TMS7000 Family Data Manual

8-bit Microcomputer Family

TEXAS INSTRUMENTS

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes in the devices or the device specifications identified in this publication without notice. TI advises its customers to obtain the latest version of device specifications to verify, before placing orders, that the information being relied upon by the customer is current.

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

Copyright © 1986, Texas Instruments Incorporated

Contents

Illustrations

Tables

Preface

This book replaces the following manuals:

- TMS7000 Family Data Manual, SPND001A
- TMS7000 Assembly Language Programmer's Guide, SPNU002B
- TMS7000 Software Development System Installation Guide, MPB52
- TMS7000 IBM CrossWare Support Reference Guide, MPB10
- TMS700 VAX/VMS CrossWare Support Reference Guide, MPB53

The following table lists related publications.

TMS7000 DATA SHEETS AND DATA MANUALS	LITERATURE NUMBER
TMS7002/7042 Data Sheet	SPNS007
TMS7742 Data Sheet	SPNS008
TMS70C42/TMS70C02 Data Sheet	SPNS009
TMS7000 USER'S GUIDES	**LITERATURE NUMBER**
8051-TMS7041 System Conversion User's Guide	SPNU003
TMS7500/TMS75C00 Data Encryption Device User's Guide	SPNU004
Link Editor User's Guide	SPDU037C
TMS7000 EVM User's Guide	
TMS7000 FAMILY DEVELOPMENT SYSTEM SUPPORT	**LITERATURE NUMBER**
XDS/7042 User's Guide	SPDU047
XDS/22 with the TMS7042 Emulator Pocket Reference	SPDF010
TMS7000 FAMILY APPLICATION NOTES	**LITERATURE NUMBER**
TMS7000 Bus Activity Tables	SPNA002
TMS7000 Keyboard Interface	SPNA003

1. Introduction

The TMS7000 is a family[1] of 8-bit single-chip microcomputers. These microcomputers incorporate a CPU, memory (ROM, RAM, EPROM), bit I/O, serial communication port, timers, interrupts, and external bus interface logic, all on a single chip. The products are available in varying complexity of functions, process technology, performance, and packaging to meet end equipment cost goals and application requirements.

Typical applications of TMS7000 family devices include:

AUTOMOTIVE	TELECOM
• Instrumentation • Audio entertainment control • Cruise control • Anti-skid braking system • Climate control • Engine control • Trip computer	• Feature phones • Autodialers • Answering machines • Modem control • Digital switches • Digital subsets
COMPUTER	**INDUSTRIAL**
• Printers and plotters • Disk controllers • Tape drive control • Keyboards • Touch screen and mouse	• Motor control • Stepper motors • Metering and measurement • Robotics
CONSUMER	**BUSINESS**
• Home security • Cable TV systems • Appliance control	• Cash registers • Automatic bank tellers • Barcode readers

---

1 The terms *TMS7000* and *TMS7000 family* refer to all TMS7000 devices: TMS7000, TMS7020, TMS7040, TMS7002, TMS7042, TMS70C00, TMS70C20, TMS70C40, TMS70C02, TMS70C42, TMS7742, and all future members, unless otherwise stated.

1.1 How to Use this Manual

This manual is divided into four major parts:

- Hardware (Sections 2–4)
- Software (Sections 5–8)
- Development Support (Sections 9–11)
- Customer Information (Section 12)

The sections and their contents are summarized below.

Section 1 - Introduction

- Introduces the TMS7000 family devices.
- Describes the different manual sections and their contents.

Section 2 - TMS7000 Family Devices

- Details each TMS7000 family category and their key features.
- Summarizes the categories and compares their features.
- Provides key features, pinouts, and pin descriptions for each category of devices.

Section 3 - TMS7000 Family Architecture

- Discusses operation of the microcomputers' hardware features:
  - Registers
  - I/O
  - Memory and memory modes
  - Clock options
  - CMOS low-power modes
  - Interrupts
  - Timer/event counters
  - Serial port (TMS70x2 and TMS70Cx2 devices only)

Section 4 - Electrical Specifications

Discusses for all device groups:

- Absolute maximum ratings
- Recommended operating characteristics
- Recommended crystal/clockin operating characteristics
- Memory interface timing
- Read and write cycle timing
- Ceramic resonator circuit application (where applicable)
- Serial port timing (where applicable)

Section 5 - TMS7000 Assembler

- Discusses basic assembler information, including:
  - Source statement format (placement of various fields in code)
  - Constants, symbols, terms, and expressions

- Discusses the various assembler directives, grouped in the following categories:
  - Directives that affect the location counter
  - Directives that affect assembler output
  - Directives that initialize constants
  - Directives for linking programs
  - Miscellaneous directives
- Assembler Output
  - Explains source listing format and resulting object code.
  - Presents normal completion and abnormal completion error messages.
  - Shows a sample cross reference listing.
  - Discusses object code and the various fields in object code format, and changing object code.

Section 6 - Assembly Language Instruction Set

- Provides general instruction set information, such as symbol definitions.
- Defines eight addressing modes used by the instructions.
- Summarizes the instruction set in table form.
- Presents the TMS7000 assembly language instruction set in alphabetical order.

Section 7 - Linking Program Modules

- Discusses relocation capability, absolute and relocatable code.
- Discusses the Link Editor and includes a sample link control file.
- Reviews directives needed for linking programs.

Section 8 - Macro Language

- Defines the TMS7000 Macro Assembler.
- Tells how to define macros and use macro libraries.
- Shows how strings, constants, and operators are used in macros.
- Discusses variables, parameters, substitution, and keywords.
- Presents the macro definition verbs.
- Provides macro examples.

Section 9 - Design Aids

Includes several examples to help you use the TMS7000 family devices:

- Interfacing the TMS7000 to peripheral and memory devices such as extra EPROM and RAM
- Programming the TMS7742
- Serial communication using the UART (serial port)
- Instruction set application notes
- Sample routines

Section 10 - Development Support

Discusses several products manufactured by Texas Instruments that enhance TMS7000 family design development, including:

- XDS (Extended Development Support) Emulator
- EVM (evaluation module)
- Prototyping devices

Section 11 - Independent Support

Discusses several products manufactured by Texas Instruments that enhance TMS7000 family design development, including assemblers, text editors, simulators, EEROM, and EPROM support.

Section 12 - Customer Information

- Discusses quality and reliability.
- Discusses prototype manufacture and production flow, including device prefix designators - TMS, TMP, TMX, and SE.
- Illustrates mechanical package information for all TMS7000 family members
- Provides ordering information for the TMS7000 microcomputers and the Texas Instruments development support products.

Appendix A - TMS7000 Bus Activity Tables

Appendix B - TMS7500/TMS75C00 Data Encryption Device

Appendix C - TMS70x1 Devices

Appendix D - Character Sets

Appendix E - Hexadecimal Instruction Table/Opcode Map

Appendix F - Instruction Opcode Set

Appendix G - CrossWare Installation

Appendix H - Glossary

Index

2. TMS7000 Family Devices

This section discusses the features of the TMS7000 family[2] of microcomputers. All family members are software compatible, allowing easy migration within the TMS7000 family by maintaining a software base, development tools, and design expertise.

The TMS7000 family devices are divided into several categories:

- **TMS70x0 devices** include the TMS7000, TMS7020, and TMS7040
- **TMS70x2 devices** include the TMS7002 and TMS7042
- **TMS70Cx0 devices** include the TMS70C00, TMS70C20, and TMS70C40
- **TMS70Cx2 devices** include the TMS70C02 and TMS70C42
- **Prototyping devices** include the TMS7742 (EPROM), the TMS77C82 (see note below) the SE70P162, SE70CP160, and SE70CP162 (piggybacks)

This section begins with a summary and comparison of the TMS7000 family devices, and then provides key features, pinouts, and pin descriptions for the individual categories.

Section **Page**

Note:

Information regarding the TMS77C82 is classified as Advance Information, which means that it is information on a new product in the sampling or preproduction phase of development. Characteristic data and other specifications are subject to change without notice.

---

2 Throughout this manual, the term *TMS7000* or *TMS7000 family* refers to all members of the group.

2.1 Summary and Device Comparison

The TMS7000 family **NMOS** devices can be summarized as follows:

- The **TMS7000** is the basic 8-bit, single-chip microcomputer, containing a CPU, a timer, flexible I/O, and 128 bytes of on-chip RAM, but no on-chip ROM.
- The **TMS7020** and **TMS7040** have the same basic features as the TMS7000, with the addition of 2K and 4K bytes of on-chip ROM, respectively.
- The **TMS7002** (ROMless) and **TMS7042** (4K bytes on-chip ROM) have the same features as the TMS70x0 devices with the addition of a serial port (UART), a 13-bit timer (Timer 2), a 10-bit timer (Timer 3), and 256 bytes of on-chip RAM.
- NMOS prototyping devices include the **TMS7742** and the **SE70P162**. The TMS7742 is an EPROM version of the TMS7042 and contains 4K bytes of on-chip EPROM. The SE70P162 piggyback device is based on the TMS70x2 architecture and acts like a ROM-coded TMS70x2 device.

Table 2-1. TMS7000 NMOS Family Feature Summary

	TMS7040 TMS7020 TMS7000			TMS7042 TMS7002		TMS7742
Maximum oscillator frequency	5 MHz			8 MHz		5MHz
Voltage	5 V ± 10%			5 V ± 10%		5 V ± 10%
Operating temperature	0°C to 70°C			0°C to 70°C		0°C to 70°C
On-chip ROM (Kbytes)	4	2	0	4	0	4 (EPROM)
Internal RAM (bytes)	128			256		256
Interrupt levels:						
External	2			2		2
Total	4			6		6
Timers/event counters:						
13-bit	1			2		2
10-bit	–			1		1
I/O lines: Bidirectional	16			22		22
Input only	8			2		2
Output only	8			8		8
Additional features	–			Serial Port		Serial Port
Development support:						
Prototyping:						
EPROM	TMS7742			TMS7742		–
Piggyback	SE70P162			SE70P162		SE70P162
XDS	Yes			Yes		Yes
EVM	Yes			Yes		Yes

The TMS7000 family **CMOS** devices can be summarized as follows:

- The CMOS **TMS70Cx0** devices have the same features as the TMS70x0 devices, adding low power requirements to the list of features.
- The CMOS **TMS70Cx2** devices contain the same features as the TMS70x2 devices with the addition of programmable-sense interrupts and two 21-bit timers.
- Prototyping devices include the **SE70CP160** and **SE70CP162** (piggyback) devices, which are based on the TMS70Cxx architecture and act like ROM-coded TMS70xx or TMS70Cxx devices.

Table 2-2. TMS7000 CMOS Family Feature Summary

	TMS70C40A TMS70C20A TMS70C00A	**TMS70C42 TMS70C02**	**TMS77C82†**
Max osc freq at 5V ± 10 %	5 MHz	6 MHz	7.5 MHz
Voltage	5 V ± 10%	2.5 to 6 V	2.5 to 6 V
Operating temperature Industrial Commercial	 -40°C to 85°C 0°C to 70°C	 -40°C to 85°C 0°C to 70°C	 -40°C to 85°C 0°C to 70°C
On-chip ROM (Kbytes)	4 | 2 | 0	4 | 0	8 (EPROM)
Internal RAM (bytes)	128	256	256
Interrupt levels: External Total	 2 4	 2 6	 2 6
Timers/event counters: 21-bit 13-bit 10-bit	 – 1 –	 2 – 1	 2 – 1
I/O lines: Bidirectional Input only Output only	16 8 8	24 – 8	24 – 8
Additional features	–	Serial Port	Serial Port
Development support: Prototyping: EPROM Piggyback XDS EVM	 – SE70CP160A Yes Yes	 TMS77C82† SE70CP162 Yes Yes	 – SE70CP162 Yes Yes

† Advance information

2.2 TMS70x0 and TMS70Cx0 Devices

2.2.1 TMS70x0 (NMOS) Key Features

	TMS7040/20/00			TMS7042/02		TMS7742
Maximum oscillator frequency	5 MHz			8 MHz		5MHz
On-chip ROM (Kbytes)	4	2	0	4	0	4 (EPROM)
Internal RAM (bytes)	128			256		256
Interrupt levels:						
External	2			2		2
Total	4			6		6
Timers/event counters:						
13-bit	1			2		2
10-bit	–			1		1
I/O lines: Bidirectional	16			22		22
Input only	8			2		2
Output only	8			8		8
Additional features	–			Serial Port		Serial Port
Development support:						
Prototyping:						
EPROM	TMS7742			TMS7742		–
Piggyback	SE70P162			SE70P162		SE70P162
XDS	Yes			Yes		Yes
EVM	Yes			Yes		Yes

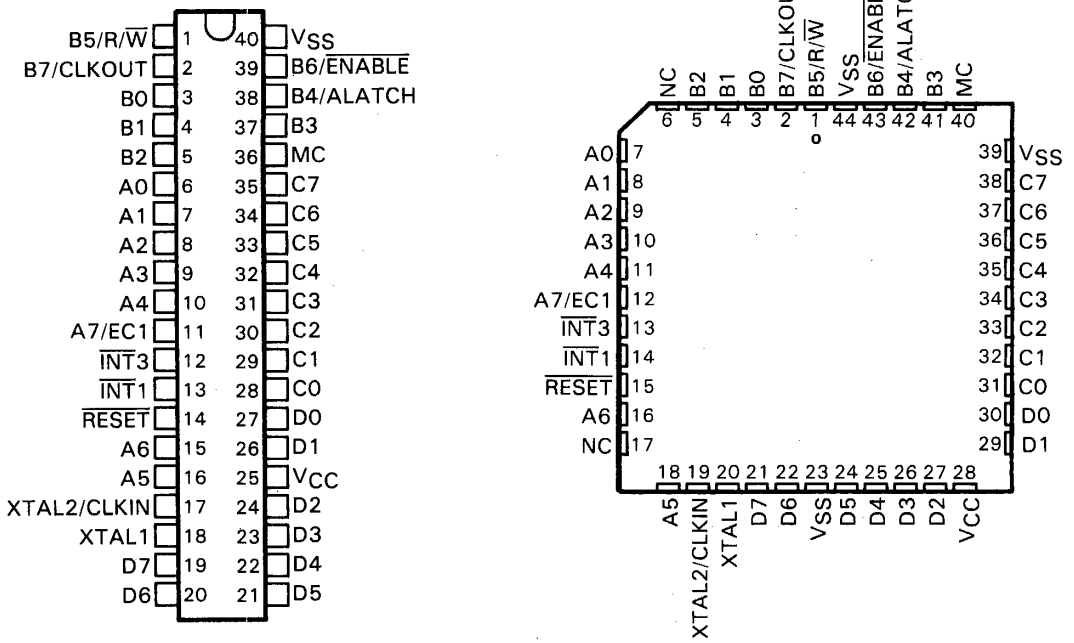
- Register-to-register architecture
- Memory-mapped ports for easy addressing
- Eight addressing formats, including:
  - Register-to-register arithmetic
  - Indirect addressing
  - Indexed and indirect branches and calls
- Single-instruction binary-coded decimal (BCD) add and subtract
- Two external maskable interrupts
- Flexible interrupt handling:
  - Priority servicing of simultaneous interrupts
  - Software calls through interrupt vectors
  - Precise timing of interrupts with the capture latch
  - Software monitoring of interrupt status
- Supports all TMS7000 family expansion modes
- N-channel silicon-gate MOS technology
- 40-pin, 600 mil, dual-inline package

2.2.2 TMS70Cx0 (CMOS) Key Features

	TMS70C40A/ C20A/C00A	TMS70C42/C02	TMS77C82†
Max osc freq at 5 V ± 10 %	5 MHz	6 MHz	7.5 MHz
On-chip ROM (Kbytes)	4 | 2 | 0	4 | 0	8 (EPROM)
Internal RAM (bytes)	128	256	256
Interrupt levels:			
External	2	2	2
Total	4	6	6
Timers/event counters:			
21-bit	–	2	2
13-bit	1	–	–
10-bit	–	1	1
I/O lines: Bidirectional	16	24	24
Input only	8	–	–
Output only	8	8	8
Additional features	–	Serial Port	Serial Port
Development support:			
Prototyping:			
EPROM	–	TMS77C82†	–
Piggyback	SE70CP160	SE70CP162	SE70CP162
XDS	Yes	Yes	Yes
EVM	Yes	Yes	Yes

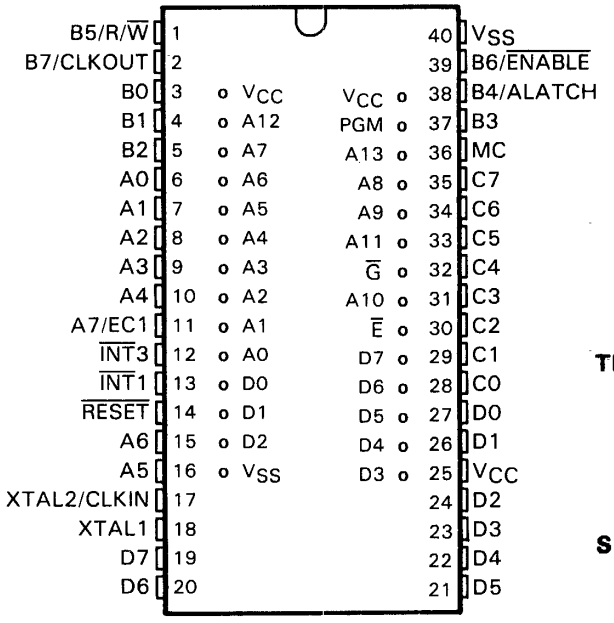
† Advance information

- Register-to-register architecture
- Memory-mapped ports for easy addressing
- Eight addressing formats, including:
  - Register-to-register arithmetic
  - Indirect addressing
  - Indexed and indirect branches and calls
- Single-instruction binary-coded decimal (BCD) add and subtract
- Two external maskable interrupts
- Flexible interrupt handling:
  - Priority servicing of simultaneous interrupts
  - Software calls through interrupt vectors
  - Precise timing of interrupts with the capture latch
  - Software monitoring of interrupt status
- Wide voltage operating range, frequency range:
  - 2.5 V – 0.8 MHz maximum
  - 6 V – 6.5 MHz maximum
- Two power-down modes:
  - Wake-Up (160 µA at 1 MHz typical)
  - Halt, XTAL/CLKIN=GND (10 µA typical)
- Silicon-gate CMOS technology
- 40-pin, 600 mil, dual-inline package
- 44-pin PLCC



A. Plastic 40-Pin DIP **B. 44-Pin PLCC (CMOS only)**

Figure 2-1. Pinouts for TMS7000, TMS7020, TMS7040, TMS70C00, TMS70C20 and TMS70C40



SE70CP160
Supports the TMS70Cx0 devices

TMS7742: 8-bit EPROM microcomputer which supports prototyping development for the TMS70x0 devices (pinout on page 2-18).

SE70P162: 8-bit piggyback microcomputer which supports prototyping development for the TMS70x0 devices (pinout on page 2-18).

Figure 2-2. Prototyping Devices Available for TMS70x0 and TMS70Cx0 Devices

Table 2-3. TMS70x0 and TMS70Cx0 Pin Descriptions

SIGNAL	PIN		I/O	DESCRIPTION
	PLCC	DIP		
A0 LSb	7	6	I	Port A. All pins may be used as high-impedance input-only lines. Pin A7/EC1 may also be used as the timer/event counter input.
A1	8	7	I	
A2	9	8	I	
A3	10	9	I	
A4	11	10	I	
A5	18	16	I	
A6	16	15	I	
A7/EC1	12	11	I	
B0	3	3	O	Port B. B0–B7 are general-purpose output-only pins. B4–B7 become memory-expansion control signals in Peripheral-Expansion, Full-Expansion, and Microprocessor modes.
B1	4	4	O	
B2	5	5	O	
B3	41	37	O	
B4/ALATCH	42	38	O	Data output/Memory interface address latch strobe
B5/R/$\overline{W}$	1	1	O	Data output/Memory read/write signal
B6/$\overline{ENABLE}$	43	39	O	Data output/Memory interface enable strobe
B7/CLKOUT	2	2	O	Data output/Internal clockout
C0	31	28	I/O	Port C. C0–C7 can be individually selected in software as general-purpose input or output pins in Single-Chip mode. C0–C7 become the LSB address/data bus in Peripheral-Expansion, Full-Expansion, and Microprocessor modes.
C1	32	29	I/O	
C2	33	30	I/O	
C3	34	31	I/O	
C4	35	32	I/O	
C5	36	33	I/O	
C6	37	34	I/O	
C7	38	35	I/O	
D0	30	27	I/O	Port D. D0–D7 can be individually selected in software as general-purpose input or output pins in Single-Chip or Peripheral-Expansion modes. D0–D7 become the MSB address/data bus in Full-Expansion and Microprocessor modes.
D1	29	26	I/O	
D2	27	24	I/O	
D3	26	23	I/O	
D4	25	22	I/O	
D5	24	21	I/O	
D6	22	20	I/O	
D7	21	19	I/O	
$\overline{INT1}$	14	13	I	Highest priority maskable interrupt
$\overline{INT3}$	13	12	I	Lowest priority maskable interrupt
$\overline{RESET}$	15	14	I	Device reset
MC	40	36	I	Mode control pin, V_{CC} for microprocessor mode
XTAL2/CLKIN	19	17	I	Crystal input for control of internal oscillator
XTAL1	20	18	O	Crystal output for control of internal oscillator
V_{CC}	28	25		Supply voltage (positive)
V_{SS}	44 39 23	40		Ground reference

2.3 TMS70x2 and TMS7742 Devices

2.3.1 TMS70x2 (NMOS) Key Features

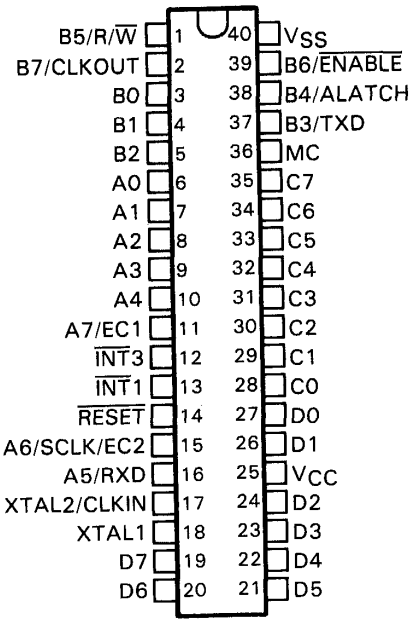
	TMS7040/20/00			TMS7042/02		TMS7742
Maximum oscillator frequency	5 MHz			8 MHz		5MHz
On-chip ROM (Kbytes)	4	2	0	4	0	4 (EPROM)
Internal RAM (bytes)	128			256		256
Interrupt levels:						
External	2			2		2
Total	4			6		6
Timers/event counters:						
13-bit	1			2		2
10-bit	–			1		1
I/O lines: Bidirectional	16			22		22
Input only	8			2		2
Output only	8			8		8
Additional features	–			Serial Port		Serial Port
Development support:						
Prototyping:						
EPROM	TMS7742			TMS7742		–
Piggyback	SE70P162			SE70P162		SE70P162
XDS	Yes			Yes		Yes
EVM	Yes			Yes		Yes

- Flexible on-chip serial port:
  - Asynchronous, Isosynchronous, or Serial I/O modes
  - Two multiprocessor communication formats
  - Error detection flags
  - Fully software programmable (bits/character, parity, and stop bits)
  - Internal or external baud-rate generator
  - Separate baud-rate timer useable as a third timer
- Register-to-register architecture
- Memory-mapped ports for easy addressing
- Eight addressing formats, including:
  - Register-to-register arithmetic
  - Indirect addressing
  - Indexed and indirect branches and calls
- Single-instruction binary-coded decimal (BCD) add and subtract
- Two external maskable interrupts
- Flexible interrupt handling:
  - Priority servicing of simultaneous interrupts
  - Software calls through interrupt vectors
  - Precise timing of interrupts with the capture latch
  - Software monitoring of interrupt status
- Supports all TMS7000 family expansion modes
- N-channel silicon-gate MOS technology
- 40-pin, 600 mil, dual-inline package

2.3.2 TMS7742 EPROM (NMOS) Device Key Features

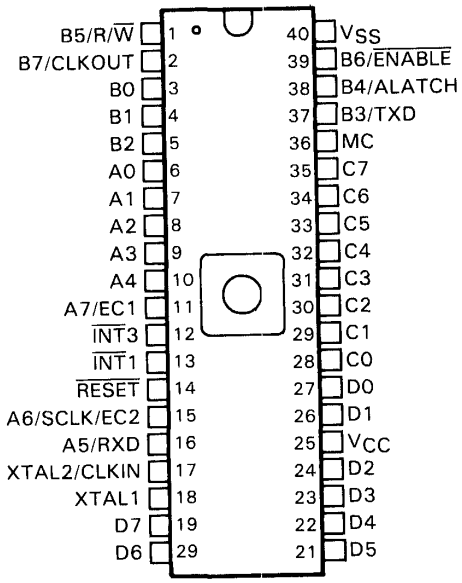
	TMS7040/20/00	TMS7042/02	TMS7742
Maximum oscillator frequency	5 MHz	8 MHz	5MHz
On-chip ROM (Kbytes)	4 | 2 | 0	4 | 0	4 (EPROM)
Internal RAM (bytes)	128	256	256
Interrupt levels:			
External	2	2	2
Total	4	6	6
Timers/event counters:			
13-bit	1	2	2
10-bit	–	1	1
I/O lines: Bidirectional	16	22	22
Input only	8	2	2
Output only	8	8	8
Additional features	–	Serial Port	Serial Port
Development support:			
Prototyping:			
EPROM	TMS7742	TMS7742	–
Piggyback	SE70P162	SE70P162	SE70P162
XDS	Yes	Yes	Yes
EVM	Yes	Yes	Yes

- EPROM programming procedure compatible with the TMS2732
- Flexible on-chip serial port:
  - Asynchronous, Isosynchronous, or Serial I/O modes
  - Two multiprocessor communication formats
  - Error detection flags
  - Fully software programmable (bits/character, parity, and stop bits)
  - Internal or external baud-rate generator
  - Separate baud-rate timer useable as a third timer
- Register-to-register architecture
- Memory-mapped ports for easy addressing
- Eight addressing formats, including:
  - Register-to-register arithmetic
  - Indirect addressing
  - Indexed and indirect branches and calls
- Single-instruction binary-coded decimal (BCD) add and subtract
- Two external maskable interrupts
- Flexible interrupt handling:
  - Priority servicing of simultaneous interrupts
  - Software calls through interrupt vectors
  - Precise timing of interrupts with the capture latch
  - Software monitoring of interrupt status
- Fully compatible with TMS7020, TMS7040, and TMS7042
- Supports all TMS7000 family expansion modes
- N-channel silicon-gate MOS technology
- 40-pin, 600 mil, dual-inline package



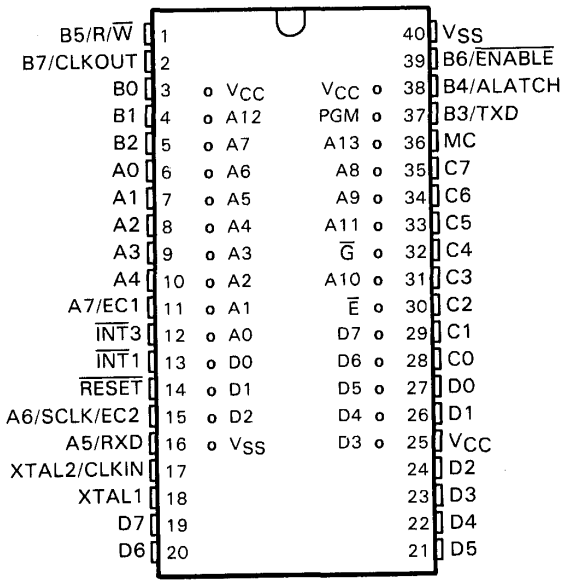
40-Pin DIP

Figure 2-3. Pinouts for TMS7002, TMS7042, and TMS7742 EPROM Device



TMS7742

Supports TMS70x2 devices



SE70P162
(8 MHz)
Supports TMS70x2 and TMS7742 devices

Figure 2-4. Prototyping Devices Available for TMS70x2 and TMS7742 Devices

Table 2-4. TMS70x2 and TMS7742 Pin Descriptions

OPERATION MODES				EPROM MODE (TMS7742 ONLY)		
SIGNAL	PIN	I/O	DESCRIPTION	SIGNAL	I/O	DESCRIPTION
A0 LSb A1 A2 A3 A4 A5/RXD A6/SCLK/EC2 A7/EC1	6 7 8 9 10 16 15 11	I/O I/O I/O I/O I/O I I/O I/O	A0–A4 and A7 are general-purpose bidirectional pins. A5 and A6 are input-only data pins. Data input/Serial port receiver Data input/Serial port clock/ Timer 2 event counter Data I/O/Timer 1 event counter	A7 A6 A5 A4 A3	I I I I I	A3–A7 are address lines.
B0 B1 B2 B3/TXD B4/ALATCH B5/R/$\overline{W}$ B6/$\overline{ENABLE}$ B7/CLKOUT	3 4 5 37 38 1 39 2	O O O O O O O O	B0–B3 are outputs. B4–B7 are outputs in Single-Chip mode and memory interface pins in all other modes. Data output/Serial port transmitter Data output/Memory interface address latch strobe Data output/Memory read/write signal Data output/Memory interface enable strobe Data output/Internal clockout			
C0 C1 C2 C3 C4 C5 C6 C7	28 29 30 31 32 33 34 35	I/O I/O I/O I/O I/O I/O I/O I/O	Port C is a bidirectional data port. In Microprocessor, Peripheral-Expansion, and Full-Expansion modes, Port C is a multiplexed low address and data bus.	Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8	I/O I/O I/O I/O I/O I/O I/O I/O	Q1–Q8 are bidirectional data lines.
D0 D1 D2 D3 D4 D5 D6 D7	27 26 24 23 22 21 20 19	I/O I/O I/O I/O I/O I/O I/O I/O	Port D is a bidirectional data port. In Microprocessor or Full-Expansion mode, it is the high address bus.	A8 A9 A11 A10 $\overline{E}$ A0 A1 A2	I I I I I I I I	A0–A2 and A8–A11 are address lines. Chip enable
$\overline{INT1}$	13	I	Highest priority external maskable interrupt			
$\overline{INT3}$	12	I	Lowest priority external maskable interrupt			
$\overline{RESET}$	14	I	Reset	GND		V_{SS} for EPROM mode
MC	36	I	Mode control pin, V_{CC} for Microprocessor mode	$\overline{G}/V_{PP}$		Program enable (21 V to program, (0 V to verify)
XTAL2/CLKIN	17	I	Crystal input for control of internal oscillator	GND		V_{SS} for EPROM mode
XTAL1	18	O	Crystal output for control of internal oscillator			
V_{CC}	25		Supply voltage (5 V)	V_{CC}		Supply voltage (5 V)
V_{SS}	40		Ground reference	GND		Ground reference

2.4 TMS70Cx2 and TMS77C82 Devices

2.4.1 TMS70Cx2 (CMOS) Key Features

	TMS70C40A/ C20A/C00A	TMS70C42/C02	TMS77C82†
Max osc freq at 5 V ± 10 %	5 MHz	6 MHz	7.5 MHz
On-chip ROM (Kbytes)	4 | 2 | 0	4 | 0	8 (EPROM)
Internal RAM (bytes)	128	256	256
Interrupt levels: External Total	 2 4	 2 6	 2 6
Timers/event counters: 21-bit 13-bit 10-bit	 – 1 –	 2 – 1	 2 – 1
I/O lines: Bidirectional Input only Output only	 16 8 8	 24 – 8	 24 – 8
Additional features	–	Serial Port	Serial Port
Development support: Prototyping: EPROM Piggyback XDS EVM	 – SE70CP160A Yes Yes	 TMS77C82† SE70CP162 Yes Yes	 – SE70CP162 Yes Yes

† Advance information

- Flexible on-chip serial port:
  - Asynchronous, Isosynchronous, or Serial I/O modes
  - Two multiprocessor communication formats
  - Error detection flags
  - Fully software programmable (bits/char, parity, and stop bits)
  - Internal or external baud-rate generator
  - Separate baud-rate timer useable as a third timer
- Memory-mapped ports for easy addressing
- Eight addressing formats, including:
  - Register-to-register arithmetic
  - Indirect addressing
  - Indexed and indirect branches and calls
- Two external maskable interrupts
- Flexible interrupt handling:
  - Priority servicing of simultaneous interrupts
  - Software calls through interrupt vectors
  - Precise timing of interrupts with the capture latch
  - Software monitoring of interrupt status
- Wide voltage operating range, frequency range:
  - 2.5 V – 0.8 MHz maximum
  - 6 V – 7.5 MHz maximum
- Wake-Up power-down mode
- Silicon-gate CMOS technology
- 40-pin, 600 mil, dual-inline package, 44-pin PLCC

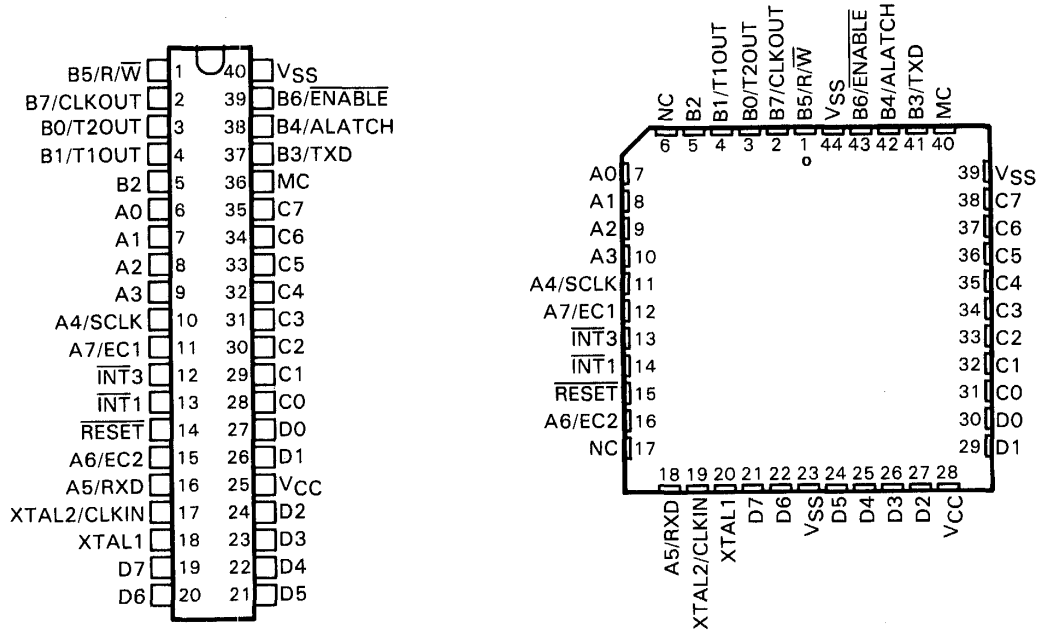
2.4.2 TMS77C82 (CMOS) Key Features (Advance Information)

This is advance information on a new product in the sampling or preproduction phase of development. Characteristic data and other specifications are subject to change without notice.

