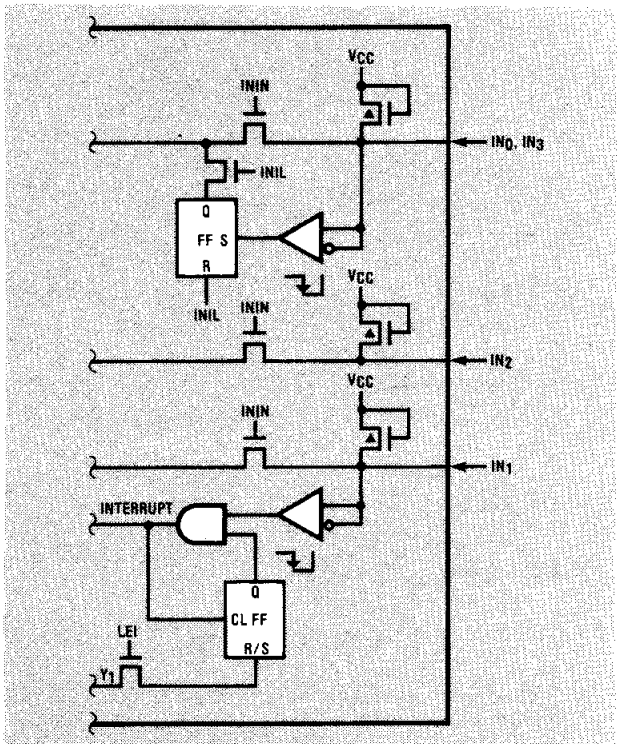


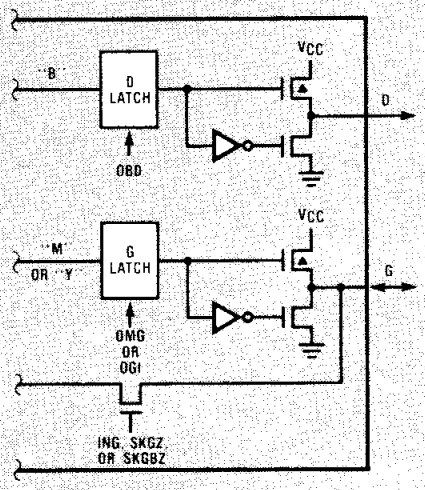
Figure 5.4 COP420 I/O Interconnect Examples



IN Inputs

1. Four general purpose inputs read directly into A (ININ instruction);
2. IN_1 can be enabled as an interrupt input (by setting EN_1);
3. IN_0 and IN_3 can "catch" low-going pulses, read into A_0 and A_3 (INIL instruction);
4. All inputs have optional pull-up load device to V_{CC} (shown in diagram), or Hi-Z (floating) inputs.

Figure 5.5 COP420 IN Port Characteristics



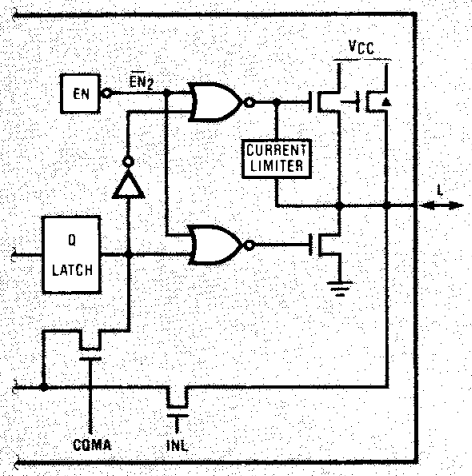
D Outputs

1. Four general purpose outputs loaded from B (OBD) instruction;
2. Standard (as shown) or open-drain outputs.

G Inputs

1. Four general purpose I/O lines loaded from memory (M) by OMG instruction or loaded with immediate data (Y) by OGI instruction;
2. Read inputs into accumulator (ING instruction), test individually (SKGBZ instruction), collectively (SKGZ instruction) for zero — seg G latch to "1" when using as input;
3. Standard (as shown) or open-drain outputs.

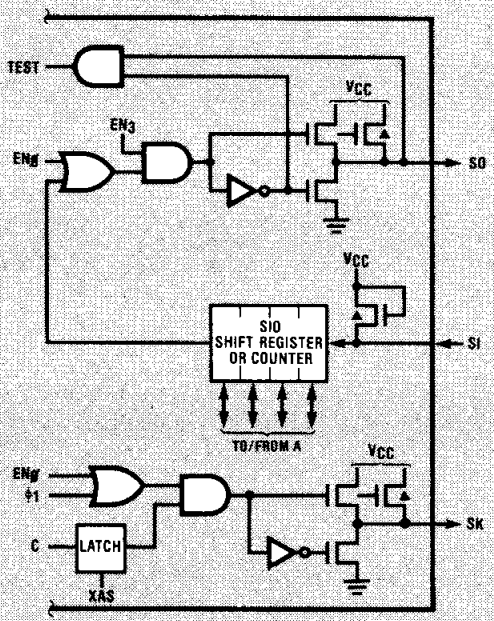
Figure 5.6 COP420 D and G Port Characteristics



L TRI-STATE® Inputs/Outputs

1. Eight TRI-STATE inputs/outputs, loaded with Q latch data by setting EN_2 or direct input of L port data to M and A (INL instruction); Q latch loaded from A and M by CAMQ instruction and read into M and A by CQMA instruction;
2. L ports TRI-STATED with $EN_2 = 0$ (if output contains depletion-load device to V_{CC}, $I_{OL} \approx 0.2$ mA @ 0V in);
3. All output options available:
   a. Standard
   b. Open-Drain
   c. Push-Pull
   d. LED Direct Drive (as shown)
   e. TRI-STATE Push-Pull

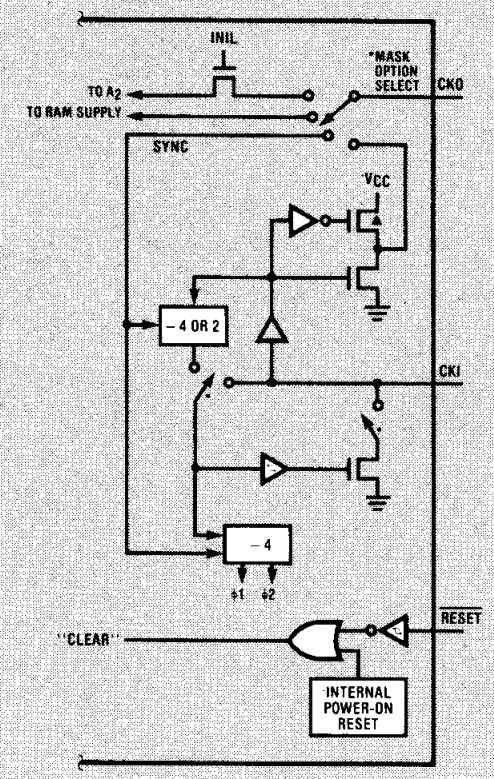
Figure 5.7 COP420 L I/O Port Characteristics



SI Input, SO, SK Outputs

1. SI is a single-pin input to the SIO register. SIO can be enabled as a 4-bit serial shift register or a 4-bit binary counter, selected by EN_0.
2. If SIO is selected as a counter, SO outputs the value of EN_3, SK outputs the value of C upon the execution of an XAS instruction.
3. If SIO is selected as a shift register, SO may be used as a serial data output and SK may be a logic controlled clock selected by EN_3.
4. The contents of SIO may be exchanged with A using an XAS instruction.
5. SI, SO and SK are also used for "in-house" standardized testing of the COP420.
6. SI may be configured with a load device to V_{CC} (as shown) or as a Hi-Z input.
7. SO and SK may be configured as:
   a. Standard
   b. Open-Drain, or
   c. Push-Pull (as shown) outputs.

Figure 5.8 SI, SO, SK Characteristics



CKO, CKI and RESET Pins

1. The COP420 CKO pin has the following options:
   a. output to crystal oscillator;
   b. general purpose input (read into A_2 by an INIL instruction);
   c. synchronization (SYNC) input;
   d. RAM power supply pin.
2. CKI has the following options:
   a. crystal oscillator input;
   b. external oscillator input;
   c. RC controlled oscillator input.
3. RESET may be used as an external reset pin or, if the power supply rise time is greater than 1 ma, as a power-on clear input.

Figure 5.9 COP420 CKO, CKI, RESET Characteristics

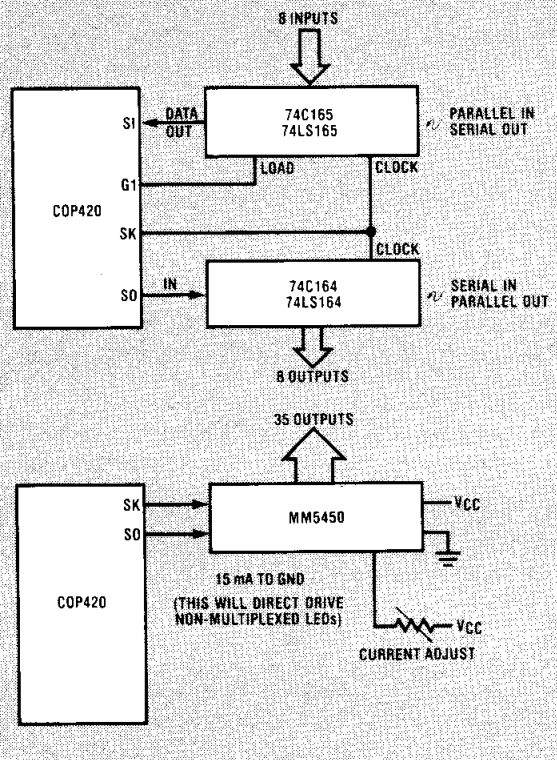
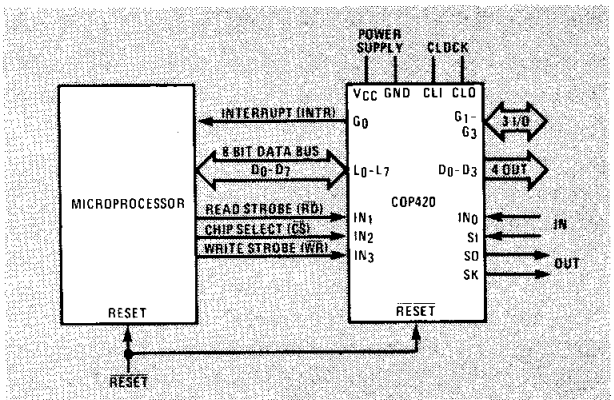


Figure 5.10 COP420 I/O Expansion



Direct hand onto National's MICROBUS™

Use: L Bus for data I/O

IN_1 for read strobe (RD)

IN_2 for chip select (CS)

IN_3 for write strobe (WR)

G_0 for interrupt request (COP420 "ready" output)

L is TRI-STATED unless both CS and WR are low.

G_0 does not TRI-STATE

Figure 5.13 MICROBUS™ Interconnect

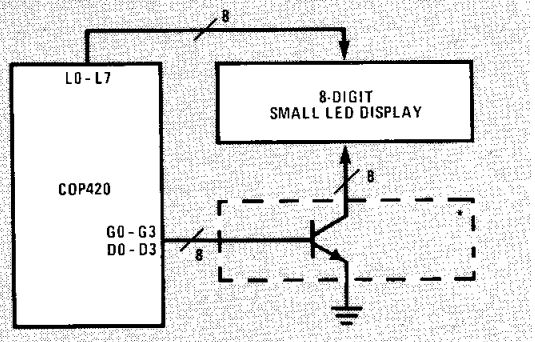


Figure 5.11 COP420 LED Display System

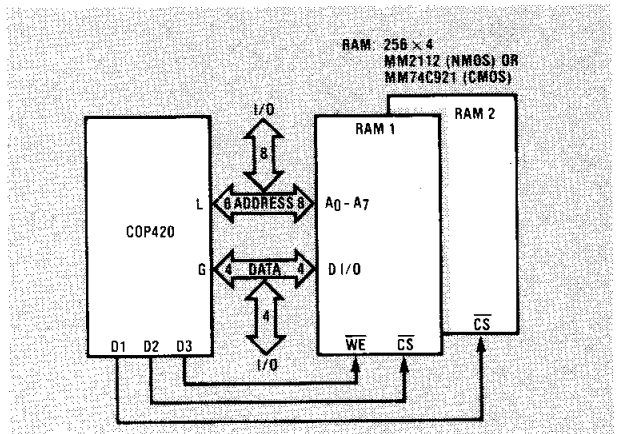


Figure 5.14 COP420 Add-On RAM

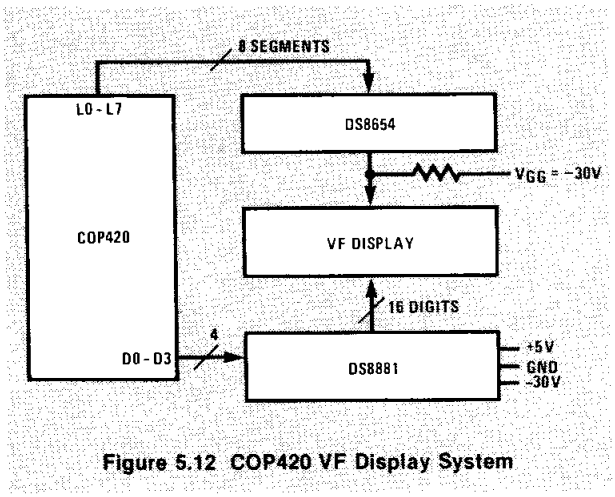


Figure 5.12 COP420 VF Display System

COP400 I/O Comparison Table

Table 5.1 provides a comparison table of the I/O capabilities of COP400 series devices. It should be understood that this is a partial listing of COP400 devices, since more inclusive parts (the COP440 and its related devices) as well as other devices will be available in the near future. For complete information on the listed devices, as well as other members of the COP400 Microcontroller family, consult the appropriate data sheets.

5.2 Software I/O Techniques

The following sections of this chapter provide several software I/O examples and techniques for interfacing the COP420 to external I/O, including program code necessary to service these peripherals.

National Semiconductor

COP420/COP421/COP422 and COP320/COP321/COP322 Single-Chip N-Channel Microcontrollers

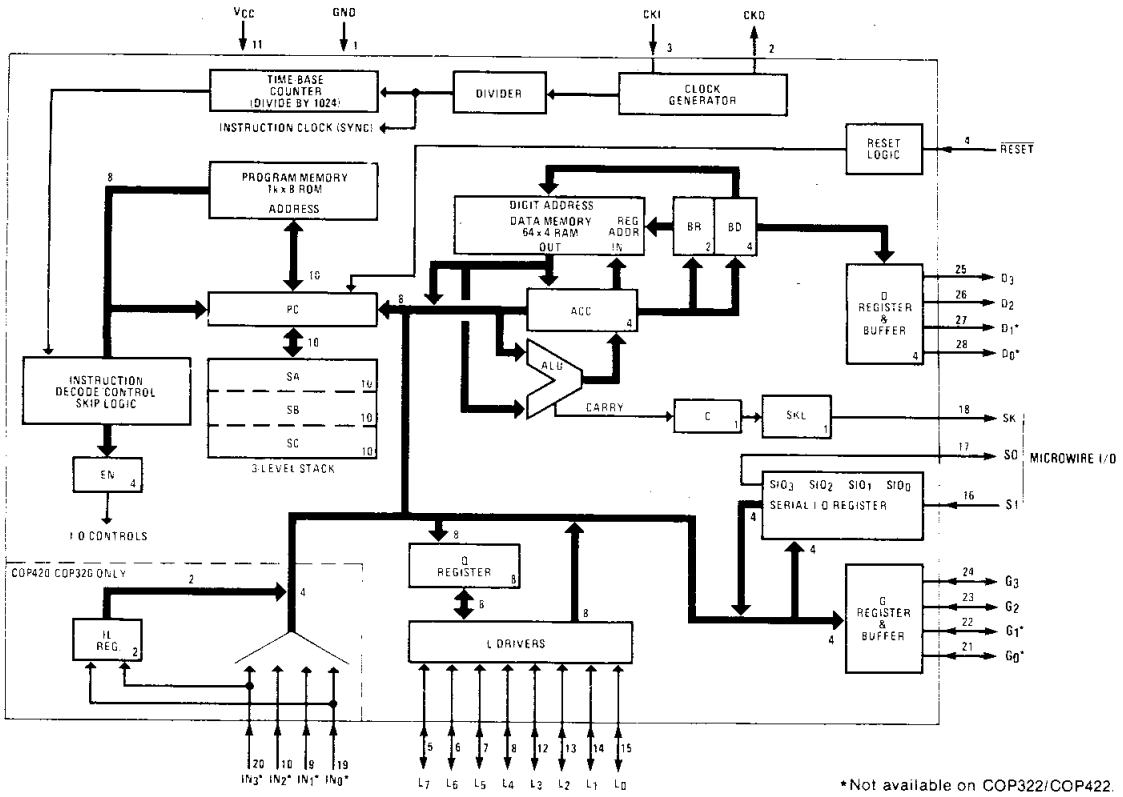
General Description

The COP420, COP421, COP422, COP320, COP321 and COP322 Single-Chip N-Channel Microcontrollers are members of the COPS™ family, fabricated using N-channel, silicon gate MOS technology. They are complete microcomputers containing all system timing, internal logic, ROM, RAM and I/O necessary to implement dedicated control functions in a variety of applications. Features include single supply operation, a variety of output configuration options, with an instruction set, internal architecture and I/O scheme designed to facilitate keyboard input, display output and BCD data manipulation. The COP421 is identical to the COP420, except with 19 I/O lines instead of 23; the COP422 has 15 I/O lines. They are an appropriate choice for use in numerous human interface control environments. Standard test procedures and reliable high-density fabrication techniques provide the medium to large volume customers with a customized Controller Oriented Processor at a low end-product cost.

The COP320 is the extended temperature range version of the COP420 (likewise the COP321 and COP322 are the extended temperature range versions of the COP421/COP422). The COP320/321/322 are exact functional equivalents of the COP420/421/422.

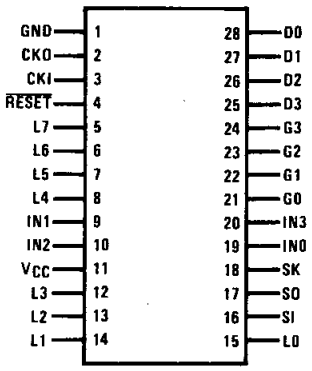
Features

- Low cost
- Powerful instruction set
- 1k × 8 ROM, 64 × 4 RAM
- 23 I/O lines (COP420, COP320)
- True vectored interrupt, plus restart
- Three-level subroutine stack
- 4.0μs instruction time
- Single supply operation
- Internal time-base counter for real-time processing
- Internal binary counter register with MICROWIRE™ compatible serial I/O capability
- General purpose and TRI-STATE® outputs
- TTL/CMOS compatible in and out
- LED direct drive outputs
- MICROBUS™ compatible
- Software/hardware compatible with other members of COP400 family
- Extended temperature range device COP320/COP321/COP322 (−40°C to +85°C)



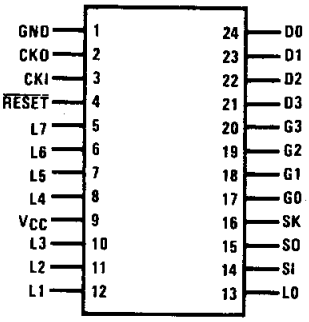
*Not available on COP322/COP422.