	TMS70C40A/ C20A/C00A			TMS70C42/C02		TMS77C82†
Max osc freq at 5 V ± 10 %	5 MHz			6 MHz		7.5 MHz
On-chip ROM (Kbytes)	4	2	0	4	0	8 (EPROM)
Internal RAM (bytes)	128			256		256
Interrupt levels:						
External	2			2		2
Total	4			6		6
Timers/event counters:						
21-bit	–			2		2
13-bit	1			–		–
10-bit	–			1		1
I/O lines: Bidirectional	16			24		24
Input only	8			–		–
Output only	8			8		8
Additional features	–			Serial Port		Serial Port
Development support:						
Prototyping:						
EPROM	–			TMS77C82†		–
Piggyback	SE70CP160A			SE70CP162		SE70CP162
XDS	Yes			Yes		Yes
EVM	Yes			Yes		Yes

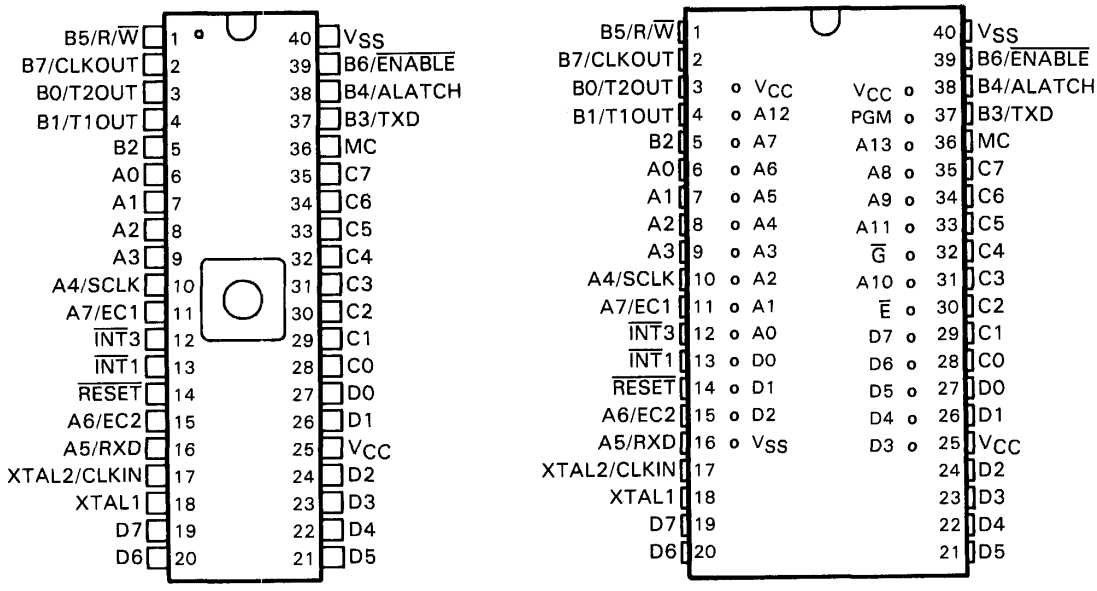
† Advance information

- EPROM programming procedure compatible with '27C64 or '27C128
- Prototyping support for the TMS70C42
- Flexible on-chip serial port:
  - Asynchronous, Isosynchronous, or Serial I/O modes
  - Two multiprocessor communication formats
  - Error detection flags
  - Fully software programmable (bits/char, parity, and stop bits)
  - Internal or external baud-rate generator
  - Separate baud-rate timer useable as a third timer
- Memory-mapped ports for easy addressing
- Eight addressing formats, including:
  - Register-to-register arithmetic
  - Indirect addressing
  - Indexed and indirect branches and calls
- Two external maskable interrupts
- Flexible interrupt handling:
  - Priority servicing of simultaneous interrupts
  - Software calls through interrupt vectors
  - Precise timing of interrupts with the capture latch
  - Software monitoring of interrupt status
- Silicon-gate CMOS technology, 40-pin, 600 mil, dual-inline package



A. 40-Pin DIP **B. 44-Pin PLCC**

Figure 2-5. Pinouts for TMS70C02 and TMS70C42 Devices



TMS77C82 **SE70CP162**

Figure 2-6. Prototyping Devices Available for TMS70Cx2 and TMS77C82 Devices

Table 2-5. TMS70Cx2 and TMS77C82† Pin Descriptions

OPERATION MODES					EPROM MODE (TMS77C82 ONLY)		
SIGNAL	PIN NO. PLCC	PIN NO. DIP	I/O	DESCRIPTION	SIGNAL	I/O	DESCRIPTION
A0 LSb	7	6	I/O	A0–A7 are general-purpose	A7	I	A3–A7 are
A1	8	7	I/O	bidirectional pins.	A6	I	address lines.
A2	9	8	I/O		A5	I	
A3	10	9	I/O		A4	I	
A4/SCLK	11	10	I/O	Data I/O/Serial port clock	A3	I	
A5/RXD	18	16	I/O	Data I/O/Serial port receiver	$\overline{A12}$	I	
A6/EC2	16	15	I/O	Data I/O/Timer 2 event counter	$\overline{PGM}$	I	
A7/EC1	12	11	I/O	Data I/O/Timer 1 event counter	$\overline{G}$	I	
B0/T2OUT	3	3	O	B0–B3 are outputs. B4–B7 are outputs in Single-Chip mode and memory			
B1/T1OUT	4	4	O	interface pins in all other modes. B0 and B1 are outputs for Timer 2			
B2	5	5	O	and Timer 1.			
B3/TXD	41	37	O	Data output/Serial port transmitter			
B4/ALATCH	42	38	O	Data output/Memory interface address latch strobe			
B5/$\overline{R/W}$	1	1	O	Data output/Memory read/write signal			
B6/$\overline{ENABLE}$	43	39	O	Data output/Memory interface enable strobe			
B7/CLKOUT	2	2	O	Data output/Internal clockout			
C0	31	28	I/O	Port C is a bidirectional data port. In	Q1	I/O	Q1–Q8 are
C1	32	29	I/O	Microprocessor, Peripheral-Expansion,	Q2	I/O	bidirectional
C2	33	30	I/O	and Full-Expansion modes, Port C is	Q3	I/O	data lines.
C3	34	31	I/O	a multiplexed low address and data	Q4	I/O	
C4	35	32	I/O	bus.	Q5	I/O	
C5	36	33	I/O		Q6	I/O	
C6	37	34	I/O		Q7	I/O	
C7	38	35	I/O		Q8	I/O	
D0	30	27	I/O	Port D is a bidirectional data port. In	A8	I	A0–A2 and
D1	29	26	I/O	Microprocessor and Full-Expansion	A9	I	A8–A10 are
D2	27	24	I/O	modes, it is the high address bus.	A11	I	address lines.
D3	26	23	I/O		A10	I	
D4	25	22	I/O		$\overline{E}$	I	Chip enable
D5	24	21	I/O		A0	I	
D6	22	20	I/O		A1	I	
D7	21	19	I/O		A2	I	
$\overline{INT1}$	14	13	I	Highest priority maskable interrupt			
$\overline{INT3}$	13	12	I	Lowest priority maskable interrupt			
$\overline{RESET}$	15	14	I	Reset	GND		V_{SS} for EPROM mode
MC	40	36	I	Mode control pin, V_{CC} for Microprocessor mode	V_{PP}		Program enable (21 V to program, (0 V to verify)
XTAL2/CLKIN	19	17	I	Crystal input for control of internal oscillator	GND		V_{SS} for EPROM mode
XTAL1	20	18	O	Crystal output for control of internal oscillator			
V_{CC}	28	25		Supply voltage (positive)	V_{CC}		Supply voltage (5 V)
V_{SS}	23, 39, 44	40		Ground reference	GND		Ground reference

† Advance information

2.5 TMS7742 and SE70P162 Prototyping Devices

2.5.1 TMS7742 EPROM (NMOS) Prototyping Device Key Features

The TMS7742 supports prototyping for the TMS7020, TMS7040, and the TMS7042 up to a maximum operational frequency of 5 MHz.

	TMS7040/20/00			TMS7042/02		TMS7742
Maximum oscillator frequency	5 MHz			8 MHz		5MHz
On-chip ROM (Kbytes)	4	2	0	4	0	4 (EPROM)
Internal RAM (bytes)	128			256		256
Interrupt levels:						
External	2			2		2
Total	4			6		6
Timers/event counters:						
13-bit	1			2		2
10-bit	–			1		1
I/O lines: Bidirectional	16			22		22
Input only	8			2		2
Output only	8			8		8
Additional features	–			Serial Port		Serial Port
Development support:						
Prototyping:						
EPROM	TMS7742			TMS7742		–
Piggyback	SE70P162			SE70P162		SE70P162
XDS	Yes			Yes		Yes
EVM	Yes			Yes		Yes

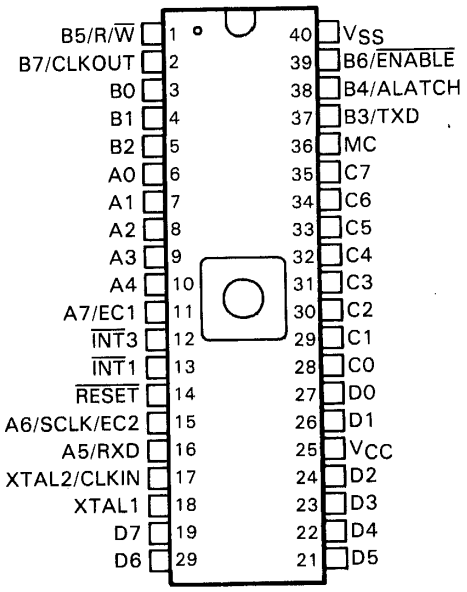
- EPROM programming procedure compatible with the TMS2732
- Flexible on-chip serial port:
  - Asynchronous, Isosynchronous, or Serial I/O modes
  - Two multiprocessor communication formats
  - Error detection flags
  - Fully software programmable
  - Internal or external baud-rate generator
  - Separate baud-rate timer useable as a third timer
- Register-to-register architecture
- Memory-mapped ports for easy addressing
- Eight addressing formats, including:
  - Register-to-register arithmetic
  - Indirect addressing
  - Indexed and indirect branches and calls
- Single-instruction binary-coded decimal (BCD) add and subtract
- Two external maskable interrupts
- Flexible interrupt handling:
  - Priority servicing of simultaneous interrupts
  - Software calls through interrupt vectors
  - Precise timing of interrupts with the capture latch
  - Software monitoring of interrupt status
- Supports all TMS7000 family expansion modes
- N-channel silicon-gate MOS technology
- 40-pin, 600 mil, dual-inline package

2.5.2 SE70P162 (NMOS) Piggyback Prototyping Device Key Features

The SE70P162 supports full-frequency prototyping for the TMS7020, TMS7040, and TMS7042.

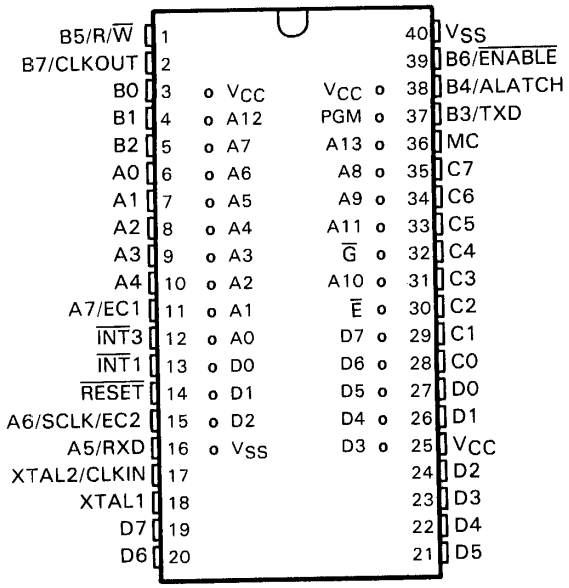
	TMS7040/20/00	TMS7042/02	TMS7742
Maximum oscillator frequency	5 MHz	8 MHz	5MHz
On-chip ROM (Kbytes)	4 | 2 | 0	4 | 0	4 (EPROM)
Internal RAM (bytes)	128	256	256
Interrupt levels:			
External	2	2	2
Total	4	6	6
Timers/event counters:			
13-bit	1	2	2
10-bit	–	1	1
I/O lines: Bidirectional	16	22	22
Input only	8	2	2
Output only	8	8	8
Additional features	–	Serial Port	Serial Port
Development support:			
Prototyping:			
EPROM	TMS7742	TMS7742	–
Piggyback	SE70P162	SE70P162	SE70P162
XDS	Yes	Yes	Yes
EVM	Yes	Yes	Yes

- Uses TMS2764 or TMS27128 EPROMs in a piggyback socket
- Register-to-register architecture
- Flexible on-chip serial port:
  - Asynchronous, Isosynchronous, or Serial I/O modes
  - Two multiprocessor communication formats
  - Error detection flags
  - Fully software programmable
  - Internal or external baud-rate generator
  - Separate baud-rate timer useable as a third timer
- Memory-mapped ports for easy addressing
- Eight addressing formats, including:
  - Register-to-register arithmetic
  - Indirect addressing
  - Indexed and indirect branches and calls
- Single-instruction binary-coded decimal (BCD) add and subtract
- Two external maskable interrupts
- Flexible interrupt handling:
  - Priority servicing of simultaneous interrupts
  - Software calls through interrupt vectors
  - Precise timing of interrupts with the capture latch
  - Software monitoring of interrupt status
- Fully compatible with TMS7042 at 8 MHz
- 40-pin, 600 mil, dual-inline package



Ceramic 40-Pin DIP

Figure 2-7. TMS7742 Pinout



Ceramic 40-Pin DIP

Figure 2-8. SE70P162 Pinout

Table 2-6. TMS7742 and SE70P162 Pin Descriptions

OPERATION MODES				EPROM MODE (TMS7742 ONLY)		
SIGNAL	PIN	I/O	DESCRIPTION	SIGNAL	I/O	DESCRIPTION
A0 LSb	6	I/O	A0–A4 and A7 are general-purpose	A7	I	A3–A7 are
A1	7	I/O	bidirectional pins. A5 and A6 are	A6	I	address lines.
A2	8	I/O	input-only data pins.	A5	I	
A3	9	I/O		A4	I	
A4	10	I/O		A3	I	
A5/RXD	16	I	Data input/Serial port receiver			
A6/SCLK/EC2	15	I/O	Data input/Serial port clock/ Timer 2 event counter			
A7/EC1	11	I/O	Data I/O/Timer 1 event counter			
B0	3	O	B0–B3 are outputs. B4–B7 are outputs in Single-Chip mode			
B1	4	O	and memory interface pins in all other modes.			
B2	5	O				
B3/TXD	37	O	Data output/Serial port transmitter			
B4/ALATCH	38	O	Data output/Memory interface address latch strobe			
B5/R/$\overline{W}$	1	O	Data output/Memory read/write signal			
B6/$\overline{ENABLE}$	39	O	Data output/Memory interface enable strobe			
B7/CLKOUT	2	O	Data output/Internal clockout			
C0	28	I/O	Port C is a bidirectional data port. In	Q1	I/O	Q1–Q8 are
C1	29	I/O	Microprocessor, Peripheral-Expansion,	Q2	I/O	bidirectional
C2	30	I/O	and Full-Expansion modes, Port C is	Q3	I/O	data lines.
C3	31	I/O	a multiplexed low address and data	Q4	I/O	
C4	32	I/O	bus.	Q5	I/O	
C5	33	I/O		Q6	I/O	
C6	34	I/O		Q7	I/O	
C7	35	I/O		Q8	I/O	
D0	27	I/O	Port D is a bidirectional data port. In	A8	I	A0–A2 and
D1	26	I/O	Microprocessor and Full-Expansion	A9	I	A8–A11 are
D2	24	I	modes, it is the high address bus.	A11	I	address lines.
D3	23	I		A10	I	
D4	22	I		$\overline{E}$	I	Chip enable
D5	21	I		A0	I	
D6	20	I		A1	I	
D7	19	I		A2	I	
$\overline{INT1}$	13	I	Highest priority external maskable interrupt			
$\overline{INT3}$	12	I	Lowest priority external maskable interrupt			
$\overline{RESET}$	14	I	Reset	GND		V_{SS} for EPROM mode
MC	36	I	Mode control pin, V_{CC} for Microprocessor mode	$\overline{G}/V_{PP}$		Program enable (21 V to program, (0 V to verify)
XTAL2/CLKIN	17	I	Crystal input for control of internal oscillator	GND		V_{SS} for EPROM mode
XTAL1	18	O	Crystal output for control of internal oscillator			
V_{CC}	25		Supply voltage (5 V)	V_{CC}		Supply voltage (5 V)
V_{SS}	40		Ground reference	GND		Ground reference

2.6 SE70CP160 and SE70CP162 Prototyping Devices

2.6.1 SE70CP160 (CMOS) Piggyback Prototyping Device Key Features

The SE70CP160 supports prototyping development for the TMS70C20 and the TMS70C40.

	TMS70C40A/ C20A/C00A			TMS70C42/C02		TMS77C82†
Max osc freq at 5 V ± 10 %	6 MHz			6 MHz		7.5 MHz
On-chip ROM (Kbytes)	4	2	0	4	0	8 (EPROM)
Internal RAM (bytes)	128			256		256
Interrupt levels:						
External	2			2		2
Total	4			6		6
Timers/event counters:						
21-bit	–			2		2
13-bit	1			–		–
10-bit	–			1		1
I/O lines: Bidirectional	16			24		24
Input only	8			–		–
Output only	8			8		8
Additional features	–			Serial Port		Serial Port
Development support:						
Prototyping:						
EPROM	–			TMS77C821		–
Piggyback	SE70CP160A			SE70CP162		SE70CP162
XDS	Yes			Yes		Yes
EVM	Yes			Yes		Yes

† Advance information

- Uses '27C64, '27C128, or compatible EPROMs in a piggyback socket
- Register-to-register architecture
- Memory-mapped ports for easy addressing
- Eight addressing formats, including:
  - Register-to-register arithmetic
  - Indirect addressing
  - Indexed and indirect branches and calls
- Single-instruction binary-coded decimal (BCD) add and subtract
- Two external maskable interrupts
- Flexible interrupt handling:
  - Priority servicing of simultaneous interrupts
  - Software calls through interrupt vectors
  - Precise timing of interrupts with the capture latch
  - Software monitoring of interrupt status
- Wide voltage operating range, frequency range:
  - 2.5 V – 0.8 MHz maximum
  - 6 V – 6.5 MHz maximum
- Two power-down modes:
  - Wake-up (160 µA at 1 MHz typical)
  - Halt (10 µA typical)
- Fully compatible with TMS70Cx0 devices
- Silicon-gate CMOS technology
- 40-pin, 600 mil, dual-inline package

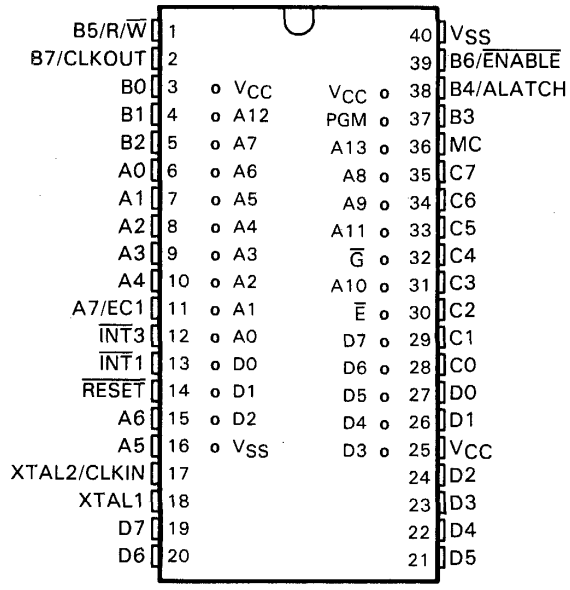
2.6.2 SE70CP162 (CMOS) Piggyback Prototyping Device Key Features

The SE70CP162 supports prototyping development for the TMS70C42.

	TMS70C40A/ C20A/C00A	TMS70C42/C02	TMS77C82†
Max osc freq at 5 V ± 10 %	5 MHz	6 MHz	7.5 MHz
On-chip ROM (Kbytes)	4 | 2 | 0	4 | 0	8 (EPROM)
Internal RAM (bytes)	128	256	256
Interrupt levels: External Total	 2 4	 2 6	 2 6
Timers/event counters: 21-bit 13-bit 10-bit	 – 1 –	 2 – 1	 2 – 1
I/O lines: Bidirectional Input only Output only	16 8 8	24 – 8	24 – 8
Additional features	–	Serial Port	Serial Port
Development support: Prototyping: EPROM Piggyback XDS EVM	 – SE70CP160A Yes Yes	 TMS77C82† SE70CP162 Yes Yes	 – SE70CP162 Yes Yes

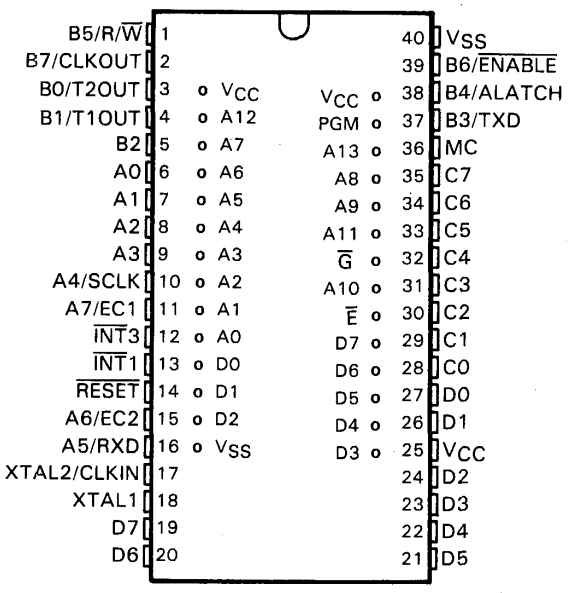
† Advance information

- Uses '27C64, '27C128, or compatible EPROMs in a piggyback socket
- Flexible on-chip serial port:
  - Asynchronous, Isosynchronous, or Serial I/O modes
  - Two multiprocessor communication formats
  - Error detection flags
  - Fully software programmable (bits/character, parity, and stop bits)
  - Internal or external baud-rate generator
  - Separate baud-rate timer useable as a third timer
- Register-to-register architecture
- Memory-mapped ports for easy addressing
- Eight addressing formats
- Single-instruction binary-coded decimal (BCD) add and subtract
- Two external maskable interrupts
- Flexible interrupt handling:
  - Priority servicing of simultaneous interrupts
  - Software calls through interrupt vectors
  - Precise timing of interrupts with the capture latch
  - Software monitoring of interrupt status
- Wide voltage operating range, frequency range:
  - 2.5 V – 0.8 MHz maximum
  - 6 V – 7.5 MHz maximum
- Wake-Up power-down mode
- Fully compatible with TMS70Cx0 devices
- Silicon-gate CMOS technology
- 40-pin, 600 mil, dual-inline package



Ceramic 40-Pin DIP

Figure 2-9. SE70CP160 Pinout



Ceramic 40-Pin DIP

Figure 2-10. SE70CP162 Pinout

Table 2-7. SE70CP162 Pin Descriptions†

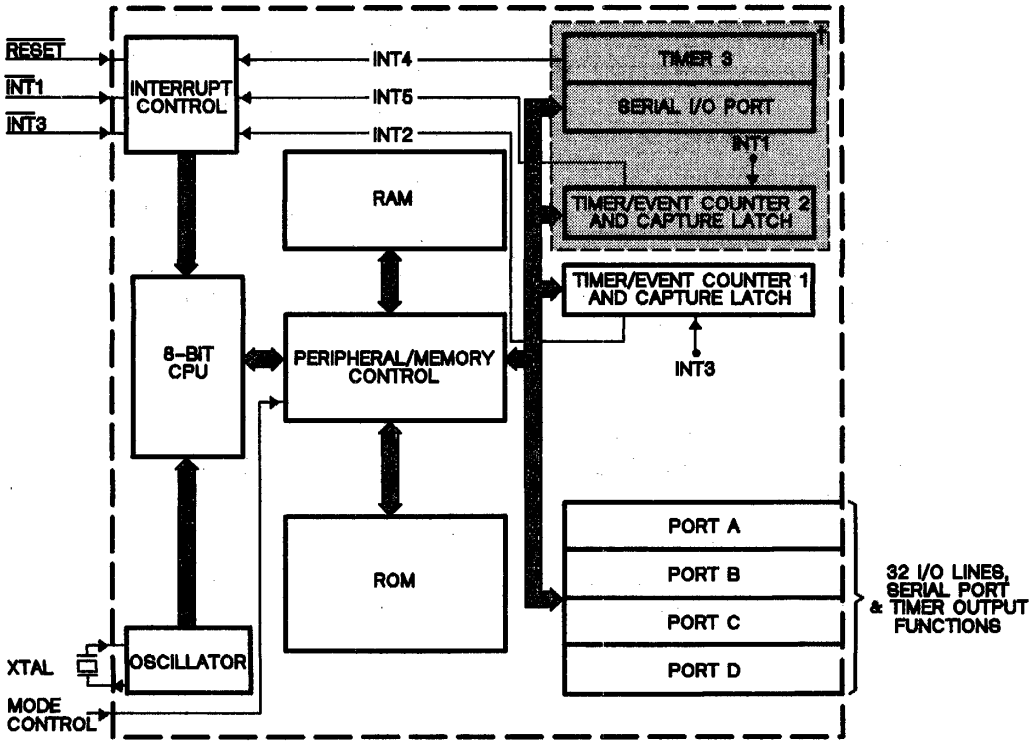
SIGNAL	PIN	I/O	DESCRIPTION
A0 LSb	6	I/O	A0–A4 and A7 are general-purpose bidirectional pins.
A1	7	I/O	
A2	8	I/O	
A3	9	I/O	
A4/SCLK	10	I/O	
A5/RXD	16	I/O	Data I/O/Serial port receiver
A6/EC2	15	I/O	Data I/O/Serial port clock/Timer 2 event counter
A7/EC1	11	I/O	Data I/O/Timer 1 event counter
B0/T2OUT	3	O	B0–B3 are outputs. B4–B7 are outputs in Single-Chip mode and memory interface pins in all other modes. B0 and B1 also contain the timer output functions.
B1/T1OUT	4	O	
B2	5	O	
B3/TXD	37	O	Data output/Serial port transmitter
B4/ALATCH	38	O	Data output/Memory interface address latch strobe
B5/R/$\overline{W}$	1	O	Data output/Memory interface read/write signal
B6/$\overline{ENABLE}$	39	O	Data output/Memory interface enable strobe
B7/CLKOUT	2	O	Data output/Internal clockout
C0	28	I/O	Port C is a bidirectional data port. In Microprocessor, Peripheral-Expansion, and Full-Expansion modes, Port C is a multiplexed low address/data bus.
C1	29	I/O	
C2	30	I/O	
C3	31	I/O	
C4	32	I/O	
C5	33	I/O	
C6	34	I/O	
C7	35	I/O	
D0	27	I/O	Port D is a bidirectional data port. In Microprocessor and Full-Expansion modes, it is the high address bus.
D1	26	I/O	
D2	24	I/O	
D3	23	I/O	
D4	22	I/O	
D5	21	I/O	
D6	20	I/O	
D7	19	I/O	
$\overline{INT1}$	13	I	Highest priority maskable interrupt
$\overline{INT3}$	12	I	Lowest priority maskable interrupt
$\overline{RESET}$	14	I	Device reset
MC	36	I	Mode control pin, V_{CC} for Microprocessor mode
XTAL2/CLKIN	17	I	Crystal input for control of internal oscillator
XTAL1	18	O	Crystal output for control of internal oscillator
V_{CC}	25		Supply voltage (5 V)
V_{SS}	40		Ground reference

† For SE70CP160 pin descriptions, refer to the TMS70Cx0 device pin description table on page 2-7.

3. TMS7000 Family Architecture

This section discusses the internal architecture of the TMS7000 family[3] devices. Topics in this section include:

Figure 3-1 shows the major components of the TMS7000 family devices' internal architecture.



† Timer 3, serial port, and timer/event counter 2 available for TMS70x2 and TMS70Cx2 devices only

Figure 3-1. TMS7000 Family Block Diagram

---

3 *TMS7000* and *TMS7000 family* refer to all TMS7000 devices as described in Section 2.

3.1 On-Chip RAM and Registers

TMS7000 family devices have a 64K-byte maximum memory address space. On-chip and off-chip memory address space varies according to the particular family member used and mode selected (see Section 3.3, Memory Modes). The following sections discuss the Register File (RF), the Peripheral File (PF), and three CPU registers: the Stack Pointer (SP), the Status Register (ST), and the Program Counter (PC).

3.1.1 Register File (RF)

On-chip RAM is called the **Register File** (RF). Depending upon the device used, the RF has either 128 or 256 bytes of memory treated as registers R0–R127 or R0–R255. These are located in lower memory as follows:

Device	Number of Registers	Register Range	Memory Address
TMS70x0	128	R0–R127	>0000 – >007F
TMS70Cx0	128	R0–R127	>0000 – >007F
TMS70x2	256	R0–R255	>0000 – >00FF
TMS70Cx2	256	R0–R255	>0000 – >00FF

The first two registers, **R0** and **R1**, are also referred to as **Register A** and **Register B**, respectively. Several instructions use Register A or B implicitly as either the source or destination register. For example, the STSP instruction stores the contents of the Stack Pointer in Register B. Other instructions may use Registers A or B to save memory or increase execution speed. Unless otherwise indicated, any register in the Register File can be used as a source or destination register.

3.1.2 Peripheral File (PF)

The **Peripheral File** (PF) is mapped into locations >0100 to >01FF, which are referred to as P0–P255. These Peripheral-File locations contain the 8-bit PF registers, used for interrupt control, parallel I/O ports, timer control, memory-expansion control, and serial port control. All PF addresses not used onboard the TMS7000 are mapped externally in all modes except Single-Chip. Several instructions, called Peripheral-File instructions, communicate with the PF registers, allowing easy use of externally-mapped peripheral devices.

3.1.3 Stack Pointer (SP)

The **Stack Pointer** (SP) is an 8-bit CPU register that points to the top of the stack. The stack is physically located in the on-chip RAM, or RF. When the stack is used, the SP points to the last or top entry on the stack. During reset, the SP is loaded with >01. The SP is loaded from Register B (R1) via the LDSP instruction and initialized to any other value by executing a stack initialization program such as the one illustrated in Figure 3-2. This feature allows the stack to be located anywhere in the Register File. The SP is *loaded* into Register B via the STSP command. The SP is automatically incremented when data is pushed onto the stack and automatically decremented after data is popped from the stack.

```
INIT    MOV     %>60,B
        LDSP
```

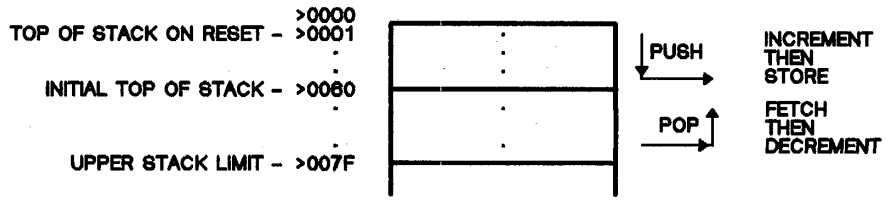


Figure 3-2. Example of Stack Initialization in the Register File

3.1.4 Status Register (ST)

The **Status Register** (ST) is an 8-bit CPU register that contains three conditional status bits - carry (C), sign (N), zero (Z) - and a global interrupt enable bit (I). The C, N, and Z bits are used for arithmetic operations, bit rotating, and conditional branching.

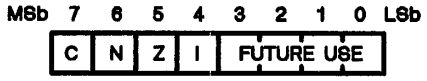


Figure 3-3. Status Register (ST)

Carry (C) Bit Used as carry-in/carry-out for most rotate and arithmetic instructions.

Negative (N) Bit
Contains the most significant bit of the destination operand contents after instruction execution.

Zero (Z) Bit Contains a 1 when the destination operand equals zero after instruction execution.

Global Interrupt Enable (I) Bit

Enables/disables all interrupts. The EINT (Enable Interrupts) instruction sets this bit to 1; the DINT (Disable Interrupts) instruction clears it.

This bit must be set to a 1 for interrupts to be acknowledged. However, the individual interrupt flag bits can be set whether this bit is set to a 1 or a 0.

Jump-on-condition instructions are also associated with the C, N, and Z status bits to provide conditional program-flow options.

During reset all bits in the Status Register are cleared. During other interrupts, the Status Register is saved on the stack and can be accessed via the PUSHST and POPST instructions.

3.1.5 Program Counter (PC)

The 16-bit **Program Counter** (PC) consists of two 8-bit registers in the CPU. These registers contain the MSB and the LSB of a 16-bit address: the **Program Counter High** (PCH) and **Program Counter Low** (PCL).

The PC acts as the 16-bit address pointer of the opcodes and operands in memory of the currently executing instruction. During reset, the MSB and the LSB of the PC are loaded into Register A and Register B, respectively.

3.2 On-Chip General Purpose I/O Ports

TMS7000 devices have 32 I/O pins organized as four 8-bit parallel Ports A, B, C, and D.

Port A — **TMS70x0** and **TMS70Cx0** devices – Port A is an input-only port

TMS70x2 devices – A0–A4 and A7 are bidirectional data pins; A5 and A6 are input-only data pins

TMS70Cx2 devices – Port A is fully bidirectional

Port B — **All devices** – Port B is an output-only port

Port C, Port D — **All devices** – both ports are bidirectional; they are also used as the address/data bus for memory expansion

Ports A, C, and D are each controlled and accessed via individual **Data-Direction Registers** and **Data Registers** in the Peripheral File. Output-only port B has only a Data Register. The *Data Register* contains the value to be input or output; the *Data-Direction Register* indicates whether the value is an input or an output. I/O pins can be individually designated as input or output by writing a 1 or 0 to a corresponding bit in their PF Data-Direction Register. A **1** makes the pin an *output*, a **0** makes the pin an *input*.

Writing to the Data-Direction Register does not affect the value in the Data Register. This allows all bidirectional pins to be used for either input or output by only changing the Data-Direction Register.

During a hardware reset, all Data-Direction Registers are cleared, forcing all bidirectional ports to their high-impedance input state. It is good practice to load Ports A, C, and D Data Registers before programming any bidirectional bits as outputs. During a hardware reset, Port B is set to all 1s.

Caution:

When any port is configured as an output-only port, applying an external potential to its pins may affect system reliability. The value read at the port pins will be the same as the last value internally written to the port. Reading the port returns the value at the pins, which can override the data written to the port.

Figure 3-4 (page 3-6) shows the logic for each bidirectional I/O line.