Figure 1. COP420/COP421/COP422, COP320/COP321/COP322 Block Diagram



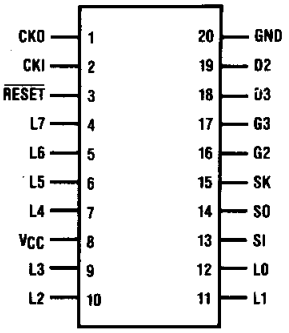
COP420, COP320

Order Number COP420N, COP320N
NS Package N28A



COP421, COP321

Order Number COP421N, COP 321N
NS Package N24A



COP422, COP322

Order Number COP422N, COP322N
NS Package N20A

Figure 2. Connection Diagrams

Pin	Description	Pin	Description
L_7-L_0	8 bidirectional I/O ports with TRI-STATE®	SK	Logic-controlled clock (or general purpose output)
G_3-G_0	4 bidirectional I/O ports	CKI	System oscillator input
D_3-D_0	4 general purpose outputs	CKO	System oscillator output (or general purpose input or RAM power supply)
IN_3-IN_0	4 general purpose inputs (COP420/320 only)	$\overline{RESET}$	System reset input
SI	Serial input (or counter input)	V_{CC}	Power supply
SO	Serial output (or general purpose output)	GND	Ground

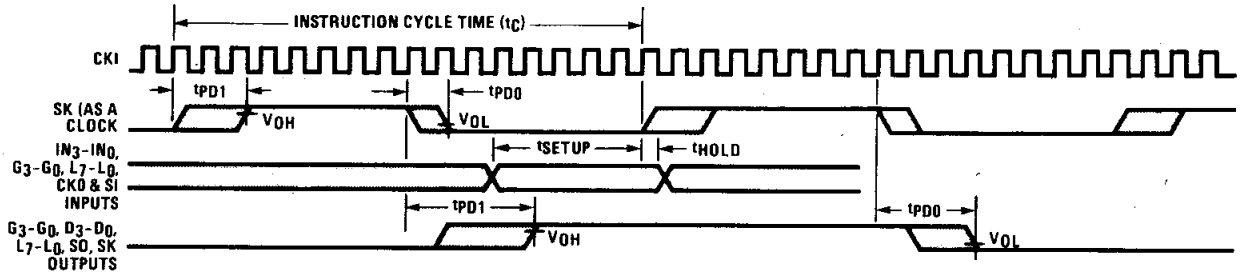


Figure 3. Input/Output Timing Diagrams (crystal divide by 16 mode)

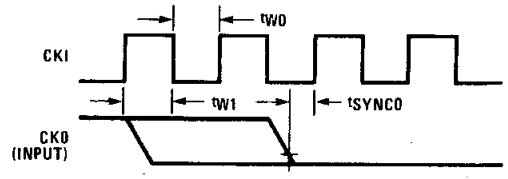


Figure 3A. Synchronization Timing

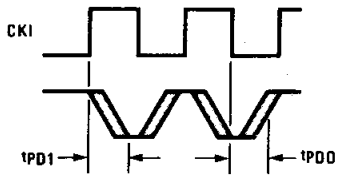


Figure 3B. CKO Output Timing

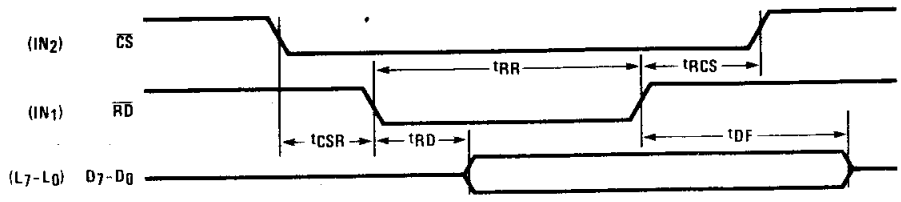


Figure 4. MICROBUS™ Read Operation Timing

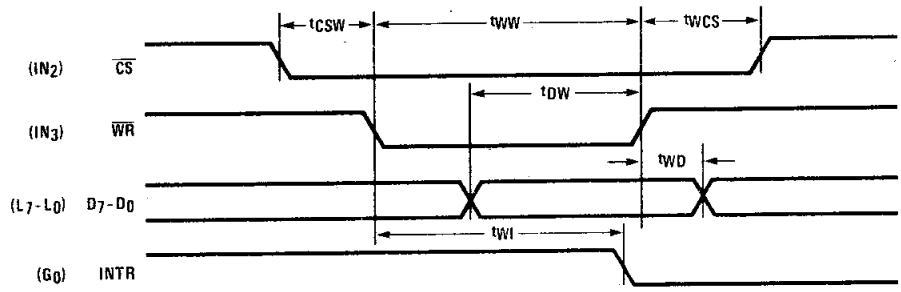


Figure 5. MICROBUS™ Write Operation Timing

Functional Description COP420/COP421/COP422, COP320/COP321/COP322

For ease of reading this description, only COP420 and/or COP421 are referenced; however, all such references apply equally to the COP422, COP322, COP320 and/or COP321, respectively.

A block diagram of the COP420 is given in figure 1. Data paths are illustrated in simplified form to depict how the various logic elements communicate with each other in implementing the instruction set of the device. Positive logic is used. When a bit is set, it is a logic "1" (greater than 2 volts). When a bit is reset, it is a logic "0" (less than 0.8 volts).

Program Memory

Program Memory consists of a 1,024 byte ROM. As can be seen by an examination of the COP420/421 instruction set, these words may be program instructions, program data or ROM addressing data. Because of the special characteristics associated with the JP, JSRP, JID and LQID instructions, ROM must often be thought of as being organized into 16 pages of 64 words each.

ROM addressing is accomplished by a 10-bit PC register. Its binary value selects one of the 1,024 8-bit words contained in ROM. A new address is loaded into the PC register during each instruction cycle. Unless the instruction is a transfer of control instruction, the PC register is loaded with the next sequential 10-bit binary count value. Three levels of subroutine nesting are implemented by the 10-bit subroutine save registers, SA, SB and SC, providing a last-in, first-out (LIFO) hardware subroutine stack.

ROM instruction words are fetched, decoded and executed by the Instruction Decode, Control and Skip Logic circuitry.

Data Memory

Data memory consists of a 256-bit RAM, organized as 4 data registers of 16 4-bit digits. RAM addressing is implemented by a 6-bit **B register** whose upper 2 bits (Br) select 1 of 4 data registers and lower 4 bits (Bd) select 1 of 16 4-bit digits in the selected data register. While the 4-bit contents of the selected RAM digit (M) is usually loaded into or from, or exchanged with, the A register (accumulator), it may also be loaded into or from the Q latches or loaded from the L ports. RAM addressing may also be performed directly by the LDD and XAD instructions based upon the 6-bit contents of the operand field of these instructions. The Bd register also serves as a source register for 4-bit data sent directly to the D outputs.

Internal Logic

The 4-bit **A register** (accumulator) is the source and destination register for most I/O, arithmetic, logic and data memory access operations. It can also be used to load the Br and Bd portions of the B register, to load and input 4 bits of the 8-bit Q latch data, to input 4 bits of the 8-bit L I/O port data and to perform data exchanges with the SIO register.

A **4-bit adder** performs the arithmetic and logic functions of the COP420/421, storing its results in A. It also outputs a carry bit to the 1-bit **C register**, most often employed to indicate arithmetic overflow. The C register, in conjunction with the XAS instruction and the EN register, also serves to control the SK output. C can be outputted directly to SK or can enable SK to be a sync clock each instruction cycle time. (See XAS instruction and EN register description, below.)

Four **general-purpose inputs, IN_3-IN_0**, are provided; IN_1, IN_2 and IN_3 may be selected, by a mask-programmable option, as Read Strobe, Chip Select and Write Strobe inputs, respectively, for use in MICROBUS™ applications.

The **D register** provides 4 general-purpose outputs and is used as the destination register for the 4-bit contents of Bd.

The **G register** contents are outputs to 4 general-purpose bidirectional I/O ports. G_0 may be mask-programmed as an output for MICROBUS™ applications.

The **Q register** is an internal, latched, 8-bit register, used to hold data loaded to or from M and A, as well as 8-bit data from ROM. Its contents are output to the L I/O ports when the L drivers are enabled under program control. (See LEI instruction). With the MICROBUS™ option selected, Q can also be loaded with the 8-bit contents of the L I/O ports upon the occurence of a write strobe from the host CPU.

The **8 L drivers**,when enabled, output the contents of latched Q data to the L I/O ports. Also, the contents of L may be read directly into A and M. As explained above, the MICROBUS™ option allows L I/O port data to be latched into the Q register. L I/O ports can be directly connected to the segments of a multiplexed LED display (using the LED Direct Drive output configuration option) with Q data being outputted to the Sa-Sg and decimal point segments of the display.

The **SIO register** functions as a 4-bit serial-in/serial-out shift register or as a binary counter depending on the contents of the EN register. (See EN register description, below.) Its contents can be exchanged with A, allowing it to input or output a continuous serial data stream. SIO may also be used to provide additional parallel I/O by connecting SO to external serial-in/parallel-out shift registers. For example of additional parallel output capacity see **Application #2.**

The XAS instruction copies C into the **SKL latch**. In the counter mode, SK is the output of SKL; in the shift register mode, SK outputs SKL ANDed with the clock.

The **EN register** is an internal 4-bit register loaded under program control by the LEI instruction. The state of each bit of this register selects or deselects the particular feature associated with each bit of the EN register (EN_3-EN_0).