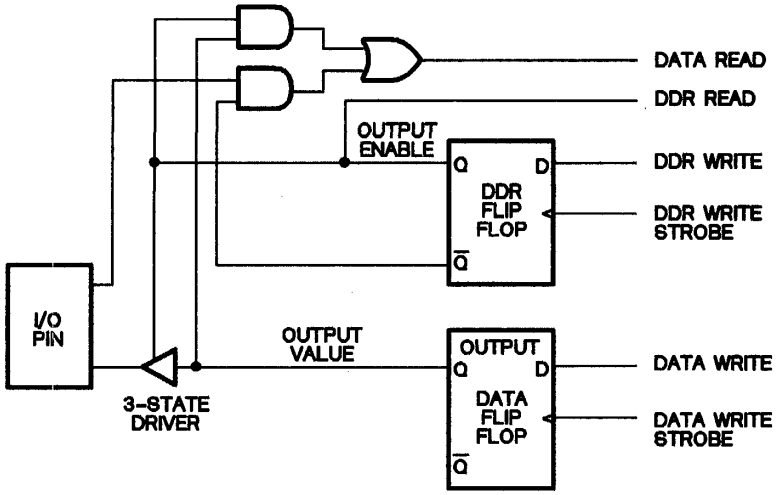


Figure 3-4. Bidirectional I/O Logic

Table 3-1. TMS70x0 and TMS70Cx0 Port Configuration

I/O	SINGLE-CHIP MODE	PERIPHERAL-EXPANSION MODE	FULL-EXPANSION MODE	MICROPROCESSOR MODE
Port A	8 input pins A7=A7/EC1	8 input pins A7=A7/EC1	8 input pins A7=A7/EC1	8 input pins A7=A7/EC1
Port B	8 output pins	4 output pins 4 bus control signals	4 output pins 4 bus control signals	4 output pins 4 bus control signals
Port C	8 I/O pins	8-bit address/data bus	8-bit low address/data bus (LSB)	8-bit low address/data bus (LSB)
Port D	8 I/O pins	8 I/O pins	8-bit high address bus (MSB)	8-bit high address bus (MSB)
Total I/O Pins Available	8 input pins 8 output pins 16 I/O pins	8 input pins 4 output pins 8 I/O pins	8 input pins 4 output pins	8 input pins 4 output pins
Total Memory Pins	None	8 address/data (multiplexed) 4 memory control	16 address/data 4 memory control	16 address/data 4 memory control

Table 3-2. TMS70x2 Port Configuration

I/O	SINGLE-CHIP MODE	PERIPHERAL-EXPANSION MODE	FULL-EXPANSION MODE	MICROPROCESSOR MODE
Port A	6 I/O pins 2 input pins A5=A5/RX A6=A6/SCLK/EC2 A7=A7/EC1	6 I/O pins 2 input pins A5=A5/RX A6=A6/SCLK/EC2 A7=A7/EC1	6 I/O pins 2 input pins A5=A5/RX A6=A6/SCLK/EC2 A7=A7/EC1	6 I/O pins 2 input pins A5=A5/RX A6=A6/SCLK/EC2 A7=A7/EC1
Port B	8 output pins B3=B3/TX	4 output pins 4 bus control signals B3=B3/TX	4 output pins 4 bus control signals B3=B3/TX	4 output pins 4 bus control signals B3=B3/TX
Port C	8 I/O pins	8-bit address/data bus	8-bit low address/data bus (LSB)	8-bit low address/data bus (LSB)
Port D	8 I/O pins	8 I/O pins	8-bit high address bus (MSB)	8-bit high address bus (MSB)
Total I/O Pins Available	2 input pins 8 output pins 22 I/O pins	2 input pins 4 output pins 14 I/O pins	2 input pins 4 output pins 6 I/O pins	2 input pins 4 output pins 6 I/O pins
Total Memory Pins	None	8 address/data (multiplexed) 4 memory control	16 address/data 4 memory control	16 address/data 4 memory control

Table 3-3. TMS70Cx2 Port Configuration

I/O	SINGLE-CHIP MODE	PERIPHERAL-EXPANSION MODE	FULL-EXPANSION MODE	MICROPROCESSOR MODE
Port A	8 I/O pins A4=A4/SCLK A5=A5/RXD A6=A6/EC2 A7=A7/EC1	8 I/O pins A4=A4/SCLK A5=A5/RXD A6=A6/EC2 A7=A7/EC1	8 I/O pins A4=A4/SCLK A5=A5/RXD A6=A6/EC2 A7=A7/EC1	8 I/O pins A4=A4/SCLK A5=A5/RXD A6=A6/EC2 A7=A7/EC1
Port B	8 output pins B3=B3/TXD B1=B1/T1OUT B0=B0/T2OUT	4 output pins 4 bus control signals B3=B3/TXD B1=B1/T1OUT B0=B0/T2OUT	4 output pins 4 bus control signals B3=B3/TXD B1=B1/T1OUT B0=B0/T2OUT	4 output pins 4 bus control signals B3=B3/TXD B1=B1/T1OUT B0=B0/T2OUT
Port C	8 I/O pins	8-bit address/data bus	8-bit low address/data bus (LSB)	8-bit low address/data bus (LSB)
Port D	8 I/O pins	8 I/O pins	8-bit high address bus (MSB)	8-bit high address bus (MSB)
Total I/O Pins Available	8 output pins 24 I/O pins	4 output pins 16 I/O pins	4 output pins 8 I/O pins	4 output pins 8 I/O pins
Total Memory Pins	None	8 address/data (multiplexed) 4 memory control	16 address/data 4 memory control	16 address/data 4 memory control

3.2.1 Port A

On **TMS70x0** and **TMS70Cx0** parts, Port A is an 8-bit high-impedance input-only port, providing eight general-purpose input lines. Pin A7/EC1 may also be used to clock the on-chip timer/event counter (see Section 3.7, Programmable Timer/Event Counters).

On **TMS70x2** parts, pins A0–A4 and pin A7/EC1 of Port A are bidirectional I/O lines. Pins A5 and A6 are general-purpose input-only pins that also have other functions when using the serial port. Pin A5/RXD receives incoming serial data and pin A6/SCLK/EC2 is the serial clock input or output. Pins A6/SCLK/EC2 and A7/EC1 may also be used to clock the on-chip timer/event counters, Timer 2 and Timer 1, respectively.

On **TMS70Cx2** devices, Port A is a fully-bidirectional I/O port. However, pins A5/RXD and A4/SCLK serve as the serial data receive pin and serial clock, respectively, when the serial port is used. Pins A6/EC2 and A7/EC1 may be used to clock the on-chip timer/event counters, Timer 2 and Timer 1, respectively. Note that SCLK has been moved to A4 on the TMS70Cx2 devices from A6 on the TMS70x2 devices. This frees up EC2 to be used at the same time as SCLK.

3.2.2 Port B

In ***Single-Chip mode***, Port B is an 8-bit general-purpose output port. Reading Port B returns the value written to the pins unless modified by an external value at the pins.

In ***all other memory modes***, Port B is split into two parts. The lower nibble (pins B0–B3) are general-purpose output-only pins. The most significant nibble (pins B4–B7) contains the bus control signals: ALATCH, R/$\overline{W}$, $\overline{ENABLE}$, and CLKOUT.

On **TMS70x2** and **TMS70Cx2** devices, pin B3 is also the serial output line (TXD) for the serial port.

3.2.3 Port C

In ***Single-Chip mode***, Port C is an 8-bit bidirectional I/O port. Any of its eight pins may be individually programmed as an input or output line.

In ***all other memory modes***, Port C becomes a multiplexed address/data port for the off-chip memory bus. In this case, Port C provides the least significant byte of a 16-bit address, followed by eight bits of read or write data. (Port D provides the most significant byte of the 16-bit address.)

3.2.4 Port D

In ***Single-Chip*** or ***Peripheral-Expansion mode***, Port D is an 8-bit bidirectional I/O port. Any of its eight pins may be individually programmed as an input or output line under software control.

In ***Full-Expansion*** and ***Microprocessor modes***, Port D becomes a multiplexed address/data port for the off-chip memory bus. In this case, Port D provides the most significant byte of a 16-bit address. (Port C provides the least significant byte of the 16-bit address.)

3.3 Memory Modes

The TMS7000 can address up to 64K bytes. Four memory modes can be selected by a combination of software and hardware: the **Single-Chip, Peripheral-Expansion, Full-Expansion**, and **Microprocessor modes**.

The **Mode Control** (MC) input pin forces the TMS7000 into Microprocessor mode when set to a V_{CC}. If the MC pin is held at V_{SS}, the remaining memory modes can be selected by bits 6 and 7 of the Peripheral File I/O Control Register (IOCNT0 – P0), as shown in Table 3-4.

Table 3-4. Mode Selection Conditions (MC Pin)

MODE	MODE SELECT CONDITIONS		
	MODE CONTROL PIN (MC)	IOCNT0 BITS 7,6	
Single-Chip	V_{SS}	0	0
Peripheral-Expansion	V_{SS}	0	1
Full-Expansion	V_{SS}	1	0
Microprocessor	V_{CC}	X	X

Note: X = Don't Care

During reset the IOCNT0 register is set to a 0. (Refer to Section 3.6 for a detailed description of reset and the initialization procedure for the IOCNT0 register.) Table 3-5 and Table 3-6 summarize the four memory modes.

Table 3-5. TMS70x0 and TMS70Cx0 Memory Map

	SINGLE-CHIP		PERIPHERAL-EXPANSION		FULL EXPANSION		MICROPROCESSOR
>0000 >007F	Register File		Register File		Register File		Register File
>0080 >00FF	Reserved		Reserved		Reserved		Reserved
>0100 >010B	On-Chip I/O		On-Chip I/O		On-Chip I/O		On-Chip I/O
>010C >0200	Not Available		Peripheral Expansion		Peripheral Expansion		Peripheral Expansion
>0201			Not Available		Memory Expansion		Memory Expansion
>F000 >F800 >FFFF	4K ROM	2K ROM	4K ROM	2K ROM	4K ROM	2K ROM	

Table 3-6. TMS70x2 and TMS70Cx2 Memory Map

'70x2	SINGLE-CHIP	PERIPHERAL-EXPANSION	FULL EXPANSION	MICRO-PROCESSOR	'70Cx2
>0000 >00FF	Register File	Register File	Register File	Register File	>0000 >00FF
>0100 >0117	On-Chip I/O	On-Chip I/O	On-Chip I/O	On-Chip I/O	>0100 >0123
>0118 >01FF	Not Available	Peripheral Expansion	Peripheral Expansion	Memory Expansion	>0124 >01FF
>0200 >EFFF		Not Available	Not Available		>0200 >EFFF
>F000 >FFFF	4K ROM	4K ROM	4K ROM		>F000 >FFFF

Table 3-7. TMS70x0 and TMS70Cx0 Peripheral Memory Map

			SINGLE-CHIP	PERIPHERAL-EXPANSION	FULL-EXPANSION	MICRO-PROCESSOR
P0	>0100	IOCNT0	I/O Control register			
P1	>0101	–	Reserved			
P2	>0102	T1DATA	Timer 1 data			
P3	>0103	T1CTL	Timer 1 control			
P4	>0104	APORT	Port A data			
P5	>0105	–	Reserved			
P6	>0106	BPORT	Port B Data	†		
P7	>0107	–	Reserved			
P8	>0108	CPORT	Port C Data			
P9	>0109	CDDR	Port C Data-Direction Register	Peripheral Expansion		
P10	>010A	DPORT	Port D Data			
P11	>010B	DDDR	Port D Data-Direction Register			
P12–P255	>010C–>01FF		Not available	Peripheral Expansion		

† In expansion modes, Port B is referenced in a special manner. See the Port B discussion on page 3-17.

Table 3-8. TMS70x2 Peripheral Memory Map

			SINGLE-CHIP	PERIPHERAL-EXPANSION	FULL-EXPANSION	MICRO-PROCESSOR
P0	>0100	IOCNT0	I/O Control register 0			
P1	>0101	–	Reserved			
P2	>0102	T1DATA	Timer 1 Data			
P3	>0103	T1CTL	Timer 1 Control			
P4	>0104	APORT	Port A Data			
P5	>0105	ADDR	Port A Data-Direction Register			
P6	>0106	BPORT	Port B Data	†		
P7	>0107	–	Reserved			
P8	>0108	CPORT	Port C Data			
P9	>0109	CDDR	Port C Data-Direction Register	Peripheral Expansion		
P10	>010A	DPORT	Port D Data			
P11	>010B	DDDR	Port D Data-Direction Register			
P12–P15	>010C–>010F		Not available			
P16	>0110	IOCNT1	I/O Control Register 1			
P17	>0111	SMODE	First Write after reset – Serial Mode register			
		SCTL0	Write – Serial Control register 0			
		SSTAT	Read – Serial port status register			
P18	>0112	T2DATA	Timer 2 Data			
P19	>0113	T2CTL	Timer 2 Control			
P20	>0114	T3DATA	Timer 3 Data			
P21	>0115	SCTL1	Serial Control register 1			
P22	>0116	RXBUF	Receiver Buffer			
P23	>0117	TXBUF	Transmitter Buffer			
P24–P225	>0118>01FF		Not available	Peripheral Expansion		

† In expansion modes, Port B is referenced in a special manner. See the Port B discussion on page 3-17.

Table 3-9. TMS70Cx2 Peripheral Memory Map

			SINGLE-CHIP	PERIPHERAL-EXPANSION	FULL-EXPANSION	MICRO-PROCESSOR
P0	>0100	IOCNT0	I/O Control register 0			
P1	>0101	IOCNT2	I/O Control register 2			
P2	>0102	IOCNT1	I/O Control register 1			
P3	>0103	–	Reserved			
P4	>0104	APORT	Port A Data			
P5	>0105	ADDR	Port A Data-Direction Register			
P6	>0106	BPORT	Port B data	†		
P7	>0107	–	Reserved			
P8	>0108	CPORT	Port C Data	Peripheral Expansion		
P9	>0109	CDDR	Port C Data-Direction Register			
P10	>010A	DPORT	Port D Data			
P11	>010B	DDDR	Port D Data Direction Register			
P12	>010C	T1MSDATA	Timer 1 MSB decrementer reload register/MSB readout latch			
P13	>010D	T1LSDATA	Timer 1 LSB reload register/LSB decrementer value			
P14	>010E	T1CTL1	Timer 1 control register 1/MSB readout latch			
P15	>010F	T1CTL0	Timer 1 control register 0/LSB capture latch value			
P16	>0110	T2MSDATA	Timer 2 MSB decrementer reload register/MSB readout latch			
P17	>0111	T2LSDATA	Timer 2 LSB reload register/LSB decrementer value			
P18	>0112	T2CTL1	Timer 2 control register 1/MSB readout latch			
P19	>0113	T2CTL0	Timer 2 control register 0/LSB capture latch value			
P20	>0114	SMODE	Serial port mode control register			
P21	>0115	SCTL0	Serial port control register 0			
P22	>0116	SSTAT	Serial port Status Register			
P23	>0117	T3DATA	Timer 3 reload register/decrementer value			
P24	>0118	SCTL1	Serial port control register 1			
P25	>0119	RXBUF	Receiver buffer			
P26	>011A	TXBUF	Transmitter buffer			
P27–P35	>011B–>0123		Reserved			
P36–P255	>0124–>01FF		Not available	Peripheral Expansion		

† In expansion modes, Port B is referenced in a special manner. See the Port B discussion on page 3-17.

3.3.1 Single-Chip Mode

Single-Chip mode is selected when:

$$MC = V_{SS} \text{ and PF Register IOCNT0} = 00XX\ XXXX$$

In Single-Chip mode, the TMS7000 family devices function as standalone microcomputers with no off-chip memory-expansion bus. User memory consists of the RAM register file and ROM. All 32 I/O lines may be used for various purposes, such as scanning keyboards, driving displays, and controlling other mechanisms. The four ports are configured as shown in Figure 3-5.

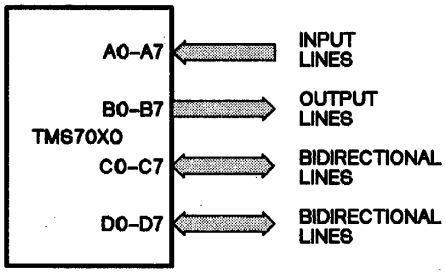


Figure 3-5. I/O Ports - Single-Chip Mode

Figure 3-6 shows the Single-Chip mode memory map. The unused Peripheral File (PF) locations and off-chip memory addresses cannot be addressed. If you attempt to read one of these locations, an undefined value is returned. Writing to these addresses has no effect. Peripheral-File registers P0–P11 reference the I/O ports and other on-chip functions. Table 3-7, Table 3-8, and Table 3-9 list the Peripheral-File registers that are available in Single-Chip mode.

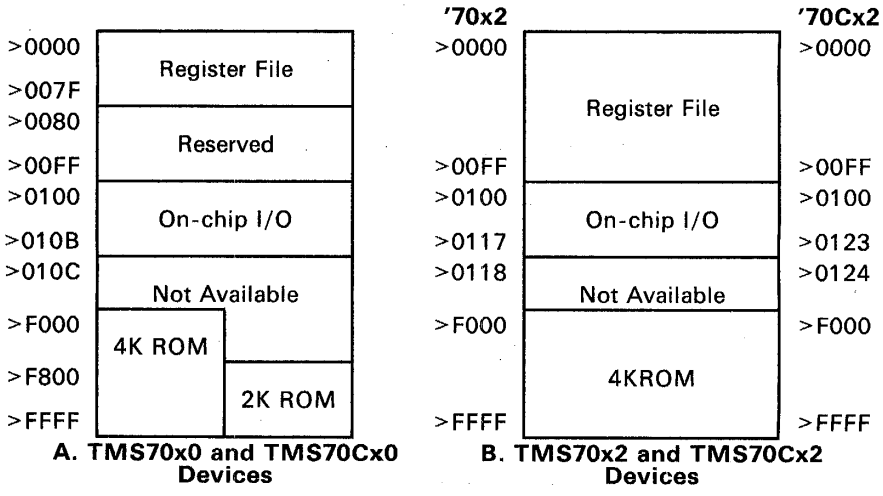


Figure 3-6. Single-Chip Mode Memory Map

Port A is accessed via PF register **P4** (APORT). When P4 is read, such as with a MOVP (Move from PF) instruction, the value on the Port A input pins is returned. The input data is read approximately two machine cycles before the completion of the instruction.

- On the **TMS70x0** and **TMS70Cx0** devices, bit 7 (A7) is the MSb and bit 0 (A0) is the LSb. When the on-chip timer/event counter is placed in the External Event-Counter mode, bit A7/EC1 serves as the external clock input, triggering the event counter on every positive-going transition.
- On **TMS70x2** parts, pins A0–A4 and pin A7/EC1 are bidirectional I/O pins. Each of these pins can become either an output or an input pin depending upon the value in the Port A Data-Direction Register (ADDR) P5:

  P5 bit = 1 Corresponding Port A pin becomes an output.
  P5 bit = 0 Corresponding Port A pin becomes a high-impedance input.

  Figure 3-4 (page 3-6) shows a diagram of the bidirectional I/O logic.

  Pins A5 and A6/SCLK/EC2 have multiple functions. Normally they are both input-only pins (as on TMS70x0 parts), but A5 can also be the serial data receiver (RXD). Pin A6/SCLK/EC2 can also be the serial clock I/O pin (SCLK) for the serial port. A6 can be either the serial clock output or it can drive the on-chip serial clock when connected to an external clock. (See the serial port section for more information, Section 3.8). Pin A6 can also be the external clock input for Timer 2.
- On **TMS70Cx2** devices, all pins are bidirectional I/O pins. Each of these pins can become an output or an input pin, depending upon the value in the Port A Data-Direction Register (ADDR) P5. Pins A4/SCLK, A5/RXD, A6/EC2, and A7/EC1 have multiple functions. Pins A4/SCLK and A5/RXD are the serial clock I/O pin and the serial data receiver pin, respectively, when the serial port is used. Pins A6/EC2 and A7/EC1 may be used to clock the on-chip timer/event counter, Timer 2 and Timer 1, respectively.

Port B output pins always assert the value of the Port B Data Register, PF register **P6** (BPORT). Writing to P6 loads the Port B register, modifying the Port B output pins. Reading from P6 provides the current value of the Port B pins. When $\overline{\text{RESET}}$ goes active, Port B register contents are set to 1s by the on-chip circuitry.

Port C, Port D (CPORT and DPORT) are bidirectional I/O pins. Data Registers are **P8** and **P10** of the Peripheral File. Each of these pins can become either an output or an input pin depending upon the value in the port C and D Data-Direction Register, locations P9 and P11 (CDDR and DDDR). A 1 causes an output and a 0 causes a high-impedance input. Writing to the Data-Direction Registers does not affect the Data Registers. Writing to the Data Registers modifies the programmed output pins. Reading the Data Register returns either the current value at the pin (when the pin is an input) or the current value of the Data Register (for pins configured as outputs). Refer to Figure 3-4 (page 3-6) for a diagram of the bidirectional I/O logic.

Peripheral-File instructions ANDP, ORP, and XORP perform a read/modify/-write cycle on PF registers. When applied to a port's Data Register, these instructions can clear, set, or complement the output pins on the port.

The following program segment illustrates the use of the I/O lines in the Single-Chip mode for all family members.

```
IOCNT0   EQU     P0              I/O control register 1
APORT    EQU     P4              Port A data register
BPORT    EQU     P6              Port B data register
CPORT    EQU     P8              Port C data register
CDDR     EQU     P9              Port C data-direction register
DPORT    EQU     P10             Port D data register
DDDR     EQU     P11             Port D data-direction register
*
RESET    MOVP    %>3F,IOCNT0     Set Single-Chip mode, enable all
*                                interrupts, clear all pulse
*                                flip-flops
L1       MOVP    %>02,DPORT      Load Port D with 0000 0010
*                                (D7-D0)
L2       MOVP    %>00,CPORT      Load Port C with 0000 0000
*                                (C7-C1)
         MOVP    %>F0,CDDR       Config C7-C4 outputs, C3-C0 inputs
         MOVP    %>0F,DDDR       Config D7-D4 inputs, D3-D0 outputs
         ORP     %>04,DPORT      Set pin D2 to 1
         ANDP    %>7F,CPORT      Clear pin C7
         BTJZP   %>08,CPORT,L1   Jump if C3 is 0
         MOVP    %>55,BPORT      Set Port B to 0101 0101 (B7-B0)
         XORP    %1,BPORT        Toggle bit B0
         BTJOP   %>41,APORT,L2   Jump if either A6 or A1 is a 1
```

Note:

The percent sign (%) indicates the Immediate Addressing mode. The instruction set is described in Section 6.

3.3.2 Peripheral-Expansion Mode

Peripheral-Expansion mode is selected when:

MC = V_{SS} and PF Register IOCNT0 = 01XX XXXX

Peripheral-Expansion mode incorporates features of both the I/O-intensive Single-Chip mode and the memory-intensive Full-Expansion mode. References to Peripheral-File addresses (locations >0100 to >01FF) not corresponding to on-chip PF registers produce off-chip memory cycles. During Peripheral-File instructions, a PF port is read, even if the value is not needed, such as in a `MOVP A,P6`. If a hardware configuration makes this read undesirable, use a STA (Store A) instruction with the memory-mapped address of the PF register. The ability to reference off-chip addresses allows the TMS7000 to be directly connected to most of the popular peripheral devices developed for 8-bit microprocessors. The TMS7000 PF instructions reference these off-chip peripherals just as easily as they access on-chip PF registers.

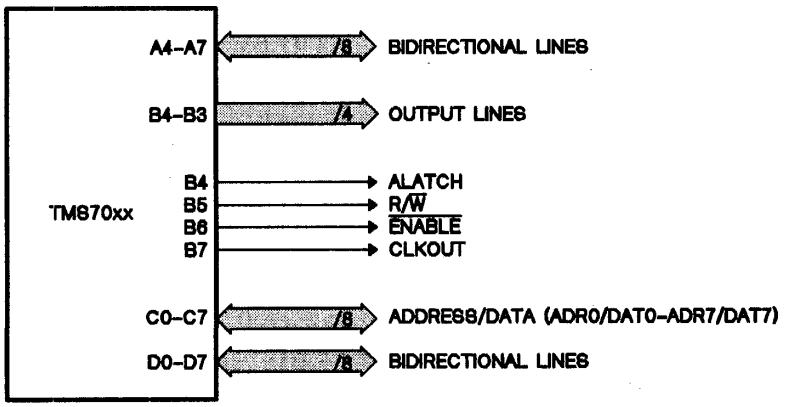
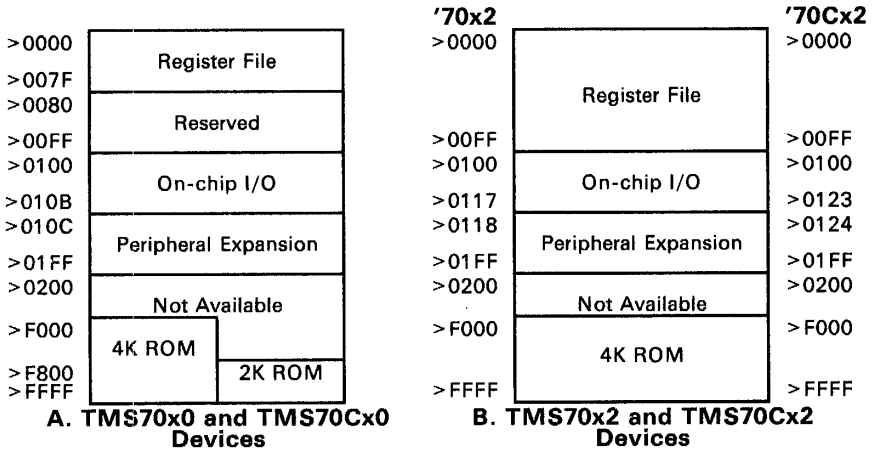


Figure 3-7. I/O Ports - Peripheral-Expansion Mode



A. TMS70x0 and TMS70Cx0 Devices

B. TMS70x2 and TMS70Cx2 Devices

Figure 3-8. Peripheral-Expansion Mode Memory Map

Port A functions the same as in Single-Chip mode.

Port B is divided into two sections: pins B0–B3 function as individual output pins, the same as in Single-Chip mode; pins B4–B7, however, function as external memory bus controls:

- Pin B4/ALATCH is strobed to logic 1 while Port C asserts the memory address.
- Pin B5/R/$\overline{W}$ is driven to logic 1 for a read cycle and to logic zero for a write cycle.
- Pin B6/$\overline{ENABLE}$ is asserted at logic 0 whenever an external memory cycle is in progress.
- Pin B7/CLKOUT is an output clock intended for general memory control timing.

Exact signal timing is described in Section 4.

References to the Port B Data Register, P6, are handled in a special manner. When a value is written to P6, pins B0–B3 output the new value. Pins B4–B7 ignore the new value and continue to output memory bus signals. An external memory write cycle will also write the entire 8 bits of the new value to the external address >0106. When P6 is read, the least significant nibble (B0–B3) is taken from the current value on pins B0–B3. The most significant nibble is obtained by reading the external address >0106.

Port C functions as a multiplexed address/data port for the memory-expansion bus. In normal configurations, Port C is attached to the input of an 8-bit latch such as an SN74LS373. The B4/ALATCH signal drives the G input of the latch, so that the latch's Q outputs follow the D inputs while B4/ALATCH is high, and ouputs become latched when it falls. After B4/ALATCH falls and data (such as a memory address) is latched, Port C either becomes a high-impedance input for read cycles or it asserts the output data for write cycles.

Port D functions identically to a bit-programmable, bidirectional I/O port, as in the Single-Chip mode.

Notes:

1. The Port C Data-Direction Register is mapped into external memory. The Port C input or output function can be recreated externally by mapping a latch at location >0108.
2. Because B4/ALATCH, B5/R/$\overline{W}$, and Port C are active for both external and internal (ROM and RAM) memory cycles, it is recommended that B6/$\overline{ENABLE}$ be gated with the chip-select input of all external memory devices to prevent external bus conflicts.

3.3.3 Full-Expansion Mode

Full-Expansion mode is selected when:

$$MC = V_{SS} \text{ and } PF \text{ Register IOCNT0} = 10XX\ XXXX$$

Full-Expansion mode uses a 16-bit address to extend the memory addressing capability of the TMS7000 to its full 64K-byte limit. External memory may be accessed with instructions using the Direct, Register File Indirect, and Indexed Addressing modes of the instruction set. This meets a variety of application requirements by expanding the external program or data storage.

Full-Expansion mode I/O is identical to the Peripheral-Expansion mode *except that Port D is used to output the most significant byte (MSB) of the 16-bit address*. Thus, Port D is not available as an I/O port. The four ports are configured as shown in Figure 3-9. Figure 3-10 shows the I/O memory assignments for the Full-Expansion mode.

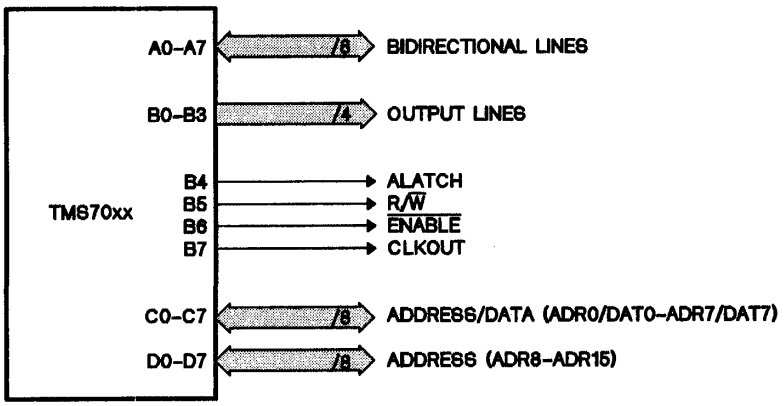
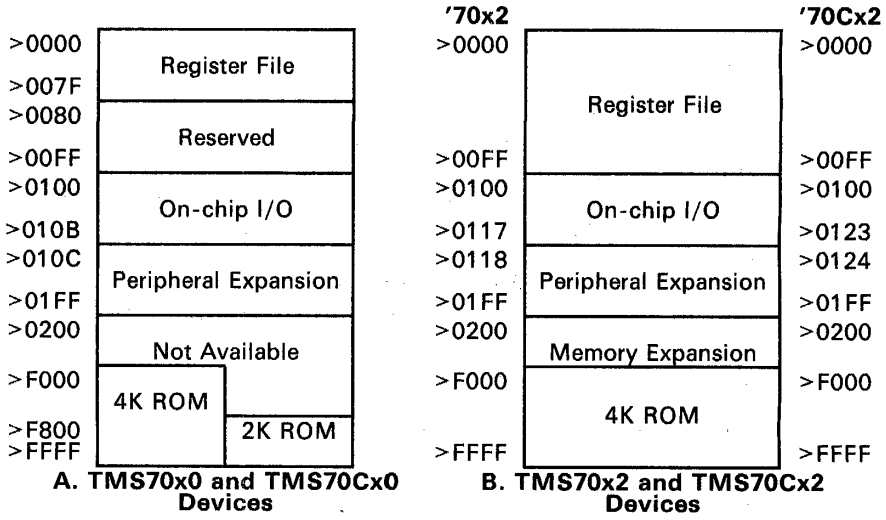


Figure 3-9. I/O Ports - Full-Expansion Mode

As in the Peripheral-Expansion mode, accesses to Peripheral-File registers (locations >0100 to >01FF) which are not directly implemented as on-chip registers produce off-chip memory cycles. The on-chip Peripheral-File registers are listed in Table 3-7, Table 3-8, and Table 3-9. Note that the Port D Data Register (DPORT) and the Port D Data-Direction Register (DDDR) are implemented as off-chip addresses in the Full-Expansion mode. The port D input or output function can be recreated externally by mapping a latch at location >010A.



A. TMS70x0 and TMS70Cx0 Devices **B. TMS70x2 and TMS70Cx2 Devices**

Figure 3-10. Full-Expansion Mode Memory Map

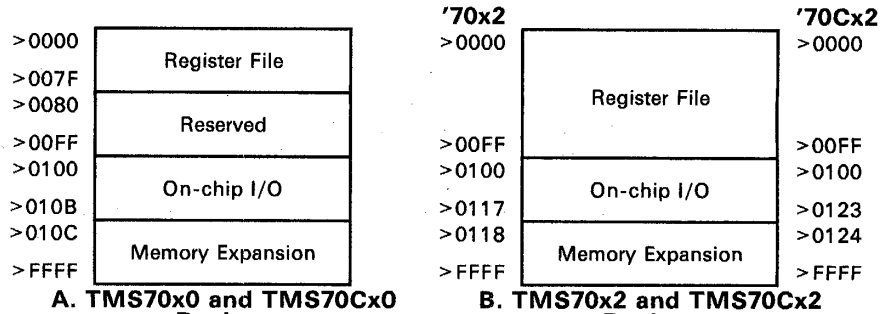
3.3.4 Microprocessor Mode

Microprocessor mode is selected when:

MC = V_{CC} and PF Register IOCNT0 = XXXX XXXX

Microprocessor mode is intended for applications that do not justify the use of on-chip ROM. The port pins are configured exactly as in Full-Expansion mode (see Figure 3-9). Unlike Full-Expansion mode, no on-chip ROM is referenced in Microprocessor mode. All memory accesses except for internal RAM and on-chip Peripheral-File locations are now addressed externally.

The MC pin must be held at logic 1 (V_{CC}) to place the device in this mode. There are no restrictions on when the value of the MC pin may change, but it is recommended that the value be changed only when the device is in reset. Indeterminant results can occur if the MC pin is changed while the device is accessing memory locations whose internal/external status may change.



A. TMS70x0 and TMS70Cx0 Devices **B. TMS70x2 and TMS70Cx2 Devices**

Figure 3-11. Microprocessor Mode Memory Map

3.4 System Clock Options

The internal state cycle period, called $t_{c(C)}$, is derived from either a crystal or an external clock source. Both NMOS and CMOS devices can use a crystal, ceramic resonator, or another approximately 50% duty cycle clock as an external clock source. The CMOS devices can also use an R-C circuit with the OSC-OFF low-power mask option (see Section 3.4.2). The internal clock then divides the external clock source frequency by two to produce the internal state frequency. For example, a 5 MHz crystal produces an internal frequency of 2.5 MHz, which drives a 400-ns machine cycle.

3.4.1 System Clock Connections

The TMS7000 devices use the following methods to implement the system clock options:

Crystals: Crystals are connected between pins XTAL1 and XTAL2/CLKIN. To optimize the crystal waveform, a 15-pF capacitor should be connected between XTAL1 and ground, and a 30-pF capacitor should be connected between XTAL2/CLKIN and ground. This connection is illustrated in Figure 3-12 *a*.

Ceramic Resonators:
Ceramic resonators are connected between pins XTAL1 and XTAL2/CLKIN. A resistor and two capacitors, with values determined by the selected ceramic resonator, must be connected as shown in Figure 3-12 *b*..

External Clock Source:
As shown in Figure 3-12 *c*, external clock sources are connected to XTAL2/CLKIN and XTAL1 is not connected.

R-C Circuits:
R-C circuits provide a simple, low-cost oscillator for applications in which frequency toleration is not a concern. R-C circuits also provide immediate start-up oscillation for the CMOS device upon exiting the Halt OSC-OFF mode of operation (see Section 3.4.2).

R-C circuits are connected as shown in Figure 3-12 *d*. The recommended value for the capacitor **C** is 47 pf. The value of the resistor **R** required for the desired frequency must be selected with respect to V_{CC}, ambient temperature, and the tolerance of the R-C components. Recommended values for the resistor in the R-C network fall in the range of 1KΩ–100KΩ.

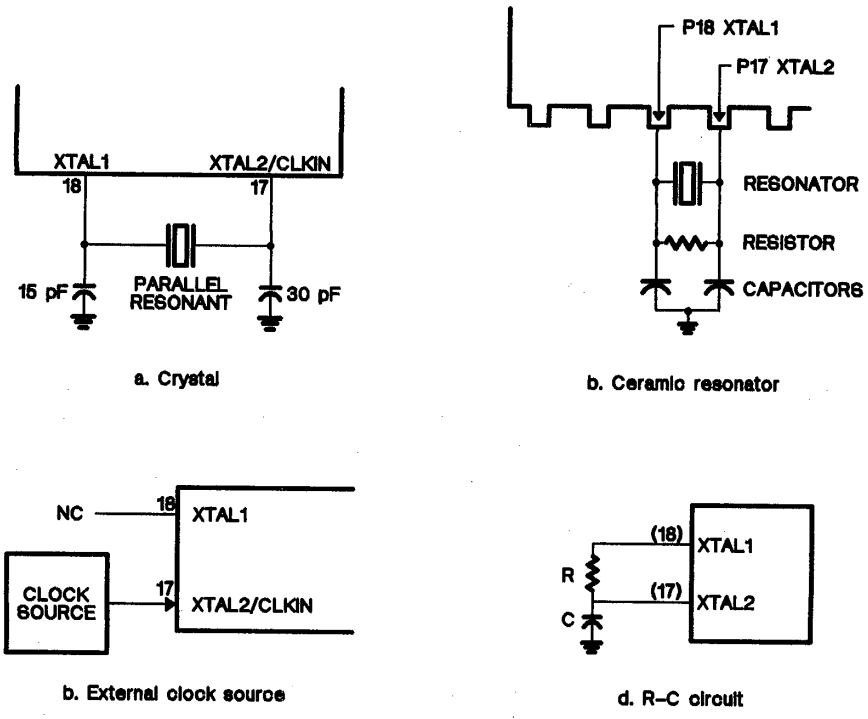


Figure 3-12. System Clock Connections

3.4.2 Low-Power Mask Options for CMOS Devices

The TMS7000 CMOS devices may use oscillator mask options which provide different levels of functionality and power consumption during the Halt low-power mode. These oscillator options are called OSC-ON and OSC-OFF.