1. The least significant bit of the enable register, EN_0, selects the SIO register as either a 4-bit shift register or a 4-bit binary counter. With EN_0 set, SIO is an asynchronous binary counter, *decrementing* its value by one upon each low-going pulse ("1" to "0") ocurring on the SI input. Each pulse must be at least two instruction cycles wide. SK outputs the value of SKL. The SO output is equal to the value of EN_3. With EN_0 reset, SIO is a serial shift register shifting left each instruction cycle time. The data present at SI goes into the least significant bit of SIO. SO can be enabled to output the most significant bit of SIO each cycle time. (See 4 below.) The SK output becomes a logic-controlled clock.
2. With EN_1 set the IN_1 input is enabled as an interrupt input. Immediately following an interrupt, EN_1 is reset to disable further interrupts.
3. With EN_2 set, the L drivers are enabled to output the data in Q to the L I/O ports. Resetting EN_2 disables the L drivers, placing the L I/O ports in a high-impedance input state.
4. EN_3, in conjunction with EN_0, affects the SO output. With EN_0 set (binary counter option selected) SO will output the value loaded into EN_3. With EN_0 reset (serial shift register option selected), setting EN_3 enables SO as the output of the SIO shift register, outputting serial shifted data each instruction time. Resetting EN_3 with the serial shift register option selected disables SO as the shift register output; data continues to be shifted through SIO and can be exchanged with A via an XAS instruction but SO remains reset to "0." The table below provides a summary of the modes associated with EN_3 and EN_0.

Enable Register Modes — Bits EN_3 and EN_0

EN_3	EN_0	SIO	SI	SO	SK
0	0	Shift Register	Input to Shift Register	0	If SKL = 1, SK = CLOCK If SKL = 0, SK = 0
1	0	Shift Register	Input to Shift Register	Serial Out	If SKL = 1, SK = CLOCK If SKL = 0, SK = 0
0	1	Binary Counter	Input to Binary Counter	0	If SKL = 1, SK = 1 If SKL = 0, SK = 0
1	1	Binary Counter	Input to Binary Counter	1	If SKL = 1, SK = 1 If SKL = 0, SK = 0

Interrupt

The following features are associated with the IN_1 interrupt procedure and protocol and must be considered by the programmer when utilizing interrupts.

a. The interrupt, once acknowledged as explained below, pushes the next sequential program counter address (PC + 1) onto the stack, pushing in turn the contents of the other subroutine-save registers to the next lower level (PC + 1 → SA → SB → SC). Any previous contents of SC are lost. The program counter is set to hex address 0FF (the last word of page 3) and EN_1 is reset.

b. An interrupt will be acknowledged only after the following conditions are met:
   1. EN_1 has been set.
   2. A low-going pulse ("1" to "0") at least two instruction cycles wide occurs on the IN_1 input.
   3. A currently executing instruction has been completed.
   4. All successive transfer of control instructions and successive LBIs have been completed (e.g., if the main program is executing a JP instruction which transfers program control to another JP instruction, the interrupt will not be acknowledged until the second JP instruction has been executed.

c. Upon acknowledgement of an interrupt, the skip logic status is saved and later restored upon popping of the stack. For example, if an interrupt occurs during the execution of ASC (Add with Carry, Skip on Carry) instruction which results in carry, the skip logic status is saved and program control is transferred to the interrupt servicing routine at hex address 0FF. At the *end* of the interrupt routine, a RET instruction is executed to "pop" the stack and return program control to the instruction following the original ASC. *At this time*, the skip logic is enabled and skips this instruction because of the previous ASC carry. Subroutines and LQID instructions should not be nested within the interrupt service routine, since their popping the stack will enable any previously saved main program skips, interfering with the orderly execution of the interrupt routine.

d. The first instruction of the interrupt routine at hex address 0FF must be a NOP.

e. A LEI instruction can be put immediately before the RET to re-enable interrupts.

Microbus™ Interface

The COP420 has an option which allows it to be used as a peripheral microprocessor device, inputting and outputting data from and to a host microprocessor (μP). IN_1, IN_2 and IN_3 general purpose inputs become **MICROBUS™ compatible** read-strobe, chip-select, and write-strobe lines, respectively. IN_1 becomes $\overline{RD}$ — a logic "0" on this input will cause Q latch data to be enabled to the L ports for input to the μP. IN_2 becomes $\overline{CS}$ — a logic "0" on this line selects the COP420 as the μP peripheral device by enabling the operation of the $\overline{RD}$ and $\overline{WR}$ lines and allows for the selection of one of several peripheral components. IN_3 becomes $\overline{WR}$ — a logic "0" on this line will write bus data from the L ports to the Q latches for input to the COP420. G_0 becomes INTR a "ready" output, reset by a write pulse from the μP on the $\overline{WR}$ line, providing the "handshaking capability necessary for asynchronous data transfer between the host CPU and the COP420.

This option has been designed for compatibility with National's MICROBUS™ — a standard interconnect system for 8-bit parallel data transfer between MOS/LSI CPUs and interfacing devices. (See MICROBUS™ National Publication.) The functioning and timing relationships between the COP420 signal lines affected by this option are as specified for the MICROBUS™ interface, and are given in the AC electrical characteristics and shown in the timing diagrams (figures 4 and 5). Connection of the COP420 to the MICROBUS™ is shown in Figure 6.

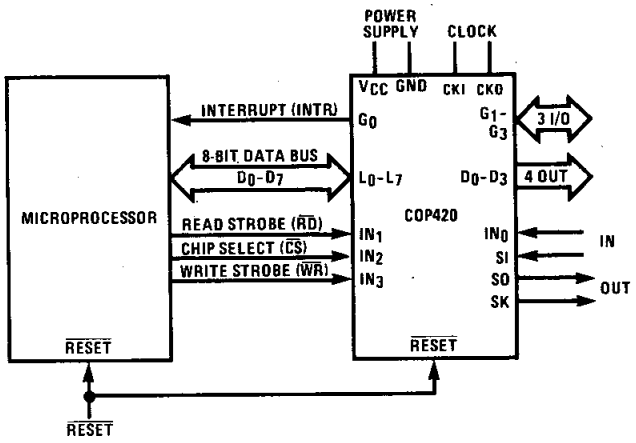


Figure 6. MICROBUS™ Option Interconnect

Initialization

The Reset Logic, internal to the COP420/421, will initialize (clear) the device upon power-up if the power supply rise time is less than 1ms and greater than 1μs. If the power supply rise time is greater than 1ms, the user must provide an external RC network and diode to the $\overline{RESET}$ pin as shown below. The $\overline{RESET}$ pin is configured as a Schmitt trigger input. If not used it should be connected to V_{CC}. Initialization will occur whenever a logic "0" is applied to the RESET input, provided it stays low for at least three instruction cycle times.

Upon initialization, the PC register is cleared to 0 (ROM address 0) and the A, B, C, D, EN, and G registers are cleared. The SK output is enabled as a SYNC output, providing a pulse each instruction cycle time. *Data Memory (RAM) is not cleared upon initialization.* The first instruction at address 0 must be a CLRA.

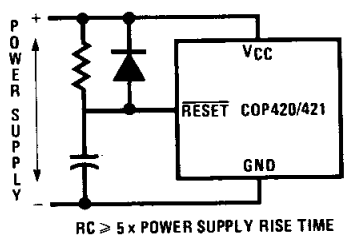


Figure 7. Power-Up Clear Circuit

Oscillator

There are four basic clock oscillator configurations available as shown by figure 8.

a. **Crystal Controlled Oscillator.** CKI and CKO are connected to an external crystal. The instruction cycle time equals the crystal frequency divided by 16 (optional by 8).

b. **External Oscillator.** CKI is an external clock input signal. The external frequency is divided by 16 (optional by 8) to give the instruction cycle time. CKO is now available to be used as the RAM power supply (V_R) or as a general purpose input.

c. **RC Controlled Oscillator.** CKI is configured as a single pin RC controlled Schmitt trigger oscillator. The instruction cycle equals the oscillation frequency divided by 4. CKO is available for non-timing functions.

d. **Externally Synchronized Oscillator.** Intended for use in multi-COP systems, CKO is programmed to function as an input connected to the SK output of another COP420/421 with CKI connected as shown. In this configuration, the SK output connected to CKO must provide a SYNC (instruction cycle) signal to CKO, thereby allowing synchronous data transfer between the COPs using only the SI and SO serial I/O pins in conjunction with the XAS instruction. Note that on power-up SK is automatically enabled as a SYNC output (See Functional Description, Initialization, above).

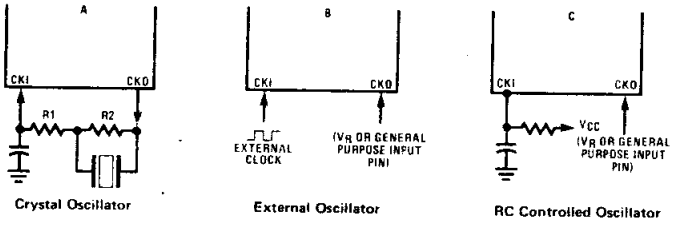
CKO Pin Options

In a crystal controlled oscillator system, CKO is used as an output to the crystal network. As an option CKO can be a SYNC input as described above. As another option CKO can be a general purpose input, read into bit 2 of A (accumulator) upon execution of an INIL instruction. As another option, CKO can be a RAM power supply pin (V_R), allowing its connection to a standby/backup power supply to maintain the integrity of RAM data with minimum power drain when the main supply is inoperative or shut down to conserve power. Using either option is appropriate in applications where the COP420/421 system timing configuration does not require use of the CKO pin.

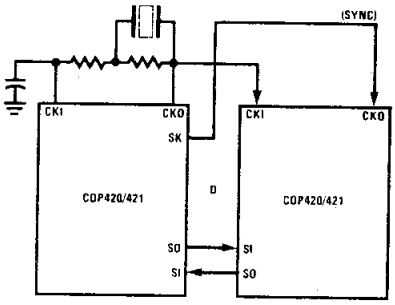
RAM Keep-Alive Option (Not available on COP422)

Selecting CKO as the RAM power supply (V_R) allows the user to shut off the chip power supply (V_{CC}) and maintain data in the RAM. To insure that RAM data integrity is maintained, the following conditions must be met:

1. $\overline{\text{RESET}}$ must go low before V_{CC} goes below spec during power off; V_{CC} must be within spec before $\overline{\text{RESET}}$ goes high on power up.
2. V_R must be within the operating range of the chip, and equal to $V_{CC} \pm 1V$ during normal operation.
3. V_R must be $\geq 3.3V$ with V_{CC} off.



Crystal Oscillator | External Oscillator | RC Controlled Oscillator



Externally Synchronized Oscillator

Crystal Oscillator

Crystal Value	Component Values		
	R1 (Ω)	R2 (Ω)	C (pF)
4 MHz	1k	1M	27
3.58 MHz	1k	1M	27
2.09 MHz	1k	1M	56

RC Controlled Oscillator

R (kΩ)	C (pF)	Instruction Cycle Time (μs)
12	100	5 ± 20%
6.8	220	5.3 ± 23%
8.2	300	8 ± 29%
22	100	8.6 ± 16%

Note: $50k\Omega \geq R \geq 5k\Omega$
$360pF \geq C \geq 50pF$

Figure 8. COP420/421/COP320/321 Oscillator

I/O Options

COP420/421 outputs have the following optional configurations, illustrated in Figure 9a:

a. Standard — an enhancement mode device to ground in conjunction with a depletion-mode device to V_{CC}, compatible with TTL and CMOS input requirements. Available on SO, SK, and all D and G outputs.

b. Open-Drain — an enhancement-mode device to ground only, allowing external pull-up as required by the user's application. Available on SO, SK, and all D and G outputs.

c. Push-Pull — An enhancement-mode device to ground in conjunction with a depletion-mode device paralleled by an enhancement-mode device to V_{CC}. This configuration has been provided to allow for fast rise and fall times when driving capacitive loads. Available on SO and SK outputs only.

d. Standard L — same as **a.**, but may be disabled. Available on L outputs only.

e. Open Drain L — same as **b.**, but may be disabled. Available on L outputs only.

f. LED Direct Drive — an enhancement-mode device to ground and to V_{CC}, meeting the typical current sourcing requirements of the segments of an LED display. The sourcing device is clamped to limit current flow. These devices may be turned off under program control (See Functional Description, EN Register), placing the outputs in a high-impedance state to provide required LED segment blanking for a multiplexed display.

g. TRI-STATE® Push-Pull — an enhancement-mode device to ground and V_{CC}. These outputs are TRI-STATE outputs, allowing for connection of these outputs to a data bus shared by other bus drivers.

COP420/COP421 inputs have the following optional configurations:

h. An on-chip depletion load device to V_{CC}.

i. A Hi-Z input which must be driven to a "1" or "0" by external components.

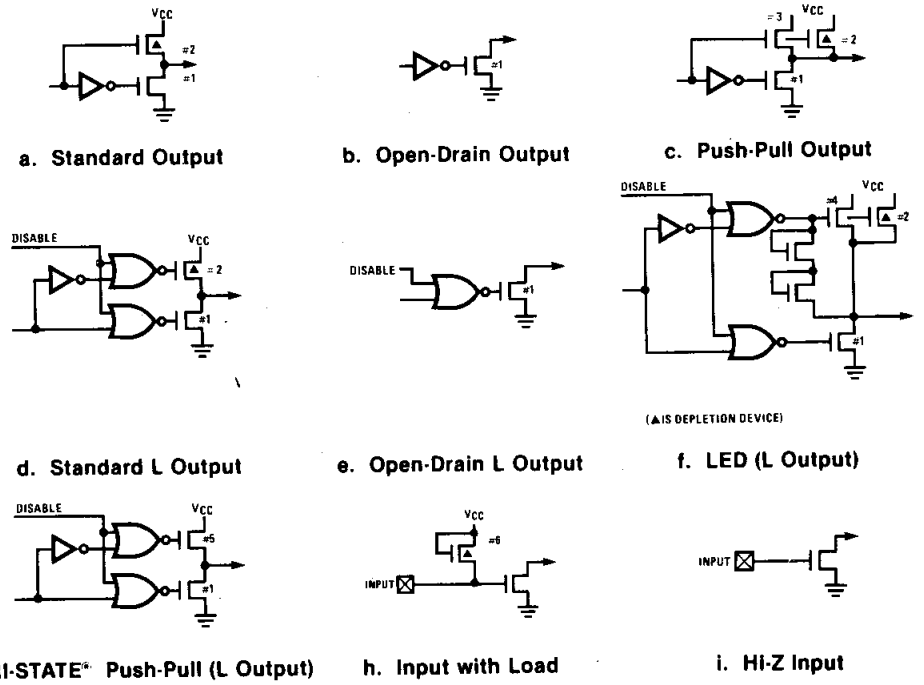
The above input and output configurations share common enhancement-mode and depletion-mode devices. Specifically, all configurations use one or more of six devices (numbered 1-6, respectively). Minimum and maximum current (I_{OUT} and V_{OUT}) curves are given in Figure 9b for each of these devices to allow the designer to effectively use these I/O configurations in designing a COP420/421 system.

The SO, SK outputs can be configured as shown in **a.**, **b.**, or **c.** The D and G outputs can be configured as shown in **a.** or **b.** Note that when inputting data to the G ports, the G outputs should be set to "1." The L outputs can be configured as in **d.**, **e.**, **f.** or **g.**

An important point to remember if using configuration **d.** or **f.** with the L drivers is that even when the L drivers are disabled, the depletion load device will source a small amount of current (see Figure 9b, device 2); however, when the L lines are used as inputs, the disabled depletion device can *not* be relied on to source sufficient current to pull an input to logic "1".

COP421

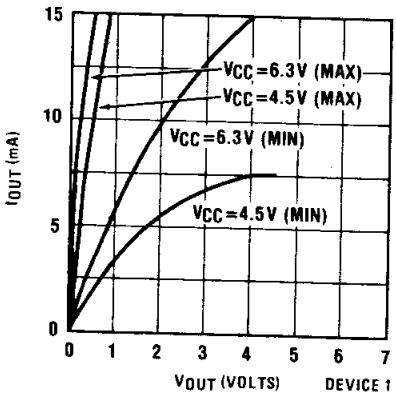
If the COP420 is bonded as a 24-pin device, it becomes the COP421, illustrated in Figure 2, COP420/421 Connection Diagrams. Note that the COP421 does not contain the four general purpose IN inputs (IN_3-IN_0). Use of this option precludes, of course, use of the IN options, interrupt feature, and the MICROBUS™ option which uses IN_1-IN_3. All other options are available for the COP421.



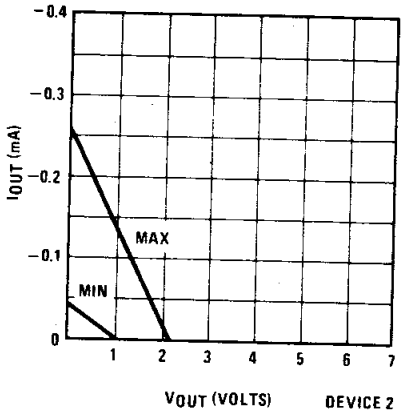
a. Standard Output
b. Open-Drain Output
c. Push-Pull Output
d. Standard L Output
e. Open-Drain L Output
f. LED (L Output)
g. TRI-STATE® Push-Pull (L Output)
h. Input with Load
i. Hi-Z Input

Figure 9a. Input/Output Configurations

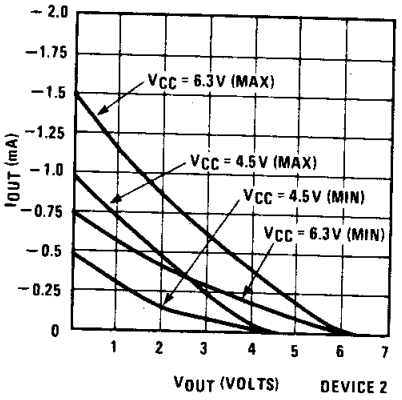
Output Sink Current



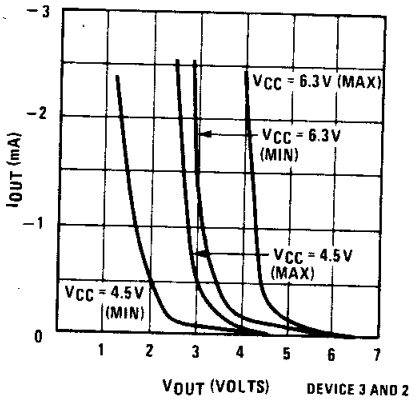
L Output Depletion Load OFF Source Current