The **TMS70Cx0** devices are mask programmable with either the OSC-ON option or the OSC-OFF mask options.

The **TMS70Cx2** devices are mask programmable with the OSC-ON mask option.

The OSC-On option will keep the on-chip oscillator active during the Halt low-power mode. Since the oscillator is still active, typical power consumption will be 80 µA/MHz for the TMS70Cx0 devices. When the device is brought out of Halt mode, there will be no delay in restoring the full operation since the oscillator is already running. The OSC-ON option is useful in applications where no delay in restoring full operation after Halt mode is more important than the lower power consumption of the OSC-OFF mode.

The OSC-OFF option is useful in applications where very low power consumption is requred in Halt mode. The OSC-OFF option causes the oscillator to cease oscillation when Halt mode is entered. This offers the lowest power consumption, typically 1 µA for the TMS70Cx0 devices. The OSC-OFF mask-programmable option supports an R-C circuit as well as a crystal, ceramic resonator, or other approximately 50% duty cycle CLKIN signal. If an R-C network is used with this option, it will restart full oscillation immediately upon exiting Halt mode. If a ceramic resonator or crystal is used, there will be a period before the oscillations stabilize, causing a delay in the response to $\overline{\text{RESET}}$ of approximately 10 milliseconds. Because of this stabilization time requirement, an external time constant of at least 10 milliseconds is recommended for $\overline{\text{RESET}}$ when using a crystal or ceramic resonator with the OSC-OFF low-power mask option.

Table 3-10. Low-Power Mask Options for CMOS Devices

MASK OPTION	HALT POWER CONSUMPTION	CLOCK SOURCE	OSCILLATOR START UP
OSC-OFF	Lowest	Ceramic resonator, crystal, or external clock source	10 millisecond delay
		R-C circuit	No delay
OSC-ON	Low	Ceramic resonator, crystal, or external clock source	No delay

Note:

OSC-ON and OSC-OFF are *mask options*, which means the option is placed on a manufacturing template, or mask, that copies the actual circuit onto the silicon device. This means the oscillator option is finalized at the start of manufacture and **cannot** be changed by software or hardware.

3.5 CMOS Low-Power Modes

The TMS7000 CMOS microcomputers can be programmed to enter low-power modes of operation when the IDLE instruction is executed. The TMS70Cx0 and TMS70Cx2 devices can both enter Wake-Up (startup) low-power mode; the TMS70Cx0 devices can also enter Halt (power-down) low-power mode. For information concerning mask options associated with the Halt low-power mdoe, see Section 3.4.2.

3.5.1 TMS70Cx0 Low-Power Modes

The TMS70Cx0 devices support the Wake-Up and Halt low-power modes. These modes are entered when:

1) Bit 5 of the Timer 1 control register (T1CTL) is set (0 for Wake-Up mode, 1 for Halt mode),
   and
2) The IDLE instruction is executed.

Activating $\overline{\text{RESET}}$ or acknowledging an enabled interrupt releases the device from either mode. Both low-power modes freeze the I/O ports, retaining their conditions before the IDLE instruction was executed. Complete RAM data retention is also maintained through both low-power modes as long as power is applied. Table 3-11 describes the low-power options.

Table 3-11. Low-Power Options for TMS70Cx0 Devices

MODE	CPU STATUS	TIMER 1 STATUS	OSC	ENTER MODE VIA	EXIT MODE VIA	CLOCK SOURCE
Wake-Up	Halted	Active	Active	IDLE	$\overline{\text{RESET}}$, $\overline{\text{INT1}}$, INT2, $\overline{\text{INT3}}$ (if enabled)	Crystal, R-C Circuit, Ceramic Resonator, External Clock
Halt (OSC-ON)	Halted	Halted	Active	IDLE	$\overline{\text{RESET}}$,$\overline{\text{INT1}}$, INT2 (if enabled)	Crystal, Ceramic Resonator, External Clock
Halt (OSC-OFF)	Halted	Halted	Halted	IDLE	$\overline{\text{RESET}}$, $\overline{\text{INT1}}$, INT2 (if enabled)	Crystal, R-C Circuit, Ceramic Resonator, External Clock

In Wake-Up mode, the oscillator and timer logic remain active. The on-chip timer may be used to release the device from the low-power state. The I_{CC} current requirements in Wake-Up mode are frequency dependent for both the OSC-ON and the OSC-OFF options.

3.5.2 TMS70Cx2 Devices

The TMS70Cx2 devices support the Wake-Up low-power mode. This mode is entered when the IDLE instruction is executed. An enabled interrupt must be executed to allow the device to return to normal operation. The TMS70Cx2 devices have the ability to disable the individual onboard timers and UART during Wake-Up mode, further reducing total power consumption. To disable Timer 1 during Wake-Up mode, set the T1HALT bit (bit 5 of T1CTL0) to 1. To disable Timer 2 during Wake-Up mode, set the T2HALT bit (bit 5 of T2CTL0) to 1. The UART/Timer 3 is disabled during Wake-Up mode by setting the SPH bit (bit 7 of SCTL0) to 1.

3.6 Interrupts and System Reset

All TMS7000 family devices have a non-maskable system reset pin, $\overline{\text{RESET}}$. This signal has the highest priority in the interrupt heirarchy. $\overline{\text{RESET}}$ immediately initializes the device.

The **TMS70x0** and **TMS70Cx0** devices have three separate, maskable interrupts that are triggered from three sources. The **TMS70x2** and **TMS70Cx2** devices have five separate maskable interrupts that can be triggered from as many as seven sources. Each interrupt has a specific priority level; if two or more interrupts occur simultaneously, they are serviced according to priority – highest first, lowest last. Table 3-12 summarizes the interrupts.

Table 3-12. Interrupt Summary

INTERRUPT	EXTERNAL/ INTERNAL	SOURCE	PRIORITY	VECTOR ADDRESS MSB	LSB
$\overline{\text{RESET}}$	E	$\overline{\text{RESET}}$ pin low	Immediate (highest priority)	>FFFE	>FFFF
$\overline{\text{INT1}}$	E	$\overline{\text{INT1}}$ pin active†	Priority 1	>FFFC	>FFFD
INT2	E/I	Timer/Event counter 1 countdown past 0	Priority 2	>FFFA	>FFFB
$\overline{\text{INT3}}$	E	$\overline{\text{INT3}}$ pin active†	Priority 3	>FFF8	>FFF9
INT4	I	RX Buffer Loaded, or TX Buffer Empty, or Timer 3 countdown past 0	Priority 4	>FFF6	>FFF7
INT5	E/I	Timer/Event counter 2 countdown thru 0	Priority 5	>FFF4	>FFF5

† The external interrupts on the TMS70Cx2 devices can be programmed for level and sense detection.
Note: INT4 and INT5 apply to TMS70x2 and TMS70Cx2 devices only.

3.6.1 Device Initialization

$\overline{\text{RESET}}$, interrupt level 0, cannot be masked. The processor recognizes a $\overline{\text{RESET}}$ immediately, even in the middle of an instruction execution. To execute the reset function, the $\overline{\text{RESET}}$ pin must be held low for a minimum of $1.25 \times t_{c(C)}$ internal state clock periods. While the $\overline{\text{RESET}}$ pin is asserted (0):

1) The Data-Direction Registers for the I/0 ports are cleared.

2) On NMOS devices, the output data flip-flops of Ports A, C, and D are set to all 1s (see Figure 3-4, page 3-6). On CMOS devices, only Port A's output data flip-flop is set to all 1s; Ports C and D output data flip-flops are not altered during a $\overline{\text{RESET}}$.

3) This places Ports C and D (and Port A on TMS70x2 and TMS70Cx2 devices) in high-impedance input mode, and Port B outputs all 1s (>FF), regardless of the internal machine clock state.

The reset function does not change the INTn flag bits in the IOCNT0 register (since all zeros are written). If any of the bits in a Peripheral File Data-Direction Register (DDR) are set to a 1, the corresponding port pin would become an output, producing a 1 level. (Remember, Data-Direction Registers are set to all 0s on $\overline{\text{RESET}}$.)

It is generally a good practice to initialize the output data flip-flop with the desired output value (by writing to the port data value register) before writing to the DDR flip-flop to make the corresponding pin an output. Figure 3-13 and Figure 3-14 show examples of possible initialization routines after the assertion of $\overline{\text{RESET}}$. Device initialization requires 17 state cycles after $\overline{\text{RESET}}$ goes inactive.

When $\overline{\text{RESET}}$ returns to its inactive condition (1), the following operations are performed before the first instruction acquisition:

1) All 0s are written to the Status Register. This clears the global interrupt enable bit (I), disabling all interrupts.

2) All 0s are written to the IOCNT0 register. This disables $\overline{\text{INT}}$1, INT2, and $\overline{\text{INT}}$3 and leaves the INTn flag bits unchanged.

3) All 0s are written to the IOCNT1 register in the TMS70x2 and TMS70Cx2 devices. This disables INT4 and INT5.

4) The PC's MSB and LSB values before $\overline{\text{RESET}}$ was asserted are stored in R0 and R1 (Registers A and B), respectively.

5) The Stack Pointer is initialized to >01.

6) The MSB and LSB of the $\overline{\text{RESET}}$ interrupt vector are fetched from locations >FFFE and >FFFF, respectively (see Table 3-12, page 3-24), and loaded into the Program Counter.

7) Program execution begins from the address placed in the Program Counter.

```
RESET   MOVP   %>2E,P0      Clear INT1-, INT2, and INT3- flags,
*                           place device in Single-Chip mode,
*                           enable INT2
        MOVP   %>0F,P16     Clear INT4, INT5 flags,
*                           enable INT4 and INT5
        MOVP   %VALU1,P4    Load Port A Data Register
        MOVP   %MASK1,P5    Load Port A Data-Direction Register
        MOVP   %VALU2,P8    Load Port C Data Register
        MOVP   %MASK2,P9    Load Port C Data-Direction Register
        MOVP   %VALU3,P10   Load Port D Data Register
        MOVP   %MASK3,P11   Load Port D Data-Direction Register
        MOVP   %VALU4,P2    Load Timer 1 reload register
        MOVP   %VALU5,P3    Load Timer 1 clock source, prescaler
*                           reload register and start timer
        MOVP   %VALU6,P18   Load Timer 2 reload register
        MOVP   %VALU7,P19   Load Timer 2 clock source, prescaler
*                           reload register and start timer
        MOV    0,P17
        MOV    %>40,P17
        MOVP   %MASK4,P17   Initialize serial port configuration
        MOVP   %>05,P17     Clear UR bit, enable transmitter
*                           and receiver
        MOVP   %VALU8,P20   Load Timer 3 reload register
        MOVP   %VALU9,P21   Initialize serial port clock source,
*                           other control bits, and Timer 3
*                           prescaler reload register
        EINT                Set global interrupt enable bit to
*                           allow interrupts
```

Figure 3-13. Sample Initialization Routine for TMS70x2 Devices

```
RESET   MOVP   %>2E,P0      Clear INT1-, INT2, and INT3- flags,
*                           place device in Single-Chip mode, and
*                           enable INT2
        MOVP   %0F,P16      Clear and enable INT4 and INT5
        MOVP   VALU1,P4     Load Port A Data Register
        MOVP   MASK1,P5     Load Port A Data-Direction Register
        MOVP   VALU2,P8     Load Port C Data Register
        MOVP   MASK2,P9     Load Port C Data-Direction Register
        MOVP   VALU3,P10    Load Port D Data Register
        MOVP   MASK3,P11    Load Port D Data-Direction Register
        MOVP   VALU4,P12    Load Timer 1 MSB reload register
        MOVP   VALU5,P13    Load Timer 1 LSB reload register
        MOVP   %>40,P14     Enable the timer output on B1
        MOVP   MASK4,P15    Initialize clock start, source, halt
*                           bit and prescaler value
        MOVP   VALU6,P16    Load Timer 2 MSB reload register
        MOVP   VALU7,P17    Load Timer 2 LSB reload register
        MOVP   %>40,P18     Enable the timer output on B0
        MOVP   MASK5,P19    Initialize clock start, source, halt
*                           bit and prescaler value
        MOVP   MASK6,P20    Initialize serial port format
        MOVP   MASK7,P21    Configure serial port
        MOVP   MASK8,P23    Load Timer 3 reload register
        MOVP   MASK9,P24    Configure serial port control
        EINT                Set global interrupt enable bit
*                           to allow interrupts
```

Figure 3-14. Sample Initialization Routine for TMS70Cx2 Devices

The Stack Pointer can also be re-initialized following reset by a executing a program similar to the one below.

```
STACK   MOV   %VALUE,B    Load Register B with the stack
*                         starting point in the Register
*                         File
        LDSP              Put this value into the Stack
*                         Pointer register
```

A simple R-C circuit can provide a power-up reset, automatically resetting the TMS7000 when power is applied. The capacitor and resistor values are selected according to the clock frequency used, the minimum voltage at which the $\overline{\text{RESET}}$ signal is at logic 1, and the ramp-up time of the power to the device. The following formula calculates the minimum time required for an adequate device reset:

$$t_{rst} = 2\left[\frac{V_{CC}}{V_{IL}}\,(1.25t_{c(C)})\right] + t_{pwr} = R\text{-}C$$

where:

t_{rst} = Total time $\overline{\text{RESET}}$ pin is held at logical level 0
V_{CC} = Supply voltage
V_{IL} = Low-level input voltage
$t_{c(C)}$ = Internal machine clock period
t_{pwr} = Ramp-up time for V_{CC}
R = Resistor value in ohms (no more than 1 megohm)
C = Capacitor value in farads

3.6.2 Interrupt Operation

The TMS7000 family's interrupts can be falling-edge sensitive, falling-edge and level sensitive, rising-edge sensitive, or rising-edge and level sensitive. Table 3-13 illustrates the interrupt configurations supported by each TMS7000 family device.

Table 3-13. External Interrupt Operation

TMS7000 DEVICE AND INTERRUPTS		FALLING EDGE	FALLING EDGE AND LEVEL	RISING EDGE	RISING EDGE AND LEVEL
TMS70x0	$\overline{INT1}$		X		
	$\overline{INT3}$		X		
TMS70x2	$\overline{INT1}$	X			
	$\overline{INT3}$	X			
TMS7742	$\overline{INT1}$	X			
	$\overline{INT3}$	X			
SE70P162	$\overline{INT1}$	X			
	INT2	X			
TMS70Cx0	$\overline{INT1}$	X			
	$\overline{INT3}$		X		
SE70CP160	$\overline{INT1}$	X			
	$\overline{INT3}$		X		
TMS70Cx2†	$\overline{INT1}$	X	X	X	X
	$\overline{INT3}$	X	X	X	X
SE70CP162†	$\overline{INT1}$	X	X	X	X
	$\overline{INT3}$	X	X	X	X
TMS77C82	$\overline{INT1}$	X	X	X	X
	$\overline{INT3}$	X	X	X	X

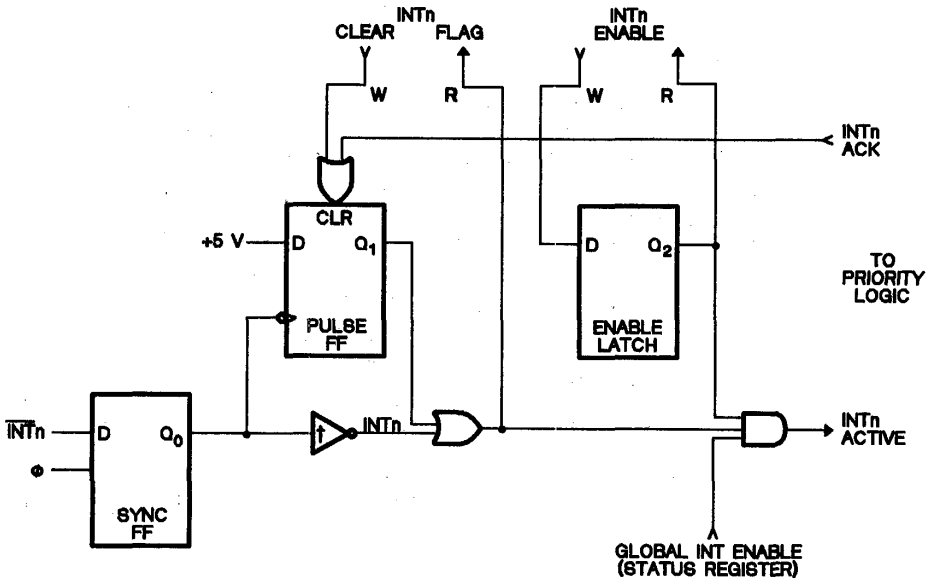
† The TMS70Cx2 and SE70CP162 devices' external interrupts edge/level-sensitive polarity are software programmable. This is accomplished via the I/O control 1 register (P1).

1) When an interrupt is first asserted, its level is gated into the Sync flip-flop by the internal state clock, $t_{c(C)}$, which has a cycle period of $2/F_{osc}$. To detect an interrupt, the INTn signal must be active for a minimum of 1.25 × $t_{c(C)}$ clock periods.

2) The negative output edge of the Sync flip-flop clocks a 1 into the Pulse flip-flop. This is the "edge" detection of the interrupt signal and is the only time a 1 is loaded into the Pulse flip-flop. The Pulse flip-flop will be set within 1.25 state clock cycles of the interrupt assertion. If the signal is removed before the CPU recognizes the interrupt, its occurrence is latched on the Pulse flip-flop output, Q1.

3) Edge-sensitive interrupts detect only the Pulse flip-flop Q1 output, not the INTn level. Once an interrupt has been asserted (INTn goes low), it becomes active if the INTn enable bit and the global interrupt enable bit (I) register are set to one.

   The "level path" logic shown in Figure 3-15 applies only to external interrupts that are both edge- and level-sensitive; it is not implemented for interrupts that are only edge-sensitive. For more information, refer to Table 3-13.



† Available only for level-sensitive interrupts

Figure 3-15. CPU Interface to Interrupt Logic

4) As shown in Figure 3-15, when the TMS7000's on-chip logic detects an active interrupt, it sends an INTn ACTIVE signal to the CPU. When the currently executing instruction is completed, the CPU acknowledges the active interrupt and routes INTA back to that interrupt's INTn ACK (interrupt acknowledge) line. If simultaneous interrupts occur, that is, more than one interrupt is active within the same instruction boundary, the interrupts are acknowledged by the CPU according to the priority levels. For example, if both INT2 and $\overline{\text{INT3}}$ occur within the same instruction boundary, INT2 is serviced first.

5) After the CPU acknowledges the interrupt, the INTn ACK line, as shown in Figure 3-15, clears the corresponding Pulse flip-flop. The CPU then pushes the Status Register contents and the Program Counter onto the stack, and clears the Status Register, including the global interrupt enable (I) bit. The CPU reads an interrupt code from the interrupt priority logic to determine which interrupt requires servicing. The 16-bit vector value is read from the two vector addresses associated with the interrupt being serviced, and is loaded into the Program Counter. The interrupt vector value is the address of the first instruction in the interrupt service routine. The interrupt vector addresses are shown in Table 3-12 on page 3-24. Instruction execution then proceeds at the new address value in the Program Counter.

Nineteen internal state clock cycles [$t_{c(C)}$] are required between the end of an instruction in the interrupted program and the start of the first instruction of the interrupt service routine. Interrupting out of the Idle state requires 17 state clock cycles.

3.6.3 Interrupt Control

The I/O control registers, IOCNT0, IOCNT1, and IOCNT2, contain the interrupt control bits. All TMS7000 family members have an IOCNT0 register. Only TMS70x2 and TMS70Cx2 devices have an IOCNT1 register, because they have two more interrupts, INT4 and INT5; only TMS70Cx2 devices have an IOCNT2 register because only they can change the polarity of their external interrupts. The I/O control registers are mapped into PF locations as follows:

Table 3-14. I/O Control Registers

PERIPHERAL FILE	TMS70x0 TMS70Cx0	TMS70x2	TMS70Cx2
IOCNT0	P0	P0	P0
IOCNT1	P16	P16	P2
IOCNT2	N/A	N/A	P1

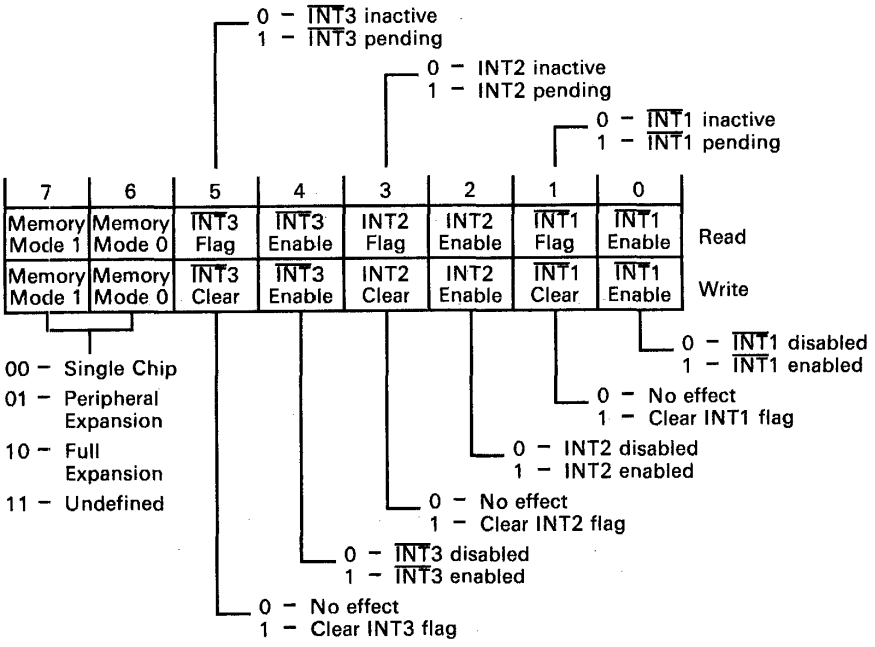


Figure 3-16. IOCNT0 - I/O Control Register 0 (P0 for All Devices)

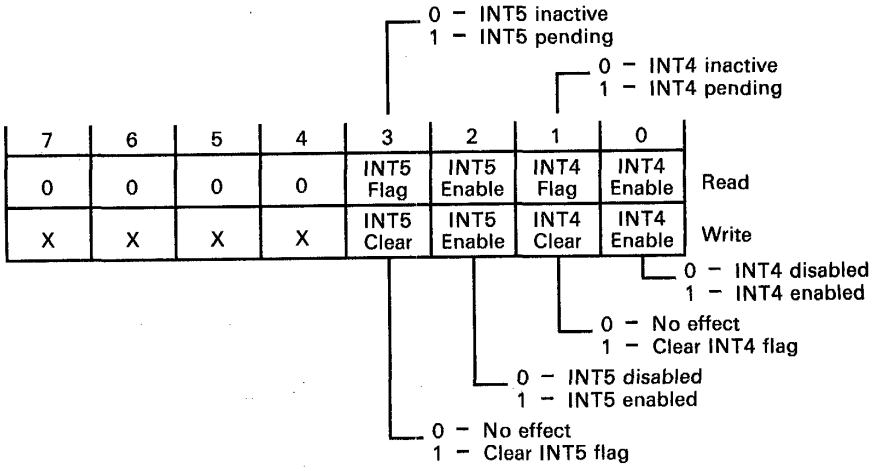


Figure 3-17. IOCNT1 - I/O Control Register 1

In the I/O control registers, each interrupt is associated with a flag bit (INTn flag) and enable bit (INTn enable). The global interrupt enable (I) bit in the Status Register allows all interrupts to be enabled or disabled at the same time. Three conditions must be met before the CPU will recognize an interrupt:

1) A 1 must be written to the INTn enable bit in the IOCNT0 or IOCNT1 register.

2) The global interrupt enable (I) bit in the Status Register must be set to 1 by the EINT instruction.

3) The interrupt must be the highest priority interrupt asserted within an instruction boundary.

Through software, the INTn enable bits can be read and written to:

- Writing a 0 individually masks the corresponding interrupt.
- Writing a 1 allows the interrupt to be recognized.

The reading of the INTn flag is handled differently (see Figure 3-15 on page 3-29):

- An active signal applied to INTn is read as a 1 from one side of an OR gate.
- INTn going active latches a 1 to the other side of the OR gate which stays latched when the signal goes inactive.

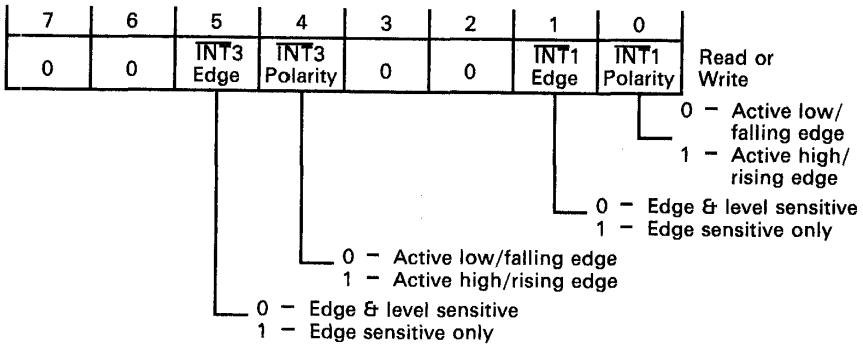


Figure 3-18. IOCNT2 - I/O Control Register 2 (TMS70Cx2 Only)

Thus, INTn going active is returned both as a latched *and* an edge-sensitive signal for the TMS70x0 and $\overline{\text{INT3}}$ of the TMS70Cx0 devices, while the TMS70Cx2 devices can choose sensitivity via IOCNT2. When a 1 is written to the INTn clear bit, the Pulse flip-flop is cleared. Writing a 0 to the INTn clear bit has no effect.

The INTn flag bit may be tested in software, regardless of whether the interrupt is enabled or disabled. For example, the following program statement waits for the active edge of the interrupt input on the $\overline{\text{INT1}}$ pin by testing INT1 flag:

```
WAIT    BTJOP    %>01,PO,WAIT    Wait for INT1-
*                                (INT1 flag = 1)
```

This allows external interrupt pins to be polled as inputs. Interrupt input pins have an advantage over the other general-purpose inputs if the input signal is a short pulse. The Pulse flip-flop of the interrupt input will always capture a pulse with a width of at least 1.25 × $t_{c(C)}$ cycles, allowing software to detect that the condition occurred, even after the pulse is gone.

Caution:

Due to the read/modify/write nature of the bit manipulation instructions (ANDP, ORP, and XORP), it is possible that the INTn flag bits in the IOCNT0 and IOCNT1 registers could be unintentionally cleared. To avoid these occurrences, *use the MOVP and the STA instructions when writing data to IOCNT0 and IOCNT1*.

Because the INTn flag and INTn clear bits are in the same bit positions, use caution when accessing these bits. For example, you may be able to use XORP to set INT1 enable without altering the state of the INT1 flag (XORP %>03,PO), as long as the INT1 flag does not change state during the instruction execution. However, if a short INT1 pulse sets the Pulse flip-flop between the read and write portions of the instruction execution, a 0 would be read from INT1 flag and a 1 would be written to INT1 clear to reclear INT1 flag. In this case, the $\overline{\text{INT1}}$ pulse would be undetected by the processor. This

same instruction would also affect the INT2 flag and INT3 flag in a similar manner as they are also located in the IOCNT0 register.

Immediately following $\overline{\text{RESET}}$, all interrupts are globally disabled because the I bit (interrupt enable) in the Status Register is reset to 0. Also, the IOCNT0 register is cleared. This clears the INTn enable bits, disabling $\overline{\text{INT1}}$, INT2, and $\overline{\text{INT3}}$ individually and putting the TMS7000 in Single-Chip mode. This does not affect the INTn flag bits from their previous condition before $\overline{\text{RESET}}$. On the TMS70x2 device, a 0 must be written by software to the INTn enable bits in the IOCNT1 register to ensure that INT4 and INT5 are also individually disabled following a $\overline{\text{RESET}}$.

3.6.4 Multiple Interrupt Servicing

When an interrupt is recognized, the global interrupt enable (I) Status Register bit is automatically cleared while the interrupt is serviced. This prevents all other interrupts from being recognized during the execution of the interrupt service routine. Once the service routine is completed by executing the RETI (Return from Interrupt) instruction, the old Status Register contents are popped from the stack. This returns the I bit back to 1, allowing any pending interrupts to be recognized.

An interrupt service routine can explicitly allow nested interrupts by executing the EINT instruction to directly set the I bit in the Status Register to a 1, thus permitting other interrupts to be recognized during service routine execution. When a nested interrupt service routine completes, it returns to the previous interrupt service routine when the RETI instruction is executed.

3.6.5 External Interrupt Servicing

The external interrupt interface consists of three discrete input lines that require no external synchronization: $\overline{\text{RESET}}$, $\overline{\text{INT1}}$, and $\overline{\text{INT3}}$.

TMS70x0 External interrupts on the TMS70x0 devices are high-impedance inputs that are both falling-edge and level sensitive, allowing multiple interrupts to be wire ORed onto one external interrupt pin.

TMS70x2, SE70P162, TMS7742 External interrupts on the TMS70x2 devices, the SE70P162 piggyback device, and the TMS7742 EPROM device are high-impedance inputs that are falling-edge sensitive only.

TMS70Cx0, SE70CP160 The external interrupt $\overline{\text{INT1}}$ on the TMS70Cx0 and SE70CP160 devices is a high-impedance falling-edge sensitive only interrupt, while $\overline{\text{INT3}}$ is a high-impedance falling-edge and level-sensitive interrupt.

TMS70Cx2, SE70CP162 The external interrupts on the TMS70Cx2 and SE70CP162 devices can be individually programmed as falling-edge sensitive only, falling-edge and level sensitive, rising-edge sensitive only, or rising-edge and level sensitive.

Certain safeguards should be observed for external interrupts that are both edge- and level-sensitive. The logical-OR of both the Pulse flip-flop output (Q1) and $\overline{\text{INTn}}$ (inverted INTn) affect the state of INTn flag, and can therefore

activate the interrupt (see Figure 3-15 on page 3-29). The Pulse flip-flop is automatically cleared when the CPU acknowledges the interrupt. However, as long as the INTn pin is low, the interrupt will remain active even when the Pulse flip-flop output is 0. This is how an external interrupt source is detected as a level signal. If INTn is active longer than the shortest path through the interrupt service routine, this same interrupt will be serviced again upon return from the service routine if no higher priority interrupts are active. In many applications this interrupt re-servicing is acceptable; however, in applications where this is a potential problem, the associated INTn enable bit must be disabled before exiting the interrupt service routine. Upon return from the service routine, INTn flag must be periodically software-polled to determine when INTn has gone inactive, and then INTn enable may be re-enabled. Note that devices with edge-sensitive only interrupts do not require the previously mentioned safeguards.

To prevent an interrupt signal from being detected as a level signal, the maximum pulse (time low) of the signal cannot exceed the following:

$$(16 + N) \times t_{c(C)}$$

where:

N = the total number of state clock cycles in the interrupt service routine, up to and including the EINT or RETI instruction
$t_{c(C)}$ = the internal state clock cycle period

This ensures that the INTn flag is cleared before the first possible instruction boundary in which the interrupt could be re-serviced. Note that this is not of any concern for $\overline{INT1}$ on the TMS70Cx0 devices and interrupts on the TMS70x2 devices, since they are edge-sensitive only, not level-sensitive.

Some applications may cause an incorrect interrupt vector to be accessed when using edge- and level-sensitive interrupts on the *TMS70x0 and TMS70Cx0 devices only*. This may happen when an INTn pulse goes inactive on the boundary condition when interrupts are being enabled. Two events are necessary for this to occur:

1) First, the Pulse flip-flop is cleared upon entry to the interrupt service routine; since the INTn pin is still active, INTn flag and INT active remain active.

2) Second, the INTn pin goes inactive on the boundary condition when interrupts are being enabled (RETI and EINT instructions or a write to IOCNT0 to enable interrupts).

When the INTn pin goes inactive, INTn flag becomes inactive and some time later INT active becomes inactive. This results in INT active being acknowledged by the CPU, but INTn flag becomes inactive before interrupt decode logic can determine which interrupt was pending. Note that INTn has already been serviced, so that re-servicing of the interrupt is not required. If this condition occurs, interrupt vector fetches from locations >FFF8 and >FFF9 (for $\overline{INT3}$) will occur for TMS70x0 and TMS70Cx0 devices. This situtation does not exist for edge-sensitive only interrupts (such as $\overline{INT1}$ on the TMS70Cx0 device and the interrupts on the TMS70x2 and TMS70Cx2 devices).

In applications where the system design cannot guarantee that the duration of the pulsed interrupt is outside this critical window, three system solutions should be considered.

- A system hardware solution uses an external D-type flip-flop or a one-shot in the interrupt path, providing a level interrupt which the TMS7000 would externally clear as part of the service routine.
- Prevent the re-servicing of the interrupt as described earlier by setting the associated INTn enable bit to 0 in the interrupt service routine.
- If only one external interrupt has the potential to cause this boundary condition, for TMS70x0 devices, this interrupt should be connected to $\overline{INT3}$ since the $\overline{INT3}$ vector is fetched when this problem occurs. This would result in $\overline{INT3}$ being re-serviced. For TMS70Cx2 devices with edge and level sensitivity enabled, a trap vector should be placed in location >FFF0 and >FFF1 which points to a RETI instruction. This will return the program to normal program flow if this condition occurs. For TMS70Cx0 devices, use $\overline{INT1}$ since this interrupt is only edge sensitive and will not exhibit the condition.

3.7 Programmable Timer/Event Counters

The programmable timer/event counters are 8-bit or 16-bit counters with a programmable, prescaled clock source. TMS70x0 and TMS70Cx0 devices contain one timer/event counter, TMS70x2 and TMS70Cx2 devices contain two timer/event counters and one timer. The data and control registers for these two timer/event counters are shown in Figure 3-19 through Figure 3-25 (pages 3-37–3-40).

- **Timer 1** is available on all TMS7000 devices.

  TMS70x0, TMS70Cx0, and TMS70x2

  Timer 1 is an 8-bit timer/event counter with a 5-bit programmable prescaler. It contains an 8-bit capture latch and is accessed through PF registers P2 and P3.

  TMS70Cx2

  Timer 1 is a 16-bit timer/event counter that contains a 5-bit programmable prescaler and a 16-bit capture latch. It is accessed through PF registers P12, P13, P14, and P15.

- **Timer 2** is available on the TMS70x2 and TMS70Cx2 devices.

  TMS70x2

  Timer 2 is an 8-bit timer/event counter with a 5-bit programmable prescaler. It is accessed through P18 and P19 of the Peripheral File.

  TMS70Cx2

  Timer 2 is a 16-bit timer/event counter that contains a 5-bit programmable prescaler and a 16-bit capture latch. It is accessed at PF registers P16, P17, P18, and P19.

- **Timer 3** is available on the **TMS70x2** and **TMS70Cx2** devices and can be used as an independent timer or as the clock source for the on-chip serial port. Because of this function, Timer 3 is described in more detail in Section 3.8, The Serial Port.

> **Note:**
>
> The contents of all registers associated with the timers are not affected by a hardware $\overline{\text{RESET}}$. These registers must be initialized by software.