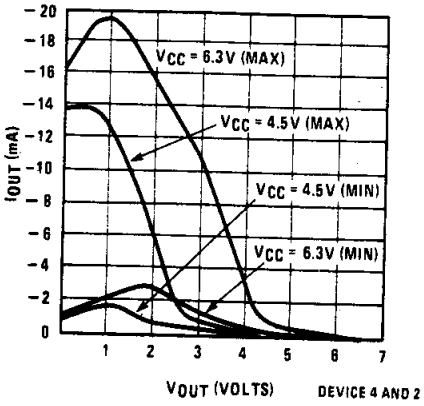
Standard Output Source Current



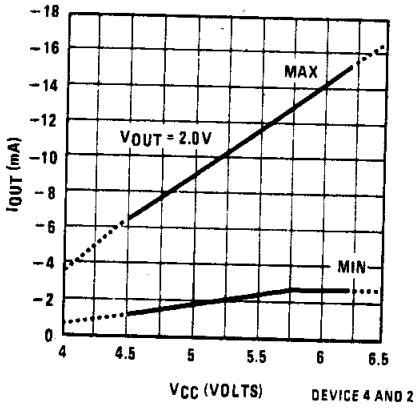
Push-Pull Source Current



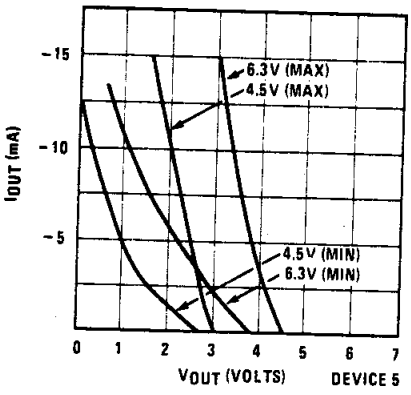
LED Output Source Current



LED Output Direct LED Drive



TRI-STATE® Output Source Current



Input Load Source Current

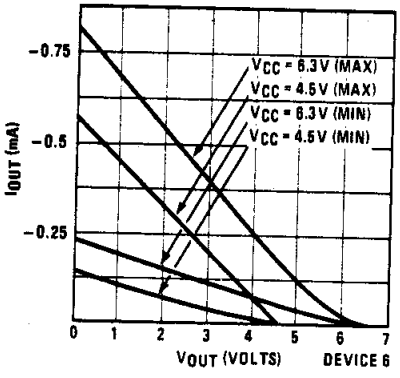
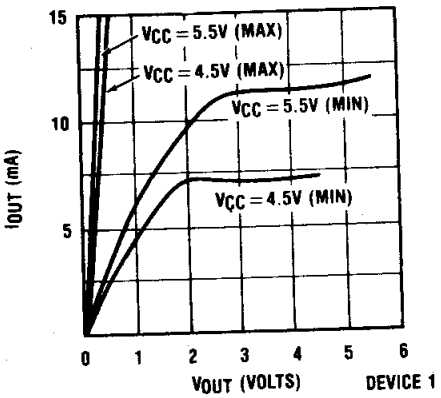
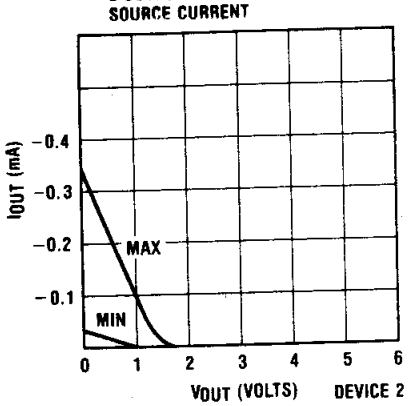


Figure 9b. COP420/COP421 Input/Output Characteristics

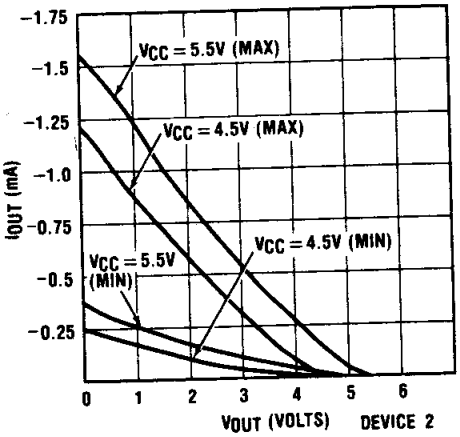
OUTPUT SINK CURRENT



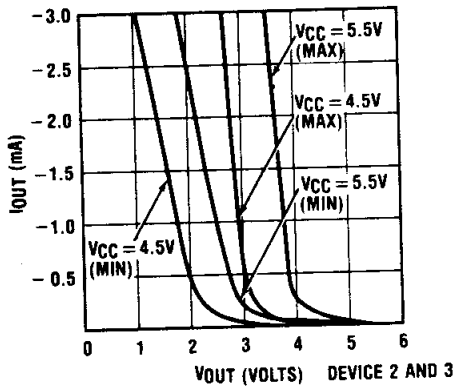
L OUTPUT DEPLETION LOAD OFF SOURCE CURRENT



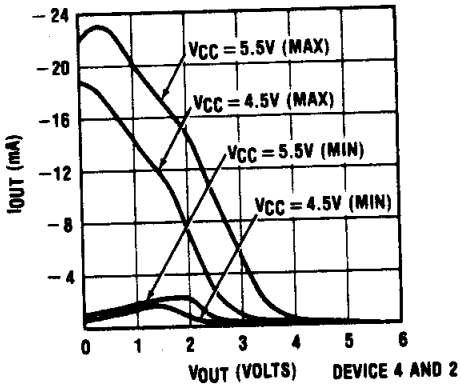
STANDARD OUTPUT SOURCE CURRENT



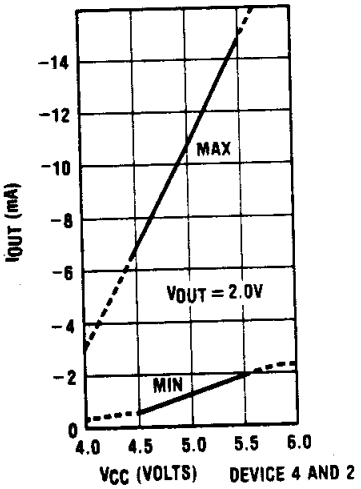
PUSH PULL SOURCE CURRENT



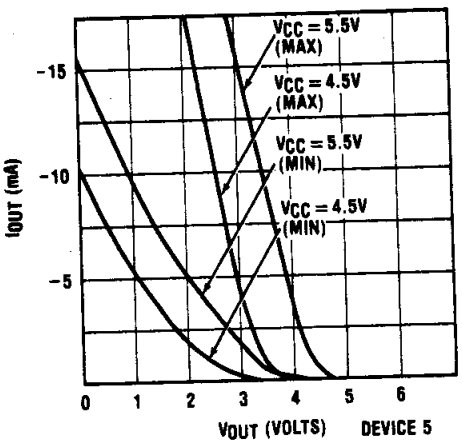
LED OUTPUT SOURCE CURRENT



LED OUTPUT DEVICE LED DRIVE



TRISTATE OUTPUT SOURCE CURRENT



INPUT LOAD SOURCE CURRENT

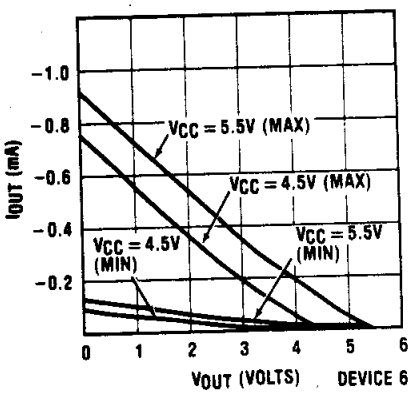


Figure 9c. COP320/COP321 Input/Output Characteristics

COP420/COP421/COP422/COP320/COP321/COP322 Instruction Set

Table 1 is a symbol table providing internal architecture, instruction operand and operational symbols used in the instruction set table.

Table 2 provides the mnemonic, operand, machine code, data flow, skip conditions, and description associated with each instruction in the COP420/COP421/COP422 instruction set.

Table 2. COP420/421/422/320/321/322 Instruction Set Table Symbols

Symbol	Definition
INTERNAL ARCHITECTURE SYMBOLS	
A	4-bit Accumulator
B	6-bit RAM Address Register
Br	Upper 2 bits of B (register address)
Bd	Lower 4 bits of B (digit address)
C	1-bit Carry Register
D	4-bit Data Output Port
EN	4-bit Enable Register
G	4-bit Register to latch data for G I/O Port
IL	Two 1-bit latches associated with the IN_3 or IN_0 inputs
IN	4-bit Input Port
L	8-bit TRI-STATE® I/O Port
M	4-bit contents of RAM Memory pointed to by B Register
PC	10-bit ROM Address Register (program counter)
Q	8-bit Register to latch data for L I/O Port
SA	10-bit Subroutine Save Register A
SB	10-bit Subroutine Save Register B
SC	10 Subroutine Save Register A
SIO	4-bit Shift Register and Counter
SK	Logic-Controlled Clock Output

Symbol	Definition
INSTRUCTION OPERAND SYMBOLS	
d	4-bit Operand Field, 0-15 binary (RAM Digit Select)
r	2-bit Operand Field, 0-3 binary (RAM Register Select)
a	10-bit Operand Field, 0-1023 binary (ROM Address)
y	4-bit Operand Field, 0-15 binary (Immediate Data)
RAM(s)	Contents of RAM location addressed by s
ROM(t)	Contents of ROM location addressed by t
OPERATIONAL SYMBOLS	
+	Plus
−	Minus
→	Replaces
↔	Is exchanged with
=	Is equal to
$\overline{A}$	The one's complement of A
⊕	Exclusive-OR
:	Range of values

Table 2. COP420/421/422/320/321/322 Instruction Set

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description
ARITHMETIC INSTRUCTIONS						
ASC		30	0011|0000	A + C + RAM(B) → A Carry → C	Carry	Add with Carry, Skip on Carry
ADD		31	0011|0001	A + RAM(B) → A	None	Add RAM to A
ADT		4A	0100|1010	A + 10_{10} → A	None	Add Ten to A
AISC	y	5-	0101| y	A + y → A	Carry	Add Immediate, Skip on Carry (y ≠ 0).
CASC		10	0001|0000	$\overline{A}$ + RAM(B) + C → A Carry → C	Carry	Complement and Add with Carry, Skip on Carry
CLRA		00	0000|0000	0 → A	None	Clear A
COMP		40	0100|0000	$\overline{A}$ → A	None	One's complement of A to A
NOP		44	0100|0100	None	None	No Operation
RC		32	0011|0010	"0" → C	None	Reset C
SC		22	0010|0010	"1" → C	None	Set C
XOR		02	0000|0010	A ⊕ RAM(B) → A	None	Exclusive-OR RAM with A

Table 2. COP420/421/422/320/321/322 Instruction Set (continued)