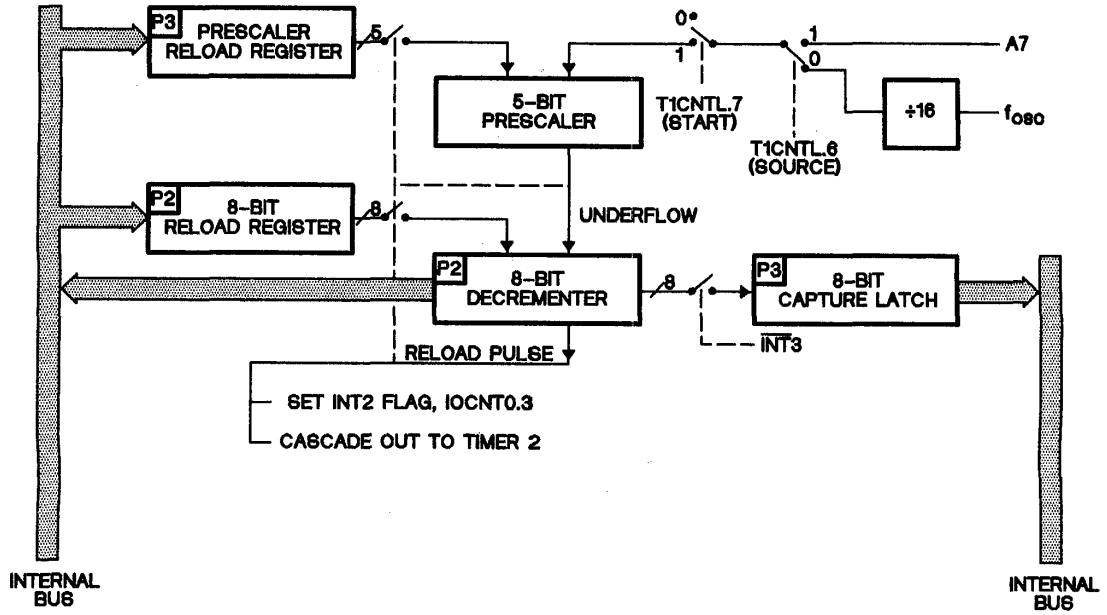


Figure 3-19. 8-Bit Programmable Timer/Event Counters - Timer 1 (TMS70x0, TMS70x2, and TMS70Cx0)

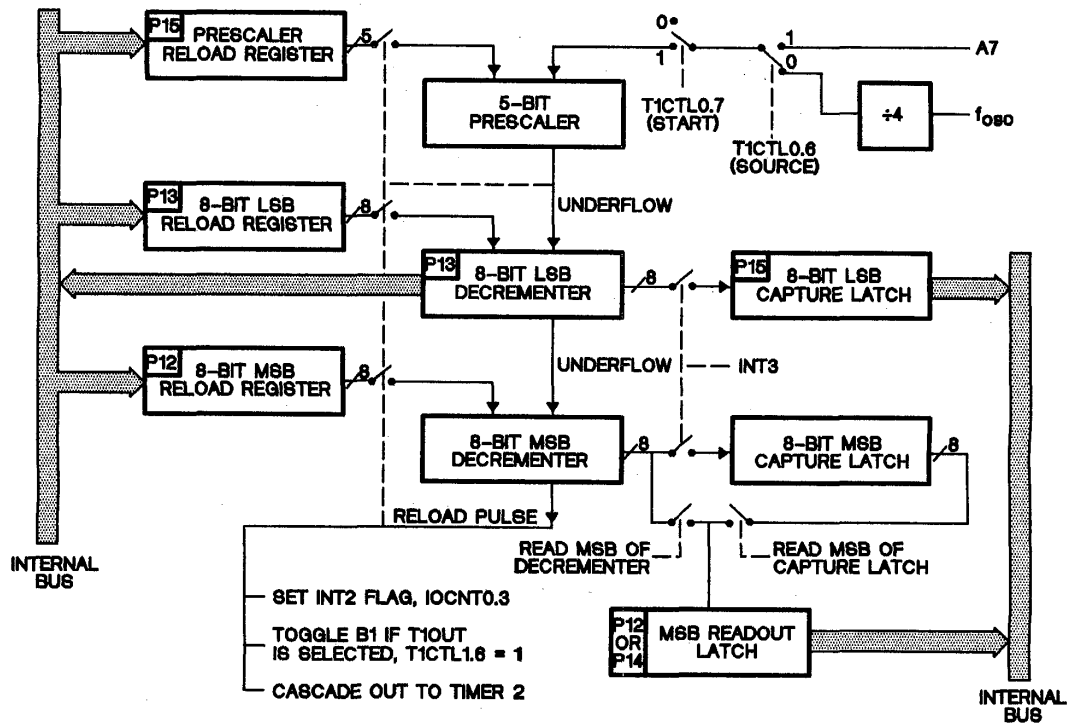


Figure 3-20. 16-Bit Programmable Timer/Event Counters - Timer 1 (TMS70Cx2)

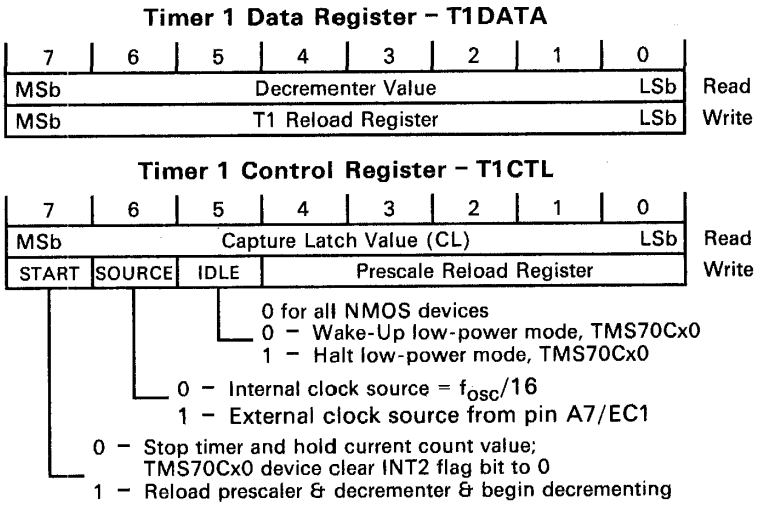


Figure 3-21. Timer 1 Data and Control Registers (TMS70x0, TMS70Cx0, and TMS70x2)

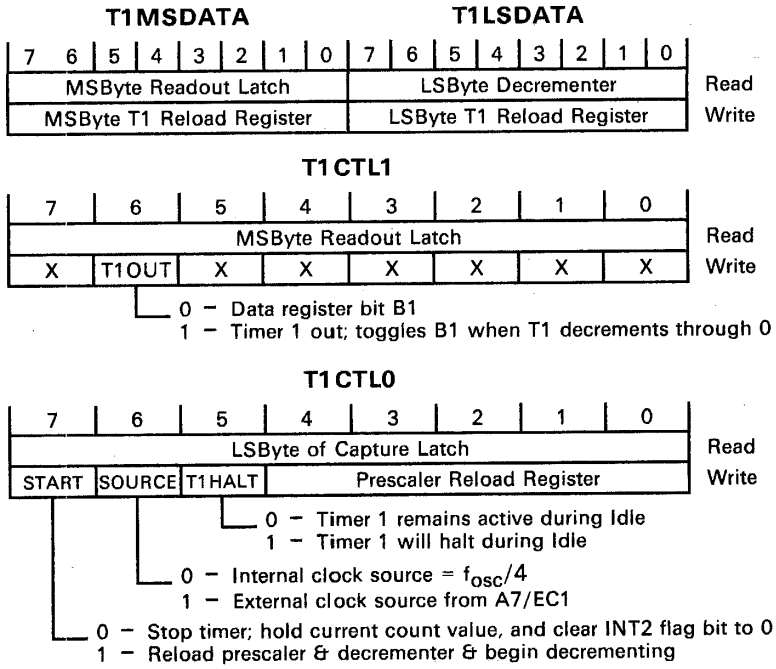


Figure 3-22. Timer 1 Data and Control Registers (TMS70Cx2)

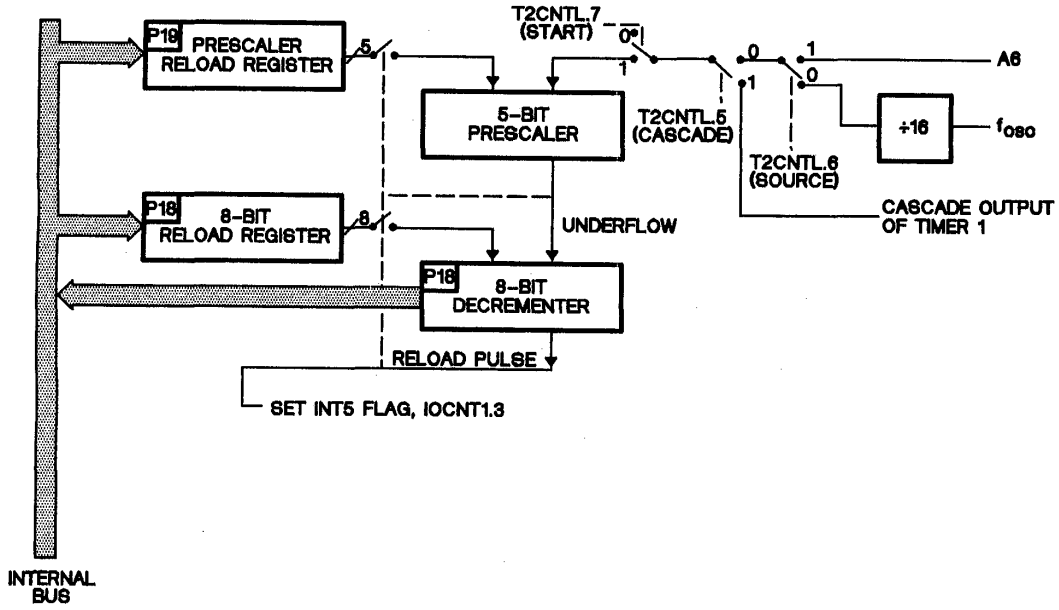


Figure 3-23. 8-Bit Programmable Timer/Event Counters - Timer 2 (TMS70x2)

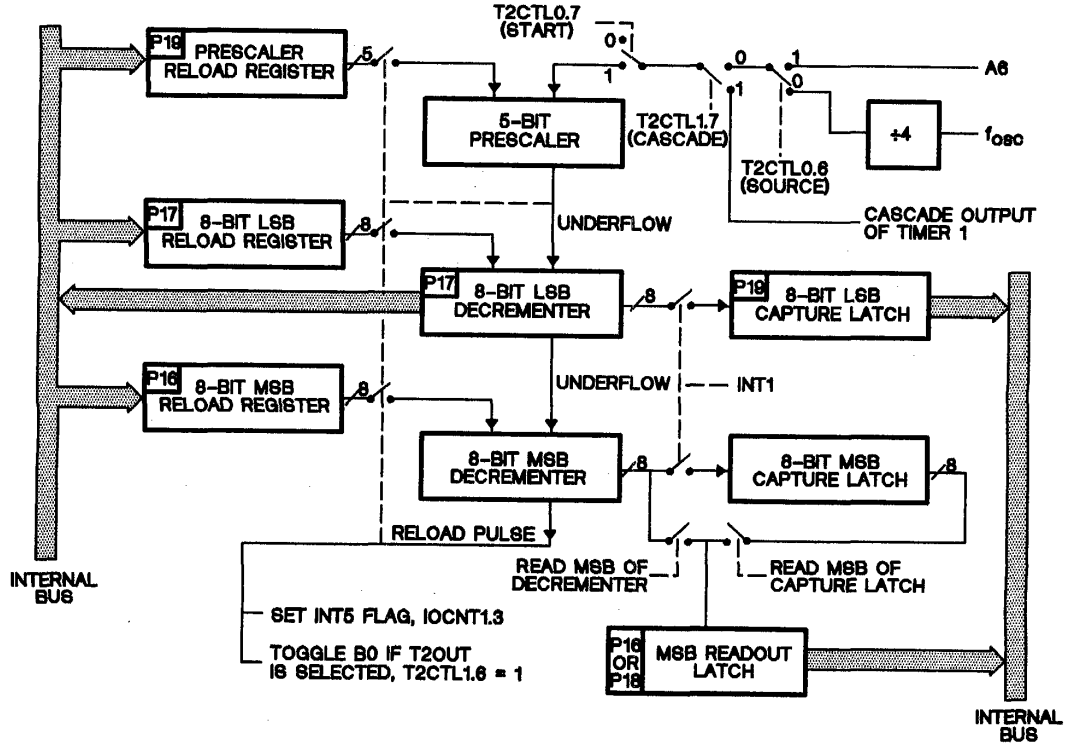


Figure 3-24. 16-Bit Programmable Timer/Event Counters - Timer 2 (TMS70Cx2)

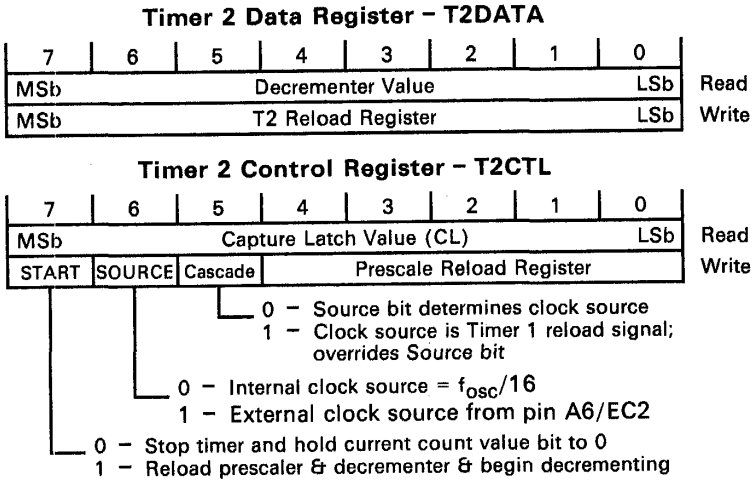


Figure 3-25. Timer 2 Data and Control Registers (TMS70x2)

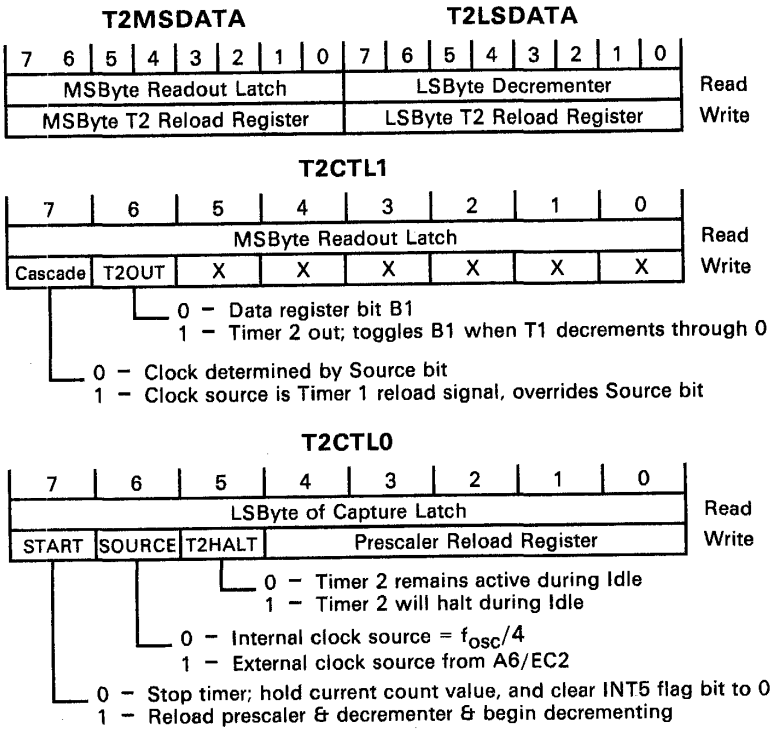


Figure 3-26. Timer 2 Data and Control Registers (TMS70Cx2)

3.7.1 Control Registers for Timer/Event Counters 1 and 2 (TMS70x0, TMS70Cx0, and TMS70x2 Devices)

The control bits and prescaling value of Timers 1 and 2 are determined by the timer control registers T1CTL (P3) and T2CTL (P19). These bits can only be written to the control registers and cannot be read by a program. When T1CTL is read, the capture-latch value associated with Timer 1 is returned. T2CTL is a write-only register and will return an irrelevant value when read. Since the control and prescale bits are write only, the read/modify/write instructions such as ANDP, ORP, and XORP should not be used. The following instructions should be used for timer control-bit manipulations.

```
MOVP    %>XX,Pn          STA   %>01XX
MOVP    A,Pn             STA   *Rn
MOVP    B,Pn             STA   >01XX(B)
```

where:

%>XX	=	Immediate 8-bit hexadecimal data value
>01XX	=	16-bit Peripheral-File hexadecimal address
A	=	Register A
B	=	Register B
Pn	=	Peripheral-File register number
Rn	=	General-purpose register pair number

The same instructions are required for writing to the timer data registers, T1DATA and T2DATA, and other write-only registers.

3.7.2 Control Registers for Timer/Event Counters 1 and 2 (TMS70Cx2 Devices)

The control bits and prescaling value of Timers 1 and 2 of the TMS70Cx2 devices are determined by the timer control registers T1CTL0 (P15), T1CTL1 (P14), T2CTL0 (P19), and T2CTL1 (P18). Data can only be written to these control registers, and cannot be read back by a program. When Timer 1 control register T1CTL0 is read, the least significant (LS) byte of the capture latch value associated with Timer 1 is returned. When T1CTL1 is read, the most significant (MS) byte of the Timer 1 readout latch is returned. When T2CTL0 is read, the least significant (LS) byte of the Timer 2 capture latch is returned. When T2CTL1 is read, the most significant (MS) byte of the Timer 2 readout latch is returned. Since the control and prescale bits are write only, the read/modify/write instructions such as ANDP, ORP, and XORP should not be used. The following instructions should be used for timer control-bit manipulations.

```
MOVP    %>XX,Pn          STA   %>01XX
MOVP    A,Pn             STA   *Rn
MOVP    B,Pn             STA   >01XX(B)
```

where:

%XX	=	Immediate 8-bit hexadecimal data value
>01XX	=	16-bit Peripheral-File hexadecimal address
A	=	Register A
B	=	Register B
Pn	=	Peripheral-File register number
Rn	=	General-purpose register pair number

The same instructions are required for writing to the timer data registers, T1LSDATA, T1MSDATA, T2LSDATA, T2MSDATA, and other write-only registers.

3.7.3 Timer Start/Stop (Bit 7) and Capture Latch

Bit 7 of the timer control registers contain a start/stop bit for the timer/event counters.

Bit 7 = 0 A start bit of **0** disables or freezes the timer chain at the current count value.

Bit 7 = 1 A start bit of **1**, regardless of whether the bit was a 0 or a 1 before, loads the prescaler and counter decrementers with the corresponding reload register values, and the timer/event counter operation begins.

3.7.3.1 Timer 1 Capture Latch (TMS70x0, TMS70Cx0, and TMS70x2 Devices)

The Timer 1 8-bit capture latch can be accessed by reading the Timer 1 control register T1CTL (P3). T1CTL will contain the "captured" current Timer 1 value whenever $\overline{\text{INT3}}$ is triggered even if $\overline{\text{INT3}}$ is disabled. Please note that when $\overline{\text{INT3}}$ is used to exit a low-power mode on the TMS70Cx0 CMOS parts, the capture latch may store an indeterminate value. This is due to the logic design of the CMOS devices. Since the value in the capture latch may not be valid when leaving either of the low-power modes via $\overline{\text{INT3}}$, it is recommended that the capture latch not be used in this situation.

3.7.3.2 Timer 1 and Timer 2 Capture Latches (TMS70Cx2 Devices)

The TMS70Cx2 contains two 16-bit capture latches, one each for Timer 1 and Timer 2. The Timer 1 16-bit capture latch can be accessed by reading the Timer 1 control registers T1CTL0 (P15) and T1CTL1 (P14). The Timer 2 16-bit capture latch can be accessed by reading the Timer 2 control registers T2CTL0 (P19) and T2CTL1 (P18). The capture latch values for Timer 1 and Timer 2 are loaded on the active edges of $\overline{\text{INT3}}$ and $\overline{\text{INT1}}$, respectively, whether the interrupts are enabled or not. Both capture latches are disabled during the IDLE instruction when their corresponding HALT bits are 1.

Reading the Timer 1 control register T1CTL1 or the Timer 2 control register T2CTL will return the value of the MSB readout latch of the respective timer. This latch is shared between MSB of the timer latch and the MSB of the capture latch. It allows the complete 16-bit value of the timer latch or the capture latch to be sampled at one moment. The LSB must be read first, which causes the MSB to be simultaneously loaded into the readout latch. This latch physically exists in only one location for each timer; however, each latch can be read from two different locations. Timer 1 MSB readout latch can be read from T1MSDATA (P12) or T1CTL1 (P14). Timer 2 MSB readout latch can be read from T2MSDATA (P16) or T2CTL1 (P18).

Reading the LSB of the decrementer or capture latch will update the contents of the readout latch. In order to correctly read the entire 16-bit value of the decrementer or capture latch, the LSB must be read first, which will load the MSB readout latch. The MSB readout latch must be read and stored before reading the LSB of either the decrementer or capture latch. The order of 16-bit read operations should be:

Timer 1: *Decrementer*: Read P13 then P12 ***or*** read P13 then P14
Capture Latch: Read P15 then P12 ***or*** read P15 then P14

Timer 2: *Decrementer*: Read P17 then P16 ***or*** read P17 then P18
Capture Latch: Read P19 then P16 ***or*** read P19 then P18

3.7.4 Clock Source Control (Bit 6)

For the **TMS70x0, TMS70Cx0,** and **TMS70x2** devices, bit 6 (SOURCE) of T1CTL and T2CTL selects the Timer 1 and Timer 2 clock sources, respectively.

For the **TMS70Cx2** devices, bit 6 (SOURCE) of T1CTL0 and T2CTL0 selects the Timer 1 and Timer 2 clock sources, respectively.

Bit 6 = 0 A source bit of **0** selects the internally generated clock and places the timer/event counter in the Realtime Clock mode using the internal clock source. Each positive transition of the timer clock signal decrements the count chain. Realtime Clock mode allows a program to periodically interrupt and call a service routine, such as a display refresh, by simply setting the prescale reload register and the timer reload register so the routine is called at the desired frequency.

Bit 6 = 1 A source bit of **1** selects the external clock source and places the timer/event counter in the Event-Counter mode. In this mode, each positive transition at the Port A event counter pins decrements the count chain (when the prescaler is decremented to zero, it is reloaded with the prescaler reload register value and the counter is decremented by one).

Summary for all TMS7000 devices:

	Event Counter Input Pin	Interrupt Level
Timer 1	Pin A7/EC1	INT2
Timer 2	Pin A6/EC2	INT5

The Event-Counter mode allows INT2 and INT5 to function as positive edge-triggered external interrupts by loading a start value of 1 into both the prescaler and timer reload register. A positive transition on A7/EC1 or A6/EC2 decrements the corresponding timer through zero and generates an INT2 or INT5. Event-Counter mode can also be used as an externally provided real-time clock if an external clock is input on the I/O pin. The minimum clock period on pins A7/EC1 or A6/EC2 must not be less than $f_{osc}/16$ for TMS70x0, TMS70x2, and TMS70Cx0 devices, or $f_{osc}/4$ for TMS70Cx2 devices. The minimum pulse width of the external signal must not be less than 1.25 state clock cycles [$1.25 \times t_{c(C)}$] to be properly detected by the device.

3.7.5 Idle/Timer Halt Bit (Bit 5)

The function of the Idle bit (bit 5) in the timer control registers varies depending on the device type.

- **TMS70x0** and **TMS70x2**

  Bit 5 is not used on any of the TMS7000 NMOS devices.

- **TMS70Cx0**

  Bit 5 of T1CTL (P3) register is the IDLE bit. This bit selects either of two low-power modes on these devices when the IDLE instruction is executed. (See Section 3.4.2 about CMOS low-power modes.)

  Bit 5 = 0 Wake-Up low-power mode
  Bit 5 = 1 Halt low-power mode

- **TMS70Cx2**

  Bit 5 of the T1CTL0 (P15) and T2CTL0 (P19) registers acts as a timer-halt bit. This bit selects either of two timer operational modes when the IDLE instruction is executed.

  Bit 5 = 0 Wake-Up low-power mode
  Bit 5 = 1 Halt timer mode

3.7.6 Cascading Timers

The TMS70x2 and TMS70Cx2 devices can have their timers cascaded together to form one large timer. The external clock input for Timer 2 is the Port A pin A6/EC2. This pin can also function as the serial clock I/O line (SCLK) for the serial port on the TMS70x2 devices (see Section 3.8, The Serial Port). Several arrangements are possible with Timer 2 in relation to Timer 3 and the serial port because of this:

- *Both SCLK and Timer 2 clock internal*: the Timer 3 output divided by 2 is driven out of the A6/EC2 pin and Timer 2 is internally clocked by 8 × $t_{c(C)}$.
- *SCLK internal and Timer 2 clock external*: the Timer 3 output divided by 2 is driven out of the A6/EC2 pin and this pin drives the Timer 2 clock. In this mode, Timer 3 and Timer 2 are cascaded together, with Timer 3 driving Timer 2. This is done by setting the Cascade bit to 0 and the Timer 2 source bit to 1. Timer 2 can then be cascaded under software control to either Timer 1 or Timer 3.
- *SCLK external and Timer 2 clock internal*: the input signal drives the serial port clock and Timer 2 is internally clocked by 8 × $t_{c(C)}$.
- *Both SCLK and Timer 2 clock external*: the input signal drives both the serial port clock and Timer 2.

The differences between the TMS70x2 and TMS70Cx2 Cascade bits are explained below.

- **TMS70x2**

  Bit 5 of the T2CTL (P19) register in the TMS70x2 devices is the Cascade bit. This bit is used in conjunction with T2CTL (P19) Source (bit 6) to determine the Timer 2 clock source.

  Bit 5 = 0 A Cascade bit of **0** allows bit 6 (source) to determine the clock source.

  Bit 5 = 1 A Cascade bit of **1** selects the output generated by the Timer 1 reload pulse as the clock input to the prescaler of Timer 2. The Cascade bit overrides the Source bit; that is, if the Cascade bit is 1, the Source bit of Timer 2 has no effect.

- **TMS70Cx2**

  Bit 7 of the T2CTL1 (P18) register is the Cascade bit. This bit is used in conjunction with the T2CTL0 (P19) Source (bit 6) to determine the Timer 2 clock source.

  Bit 7 = 0 A Cascade bit of **0** allows bit 6 of T2CTL0 to determine the clock source.

  Bit 7 = 1 A Cascade bit of **1** selects the output generated by the Timer 1 reload pulse as the clock input to the prescaler of Timer 2. The Cascade bit overrides the Source bit; that is, if the Cascade bit is 1, the Source bit of Timer 2 has no effect.

  Note that on the TMS70Cx2 devices, the Timer 2 output (T2OUT) cannot be used if Timer 1 and Timer 2 are cascaded together.

3.7.7 Timer and Prescaler Operation

The timer clock, whether internal or external, is prescaled by a 5-bit modulo-N counter. The prescaling value is determined by the least significant five bits of the timer control register. The timers decrement and an underflow occurs on the transition from 0 to >FF. Thus, a prescale value of >7 will produce an $f_{osc}/128$ clock input into the timer for a TMS70x0 device with a timer clock source of $f_{osc}/16$.

- **TMS70x0, TMS70Cx0,** and **TMS70x2**

  Timer 1 Bits 0–4 of Timer 1 control register T1CTL comprise the Timer 1 prescale reload register value.

  Timer 2 Bits 0–4 of Timer 2 control register T2CTL comprise the Timer 2 prescale reload register value.

- **TMS70Cx2**

  Timer 1 Bits 0–4 of Timer 1 control register T1CTL0 comprise the Timer 1 prescale reload register value.

  Timer 2 Bits 0–4 of Timer 2 control register T2CTL0 comprise the Timer 2 prescale reload register value.

These steps occur during timer operation:

1) Upon starting the timer, the prescaler and timer are loaded from the prescaler reload register and timer reload register, respectively.

2) Each pulse decrements the prescaler by one.

3) When the prescaler countdown decrements through zero, the timer is decremented by one. After the timer is decremented,

   If timer ≠ 0 Reload prescaler and go back to step 2.

   If timer = 0 When both the timer and the prescaler decrement through zero together, an interrupt occurs. An INT2 for Timer 1 (INT5 for Timer 2) is momentarily pulsed when both the prescaler and counter decrement past the zero value together. This sets the INT2 or INT5 Pulse flip-flop, as described in Section 3.6.2, Interrupt Operation.

4) The 5-bit prescaler and decrementer are then immediately reloaded with the contents of the prescale reload register and the timer reload register, and the timer will start decrementing with the new reload register values.

- **TMS70x0, TMS70Cx0,** and **TMS70x2**

  The 8-bit timer reload register is loaded through the Timer 1 data register T1DATA (P2) for Timer 1 and the Timer 2 data register T2DATA (P18) for Timer 2. This value is write only. When read, T1DATA and T2DATA contain the current value of the 8-bit decrementer for Timer 1 and Timer 2, respectively, and not the timer reload register value. For this reason, the read/modify/write I/O instructions should not be used to alter the data value in the timer reload register. When read, the T1CTL contains the capture latch value for Timer 1.

- **TMS70Cx2**

  The 16-bit timer reload registers are loaded through the Timer 1 data registers T1LSDATA (P13) and T1MSDATA (P12), and the Timer 2 data registers T2LSDATA (P17) and T2MSDATA (P16). This value is write only. When read, T1LSDATA and T2LSDATA return the current value of the LSB of the Timer 1 and Timer 2 decrementers, respectively, and not the LSB timer reload register value. For this reason, the read/modify/write I/O instructions should not be used to alter the data value in the timer reload registers. T1MSDATA and T2MSDATA will return the value of the MSB readout latch for Timers 1 and 2, respectively. To read the **Timer 1** capture latch, first read T1CTL0 (P15) to obtain the LSB, then read T1CTL1 (P14) to obtain the MSB. To read the **Timer 2** capture latch, first read T2CTL0 (P19) to obtain the LSB, then read T2CTL1 (P18) to obtain the MSB.

3.7.8 Timer Interrupts

When the prescaler and decrementers pass through zero together, an interrupt flag (INTn flag) is set and the prescaler and counter decrementers are immediately and automatically reloaded with the corresponding reload register values. The interrupt levels generated by the timers are INT2 for Timer 1 and INT5 for Timer 2. The period between successive timer interrupts may be calculated by the following formula:

- **TMS70x0, TMS70Cx0,** and **TMS70x2**

$$t_{INT} = t_{CLK} \times (PR+1) \times (TR+1)$$

where:

t_{INT} = Period between timer interrupts
t_{CLK} = Period of the timer input clock which is $16/f_{osc}$ for Realtime Clock mode or the period of the external input pin for Event-Counter mode
PR = 5-bit prescaler reload register value
TR = 8-bit timer reload register value

At the falling edge of the $\overline{INT3}$ input, the Timer 1 counter value is loaded into the capture latch. This feature provides the capability to determine when an external event occurred relative to the current Timer 1 decrementer value.

- **TMS70Cx2**

$$t_{INT} = t_{CLK} \times (PR+1) \times (TR+1)$$

where:

t_{INT} = Period between timer interrupts
t_{CLK} = Period of the timer input clock which is $4/f_{osc}$ for Realtime Clock mode or the period of the external input pin for Event-Counter mode
PR = 5-bit prescaler reload register value
TR = 16-bit timer reload register (value written to the MSB and LSB timer reload registers)

On the TMS70Cx2 devices, the falling edge of the $\overline{INT3}$ input will cause the 16-bit decrementer value of Timer 1 to be loaded into the Timer 1 capture latch. Likewise, the falling edge of the $\overline{INT1}$ input will cause the 16-bit decrementer value of Timer 2 to be loaded into the Timer 2 capture latch. This feature provides the capability to determine when an external event occurred relative to the current timer/counter value.

3.7.9 Timer Output Function (TMS70Cx2 Devices)

Timer 1 and Timer 2 have a timer output function which allows the B1 and B0 outputs, respectively, to be toggled every time the timer decrements through zero. This function is enabled by the T1OUT and T2OUT bits (bit 6) in the timer control registers T1CTL1 and T2CTL1.

When operating in the timer output mode, the B0 and/or B1 output cannot be changed by writing to the Port B Data Register. Writing to the appropriate timer's Start bit will reload and start the timer, and will not toggle the output. The output will toggle only when the timer decrements through zero. The timer output feature is independent of INT2 and INT5; therefore, it will operate with INT2 and INT5 enabled or disabled. Also, if the timer is active during the IDLE instruction, the timer output feature will continue to operate.

Whenever the T2OUT or T1OUT bit is returned to 0, B0 or B1 will become an output-only pin, like B2. The vaslue in the B0 or B1 data register will be the last value output by the timer output function, to that B0 or B1 will not change as the T1OUT or T2OUT bit is returned to 0.

Whenever Port B is read, the value on the B0 pin will always be returned, so the current timer output value can be read by reading Port B.

The T1OUT and T2OUT bits are set to 0 by a reset, so the timer output function will not be enabled unless the user sets T1OUT or T2OUT to 1.

The Timer 2 output (T2OUT) cannot be used if Timer 1 and Timer 2 are cascaded together (Cascade bit of T2CTL1 set to 1).

3.8 Serial Port (TMS70x2 and TMS70Cx2 Devices Only)

The TMS70x2 and TMS70Cx2 devices contain a serial port, greatly enhancing their I/O and communications capabilities. Including a hardware serial port on chip saves ROM code and allows much higher transmission rates than could be achieved through software.

The full-duplex serial port consists of a receiver (RX), transmitter (TX), and a third timer called Timer 3 (T3). The functional operation of the serial port is configured through software initialization. A set of control words are first sent out to the serial port to initialize the desired communications format. These control words will determine the baud rate, character length, even/odd/off parity, number of stop bits, and so forth.

Figure 3-27 (page 3-50) illustrates the serial port functional blocks.

The serial port provides Universal Synchronous Asynchronous Receiver/-Transmitter (USART) communications:

- **Asynchronous mode**, discussed in Section 3.8.2.1 (page 3-63) interfaces with many standard devices such as terminals and printers using RS-232-C formats.
- **Isosynchronous mode**, discussed in Section 3.8.2.2 (page 3-64) permits very high transmission rates and requires a synchronizing clock signal between the receiver and transmitter.
- **Serial I/O mode**, discussed in Section 3.8.2.3 (page 3-64) can be used to expand I/O lines and to communicate with peripheral devices requiring a non-UART serial input such as A-to-D converters, display drivers, and shift registers.

The serial port also has two multiprocessor protocols, compatible with the Motorola 6801 and Intel 8051. These protocols allow efficient data transfer between multiple processors. They are implemented using isosynchronous or standard asynchronous formats.