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description
TRANSFER OF CONTROL INSTRUCTIONS						
JID		FF	1111 1111	ROM ($PC_{9:8}$, A,M) → $PC_{7:0}$	None	Jump Indirect (Note 3)
JMP	a	6-	0110 00 $a_{9:8}$	a → PC	None	Jump
		--	$a_{7:0}$			
JP	a	--	1 $a_{6:0}$ (pages 2,3 only)	a → $PC_{6:0}$	None	Jump within Page (Note 4)
			or			
		--	11 $a_{5:0}$ (all other pages)	a → $PC_{5:0}$		
JSRP	a	--	10 $a_{5:0}$	PC + 1 → SA → SB → SC 0010 → $PC_{9:6}$ a → $PC_{5:0}$	None	Jump to Subroutine Page (Note 5)
JSR	a	6-	0110 10 $a_{9:8}$	PC + 1 → SA → SB → SC a → PC	None	Jump to Subroutine
		--	$a_{7:0}$			
RET		48	0100 1000	SC → SB → SA → PC	None	Return from Subroutine
RETSK		49	0100 1001	SC → SB → SA → PC	Always Skip on Return	Return from Subroutine then Skip
MEMORY REFERENCE INSTRUCTIONS						
CAMQ		33	0011 0011	A → $Q_{7:4}$ RAM(B) → $Q_{3:0}$	None	Copy A, RAM to Q
		3C	0011 1100			
CQMA		33	0011 0011	$Q_{7:4}$ → RAM(B) $Q_{3:0}$ → A	None	Copy Q to RAM, A
		2C	0010 1100			
LD	r	-5	00 r 0101	RAM(B) → A Br ⊕ r → Br	None	Load RAM into A, Exclusive-OR Br with r
LDD	r,d	23	0010 0011	RAM(r,d) → A	None	Load A with RAM pointed to directly by r,d
		--	00 r d			
LQID		BF	1011 1111	ROM($PC_{9:8}$,A,M) → Q SB → SC	None	Load Q Indirect (Note 3)
RMB	0	4C	0100 1100	0 → $RAM(B)_0$	None	Reset RAM Bit
	1	45	0100 0101	0 → $RAM(B)_1$		
	2	42	0100 0010	0 → $RAM(B)_2$		
	3	43	0100 0011	0 → $RAM(B)_3$		
SMB	0	4D	0100 1101	1 → $RAM(B)_0$	None	Set RAM Bit
	1	47	0100 1101	1 → $RAM(B)_1$		
	2	46	0100 0110	1 → $RAM(B)_2$		
	3	4B	0100 1011	1 → $RAM(B)_3$		

Table 2. COP420/421/422/320/321/322 Instruction Set (continued)

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description
MEMORY REFERENCE INSTRUCTIONS (continued)						
STII	y	7–	0111 | y	y → RAM(B) Bd + 1 → Bd	None	Store Memory Immediate and Increment Bd
X	r	–6	00 | r | 0110	RAM(B) ↔ A Br ⊕ r → Br	None	Exchange RAM with A, Exclusive-OR Br with r
XAD	r,d	23	0010 | 0011	RAM(r,d) ↔ A	None	Exchange A with RAM pointed to directly by r,d
		--	10 | r | d			
XDS	r	–7	00 | r | 0111	RAM(B) ↔ A Bd – 1 → Bd Br ⊕ r → Br	Bd decrements past 0	Exchange RAM with A and Decrement Bd, Exclusive-OR Br with r
XIS	r	–4	00 | r | 0100	RAM(B) ↔ A Bd + 1 → Bd Br ⊕ r → Br	Bd increments past 15	Exchange RAM with A and Increment Bd, Exclusive-OR Br with r
REGISTER REFERENCE INSTRUCTIONS						
CAB		50	0101 | 0000	A → Bd	None	Copy A to Bd
CBA		4E	0100 | 1110	Bd → A	None	Copy Bd to A
LBI	r,d	--	00 | r | (d – 1) (d = 0, 9:15)	r,d → B	Skip until not a LBI	Load B Immediate with r,d (Note 6)
			or			
		33	0011 | 0011			
		--	10 | r | d (any d)			
LEI	y	33	0011 | 0011	y → EN	None	Load EN Immediate (Note 7)
		6–	0110 | y			
XABR		12	0001 | 0010	A ↔ Br (0,0 → A_3,A_2)	None	Exchange A with Br
TEST INSTRUCTIONS						
SKC		20	0010 | 0000		C = "1"	Skip if C is True
SKE		21	0010 | 0001		A = RAM(B)	Skip if A Equals RAM
SKGZ		33	0011 | 0011		$G_{3:0} = 0$	Skip if G is Zero (all 4 bits)
		21	0010 | 0001			
SKGBZ		33	0011 | 0011	1st byte		Skip if G Bit is Zero
	0	01	0000 | 0001		$G_0 = 0$	
	1	11	0001 | 0001		$G_1 = 0$	
	2	03	0000 | 0011	2nd byte	$G_2 = 0$	
	3	13	0001 | 0011		$G_3 = 0$	
SKMBZ	0	01	0000 | 0001		$RAM(B)_0 = 0$	Skip if RAM Bit is Zero
	1	11	0001 | 0001		$RAM(B)_1 = 0$	
	2	03	0000 | 0011		$RAM(B)_2 = 0$	
	3	13	0001 | 0011		$RAM(B)_3 = 0$	
SKT		41	0100 | 0001		A time-base counter carry has occurred since last test	Skip on Timer (Note 3)

Table 2. COP420/421/422/320/321/322 Instruction Set (continued)

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description
INPUT/OUTPUT INSTRUCTIONS						
ING		33	0011 0011	G → A	None	Input G Ports to A
		2A	0010 1010			
ININ		33	0011 0011	IN → A	None	Input IN Inputs to A (Note 2)
		28	0010 1000			
INIL		33	0011 0011	IL_3, CKO, "0", IL_0 → A	None	Input IL Latches to A (Note 3)
		29	0010 1001			
INL		33	0011 0011	$L_{7:4}$ → RAM(B)	None	Input L Ports to RAM,A
		2E	0010 1110	$L_{3:0}$ → A		
OBD		33	0011 0011	Bd → D	None	Output Bd to D Outputs
		3E	0011 1110			
OGI	y	33	0011 0011	y → G	None	Output to G Ports Immediate
		5-	0101 y			
OMG		33	0011 0011	RAM(B) → G	None	Output RAM to G Ports
		3A	0011 1010			
XAS		4F	0100 1111	A ↔ SIO, C → SKL	None	Exchange A with SIO (Note 3)

Note 1: All subscripts for alphabetical symbols indicate bit numbers unless explicitly defined (e.g., Br and Bd are explicitly defined). Bits are numbered 0 to N where 0 signifies the least significant bit (low-order, right-most bit). For example, A_3 indicates the most significant (left-most) bit of the 4-bit A register.

Note 2: The ININ instruction is not available on the COP421/COP321 and COP422/COP322 since these devices do not contain the IN inputs.

Note 3: For additional information on the operation of the XAS, JID, LQID, INIL, and SKT instructions, see below.

Note 4: The JP instruction allows a jump, while in subroutine pages 2 or 3, to any ROM location within the two-page boundary of pages 2 or 3. The JP instruction, otherwise, permits a jump to a ROM location within the current 64-word page. JP may not jump to the last word of a page.

Note 5: A JSRP transfers program control to subroutine page 2 (0010 is loaded into the upper 4 bits of P). A JSRP may not be used when in pages 2 or 3. JSRP may not jump to the last word in page 2.

Note 6: LBI is a single-byte instruction if d = 0, 9, 10, 11, 12, 13, 14, or 15. The machine code for the lower 4 bits equals the binary value of the "d" data *minus 1*, e.g., to load the lower four bits of B (Bd) with the value 9 (1001_2), the lower 4 bits of the LBI instruction equal 8 (1000_2). To load 0, the lower 4 bits of the LBI instruction should equal 15 (1111_2).

Note 7: Machihe code for operand field y for LEI instruction should equal the binary value to be latched into EN, where a "1" or "0" in each bit of EN corresponds with the selection or deselection of a particular function associated with each bit. (See Functional Description, EN Register.)

The following information is provided to assist the user in understanding the operation of several unique instructions and to provide notes useful to programmers in writing COP420/421 programs.

XAS Instruction

XAS (Exchange A with SIO) exchanges the 4-bit contents of the accumulator with the 4-bit contents of the SIO register. The contents of SIO will contain serial-in/serial-out shift register or binary counter data, depending on the value of the EN register. An XAS instruction will also affect the SK output. (See Functional Description, EN Register, above.) If SIO is selected as a shift register, an XAS instruction must be performed once every 4 instruction cycles to effect a continuous data stream.

JID Instruction

JID (Jump Indirect) is an indirect addressing instruction, transferring program control to a new ROM location pointed to indirectly by A and M. It loads the lower 8 bits of the ROM address register PC with the *contents* of ROM addressed by the 10-bit word, $PC_{9:8}$, A, M. PC_9 and PC_8 are not affected by this instruction.

Note that JID requires 2 instruction cycles to execute.

INIL Instruction

INIL (Input IL Latches to A) inputs 2 latches, IL_3 and IL_0 (see figure 10) and CKO into A. The IL_3 and IL_0 latches are set if a low-going pulse ("1" to "0") has occurred on the IN_3 and IN_0 inputs since the last INIL instruction, provided the input pulse stays low for at least two instruction times. Execution of an INIL inputs IL_3 and IL_0 into A3 and A0 respectively, and resets these latches to allow them to respond to subsequent low-going pulses on the IN_3 and IN_0 lines. If CKO is mask programmed as a general purpose input, an INIL will input the state of CKO into A2. If CKO has not been so programmed, a "1" will be placed in A2. A "0" is always placed in A1 upon the execution of an INIL. The general purpose inputs IN_3-IN_0 are input to A upon execution of an ININ instruction. (See table 2, ININ instruction.) INIL is useful in recognizing pulses of short duration or pulses which occur too often to be read conveniently by an ININ instruction.

Note: IL latches are not cleared on reset.

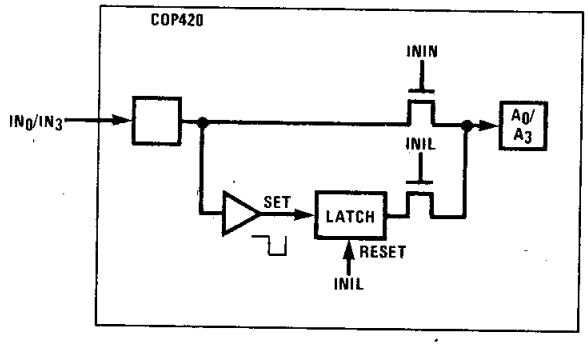


Figure 10.