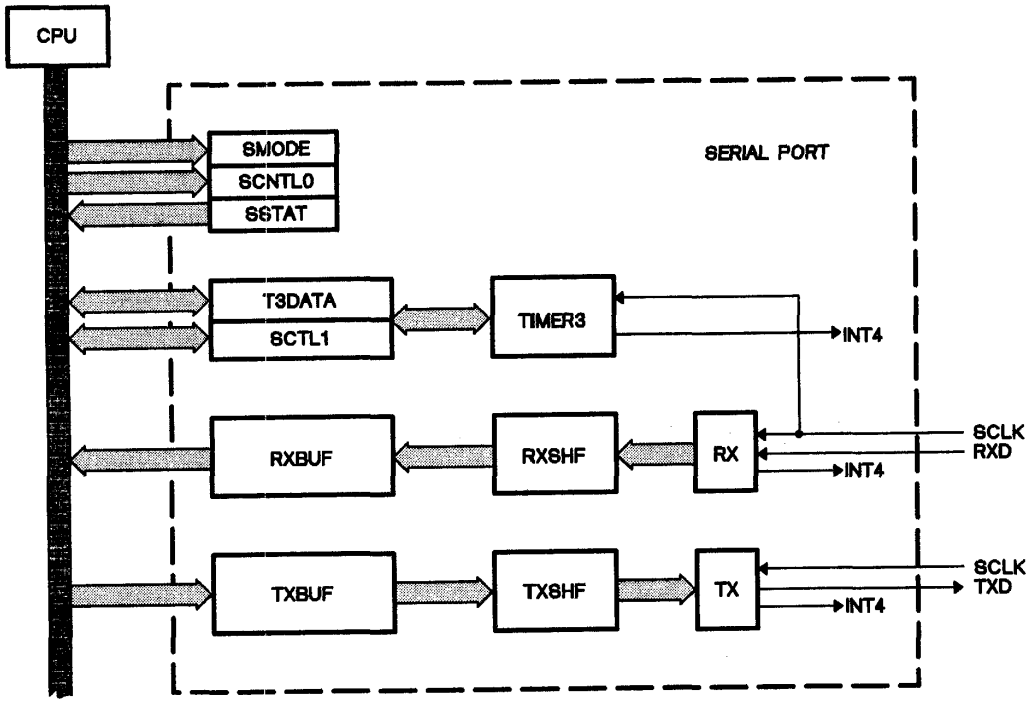


Figure 3-27. Serial Port Functional Blocks

3.8.1 Serial Port Registers

The serial port is controlled and accessed through registers in the Peripheral File. These registers are listed in Table 3-15. Figure 3-27 contains a block diagram of the serial port registers and functional blocks.

Table 3-15. Serial Port Control Registers

REGISTER		NAME	TYPE	FUNCTION
TMS70Cx2	TMS70x2			
P20	P17	SMODE	FIRST WRITE	Serial Port Mode
P21	P17	SCTL0	READ/WRITE†	Serial Port Control 0
P22	P17	SSTAT	READ	Serial Port Status
P23	P20	T3DATA	READ/WRITE	Timer 3 Data
P24	P21	SCTL1	READ/WRITE	Serial Port Control 1
P25	P22	RXBUF	READ	Receiver Buffer
P26	P23	TXBUF	WRITE	Transmission Buffer

† Write only for TMS70x2 devices

The serial mode register, **SMODE**, is the RX/TX control register that describes the character format and type of communication mode (Asynchronous, Isosynchronous, or Serial I/O).

The serial port control 0 register, **SCTL0**, is the RX/TX control register used to control the serial port functions, TX and RX enable, clearing of error flags, and S/W enable.

The serial port Status Register, **SSTAT**, is the read-only serial Status Register used to report the serial port status.

The **T3DATA** register is the read/write Timer 3 data register.

RXBUF is a read-only register containing data from RX. RXBUF is double-buffered with the internal shift register (**RXSHF**) so that the the CPU has at least a full frame to read the received data before RX can overwrite it with new data.

TXBUF is a write-only register from which TX takes the data it transmits. It is double-buffered with the TX shift register (**TXSHF**), so that the CPU has a full frame to write new data before TXBUF becomes empty.

The TXD and RXD lines use I/O pins B3/TXD and A5/RXD, respectively. This configuration allows the TXD and RXD pins to be used as I/O pins if desired. If serial port transmission is disabled, then TXD follows B3. If reception is disabled, then no receiver interrupts occur and A5 functions an an input pin on TMS70x2 devices and as a general-purpose I/O pin on TMS70Cx2 devices. The B3 I/O pin must be set to a 1 in order to enable the TXD pin.

3.8.1.1 Serial Mode Register (SMODE)

The SMODE register is the RX/TX control register that describes the character format and type of communication mode (Asynchronous, Isosynchronous, or Serial I/O).

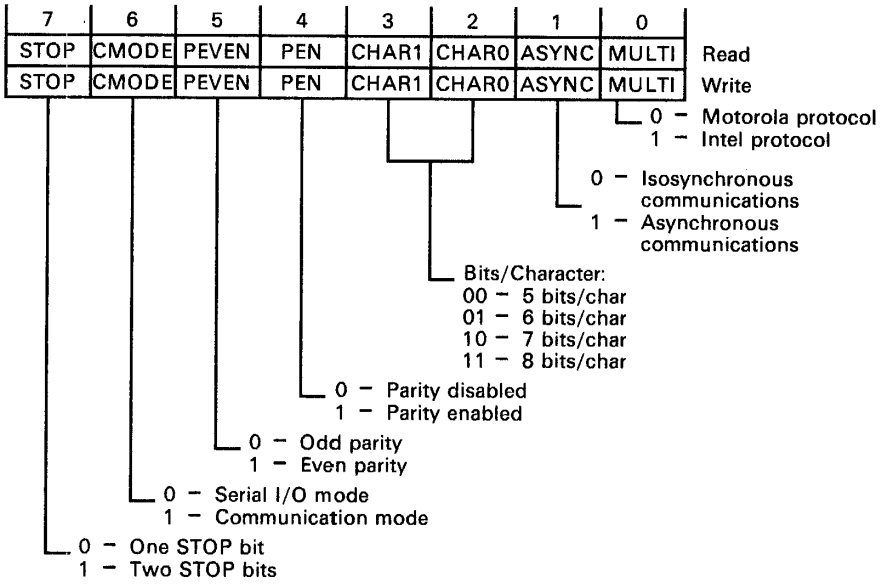


Figure 3-28. Serial Mode Register - SMODE

- **TMS70x2** (Write-only register)

  SMODE is accessed at Peripheral-File location P17 on the first write after a hardware or serial port reset. SMODE must be the first register written to in the serial port immediately following a reset. After the SMODE register is written to, it cannot be accessed again without first performing another reset operation. The first write operation to location P17 immediately following a reset accesses SMODE. All subsequent writes to P17 access the control register (SCTL0).

- **TMS70Cx2**

  SMODE is accessed anytime at Peripheral-File register P20.

Multiprocessor Mode (MULTI) Bit 0

There are two possible multiprocessor protocols, Motorola (Section 3.8.3.1) and Intel (Section 3.8.3.2).

0 - Selects the Motorola protocol.
1 - Selects the Intel protocol.

The Motorola mode is typically used for normal communications since the Intel mode adds an extra bit to the frame. The Motorola mode does not add this

extra bit and is compatible with RS-232-type communications. Multiprocessor communication is different from the other communication modes because it uses Wake-Up and Sleep functions.

Communications Mode (ASYNC) Bit 1

This bit determines the serial port communication mode.

0 – Selects Isosynchronous mode (Section 3.8.2.2). In this mode, the bit period is equal to the SCLK period; bits are read on a single value basis.
1 – Selects Asynchronous mode (Section 3.8.2.1). In this mode the bit period is 8 times the SCLK period and bits are read on a two out of three majority basis.

Number of Bits per Character (CHAR1, CHAR2) Bits 2,3

Character length is programmable to 5, 6, 7 or 8 bits. Characters less than 8 bits are right-justified in buffers RXBUF and TXBUF and padded with leading zeros. The unused leading bits in TXBUF may be written as don't *cares*. The RXBUF and TXBUF register formats are illustrated in Figure 3-33 and Figure 3-34.

Parity Enable (PEN) Bit 4

If parity is disabled (PEN set to 0), then no parity bit is generated during transmission or expected during reception. A received parity bit is not transferred to RXBUF with the received data because it is not considered one of the data bits when programming the character field. On the TMS70Cx2 devices, the parity error flag may be set even though parity is disabled.

Parity Even (PEVEN) Bit 5

If PEN is set, then this bit defines odd or even parity according to an odd or even number of 1 bits in both transmitted and received characters.

0 – Sets odd parity.
1 – Sets even parity.

Serial I/O or Communication Mode (CMODE) Bit 6

This bit determines whether the serial port operates in Serial I/O mode or one of the communication modes.

0 – Puts the serial port in Serial I/O mode which allows easy I/O expansion by using external shift registers.
1 – Selects communication mode. The ASYNC bit (bit 1) determines whether the serial port is in Asynchronous or Isosynchronous mode. The MULTI bit (bit 0) determines if the communication uses the Motorola or Intel protocol.

Number of Stop Bits (STOP) Bit 7

This bit determines the number of stop bits sent when the serial port is in Isosynchronous or Asynchronous mode.

0 – Selects one stop bit.
1 – Selects two stop bits. The receiver checks for one stop bit only.

3.8.1.2 Serial Control Register 0 (SCTL0)

The SCTL0 register is the RX/TX control register used to control the serial port functions, TX and RX enable, clearing of error flags, and S/W reset. SCTL0 is cleared by a hardware or software reset.

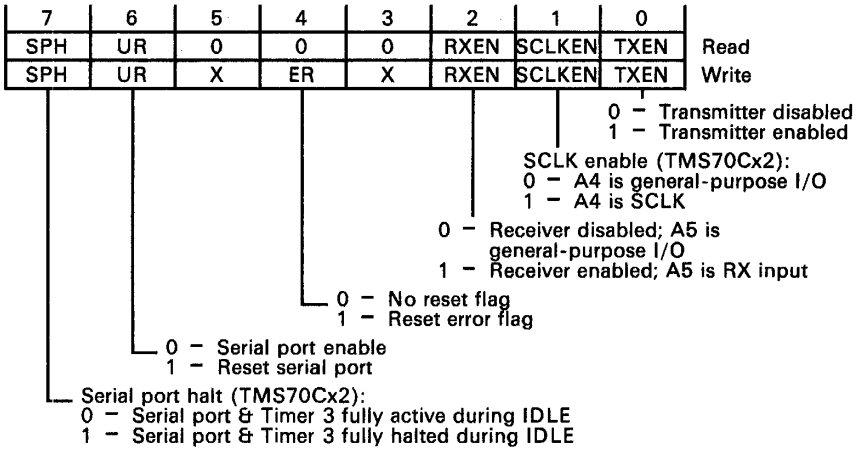


Figure 3-29. Serial Control 0 Register - SCTL0

- **TMS70x2** (Write-only register)

  SCTL0 is a write-only register, accessed at Peripheral-File location P17 on the second and subsequent write operations after a hardware or serial port reset. After a hardware or serial port reset, SMODE must be written to before the SCTL0 register can be accessed, since the SMODE and SCTL0 registers are accessed through the same location.

  Use the following procedure if you do not know if P17 is SCTL0 or SMODE. Writing a 0 to P17 puts this register at SCTL0, but the first write operation might have changed the SMODE value so it needs to be re-initialized.

```
SMODE   EQU   P17
SCTL0   EQU   P17
*
UARTRS  MOVP  %0,P17            P17 in an unknown state,
*                               ensure being at SCTL0
        MOVP  %>40,SCTL0        Reset the serial port
        MOVP  %>XX,SMODE        Set SMODE to proper
*                               values
        MOVP  %?X0XXXXXX,SCTL0  Clear the reset bit
*                               (?=binary)
*                               P17 is now SCTL0
```

- **TMS70Cx2**

  SCTL0 is a read/write register, and can be accessed anytime at Peripheral-File location P21.

Transmit Enable (TXEN) Bit 0

Data transmission through TXD (pin B3) cannot take place unless TXEN is set to 1.

When TXEN is reset to 0, transmission does not halt until all the data previously written to TXBUF is sent. Thereafter, B3/TXD can be used as general-purpose output. TXEN is set to 0 by a hardware or software reset.

In Isosynchronous mode, if an internally generated SCLK is used, the SCLK output at pin A6 (TMS70x2) or A4 (TMS70Cx2) is enabled. When the entire frame is transmitted, TX disables SCLK and sets TXRDY and INT4 flag to a 1, and TXEN to 0. TXEN has no direct effect on TXRDY or INT4 flag in this mode.

Serial Clock Enable (SCLKEN) Bit 1 - TMS70Cx2 devices only

This bit determines if the A4/SCLK pin will be used as general-purpose I/O (bit 1 = 0), or as the serial clock SCLK pin (bit 1 = 1).

Receive Enable (RXEN) Bit 2

In the **communication modes** (Asynchronous and Isosynchronous):

0 - Prevents received characters from being transferred into RXBUF, and no RXRDY interrupt is generated. However, the receiver shift register (RXSHF) continues to assemble characters. Thus, if RXEN is set during character reception, the complete character will be transferred into RXBUF.
1 - Enables RX (receiver) to set INT4 flag and enable RXRDY.

In **Serial I/O mode**:

0 - The UR bit sets RXEN to 0.
1 - Enables RX operation.

In Isosynchronous mode, if an internally generated SCLK is used, the SCLK output at pin A6 (TMS70x2) or A4 (TMS70Cx2) is enabled. When the entire frame is received, RX disables SCLK and sets RXRDY and INT4 flag to a 1, and RXEN to 0. RXEN has no direct effect on RXRDY or INT4 flag in this mode.

Error Reset (ER) Bit 4

The error reset bit is used to reset any error flags during serial port operation.

0 - No error flags are affected.
1 - Clears all three error flags in the SSTAT register (PE, OE, FE).

Software UART Reset (UR) Bit 6

Writing a 1 to this bit puts the serial port in the reset condition, enabling the SMODE register for initialization. SCLK (pin A6 on TMS70x2 devices, pin A4 on TMS70Cx2 devices) is put in the high-impedance input state. The TXD signal is held at 1 so the B3 pin may be used as a general-purpose output line. On TMS70Cx2 devices, the A5/RXD signal becomes a general-purpose I/O line; on TMS70x2 devices, it becomes an input.

Until a 0 is written to UR, all affected logic is held in the reset state. UR must be set to 0 before the CPU can write a 1 to CLK and output SCLK on Port A. UR is set to 1 by hardware $\overline{\text{RESET}}$. The UART reset affects only the items above; it is not a general device reset like the $\overline{\text{RESET}}$ pin.

Serial Port Halt Enable (SPH) Bit 7 - TMS70Cx2 devices only

This bit determines if the serial port and Timer 3 will be active or not during an IDLE instruction.

0 - Serial port and Timer 3 will be fully active during an IDLE instruction.
1 - Serial port and Timer 3 will be halted during an IDLE instruction.

3.8.1.3 Serial Port Status Register (SSTAT)

SSTAT is the read-only serial port Status Register. Bits 0, 1, and 6 of this register are cleared by a hardware or software reset.

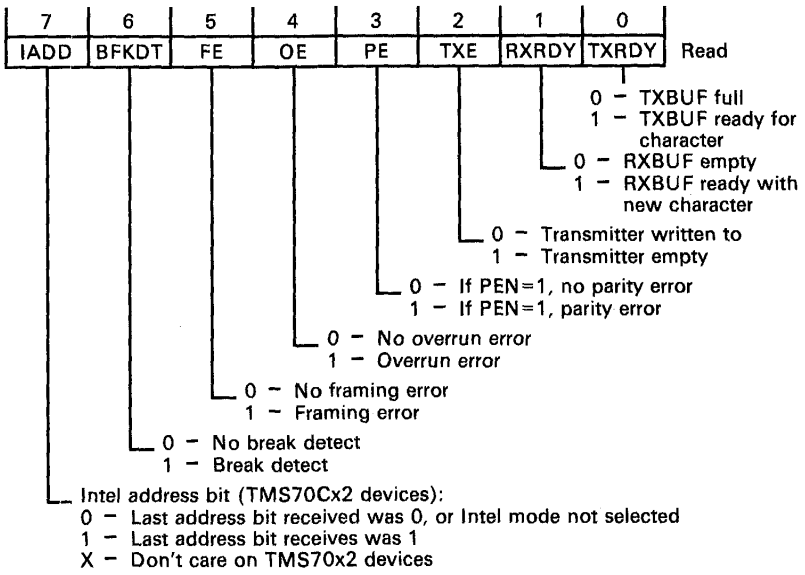


Figure 3-30. Serial Port Status Register - SSTAT

- **TMS70x2**

  The SSTAT register is accessed anytime by reading Peripheral-File location P17.

- **TMS70Cx2**

  The SSTAT register is accessed anytime by reading Peripheral-File location P22.

Transmitter Ready (TXRDY) Bit 0

The TXRDY bit is set by the transmitter to indicate that TXBUF is ready to receive another character. It is automatically reset when a character is loaded. If the serial port interrupt (INT4) is enabled, it is issued at the same time the TXRDY bit is set. Resetting the UART sets TXRDY to 1.

Receiver Ready (RXRDY) Bit 1

This bit is set by the receiver to indicate that RXBUF is ready with a new character. It is automatically reset when the character is read out. If the serial port interrupt (INT4) is enabled, it is set at the same time that the RXRDY bit is set. Resetting the UART sets RXRDY to 1.

Transmitter Empty (TXE) Bit 2

The TXE bit is set to 1 when the transmitter shift register (TXSHF) and TXBUF (shown in Figure 3-34, page 3-60) are empty. It is reset to 0 when the TXBUF is written to. Resetting the UART sets TXE to 1.

Parity Error (PE) Bit 3

PE is set when a character is received with a mismatch between the number of 1s and its parity bit. This bit is reset by the ER bit in SCTL0. Disabling the parity does not disable this flag, so this flag may be set even when the parity is disabled.

Overrun Error (OE) Bit 4

OE is set when a character is transferred into RXBUF (shown in Figure 3-34) before the previous character has been read out. The previous character is overwritten and lost. OE is reset by the ER bit in SCTL0.

Framing Error (FE) Bit 5

FE is set when a character is received with a 0 stop bit, meaning that synchronization with the start bit has been lost and the character is incorrectly framed. The ER bit in SCTL0 resets FE.

Break Detect (BRKDT) Bit 6

The BRKDT bit shows that a break condition has occurred. BRKDT is set if the RXD line remains continuously low for 10 bits or more, starting from the end of a frame (stop bit). When the break ends, BRKDT is set to a 0 immediately. In the Serial I/O mode, BRKDT remains a 0. UR (SCTL0 bit 6) sets BRKDT to 0. A break is generated by setting Port B bit 3 low. Setting B3 high again resumes TXD operation.

The TXD and RXD lines are multiplexed on I/O lines B3 and A5, respectively. This configuration allows the TXD and RXD pins to be used as I/O pins if desired. If transmission is disabled, then TXD follows B3. If reception is disabled, then no receiver interrupts occur and A5 is an input bit.

Intel Address Bit (IADD) Bit 7 - TMS70Cx2 devices only

This bit shows the last data bit received when using the Intel protocol.

0 - Last address bit received was 0, or Intel mode was not selected.
1 - Last address bit received was 1.
X - Don't care on TMS70x2 devices.

3.8.1.4 Serial Control Register 1 (SCTL1)

The SCTL1 register is the read/write serial control register 1. It is used to control the Timer 3 start/stop function, the source of SCLK, multiprocessor communication, Timer 3 interrupt, and the Timer 3 prescaler value.

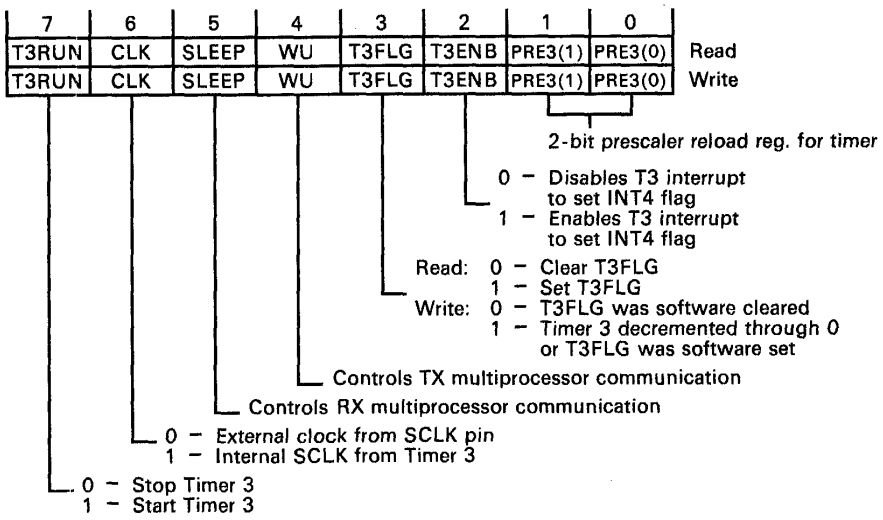


Figure 3-31. Serial Port Control 1 Register - SCTL1

- **TMS70x2**

  The SCTL1 register is accessed at Peripheral-File location P21.

- **TMS70Cx2**

  The SCTL1 register is accessed at Peripheral-File location P24.

Timer 3 Prescale Reload Register (PRE3(1), PRE3(0)) Bits 0,1
These are the prescale bits for Timer 3. The internal clock input to Timer 3 is either f_{osc}/4, /8, /16, or /32, depending on how the prescale bits are set. The Timer 3 output divided by 2 is the actual baud rate for the Isosynchronous mode; divided by 8, it is the baud rate for for the Asynchronous mode.

Timer 3 Interrupt Enable (T3ENB) Bit 2
When T3ENB is set to 1, Timer 3 sets INT4FLG to 1 when it sets T3FLG to 1. T3ENB is reset to 0 by a hardware reset, but not by UR (SCTL0 bit 6). This allows Timer 3 to operate independently of the serial port.

Timer 3 Interrupt Flag (T3FLG) Bit 3
The T3FLG bit is set to 1 when both the Timer 3 prescaler and Timer 3 decrement through zero together. T3FLG indicates that Timer 3 caused the serial port interrupt. T3FLG must be cleared by software in the T3 interrupt service routine, since it is not cleared when the INT4 vector is fetched by the CPU. T3FLG is reset to 0 by a hardware reset, but not by UR (SCTL0 bit 6). This allows Timer 3 to operate independently of the serial port.

Wake-Up (WU) Bit 4

The WU bit controls the TX features of the multiprocessor communication modes (see Section 3.8.2.2 and Section 3.8.2.1). Resetting th UART sets WU to 0; it cannot be set again until UR is cleared.

Sleep (SLEEP) Bit 5

The SLEEP bit controls the RX features of the multiprocessor modes (See Section 3.8.2.2 and Section 3.8.2.1). Resetting the UART sets SLEEP to 0.

Serial Clock Source (CLK) Bit 6

The CLK bit determines the SCLK source. Resetting the UART sets CLK to 0; it cannot be set again until UR is cleared.

0 - Selects an external SCLK, which is input on the high-impedance A6/SCLK line on the TMS70x2 devices, and pin A4/SCLK on the TMS70Cx2 devices.
1 - Selects an internal SCLK, derived from Timer 3. This signal is output on the low impedance SCLK line.

Timer 3 Start (START) Bit 7

This bit controls the starting and stopping of Timer 3.

0 - Stops Timer 3.
1 - Loads Timer 3 with the Timer 3 data value and then starts the timer. Writing a 1 will have no effect if Timer 3 is already active.

3.8.1.5 Timer 3 Data Register

The Timer 3 data register, T3DATA, is a read/write register used to store the countdown value of Timer 3.

7	6	5	4	3	2	1	0	
MSb			Current Timer Value				LSb	Read
MSb			Timer Reload Register				LSb	Write

Figure 3-32. Timer 3 Data Register - T3DATA

- **TMS70x2**

  The T3DATA register is accessed at Peripheral-File location P20.

- **TMS70Cx2**

  The T3DATA register is accessed at Peripheral-File location P23.

3.8.1.6 Receiver Buffer

The receiver buffer, RXBUF, is a read-only register used to store the current RX data. Writing has no direct effect on this register. Data in the RXBUF is right justified, padded with leading 0s.

7	6	5	4	3	2	1	0
MSb			Receiver Data				LSb
0	0	0	←	5 Data Bits			→
0	0	←	6 Data Bits				→
0	←	7 Data Bits					→
←	8 Data Bits						→

Figure 3-33. Receive Buffer - RXBUF

- **TMS70x2**

  The read-only RXBUF register is accessed at PF location P22.

- **TMS70Cx2**

  The read-only RXBUF register is accessed at PF location P25.

3.8.1.7 Transmitter Buffer

The transmitter buffer, TXBUF, is a write-only register used to store data bits to be transmitted by TX. Data written to TXBUF must be right justified because the left-most bits will be ignored for characters less than eight bits long.

7	6	5	4	3	2	1	0
MSb			Transmitter Data				LSb
X	X	X	←	5 Data Bits			→
X	X	←	6 Data Bits				→
X	←	7 Data Bits					→
←	8 Data Bits						→

Figure 3-34. Transmitter Buffer - TXBUF

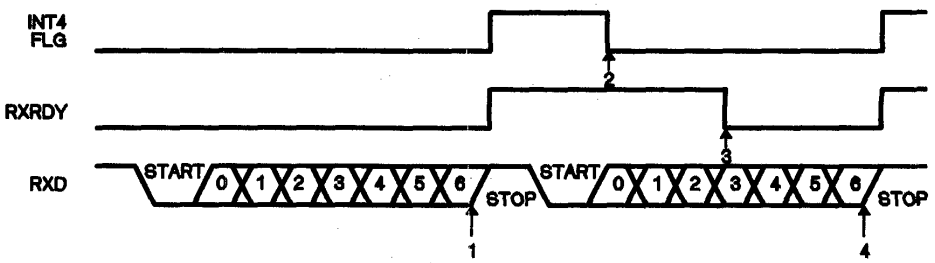
- **TMS70x2**

  The write-only TXBUF register is accessed at PF location P23.

- **TMS70Cx2**

  The write-only TXBUF register is accessed at PF location P26.

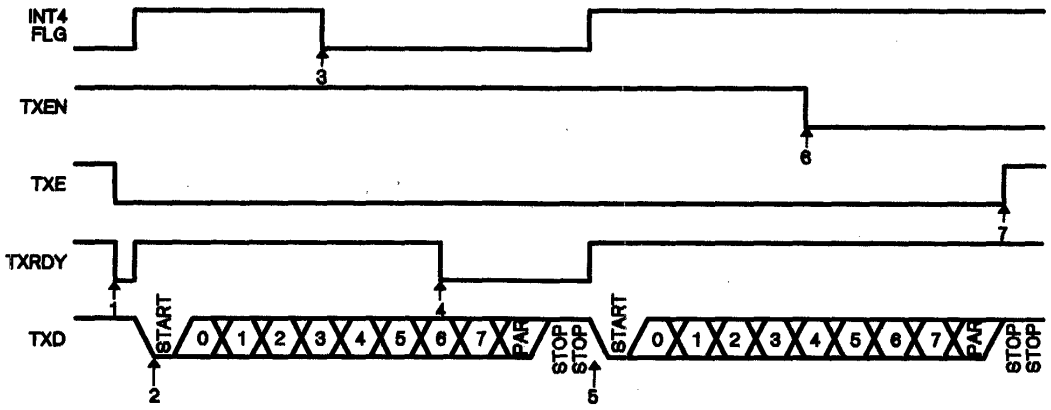
3.8.1.8 RX Signals in Communication Modes



Notes: 1) Format shown is start bit + seven data bits + stop bit.
2) SCLK is continuous, external or internal.
3) If RXEN = 0, RXSHF still receives data from RXD. However, the data is not transferred to RXBUF and RXRDY and INT4FLG are not set.

Sequence of Events:
1) RXSHF data is transferred to RXBUF. Error status bits are set if an error is detected.
2) Software writes to INT4CLR to clear INT4FLG. If not, CPU clears.
3) INT4FLG on entry to level 4 interrupt routine.
4) Software reads RXBUF.

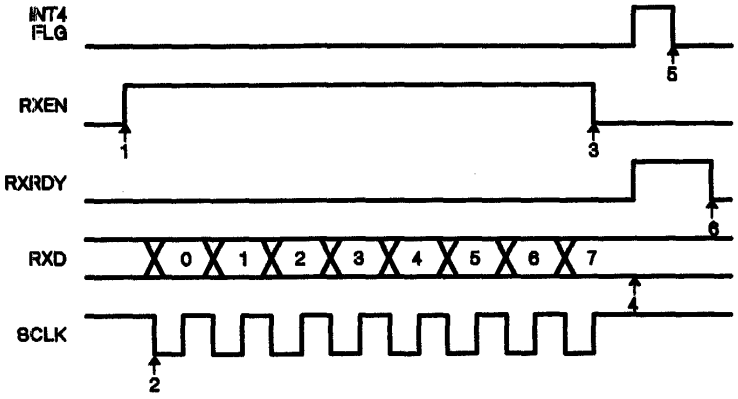
3.8.1.9 TX Signals in Communication Modes



Notes: 1 Format shown is start bit + eight data bits + parity bit + two stop bits.
2) SCLK is continuous whether internal or external.

Sequence of Events:
1) Software writes to TXBUF.
2) TXBUF and WU data are transferred to TXSHF and WUT. INT4FLG and TXRDY are set.
3) Software writes to INT4CLR to clear INT4FLG or CPU clears INT4FLG on entry to level 4 interrupt routine.
4) Software writes to TXBUF.
5) Software writes to INT4CLR to clear INT4FLG or CPU clears INT4FLG on entry to level 4 interrupt routine.
6) Software resets TXEN; current frame will finish and transmission will stop whether TXBUF is full or empty.
7) TXE is set if TXBUF and TXSFT are empty.

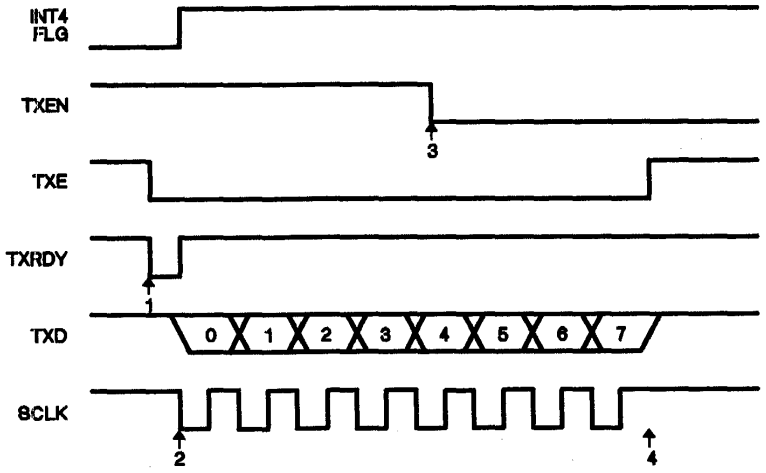
3.8.1.10 RX Signals in Serial I/O Modes



Notes: 1 RXEN has no effect on INT4FLG or RXRDY in Serial I/O mode.
2) RXD is sampled on SCLK rise; external shift registers should be clocked on SCLK fall.
3) The SCLK source should be internal as it is gated by internal circuitry.

Sequence of Events:
1) Software starts receiving by setting RXEN.
2) Gated SCLK starts and data is received.
3) RXEN is automatically cleared in last data bit.
4) RXSHF data is transferred to RXBUF, and RXRDY and INT4 are set.
5) Software writes to INT4CLR to clear INT4FLG; if not, CPU clears INT4FLG on entry to level 4 interrupt routine.
6) Software reads RXBUF.

3.8.1.11 TX Signals in Serial I/O Modes



Notes: 1 Format shown is eight data bits.
2) The SCLK source should be internal as it is gated by internal circuitry.

Sequence of Events:
1) Software writes to TXBUF.
2) TXBUF data is transferred to TXSFT; INT4FLG and TXRDY are set, and SCLK starts.
3) Software resets TXEN, current frame will finish and transmission will halt whether TXBUF is full or empty.
4) Frame ends and SCLK stops because TXEN = 0.

3.8.2 Clock Sources and Serial Port Modes

The serial port can be driven by an internal (Timer 3) or external baud rate generator. The serial clock source, SCLK, is determined by the SCTL1 clock bit (CLK) as either an input or an output. If an *external clock source* is selected, then the A6/SCLK pin (TMS70x2 devices) or A4/SCLK pin (TMS70Cx2 devices) is a high-impedance input. If an *internal clock source* is selected, then a 50% duty cycle clock signal is output on the low-impedance SCLK pin. The clock output frequency depends on the crystal frequency. The current logic level of SCLK (internal or external) can be determined by reading SCLK. RX receives data on the rising SCLK edges and TX transmits data on the falling SCLK edges.

RX/TX (receiver/transmitter) has three modes: two communication modes - Asynchronous and Isosynchronous - and Serial I/O. Serial I/O Mode links the serial port to shift registers for simple I/O expansion. The Isosynchronous and Asynchronous communication modes link to other synchronous and asynchronous devices. These two modes also have extra features for two forms of multiprocessor communication, Motorola and Intel. In all modes, I/O is NRZ (non-return to zero) format; that is, data value 1 = high level, and data value 0 = low level.

3.8.2.1 Asynchronous Communication Mode

In Asynchronous communication mode, the frame format consists of a start bit, five to eight data bits, an even/odd/no parity bit, and one or two stop bits. The bit period is eight times the SCLK period.

Receiving a valid start bit initiates RX operation. A valid start bit consists of a negative edge followed by three samples, two of which *must* be zero. If two of the three samples are not zero, then the receiver continues to search for a Start bit. These samples occur three, four, and five SCLK periods after the negative edge. This sequence provides false start bit rejection and also locates the center of bits in the frame where the bits will be read on a majority (two out of three) basis. Figure 3-35 illustrates the asynchronous communication format, with a start bit showing how edges are found and majority vote taken.

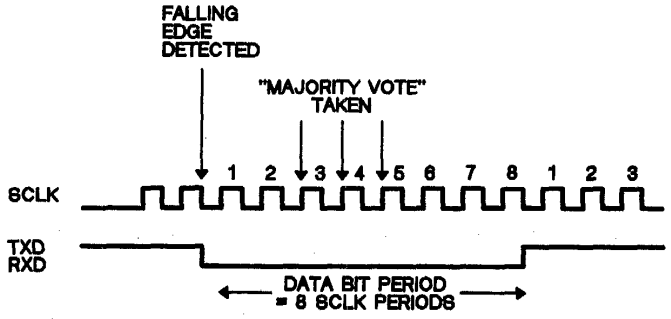


Figure 3-35. Asynchronous Communication Format

Since RX synchronizes itself to frames, the external transmitting and receiving devices do not have to use the same SCLK; it may be generated locally. If the internal SCLK is used it is output continuously on pin A6/SCLK (TMS70x2 devices) or A4/SCLK (TMS70Cx2 devices).

3.8.2.2 Isosynchronous Communication Mode

Isosynchronous communication mode is a hybrid protocol, combining features of the Asynchronous mode and the Serial I/O mode. The Isosynchronous frame format is the same as the Asynchronous mode frame format, consisting of a start bit, five to eight data bits, an even/odd/no parity bit, and one or two stop bits. However,it uses only one serial clock (SCLK) cycle per data bit as compared to 8 SCLKs per data bit for Asynchronous mode. This allows much faster transmission rates than Asynchronous mode. The bit period equals the SCLK period, as it does in Serial I/O mode. Bits are read on a single value basis. Since the RX does not synchronize itself to the data bits, the transmitter and receiver must be supplied with a common SCLK. The benefit of the Isosynchronous mode is that the frame format can be configured like the Asynchronous mode, yet the baud rate is that of the Serial I/O mode.

Receiving a valid start bit, which consists of a negative edge, initiates RX operation. Since RX does not synchronize itself to data bits, the transmitter and receiver must be supplied with a common SCLK. If the internal SCLK is used it is output continuously on pin A6/SCLK/EC2 (TMS70x2 devices) or A4/SCLK (TMS70Cx2 devices).

Figure 3-36 illustrates the Isosynchronous communication format, with a complete frame consisting of a start bit, six data bits, even parity, and two stop bits.

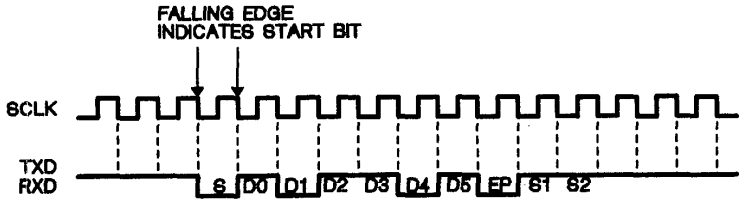


Figure 3-36. Isosynchronous Communication Format

In both the Asynchronous and Isosynchronous Communication modes, when a frame is fully received, RXBUF is loaded from RXSHF, RXRDY. and INT4 flag are set to 1, and the error status bits are set accordingly. RXRDY is reset to 0 when the CPU reads RXBUF.

Transmission is initiated after the CPU writes to TXBUF. This sets TXE to 0. TXSHF is loaded from TXBUF, setting TXRDY and INT4 flag to 1. After completing the transmission, TXSHF reloads if TXBUF is full; if not, TX idles and TXE is 1 until TXBUF is written to. Bit 3 of Port 3 must be set to a 1 to enable data transmission through the B3/TXD pin.

3.8.2.3 Serial I/O Mode

In Serial I/O mode, the frame format is five to eight data bits and one stop bit, with no corresponding clock cycle for the stop bit. An external or internal synchronizing clock signal must be supplied from either the internal Timer 3 or an external clock. An external clock must be supplied if the external SCLK option is used. The bit period is equal to the SCLK period. TX operation is initiated by writing to TXBUF when TXRDY equals 1. RX operation is initiated by writing a 1 to the RXEN bit. When the receiver has received a full frame, the RXEN bit is automatically cleared, disabling the receiver. The transmitter starts operating when the TX enable bit (TXEN) is set to 1. Data is written to TXBUF when TXRDY equals 1. Unlike the receiver, the TXEN bit is not automatically cleared when the transmitter finishes a full frame.

To start the receiver and transmitter at the same time, first write the transmitter data to TXBUF and then set both RXEN and TXEN in one instruction. Be careful that the enable bits are not set when Timer 3 rolls over past 0. This can be done by adjusting the timer rate before the bits are enabled and then setting the timer to the correct rate after enabling.

Figure 3-37 illustrates the serial I/O format for two back-to-back frames, each containing five data bits.

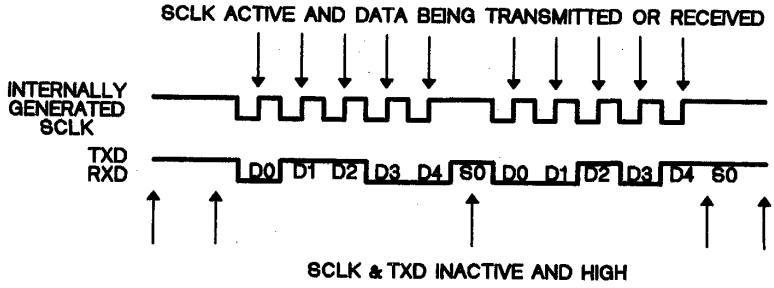


Figure 3-37. Serial I/O Communication Format

An internal SCLK source will be output on pin A6/SCLK (TMS70x2 devices) or A4/SCLK (TMS70Cx2 devices). In Serial I/O mode, SCLK is only active when data is being transmitted or received; otherwise, SCLK has a value of one.

3.8.3 Multiprocessor Communication

When the serial port is in either the Asynchronous or Isosynchronous communications mode, the multiprocessor communication formats are available. These formats efficiently transfer information between many microcomputers on the same serial link. Information is transferred as a block of frames from a particular source to some destination(s). The serial port has features to identify the start of a block of data, and suppress interrupts and status information from RX until a block start is identified.

In both multiprocessor modes the sequence is:

1) The serial port wakes up at the start of a block and reads the first frame (containing the destination address).

2) A software routine is entered through either an interrupt or polling routine and checks the incoming data byte against its address byte stored in memory.

3) If the block is addressed to the microcomputer the CPU reads the rest of the block; if not, the software routine puts the serial port to sleep again and therefore will not receive serial port interrupts until the next block start.

On the serial link, all processors set their SLEEP bit to 1 so that they will only be interrupted when the address bit in the data stream is a 1. When the processors receive the address of the current block, they compare it to their own addresses and those processors which are addressed set their SLEEP bit to a 0, so that they will read the rest of the block.

Although RX still operates when the SLEEP bit is 1, it will not set RXRDY, INT4 flag, or the error status bits to 1 unless the address bit in the received frame is a 1. The RX does not alter the SLEEP bit; this must be done in software.

To provide more flexibility, the serial port implements two multiprocessor protocols, one supported by Motorola and the other by Intel. The Motorola protocol is compatible with the Motorola MC6801 processor modes and the Intel protocol is compatible with the Intel protocol for the 8051. The multiprocessor mode is software selectable via the MULTI bit in the SMODE register (Figure 3-28). Both formats use the WU and SLEEP flags to control the TX and RX features of these modes.

Because the Intel multiprocessor mode contains an extra address/data bit, it is not as efficient as the Motorola mode in handling blocks containing more than 10 bytes of data. The Intel mode is more efficient in handling many small blocks of data because it does not have to wait between blocks of data as does the Motorola mode.

3.8.3.1 Motorola (MC6801) Protocol

In this protocol, blocks are separated by having a longer idle time between the blocks than between frames in the blocks. An idle time of 10 or more bits after a frame indicates the start of a new block.

The processor wakes up (serial port resets the SLEEP bit to 0) after the block start signal. The processor now recognizes the next serial port interrupt. The service routine then receives the address sent out by the transmitter and compares this address to its own. If the CPU is addressed, the service routine does not set the SLEEP bit, and receives the rest of the block. If the CPU is not addressed, the service routine sets the SLEEP bit (in software) to a 1. This lets the CPU continue to execute its main program without being interrupted by the serial port. The serial port sets the SLEEP bit to 0 whenever it detects a block start signal.

There are two ways to send a block start signal.

1) The first is to deliberately leave an idle time of 10 bits or more by delaying the time between the transmission of the last frame of data in the previous block and the address frame of the new block.

2) In the second method, the serial port implements a more efficient method of sending a block start signal. Using the Wake-Up (WU) bit, an idle time of exactly one frame (timed by the serial port) can be sent. The serial communications line is therefore not idle any longer than necessary.

Associated with the WU bit is the wake-up temporary (WUT) flag. WUT is an internal flag, double buffered with WU. When TXSHF is loaded from TXBUF, WUT is loaded from WU, and WU is reset to 0. This arrangement is shown in Figure 3-38.

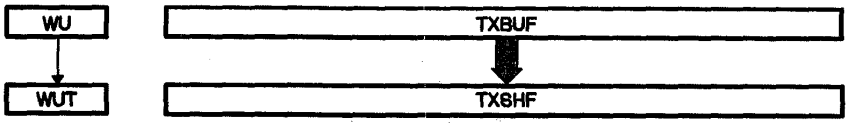


Figure 3-38. Double-Buffered WUT and TXSHF

Sending out a block start signal of exactly one frame time is accomplished as follows:

1) Write a 1 to the WU bit.

2) Write a data word (don't care) to TXBUF.

3) When TXSHF is free again, TXBUF's contents are shifted to TXSHF, and the WU value is shifted to WUT.

4) If WU was set to a 1, the start, data, and parity bits are suppressed and an idle period of one frame, timed by the serial port, is transmitted.

5) The next data word, shifted out of the serial port after the block start signal, is the second data word written to the TXBUF after writing a 1 to the WU bit.

6) The first data word written is suppressed while the block start signal is sent out, and ignored after that.

Writing the first don't care data word to the TXBUF is necessary so the WU bit value can be shifted to WUT. After the don't-care data word is shifted to the TXSHF, the TXBUF (and WU if necessary) may be written to again, since WUT and TXSHF are both double-buffered.

Although RX still operates when the SLEEP bit is 1, it will not set RXRDY, INT4 flag, or the error status bits to 1. The RX will set the SLEEP bit to 0 if it times an appropriate 10-bit idle time on RXD. The Motorola multiprocessor communication format is shown in Figure 3-39.

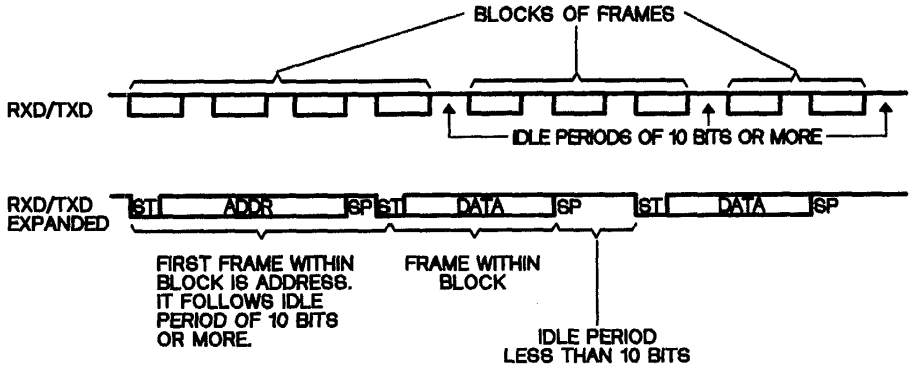


Figure 3-39. Motorola Multiprocessor Communication Format

3.8.3.2 Intel (I8051) Protocol

In the Intel protocol, the frame has an extra bit called an address bit just before the parity bit. Blocks are distinguished by the first frame(s) in the block with the address bit set to 1, and all other frames with the address bit set to 0. The idle period timing is irrelevant.

The WU bit sets the address bit. In TX, when the TXBUF and WU are loaded into TXSHF and WUT, WU is reset to 0 and WUT is the value of the address bit of the current frame. Thus, to send an address, set the WU bit to a 1, and write the appropriate address value to the TXBUF. When this address value is transferred to TXSHF and shifted out, its address bit is sent as a 1, which flags the other processors on the serial link to read the address. Since TXSHF and WUT are both double-buffered, TXBUF and WU may be written to immediately after TXSHF and WUT are loaded. To transmit non-address frames in the block, the WU bit must be left at 0. *On the TMS70Cx2 devices, the received address bit is also placed in the SSTAT IADD bit.*

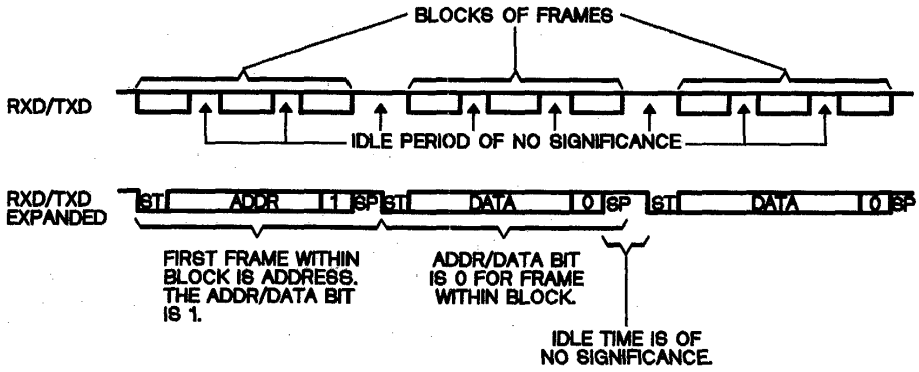


Figure 3-40. Intel Multiprocessor Communication Format

3.8.4 Serial Port Initialization

The serial port must be initialized before it can be used; then it may be operated by simply reading and writing to Peripheral-File registers. A good programming practice is not to assume that any registers have particular values at power-up or reset. A program should write to every value or register that might affect the serial port. Initialize the serial port as follows:

- **TMS70X2**

  1) Set B3 data value to 1. This allows the TXD line to transmit.

  2) Write to the SMODE register (P17). This sets the character format and the type of communication mode.

  3) Write to the SCTL0 register (second write to P17) to set the UR bit to 0. This same write can also enable the transmitter, receiver, or both.

  4) Load the Timer 3 reload register value (P20).

  5) Write to SCTL1 (P21) to initialize Timer 3, the clock source, and multiprocessor mode.

  Once the serial port is initialized it can be operated continuously in the selected operational mode. To send data, simply write to the transmit buffers (P23), making sure that the transmitter is enabled (P17). Take input data from the receive buffer (P22) with the receiver enabled (P17). If the mode must be changed, the serial port must be reset and then re-initialized for the desired mode. The serial port can be reset in two ways: hardware reset (via the RESET pin) or software reset (via the UR bit in SCTL0).

- **TMS70Cx2**

  1) Set B3 data value to 1. This allows the TXD line to transmit.

  2) Write to the SMODE register (P20). This sets the character format and the type of communication mode.

  3) Write to the SCTL0 register (P21). Enable the receiver or the transmitter or both. The UR bit must be set to 0.

  4) Load the Timer 3 reload register value (P23).

  5) Write to SCTL1 register (P24) to initialize Timer 3, the clock source, and multiprocessor mode, if desired.

  Once the serial port is initialized it can be operated continuously in the selected operational mode. To send data, simply write to the transmit buffers (P26), making sure that the transmitter is enabled (P21). Take input data from the receive buffer (P25) with the receiver enabled (P21). If the mode must be changed, the serial port must be reset and then re-initialized for the desired mode. The serial port can be reset in three ways: hardware reset (via the RESET pin) or software reset (via the UR bit in SCTL0), or by writing to the SMODE register.

3.8.5 Timer 3

Timer 3, illustrated in Figure 3-41 and Figure 3-42, can be used as a stand-alone timer or as the internal baud-rate generator on TMS70x2 and TMS70Cx2 devices.

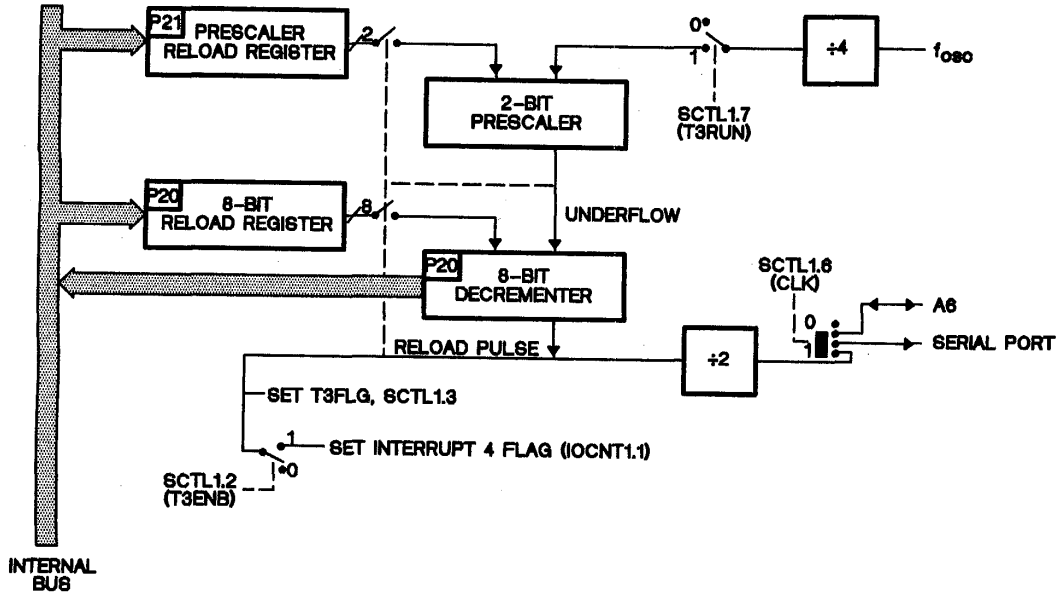


Figure 3-41. 8-Bit Timer 3 (TMS70x2)

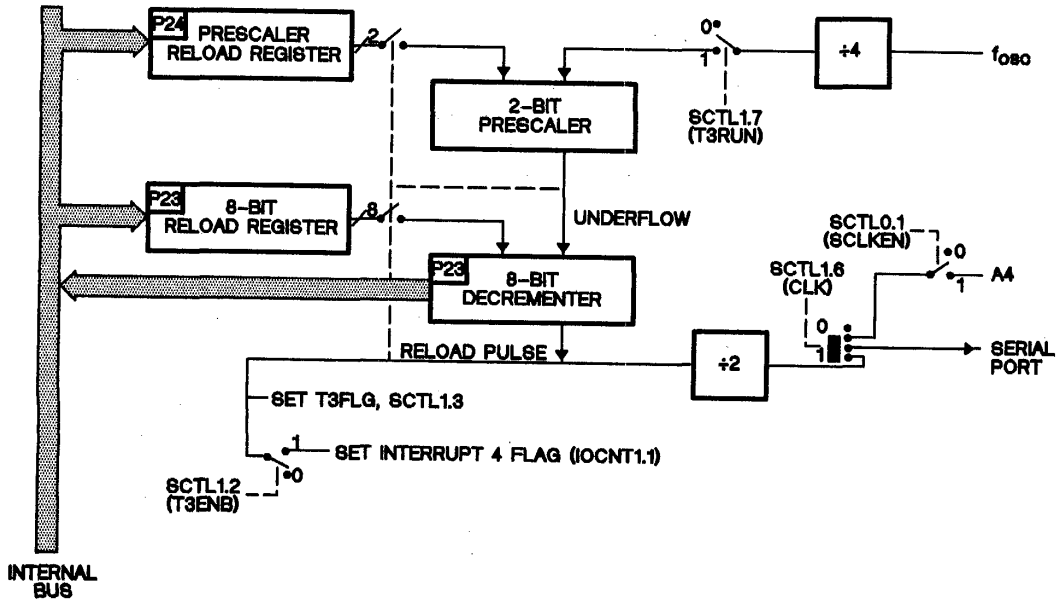


Figure 3-42. 16-Bit Timer 3 (TMS70Cx2)

Timer 3 is accessed through T3DATA (similar to T1DATA and T2DATA on the TMS70x2 devices) and SCTL1 (shared with RX/TX functions). The clock source for Timer 3 is internal only, and has a period of 2 × $t_{c(C)}$. Timer 3 is a free running clock and is updated with new timer reload values when the prescaler and decrementer pass through zero together. Timer 3 is stopped and started by bit 7 in SCTL1.

Timer 3 consists of a 2-bit prescaler (SCTL1 bits 1 and 0) and an 8-bit decrementer (register T3DATA). When they decrement through zero, both the prescaler and the decrementer are reloaded from the 2-bit and 8-bit reload registers, respectively.

The Timer 3 output goes to the serial port via a ÷2 circuit, producing an internal equal mark-space ratio SCLK. The baud rate generated by Timer 3 is user-programmable and is determined by the value of the 2-bit prescaler and the 8-bit timer reload register. The equations for determining the baud rates for both the Asynchronous and Isosynchronous modes are:

Asynchronous baud rate, TMS70x2 and TMS70Cx2 only:

$$\frac{1}{32 \times (PR + 1) \times (TR + 1) \times t_{c(C)}}$$

Isosynchronous and Serial I/O baud rate, TMS70x2 and TMS70Cx2 only:

$$\frac{1}{4 \times (PR + 1) \times (TR + 1) \times t_{c(C)}}$$

where:

$t_{c(C)}$ = $2/f_{osc}$
PR = Timer 3 prescale reload register value
TR = Timer 3 reload register value

For example, to program the serial port to operate at 300 baud in Asynchronous mode (with f_{osc}= 8 MHz), the prescaler value is set to 3 and the reload register value is set to 103 decimal, or >67. Other prescaler and timer values for common baud rates are shown in Table 3-16.

Table 3-16. Timer Values for Common Baud Rates - TMS70x2 and TMS70Cx2

BAUD RATE	3.579454 MHz		4.9152 MHz		7.158908 MHz		8 MHz	
	PS, T	ERROR	PS, T	ERROR	PS, T	ERROR	PS, T	ERROR
75	3, 186	0.2%	3, 255	.0%	–	–	–	–
110	1, 253	0.1%	3, 174	0.3%	3, 253	0.1%	–	–
300	0, 185	0.2%	0, 255	.0%	2, 123	.0%	3, 103	0.2%
600	0, 92	0.2%	0, 127	.0%	0, 185	0.2%	3, 51	0.2%
1200	0, 46	0.8%	0, 63	.0%	0, 92	0.2%	3, 25	0.2%
2400	0, 22	1.3%	0, 31	.0%	0, 46	0.8%	3, 12	0.2%
4800	0, 11	3.0%	0, 15	.0%	0, 22	1.3%	1, 12	0.2%
9600	0, 5	3.0%	0, 7	.0%	0, 11	3.0%	0, 12	0.2%
19200	0, 2	3.0%	0, 3	.0%	0, 5	3.0%	0, 6	7.0%
38400	0, 1	27.0%	0, 1	.0%	0, 2	3.0%	0, 2	.0%
125000	–	–	–	–	–	–	0–0	.0%

Note: PS = prescaler; T = timer

The Timer 3 output always sets T3FLG to 1, and sets INT4 flag to 1 if T3ENB is a 1 when the timer and prescaler decrement through 0. This allows Timer 3 to be used as a utility timer if it is not used by the serial port. Timer 3 and its flags are not affected by the serial port software reset, UR, allowing Timer 3 to be used independently of the serial port.

3.8.6 Initialization Examples

This section contains four examples that initialize the serial port. In each case the data is moved to and from the buffers in the interrupt routines.

- The first example shows a typical RS-232 application that connects to a terminal.
- The second demonstrates a system using the Serial I/O mode to connect to a shift register.
- The third example uses the baud-rate timer as an additional third timer when the serial port is not used.
- The last example illustrates use of the Intel mode in a multiprocessor application.

In all examples, assume the register mnemonics have been equated (EQU) with the corresponding Peripheral-File location.

3.8.6.1 RS-232-C Example

This example transmits and receives data from a standard RS-232-C-type terminal at 9600 baud with a data format of 7 data bits, 2 stop bits and no parity.

```
RS232   DINT                        Precaution
        ORP    %?00001000,PORTB     Enable TX pin
        MOVP   %?00001011,IOCNT1    Enable INT4
        MOVP   %0,P17               Point to SCTL0
        MOVP   %?00010000,SCTL0     Reset the UART
        MOVP   %?11001010,SMODE     Two stop, 7 data bits, no
*                                   parity, no extra Intel mode bit,
*                                   communications mode
        MOVP   %?00010101,SCTL0     Clear RESET, clear error flags,
*                                   enable TX and RX
        MOVP   %7,T3DATA            Set baud rate to 9600
*                                   (4.9152 MHz crystal)
        MOVP   %?01000000,SCTL1     Internal clock, prescale=0, no
*                                   multiprocessing, disable Timer 3
*                                   interrupt, start Timer 3
        EINT
```

3.8.6.2 Serial I/O Example

This routine sends and receives data from a shift register device at 1200 baud with 8 data bits and no parity.

```
SERIAL  DINT                            Precaution
        ORP     %?00001000,PORTB        Enable TX pin
        MOVP    %?00001011,IOCNT1       Enable INT4
        MOVP    %0,P17                  Point to SCTL0
        MOVP    %?00010000,SCTL0        Reset the UART
        MOVP    %?00001100,SMODE        One stop, 8 data bits,
*                                       no parity, no extra Intel
*                                       mode bit, Serial I/O mode
        MOVP    %?00010101,SCTL0        Clear RESET, clear error
*                                       flags, enable TX and RX
        MOVP    %64,T3DATA              Set baud rate to
*                                       1200 (5MHz crystal)
        MOVP    %?11000000,SCTL1        Internal clock, prescale=0,
*                                       no multiprocessing, disable
*                                       Timer 3 interrupt,
*                                       start Timer 3
        EINT
```

3.8.6.3 Extra Timer with No Serial Port

Timer 3 can be used as an additional timer when the serial port is not needed. INT4 occurs whenever the timer passes 0. The timer period is determined by the value TIME and the prescale bit in SCTL1. Disable the transmitter and receiver to assure no interrupts come from that source. This timer works best as a periodic interrupt, allowing a task to be performed at a fixed interval.

```
TIMER3  DINT                            Precaution
        MOVP    %?00001011,IOCNT1       Enable INT4
        MOVP    %0,P17                  Point to SCTL0
        MOVP    %?00010000,SCTL0        Reset the UART
        MOVP    %?01000010,SMODE        Asynchronous communication mode
        MOVP    %?00010000,SCTL0        Clear RESET, clear error
*                                       flags, disable TX and RX
        MOVP    %TIME,T3DATA            Set timer to selected rate
        MOVP    %?110001XX,SCTL1        Internal clock, no
*                                       multiprocessing selected
*                                       prescale, enable Timer 3
*                                       interrupts, start Timer 3
        EINT
```

3.8.6.4 Intel Multiprocessor Example

This example illustrates basic concepts of sending and receiving data in a multiprocessor system. The processors are usually close to each other so they can send at maximum speed without problems. The data is sent and received during the interrupt routines.

```
MULTI   DINT                            Precaution
        ORP    %?00001000,PORTB         Enable TX pin (?=binary)
        MOVP   %?00001011,IOCNT1        Enable INT4
        MOVP   %0,P17                   Point to SCTL0
        MOVP   %?00010000,SCTL0         Reset the UART
        MOVP   %?01111111,SMODE         One stop, 8 data bits,
*                                       odd parity, Intel mode bit,
*                                       communications mode
        MOVP   %?00010101,SCTL0         Clear RESET, clear error
*                                       flags, enable TX and RX
        MOVP   %0,T3DATA                Set baud rate to full
*                                       speed (5MHz crystal)
        MOVP   %?11100000,SCTL1         Internal clock, prescale=0,
*                                       no multiprocessing, disable
*                                       Timer 3 interrupts, put
*                                       receiver to sleep,
        EINT                            start Timer 3
*
*   Meanwhile, back at the interrupt routines
*
SENDIT  ORP    %BIT4,SCTL1              Send Wake-Up bit
*                                       (Bit4=00010000)
        MOVP   %ADDRS,TXBUF             Send address byte
*                                       wait for the transmit
*                                       complete interrupt . . . . .
        ANDP   %#BIT4,SCTL1             Clear Wake-Up bit
*                                       (# = logical NOT)
        MOVP   %DATA,TXBUF              start sending data bytes
*
*
GETIT   MOVP   RXBUF,A                  Get address byte
*                                       (it only interrupts on an
*                                       address byte when sleeping)
        CMP    %ADDRS,A                 Is it this processor's address?
        JNE    NOTIT                    If this is not the correct
*                                       address ignore the rest
*                                       of the following data bytes
        ANDP   %#BIT5,SCTL1             Clear Sleep bit and wait for
*                                       additional data bytes
*                                       Some method should determine
*                                       End of Data so that the pro-
*                                       cessor can go back to sleep
*                                       Byte count in first data byte
*                                       or special end of data byte
*                                       are two methods
```

3.8.7 Serial Port Interrupts

INT4 is dedicated to the serial port. Three sources can generate an interrupt through INT4:

1) The transmitter (TX),
2) The receiver (RX), and
3) Timer 3 (T3).

Setting TXEN to 1 allows data loaded into the TXBUF to be shifted into TXSHF. The TX sets TXRDY and INT4 flag to 1 when TXSHF is loaded from TXBUF.

In the communication modes, if RXEN is set to 1, RX sets RXRDY and INT4 flag to a 1 when RXBUF is loaded from RXSHF. If RXEN is 0, RXSHF still receives frames and shifts them into RXBUF, but RXRDY and INT4 flag are held to 0. If a character is in RXBUF, and RXEN is then set to a 1, RXRDY and INT4 flag will be set to 1.

In Serial I/O mode, RXEN is set to initiate the reception of a frame. When the last bit of the frame is received RXEN is reset to 0; however, RXRDY and INT4 flag are still set to 1 when the character is shifted from RXSHF to RXBUF. RXRDY and INT4 flag bits are not masked by RXEN.

Timer 3 sets T3FLG and INT4 flag (if T3ENB is 1) when its prescaler and timer decrement through 0 together.

When the CPU acknowledges INT4, RXRDY, TXRDY, and T3FLG are the flags that indicate its source. The INT4 service routine must determine which of these sources caused INT4 in the specific application. For example, if all three are likely sources, the INT4 service routine must check for the following possible situations:

1) RXRDY only
2) TXRDY only
3) T3 only
4) RXRDY, TXRDY, T3
5) RXRDY, TXRDY
6) RXRDY, T3
7) TXRDY, T3
8) None

The last check is necessary because RXRDY, TXRDY, or T3FLG can set INT4 flag. It is possible that one or more interrupts may occur between CPU acknowledgement of INT4 and INT4 service routine testing of RXRDY, TXRDY, and T3FLG. The CPU clears the INT4 flag bit when it acknowledges INT4. If a second INT4 source is set in the time between this clearing and the software testing, the second or third interrupts will be serviced by the current INT4 service routine. Thus, when INT4 is again acknowledged (INT4 flag was set again by the second interrupt) RXRDY, TXRDY, and T3FLG will all be set to 0.

4. Electrical Specifications

This section contains electrical and timing information for each category of TMS7000 family devices. The NMOS devices are presented first, followed by the CMOS devices. The TMS7000 CMOS devices can operate at wide voltage and frequency ranges; therefore, the CMOS specifications are presented using two separate test voltage ranges.

NMOS Devices:

CMOS Devices:

4.1 TMS7000, TMS7020, and TMS7040 Specifications

Table 4-1. Absolute Maximum Ratings over Operating Free-Air Temperature Range (unless otherwise noted)

Supply voltage range, V_{CC}† ..... -0.3 V to 7 V
Input voltage range ..... -0.3 V to 7 V
Output voltage range ..... -0.3 V to 7 V
Maximum buffer current ..... ±10 mA
Continuous power dissipation ..... 1 W
Storage temperature range ..... -55°C to 150°C

† Unless otherwise noted, all voltages are with respect to V_{SS}.

Caution:

Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the "Recommended Operating Conditions" section of this specification is not implied. Exposure to absolute maximum rated conditions for extended periods may affect device reliability.

Table 4-2. Recommended Operating Conditions

			MIN	NOM	MAX	UNIT
V_{CC}	Supply voltage		4.5	5	5.5	V
V_{IH}	High-level input voltage	CLKIN	2.6			V
		All others	2.0			V
V_{IL}	Low-level input voltage	CLKIN			0.6	V
		All others			0.8	V
T_A	Operating free-air temperature		0		70	°C

Table 4-3. Electrical Characteristics over Full Range of Operating Conditions

PARAMETER			TEST CONDITIONS	MIN	TYP†	MAX	UNIT
I_I	Input current	Port A, input-only pins	$V_I = V_{SS}$ to V_{CC}		±2	±10	µA
		I/O pins	V_I = 0.4 V to V_{CC}		±10	±100	µA
C_I	Input capacitance				2		pF
V_{OH}	High-level output voltage		I_{OH} = -400 µA	2.4	2.8		V
V_{OL}	Low-level output voltage		I_{OL} = 3.2 mA		0.2	0.4	V
$t_{r(O)}$	Output rise time‡		See Figure 4-1		9	50	ns
$t_{f(O)}$	Output fall time‡		See Figure 4-1		10	60	ns
I_{CC}	Supply current		All outputs open		80	150	mA
$P_{D(av)}$	Average power dissipation		All outputs open		400	825	mW

† All typical values are at V_{CC} = 5 V, T_A = 25°C.
‡ Rise and fall times are measured between the maximum low level and the minimum high level using the 10% and 90% points (see Figure 4-2). Measured outputs have 100-pF loads to V_{SS}.

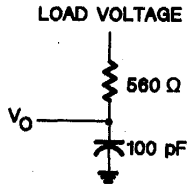


Figure 4-1. Output Loading Circuit for Test

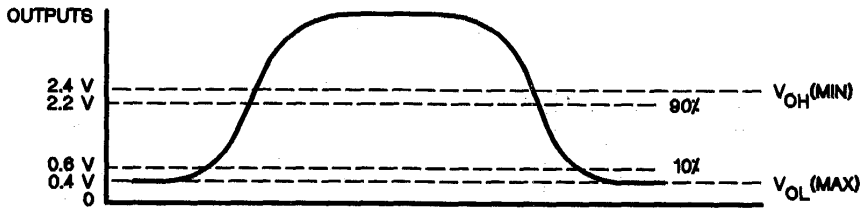


Figure 4-2. Measurement Points for Switching Characteristics

Table 4-4. Recommended Crystal Operating Conditions over Full Operating Range

PARAMETER		MIN	TYP	MAX	UNIT
f_{osc}	Crystal frequency	1.0		5.0	MHz
	CLKIN duty cycle		50		%
$t_{c(P)}$	Crystal cycle time	200		1000	ns
$t_{c(C)}$	Internal state cycle time	400		2000	ns
$t_{w(PH)}$	CLKIN pulse duration high	90			ns
$t_{w(PL)}$	CLKIN pulse duration low	90			ns
t_r	CLKIN rise time†			30	ns
t_f	CLKIN fall time†			30	ns
$t_{d(PH-CH)}$	CLKIN rise to CLKOUT rise delay		125	200	ns

† Rise and fall times are measured between the maximum low level and the minimum high level.

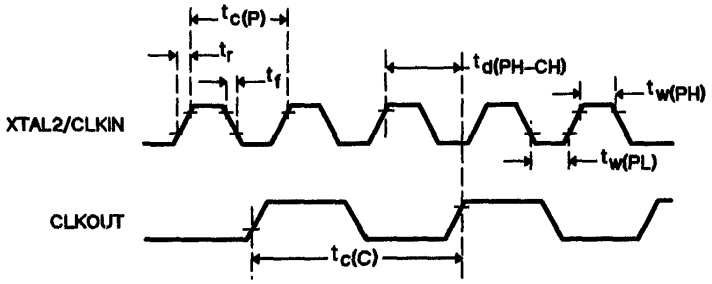


Figure 4-3. Clock Timing

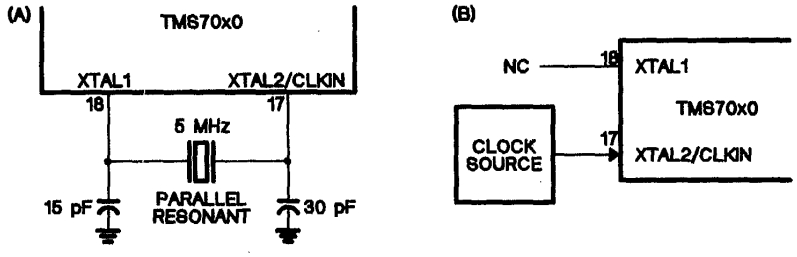


Figure 4-4. Recommended Clock Connections

Table 4-5. Memory Interface Timing at 5 MHz over Full Operating Free-Air Temperature Range

PARAMETER		MIN	TYP	MAX	UNIT
$t_{c(C)}$	CLKOUT cycle time†		400		ns
$t_{w(CH)}$	CLKOUT high pulse duration	130	170	200	ns
$t_{w(CL)}$	CLKOUT low pulse duration	150	190	240	ns
$t_{d(CH-JL)}$	Delay time, CLKOUT rising to ALATCH fall	260	300	340	ns
$t_{w(JH)}$	ALATCH high pulse duration	150	190	230	ns
$t_{su(HA-JL)}$	Setup time, high address valid before ALATCH fall	50	170	220	ns
$t_{su(LA-JL)}$	Setup time, low address valid before ALATCH fall	50	150	220	ns
$t_{h(JL-LA)}$	Hold time, low address valid after ALATCH fall	30	45	80	ns
$t_{su(RW-JL)}$	Setup time, R/$\overline{W}$ valid before ALATCH fall	50	140	200	ns
$t_{h(EH-RW)}$	Hold time, R/$\overline{W}$ valid after $\overline{ENABLE}$ rise	40	100		ns
$t_{h(EH-HA)}$	Hold time, high address valid after $\overline{ENABLE}$ rise	30	40		ns
$t_{su(Q-EH)}$	Setup time, data output valid before $\overline{ENABLE}$ rise	230	290		ns
$t_{h(EH-Q)}$	Hold time, data output valid after $\overline{ENABLE}$ rise	65	80		ns
$t_{d(EH-A)}$	Delay time, $\overline{ENABLE}$ rise to next address drive	60	85		ns
$t_{a(EL-D)}$	Access time, data input valid after $\overline{ENABLE}$ fall	155	190		ns
$t_{a(A-D)}$	Access time, address valid to data input valid	400	470		ns
$t_{d(A-EH)}$	Delay time, address valid to $\overline{ENABLE}$ rise	580		730	ns
$t_{h(EH-D)}$	Hold time, data input valid after $\overline{ENABLE}$ rise	0			ns
$t_{d(CH-EL)}$	Delay time, CLKOUT rise to $\overline{ENABLE}$ fall	-10	15	50	ns

† $t_{c(C)}$ is defined to be $2/f_{osc}$ and may be referred to as a machine state or simply a state.