LQID Instruction

LQID (Load Q Indirect) loads the 8-bit Q register with the contents of ROM pointed to by the 10-bit word PC_9, PC_8, A, M. LQID can be used for table lookup or code conversion such as BCD to seven-segment. The LQID instruction "pushes" the stack (PC + 1 → SA → SB →SC) and replaces the least significant 8 bits of PC as follows: A → $PC_{7:4}$, RAM(B) → $PC_{3:0}$, leaving PC_9 and PC_8 unchanged. The ROM data pointed to by the new address is fetched and loaded into the Q latches. Next, the stack is "popped" (SC → SB → SA →PC), restoring the saved value of PC to continue sequential program execution. Since LQID pushes SB → SC, the previous contents of SC are lost. Also, when LQID pops the stack, the previously pushed contents of SB are left in SC. The net result is that the contents of SB are placed in SC (SB → SC). Note that LQID takes two instruction cycle times to execute.

SKT Instruction

The SKT (Skip On Timer) instruction tests the state of an internal 10-bit time-base counter. This counter divides the instruction cycle clock frequency by 1024 and provides a latched indication of counter overflow. The SKT instruction tests this latch, executing the next program instruction if the latch is not set. If the latch has been set since the previous test, the next program instruction is skipped and the latch is reset. The features associated with this instruction, therefore, allow the COP420/421 to generate its own time-base for real-time processing rather than relying on an external input signal.

For example, using a 2.097 MHz crystal as the time-base to the clock generator, the instruction cycle clock frequency will be 131 kHz (crystal frequency ÷ 16) and the binary counter output pulse frequency will be 128 Hz. For time-of-day or similar real-time processing, the SKT instruction can call a routine which increments a "seconds" counter every 128 ticks.

Instruction Set Notes

a. The first word of a COP420/421 program (ROM address 0) must be a CLRA (Clear A) instruction.

b. Although skipped instructions are not executed, one instruction cycle time is devoted to skipping each byte of the skipped instruction. Thus all program paths take the same number of cycle times whether instructions are skipped or executed except JID and LQID. LQID and JID take two cycle times if executed and one if skipped.

c. The ROM is organized into 16 pages of 64 words each. The Program Counter is an 10-bit binary counter, and will count through page boundaries. If a JP, JSRP, JID or LQID instruction is located in the last word of a page, the instruction operates as if it were in the next page. For example: a JP located in the last word of a page will jump to a location in the next page. Also, a LQID or JID located in the last word of page 3, 7, 11 or 15 will access data in the next group of four pages.

Option List

The COP420/421/422 mask-programmable options are assigned numbers which correspond with the COP*420* pins.

The following is a list of COP420 options. When specifying a COP421 or COP422 chip, Options 9, 10, 19, 20 and 29 must all be set to zero. When specifying a COP422 chip, Options 21, 22, 27 and 28 must also be zero, and Option 2 must not be a 1. The options are programmed at the same time as the ROM pattern to provide the user with the hardware flexibility to interface to various I/O components using little or no external circuitry.

Option 1 = 0: Ground Pin — no options available

Option 2: CKO Pin
- = 0: clock generator output to crystal (0 not available if option 3 = 4 or 5)
- = 1: pin is RAM power supply (V_R) input (Not available on COP422/COP322)
- = 2: general purpose input with load device
- = 3: multi-COP SYNC input
- = 4: general purpose Hi Z input

Option 3: CKI Input
- = 0: crystal input divided by 16
- = 1: crystal input divided by 8
- = 2: TTL external clock input divided by 16
- = 3: TTL external clock input divided by 8
- = 4: single-pin RC controlled oscillator (÷4)
- = 5: Schmitt trigger clock input (÷4)

Option 4: RESET Pin
- = 0: Load devices to V_{CC}
- = 1: Hi-Z input

Option 5: L_7 Driver
- = 0: Standard output (figure 9D)
- = 1: Open-Drain output (E)
- = 2: LED direct drive output (F)
- = 3: TRI-STATE® push-pull output (G)

Option 6: L_6 Driver
same as Option 5

Option 7: L_5 Driver
same as Option 5

Option 8: L_4 Driver
same as Option 5

Option 9: IN_1 Input
- = 0: load device to V_{CC} (H)
- = 1: Hi-Z input (I)

Option 10: IN_2 Input
same as Option 9

Option 11 = 0: V_{CC} Pin — no options available

Option 12: L_3 Driver
same as Option 5

Option 13: L_2 Driver
same as Option 5

Option 14: L_1 Driver
same as Option 5

Option 15: L_0 Driver
same as Option 5

Option 16: SI Input
same as Option 9

Option 17: SO Driver
- = 0: standard output (A)
- = 1: open-drain output (B)
- = 2: push-pull output (C)

Option 18: SK Driver
same as Option 17

Option 19: IN_0 Input
same as Option 9

Option 20: IN_3 Input
same as Option 9

Option 21: G_0 I/O Port
- = 0: Standard output (A)
- = 1: Open-Drain output (B)

Option 22: G_1 I/O Port
same as Option 21

Option 23: G_2 I/O Port
same as Option 21

Option 24: G_3 I/O Port
same as Option 21

Option 25: D_3 Output
- = 0: Standard output (A)
- = 1: Open-Drain output (B)

Option 26: D_2 Output
same as Option 25

Option 27: D_1 Output
same as Option 25

Option 28: D_0 Output
same as Option 25

Option 29: COP Function
- = 0: normal operation
- = 1: MICROBUS™ option

Option 30: COP Bonding
- = 0: COP420 (28-pin device)
- = 1: COP421 (24-pin device)
- = 2: 28- and 24-pin device
- = 3: COP422 (20-pin device)
- = 4: 28- and 20-pin device
- = 5: 24- and 20-pin device
- = 6: 28-, 24- and 20-pin device

Option 31: IN Input Levels
- = 0: normal input levels
- = 1: Higher voltage input levels ("0" = 1.2V, "1" = 3.6V)

Option 32: G Input Levels
same as Option 31

Option 33: L Input Levels
same as Option 31

Option 34: CKO Input Levels
same as Option 31

Option 35: SI Input Levels
same as Option 31

TEST MODE (Non-Standard Operation)

The SO output has been configured to provide for standard test procedures for the custom-programmed COP420. With SO forced to logic "1," two test modes are provided, depending upon the value of SI:

a. RAM and Internal Logic Test Mode (SI = 1)

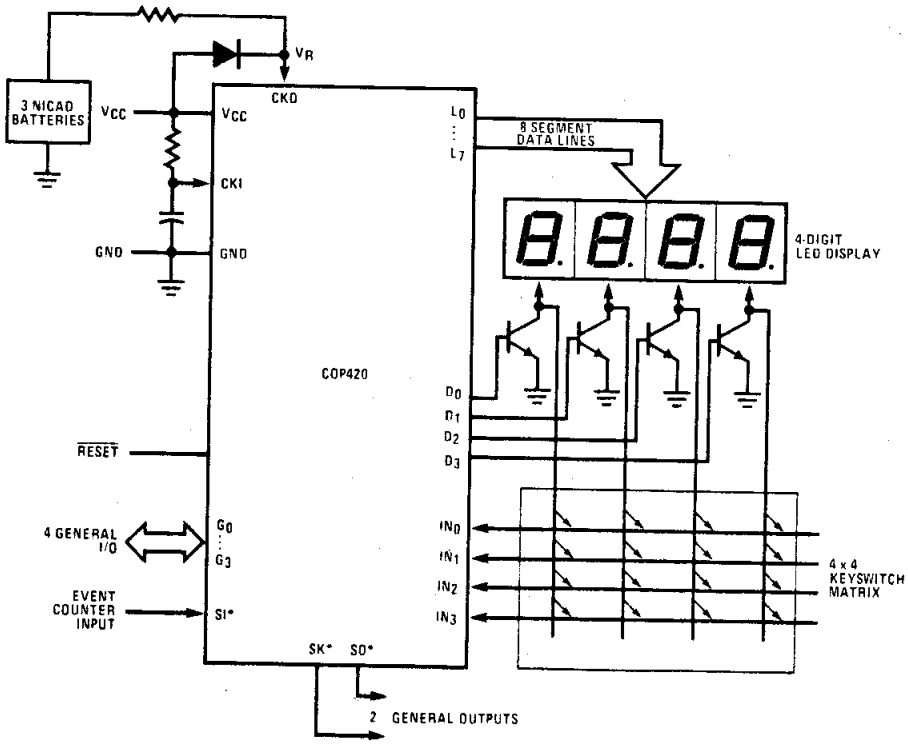
b. ROM Test Mode (SI = 0)

These special test modes should not be employed by the user; they are intended for manufacturing test only.

APPLICATION #1: COP420 General Controller

Figure 8 shows an interconnect diagram for a COP420 used as a general controller. Operation of the system is as follows:

1. The L_7-L_0 outputs are configured as LED Direct Drive outputs, allowing direct connection to the segments of the display.
2. The D_3-D_0 outputs drive the digits of the multiplexed display directly and scan the columns of the 4 × 4 keyboard matrix.
3. The IN_3-IN_0 inputs are used to input the 4 rows of the keyboard matrix. Reading the IN lines in conjunction with the current value of the D outputs allows detection, debouncing, and decoding of any one of the 16 keyswitches.
4. CKI is configured as a single-pin oscillator input allowing system timing to be controlled by a single-pin RC network. CKO is therefore available for use as a V_R RAM power supply pin. RAM data integrity is thereby assured when the main power supply is shut down (see RAM Keep-Alive Option description).
5. SI is selected as the input to a binary counter input. With SIO used as a binary counter, SO and SK can be used as general purpose outputs.
6. The 4 bidirectional G I/O ports (G_3-G_0) are available for use as required by the user's application.



*SI, SO and SK may also be used for serial I/O

Figure 11. COP420 Keyboard/Display Interface