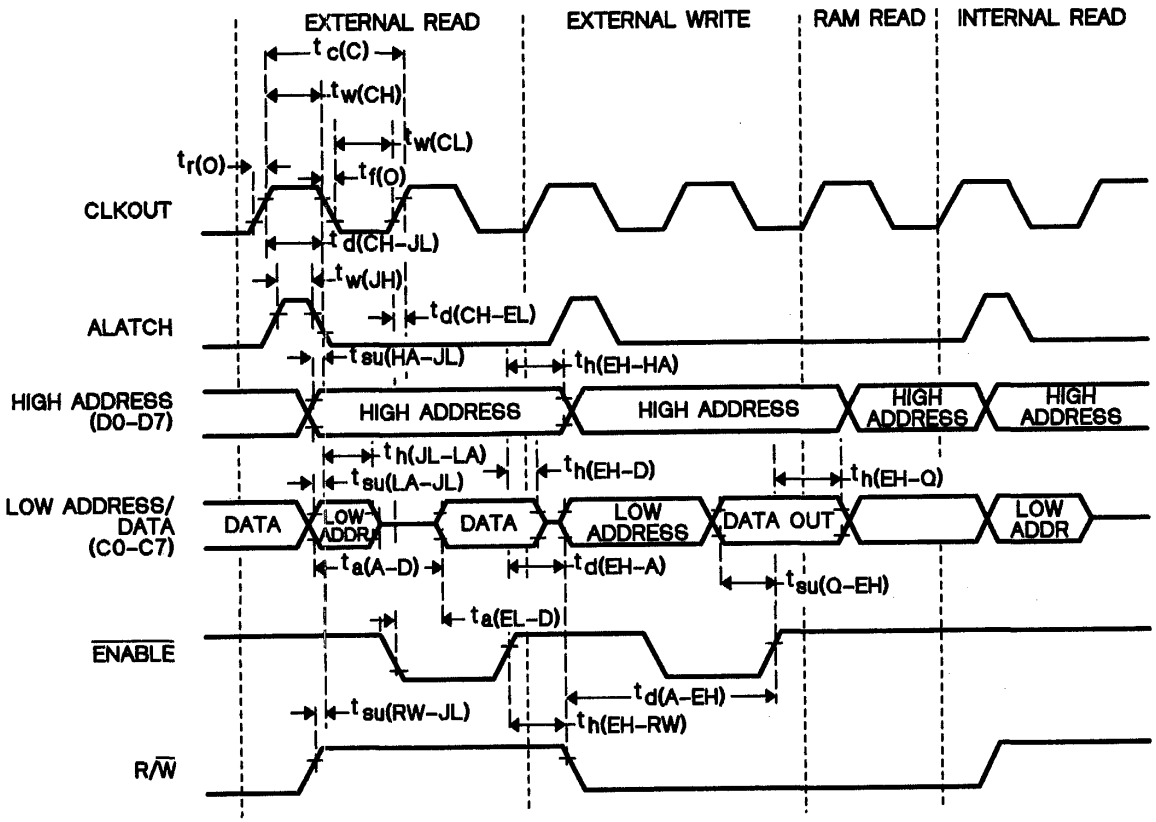


Figure 4-5. Read and Write Cycle Timing

4.1.1 Application of Ceramic Resonator

The circuit shown in Figure 4-6 provides an economical alternative to quartz crystals where frequency tolerance is not a major concern. Frequency tolerance over temperature is about 1%.

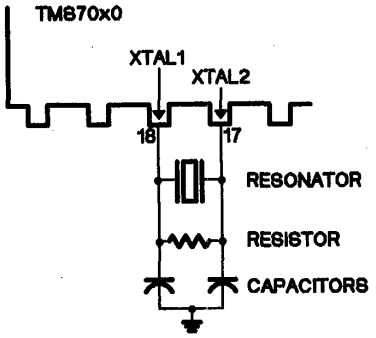


Figure 4-6. Ceramic Resonator Circuit

The following manufacturers supply ceramic resonators.

Murata Corporation of America
1148 Franklin Rd. SE
Marrietta, GA 30067
(404) 952-9777
Telex - 0542329 Murata ATL

For 5 MHz operation
Resonator ceralock CSA5.00MT
Resistor 1 MΩ 10%
Capacitors (both) 30 pF

NGK Spark Plugs (USA) Inc.
20608 Madrona Ave.
Torrance, CA 90503
(213) 328-6882
Telex - 664290

For 5 MHz operation
Resonator R5.OM
Resistor 1 MΩ 10%
Capacitors 68 pF ± 10%

Kyocera International
8611 Balboa Ave.
San Diego, CA 92123
(714) 279-8319
Telex - 697929

4.2 TMS7002 and TMS7042 Specifications

Table 4-6. Absolute Maximum Ratings over Operating Free-Air Temperature Range (unless otherwise noted)

Supply voltage range, V_{CC}† ..... -0.3 V to 7 V
Input voltages range ..... -0.3 V to 7 V
Output voltages range ..... -0.3 V to 7 V
Maximum buffer current ..... ±10 mA
Continuous power dissipation ..... 1.4 W
Storage temperature range ..... -55°C to 150°C

† Unless otherwise noted, all voltages are with respect to V_{SS}.

Caution:

Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the "Recommended Operating Conditions" section of this specification is not implied. Exposure to absolute maximum rated conditions for extended periods may affect device reliability.

Table 4-7. Recommended Operating Conditions

			MIN	NOM	MAX	UNIT
V_{CC}	Supply voltage		4.5	5	5.5	V
V_{IH}	High-level input voltage	CLKIN	2.6			V
		All other inputs	2.0			V
V_{IL}	Low-level input voltage	CLKIN			0.6	V
		All other inputs			0.8	V
T_A	Operating free-air temperature		0		70	°C

Table 4-8. Electrical Characteristics over Full Range of Operating Conditions

PARAMETER			TEST CONDITIONS	MIN	TYP†	MAX	UNIT
I_I	Input current	A5,MC,$\overline{RESET}$, $\overline{INT1}$, $\overline{INT3}$, XTAL2	$V_I = V_{SS}$ to V_{CC}		±2	±10	µA
		Ports C and D A0–A4, A6, A7	V_I = 0.4 V to V_{CC}		±10	±100	
C_I	Input capacitance				2		pF
V_{OH}	High-level output voltage		I_{OH} = -400 µA	2.4	2.8		V
V_{OL}	Low-level output voltage		I_{OH} = 3.2 mA		0.2	0.4	V
$t_{r(O)}$	Output rise time‡		See Figure 4-7		9	30	ns
$t_{f(O)}$	Output fall time‡		See Figure 4-7		10	35	ns
I_{CC}	Supply current		All outputs open		160	210	mA
$P_{D(av)}$	Average power dissipation				800	1155	mW

† All typical values are at V_{CC} = 5 V, T_A = 25°C.
‡ Rise and fall times are measured between the maximum low level and the minimum high level using the 10% and 90% points (see Figure 4-8).

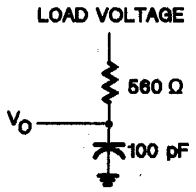


Figure 4-7. Output Loading Circuit for Test

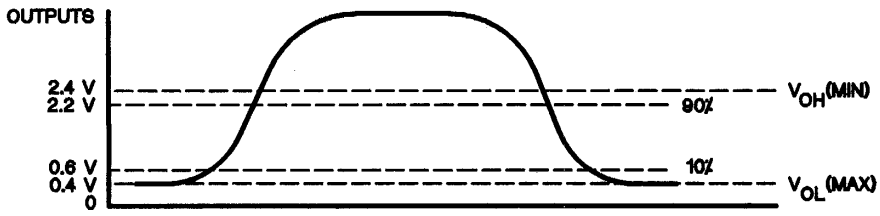


Figure 4-8. Measurement Points for Switching Characteristics

Table 4-9. Recommended Crystal Operating Conditions over Full Operating Range

PARAMETER		MIN	TYP	MAX	UNIT
f_{osc}	Crystal frequency	1.0		8.0	MHz
	CLKIN duty cycle		50		%
$t_{c(P)}$	Crystal cycle time	125		1000	ns
$t_{c(C)}$	Internal state cycle time	250		2000	ns
$t_{w(PH)}$	CLKIN pulse duration high	50			ns
$t_{w(PL)}$	CLKIN pulse duration low	50			ns
t_r	CLKIN rise time†			30	ns
t_f	CLKIN fall time†			30	ns
$t_{d(PH-CH)}$	CLKIN rise to CLKOUT rise delay		70	200	ns

† Rise and fall times are measured between the maximum low level and the minimum high level.

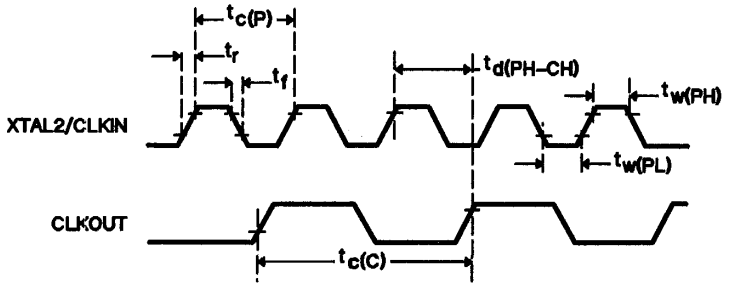


Figure 4-9. Clock Timing

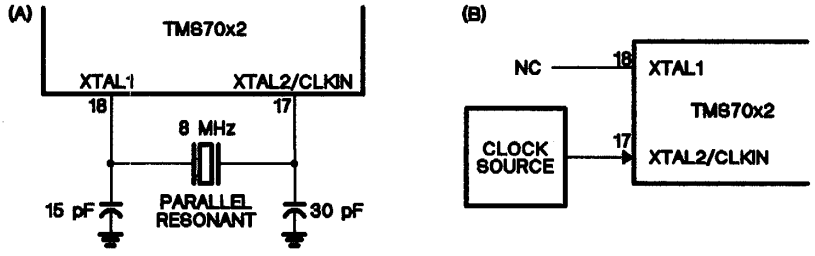


Figure 4-10. Recommended Clock Connections

Table 4-10. Memory Interface Timing

PARAMETER		MIN	MAX	UNIT
$t_{c(C)}$	CLKOUT cycle time†	250	2000	ns
$t_{w(CH)}$	CLKOUT high pulse duration	$0.5t_{c(C)}$-40	$0.5t_{c(C)}$+10	ns
$t_{w(CL)}$	CLKOUT low pulse duration	$0.5t_{c(C)}$-40	$0.5t_{c(C)}$+15	ns
$t_{d(CH-JL)}$	Delay time, CLKOUT rise to ALATCH fall	$0.5t_{c(C)}$-10	$0.5t_{c(C)}$+30	ns
$t_{w(JH)}$	ALATCH high pulse duration	$0.25t_{c(C)}$-15	$0.25t_{c(C)}$+30	ns
$t_{su(HA-JL)}$	Setup time, high address valid before ALATCH fall	$0.25t_{c(C)}$-40	$0.25t_{c(C)}$+45	ns
$t_{su(LA-JL)}$	Setup time, low address valid before ALATCH fall	$0.25t_{c(C)}$-40	$0.25t_{c(C)}$+15	ns
$t_{h(JL-LA)}$	Hold time, low address valid after ALATCH fall	$0.25t_{c(C)}$	$0.25t_{c(C)}$+45	ns
$t_{su(RW-JL)}$	Setup time, R/$\overline{W}$ valid before ALATCH fall	$0.25t_{c(C)}$-35	$0.25t_{c(C)}$+30	ns
$t_{h(EH-RW)}$	Hold time, R/$\overline{W}$ valid after $\overline{ENABLE}$ rise	$0.5t_{c(C)}$-40		ns
$t_{h(EH-HA)}$	Hold time, high address valid after $\overline{ENABLE}$ rise	$0.5t_{c(C)}$-50		ns
$t_{su(Q-EH)}$	Setup time, data output valid before $\overline{ENABLE}$ rise	$0.5t_{c(C)}$-45		ns
$t_{h(EH-Q)}$	Hold time, data output valid after $\overline{ENABLE}$ rise	$0.5t_{c(C)}$-45		ns
$t_{d(LA-EL)}$	Delay time, low address high impedance to $\overline{ENABLE}$ fall	$0.25t_{c(C)}$-45	$0.25t_{c(C)}$	ns
$t_{d(EH-A)}$	Delay time, $\overline{ENABLE}$ rise to next address drive	$0.5t_{c(C)}$-25		ns
$t_{a(EL-D)}$	Access time, data input valid after $\overline{ENABLE}$ fall	$0.75t_{c(C)}$-105		ns
$t_{a(A-D)}$	Access time, address valid to data input valid	$1.5t_{c(C)}$-115		ns
$t_{d(A-EH)}$	Delay time, address valid to $\overline{ENABLE}$ rise	$1.5t_{c(C)}$-80	$1.5t_{c(C)}$+30	ns
$t_{h(EH-D)}$	Hold time, data input valid after $\overline{ENABLE}$ rise	0		ns
$t_{d(EH-JH)}$	Delay time, $\overline{ENABLE}$ rise to ALATCH rise	$0.5t_{c(C)}$-25	$0.5t_{c(C)}$+10	ns
$t_{d(CH-EL)}$	Delay time, CLKOUT rise to $\overline{ENABLE}$ fall	-10	35	ns

† $t_{c(C)}$ is defined to be $2/f_{osc}$ and may be referred to as a machine state or simply a state.

Table 4-11. Memory Interface Timing at 8 MHz

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
$t_{c(C)}$	CLKOUT cycle time†	f = 8 MHz, 50% duty cycle		250		ns
$t_{w(CH)}$	CLKOUT high pulse duration		85	110	135	ns
$t_{w(CL)}$	CLKOUT low pulse duration		85	115	140	ns
$t_{d(CH-JL)}$	Delay time, CLKOUT rise to ALATCH fall		115	135	155	ns
$t_{w(JH)}$	ALATCH high pulse duration		47	70	92	ns
$t_{su(HA-JL)}$	Setup time, high address valid before ALATCH fall		22	65	108	ns
$t_{su(LA-JL)}$	Setup time, low address valid before ALATCH fall		22	50	78	ns
$t_{h(JL-LA)}$	Hold time, low address valid after ALATCH fall		62	90	108	ns
$t_{su(RW-JL)}$	Setup time, R/$\overline{W}$ valid before ALATCH fall		27	60	93	ns
$t_{h(EH-RW)}$	Hold time, R/$\overline{W}$ valid after $\overline{ENABLE}$ rise		85	120		ns
$t_{h(EH-HA)}$	Hold time, high address valid after $\overline{ENABLE}$ rise		75	120		ns
$t_{su(Q-EH)}$	Setup time, data output valid before $\overline{ENABLE}$ rise		80	120		ns
$t_{h(EH-Q)}$	Hold time, data output valid after $\overline{ENABLE}$ rise		80	115		ns
$t_{d(LA-EL)}$	Delay time, low address high impedance to $\overline{ENABLE}$ fall		17	40	62	ns
$t_{d(EH-A)}$	Delay time, $\overline{ENABLE}$ rise to next address drive		100	150		ns
$t_{a(EL-D)}$	Access time, data input valid after $\overline{ENABLE}$ fall		82	120		ns
$t_{a(A-D)}$	Access time, address valid to data input valid		260	300		ns
$t_{d(A-EH)}$	Delay time, address valid to $\overline{ENABLE}$ rise		295	350	405	ns
$t_{h(EH-D)}$	Hold time, data input valid after $\overline{ENABLE}$ rise		0			ns
$t_{d(EH-JH)}$	Delay time, $\overline{ENABLE}$ rise to ALATCH rise		100	105	135	ns
$t_{d(CH-EL)}$	Delay time, CLKOUT rise to $\overline{ENABLE}$ fall		-10	25	35	ns

† $t_{c(C)}$ is defined to be $2/f_{osc}$ and may be referred to as a machine state or simply a state.

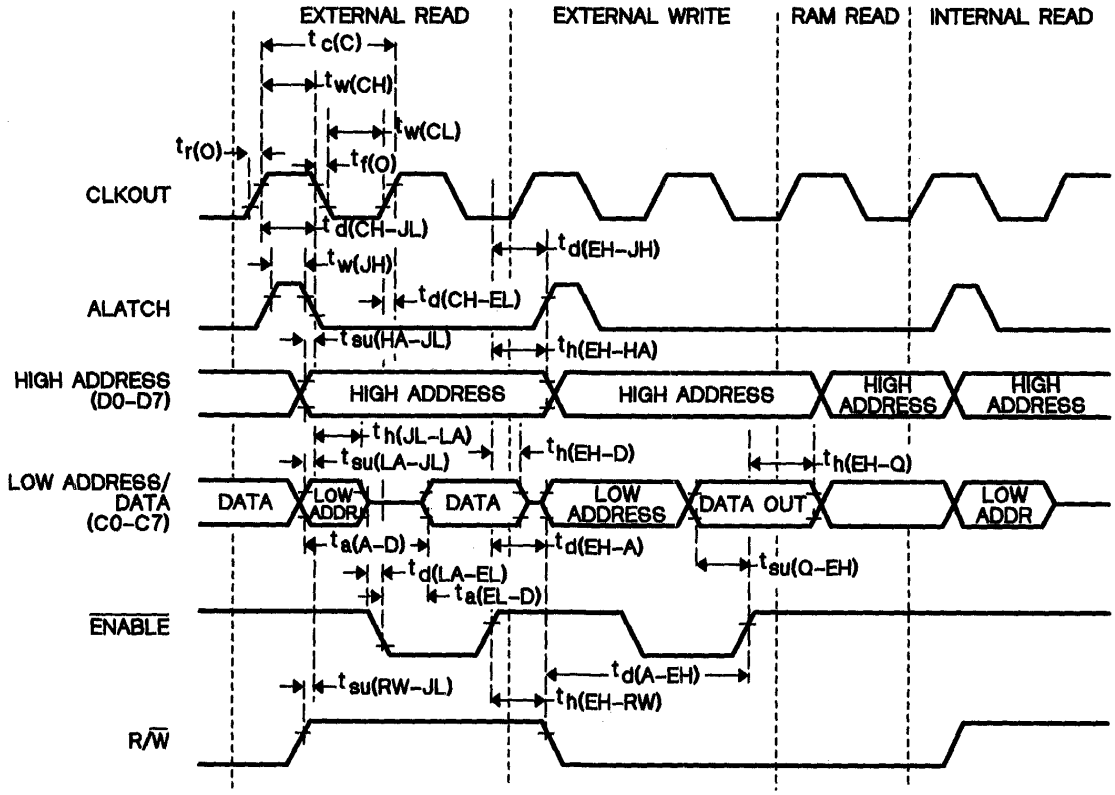


Figure 4-11. Read and Write Cycle Timing

4.2.1 Application of Ceramic Resonator

The circuit shown in Figure 4-12 provides an economical alternative to quartz crystals where frequency tolerance is not a major concern. Frequency tolerance over temperature is about 1%.

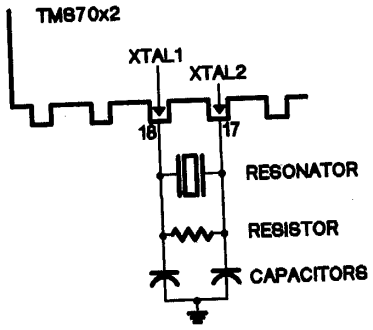


Figure 4-12. Ceramic Resonator Circuit

The following manufacturers supply ceramic resonators.

Murata Corporation of America
1148 Franklin Rd. SE
Marrietta, GA 30067
(404) 952-9777
Telex - 0542329 Murata ATL

For 5 MHz operation
Resonator ceralock CSA5.00MT
Resistor 1 MΩ 10%
Capacitors (both) 30 pF

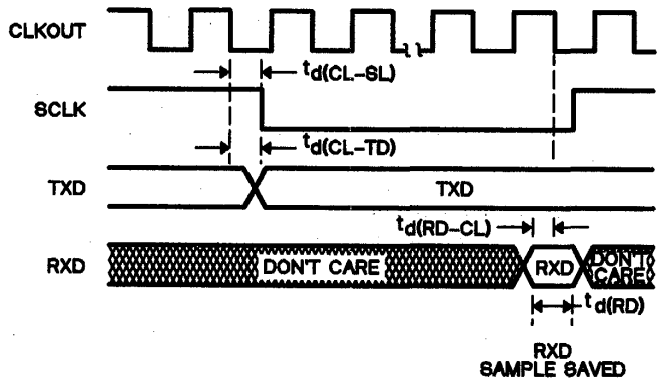
NGK Spark Plugs (USA) Inc.
20608 Madrona Ave.
Torrance, CA 90503
(213) 328-6882
Telex - 664290

For 5 MHz operation
Resonator R5.0M
Resistor 1 MΩ 10%
Capacitors 68 pF ± 10%

Kyocera International
8611 Balboa Ave.
San Diego, CA 92123
(714) 279-8319
Telex - 697929

4.2.2 Serial Port Timing

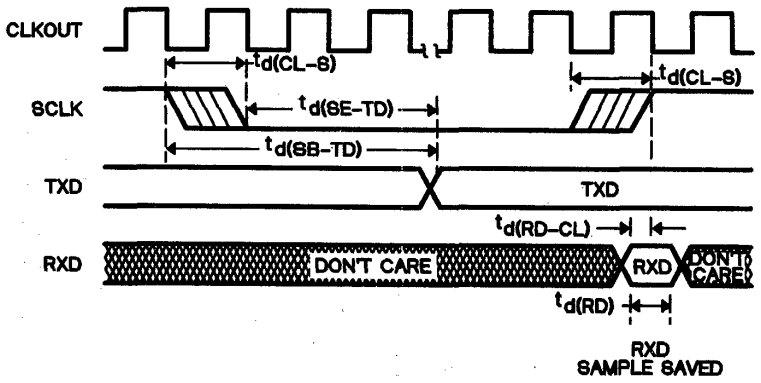
4.2.2.1 Internal Serial Clock



Notes: 1) The CLKOUT signal is not available in Single-Chip mode.
2) CLKOUT = $t_{c(C)}$.

PARAMETER		TYP	UNIT
$t_{d(CL-SL)}$	CLKOUT low to SCLK low	1/4 $t_{c(C)}$	ns
$t_{d(CL-TD)}$	CLKOUT low to new TXD data	1/4 $t_{c(C)}$	ns
$t_{d(RD-CL)}$	RXD data valid before CLKOUT low	1/4 $t_{c(C)}$	ns
$t_{d(RD)}$	RXD data valid time	1/2 $t_{c(C)}$	ns

4.2.2.2 External Serial Clock



Notes: 1) The CLKOUT signal is not available in Single-Chip mode.
2) CLKOUT = $t_{c(C)}$.
3) SCLK sampled; if SCLK = 1 then 0, fall transition found.
4) SCLK sampled; if SCLK = 0 then 1, rise transition found.

PARAMETER		TYP	UNIT
$t_{d(RD-CL)}$	RXD data valid before CLKOUT low	1/4 $t_{c(C)}$	ns
$t_{d(RD)}$	RXD data valid time	1/2 $t_{c(C)}$	ns
$t_{d(SB-TD)}$	Start of SCLK sample to new TXD data	3 1/4 $t_{c(C)}$	ns
$t_{d(SE-TD)}$	End of SCLK sample to new TXD data	2 1/4 $t_{c(C)}$	ns
$t_{d(CL-S)}$	Clockout low to SCLK transition	$t_{c(C)}$	ns

4.3 TMS7742 Specifications

Table 4-12. Absolute Maximum Ratings over Operating Free-Air Temperature Range (unless otherwise noted)

Supply voltage range, V_{CC}†	-0.3 V to 7 V
Supply voltage range, V_{PP}	-0.3 V to 22 V
Input voltage range	-0.3 V to 7 V
Output voltage range	-0.3 V to 7 V
Maximum buffer sink current	±10 mA
Continuous power dissipation	2 W
Storage temperature range	-55°C to 150°C

† Unless otherwise noted, all voltages are with respect to V_{SS}.

Caution:

Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the "Recommended Operating Conditions" section of this specification is not implied. Exposure to absolute maximum rated conditions for extended periods may affect device reliability.

Table 4-13. Recommended Operating Conditions†

			MIN	NOM	MAX	UNIT
V_{CC}	Supply voltage		4.5	5	5.5	V
V_{PP}	Program supply voltage‡		20.5	21	21.5	V
V_{IH}	High-level input voltage	CLKIN	2.6			V
		All other inputs	2.0			V
V_{IL}	Low-level input voltage	CLKIN			0.6	V
		All other inputs			0.8	V
T_A	Operating free-air temperature		0		70	°C

† Ambient light may affect operational functionality and electrical characteristics. It is recommended to use an opaque label over the window when the EPROM is not being erase.

‡ V_{PP} is applied to the MC pin in EPROM mode only.

Table 4-14. Electrical Characteristics over Full Range of Operating Conditions†

PARAMETER			TEST CONDITIONS	MIN	TYP‡	MAX	UNIT
I_I	Input current	A5, MC, $\overline{\text{RESET}}$, $\overline{\text{INT1}}$, $\overline{\text{INT3}}$, XTAL2	$V_I = V_{SS}$ to V_{CC}		±2	±10	µA
		Ports C and D A0–A4, A6, A7	V_I = 0.4 V to V_{CC}		±10	±100	
C_I	Input capacitance				2		pF
V_{OH}	High-level output voltage		I_{OH} = -400 µA	2.4	2.8		V
V_{OL}	Low-level output voltage		I_{OL} = 3.2 mA		0.2	0.4	V
$t_{r(O)}$	Output rise time§		See Figure 4-13		9	50	ns
$t_{f(O)}$	Output fall time§		See Figure 4-13		10	60	ns
I_{CC}	Supply current		All outputs open		180	250	mA
I_{PP}	Program supply current		$\overline{E} = V_{IL}$, $\overline{G} = V_{PP}$			30	mA
$P_{D(av)}$	Average power dissipation		All outputs open		900	1375	mW

† Ambient light may affect operational functionality and electrical characteristics. It is recommended to use an opaque label over the window when the EPROM is not being erased.
‡ All typical values are at V_{CC} = 5 V, T_A = 25°C.
§ Rise and fall times are measured between the maximum low level and the minimum high level using the 10% and 90% points. Measured outputs have 100-pF loads to V_{SS}.

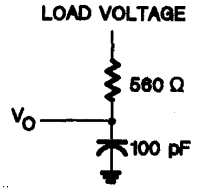


Figure 4-13. Output Loading Circuit for Test

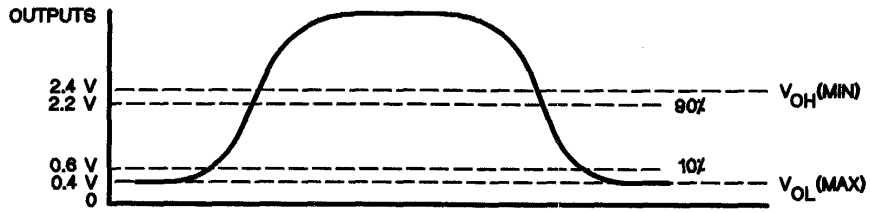


Figure 4-14. Measurement Points for Switching Characteristics

Table 4-15. Recommended Crystal Operating Conditions over Full Operating Range

PARAMETER		MIN	TYP	MAX	UNIT
f_{osc}	Crystal frequency	1		5	MHz
	CLKIN duty cycle		50		%
$t_{c(P)}$	Crystal cycle time	200		1000	ns
$t_{c(C)}$	Internal state cycle time	400		2000	ns
$t_{w(PH)}$	CLKIN pulse duration high	90			ns
$t_{w(PL)}$	CLKIN pulse duration low	90			ns
t_r	CLKIN rise time†			30	ns
t_f	CLKIN fall time†			30	ns
$t_{d(PH-CH)}$	CLKIN rise to CLKOUT rise delay		120	200	ns

† Rise and fall times are measured between the maximum low level and the minimum high level.

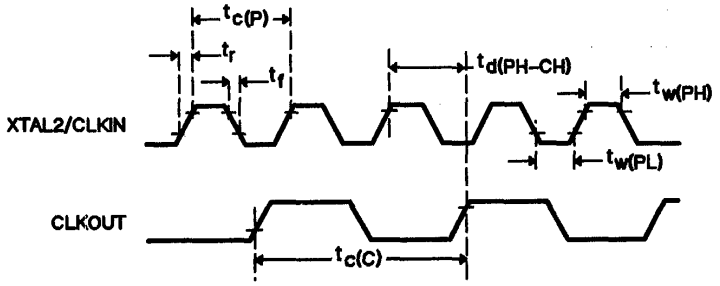


Figure 4-15. Clock Timing

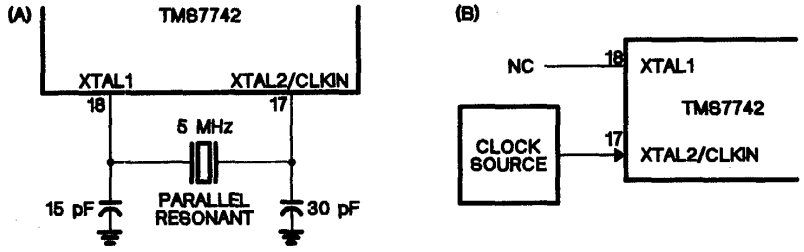


Figure 4-16. Recommended Clock Connections

Table 4-16. Memory Interface Timing

PARAMETER		MIN	MAX	UNIT
$t_{c(C)}$	CLKOUT cycle time†	400	2000	ns
$t_{w(CH)}$	CLKOUT high pulse duration	$0.5t_{c(C)}-40$	$0.5t_{c(C)}+10$	ns
$t_{w(CL)}$	CLKOUT low pulse duration	$0.5t_{c(C)}-40$	$0.5t_{c(C)}+15$	ns
$t_{d(CH-JL)}$	Delay time, CLKOUT rise to ALATCH fall	$0.5t_{c(C)}-10$	$0.5t_{c(C)}+30$	ns
$t_{w(JH)}$	ALATCH high pulse duration	$0.25t_{c(C)}-15$	$0.25t_{c(C)}+30$	ns
$t_{su(HA-JL)}$	Setup time, high address valid before ALATCH fall	$0.25t_{c(C)}-40$	$0.25t_{c(C)}+45$	ns
$t_{su(LA-JL)}$	Setup time, low address valid before ALATCH fall	$0.25t_{c(C)}-45$	$0.25t_{c(C)}+15$	ns
$t_{h(JL-LA)}$	Hold time, low address valid after ALATCH fall	$0.25t_{c(C)}$	$0.25t_{c(C)}+45$	ns
$t_{su(RW-JL)}$	Setup time, R/$\overline{W}$ valid before ALATCH fall	$0.25t_{c(C)}-35$	$0.25t_{c(C)}+30$	ns
$t_{h(EH-RW)}$	Hold time, R/$\overline{W}$ valid after $\overline{ENABLE}$ rise	$0.5t_{c(C)}-40$		ns
$t_{h(EH-HA)}$	Hold time, high address valid after $\overline{ENABLE}$ rise	$0.5t_{c(C)}-50$		ns
$t_{su(Q-EH)}$	Setup time, data output valid before $\overline{ENABLE}$ rise	$0.5t_{c(C)}-45$		ns
$t_{h(EH-Q)}$	Hold time, data output valid after $\overline{ENABLE}$ rise	$0.5t_{c(C)}-45$		ns
$t_{d(LA-EL)}$	Delay time, low address high impedance to $\overline{ENABLE}$ fall	$0.25t_{c(C)}-45$	$0.25t_{c(C)}+15$	ns
$t_{d(EH-A)}$	Delay time, $\overline{ENABLE}$ rise to next address drive	$0.5t_{c(C)}-25$		ns
$t_{a(EL-D)}$	Access time, data input valid after $\overline{ENABLE}$ fall	$0.75t_{c(C)}-135$		ns
$t_{a(A-D)}$	Access time, address valid to data input valid	$1.5t_{c(C)}-160$		ns
$t_{d(A-EH)}$	Delay time, address valid to $\overline{ENABLE}$ rise	$1.5t_{c(C)}-80$	$1.5t_{c(C)}+30$	ns
$t_{h(EH-D)}$	Hold time, data input valid after $\overline{ENABLE}$ rise	0		ns
$t_{d(EH-JH)}$	Delay time, $\overline{ENABLE}$ rise to ALATCH rise	$0.5t_{c(C)}-70$	$0.5t_{c(C)}+10$	ns
$t_{d(CH-EL)}$	Delay time, CLKOUT rise to $\overline{ENABLE}$ fall	-10	35	ns

† $t_{c(C)}$ is defined to be $2/f_{osc}$ and may may be referred to as a machine state or simply a state.

Table 4-17. Memory Interface Timing at 5 MHz

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
$t_{c(C)}$	CLKOUT cycle time†	f = 5 MHz, 50% duty cycle		400		ns
$t_{w(CH)}$	CLKOUT high pulse duration		160	185	210	ns
$t_{w(CL)}$	CLKOUT low pulse duration		160	190	215	ns
$t_{d(CH-JL)}$	Delay time, CLKOUT rise to ALATCH fall		190	210	230	ns
$t_{w(JH)}$	ALATCH high pulse duration		85	110	130	ns
$t_{su(HA-JL)}$	Setup time, high address valid before ALATCH fall		60	100	145	ns
$t_{su(LA-JL)}$	Setup time, low address valid before ALATCH fall		55	90	125	ns
$t_{h(JL-LA)}$	Hold time, low address valid after ALATCH fall		100	125	145	ns
$t_{su(RW-JL)}$	Setup time, R/$\overline{W}$ valid before ALATCH fall		65	95	130	ns
$t_{h(EH-RW)}$	Hold time, R/$\overline{W}$ valid after $\overline{ENABLE}$ rise		160	195		ns
$t_{h(EH-HA)}$	Hold time, high address valid after $\overline{ENABLE}$ rise		150	195		ns
$t_{su(Q-EH)}$	Setup time, data output valid before $\overline{ENABLE}$ rise		155	185		ns
$t_{h(EH-Q)}$	Hold time, data output valid after $\overline{ENABLE}$ rise		155	180		ns
$t_{d(LA-EL)}$	Delay time, low address high impedance to $\overline{ENABLE}$ fall		55	85	115	ns
$t_{d(EH-A)}$	Delay time, $\overline{ENABLE}$ rise to next address drive		175	205		ns
$t_{a(EL-D)}$	Access time, data input valid after $\overline{ENABLE}$ fall		165	205		ns
$t_{a(A-D)}$	Access time, address valid to data input valid		440	485		ns
$t_{d(A-EH)}$	Delay time, address valid to $\overline{ENABLE}$ rise		520	575	630	ns
$t_{h(EH-D)}$	Hold time, data input valid after $\overline{ENABLE}$ rise		0			ns
$t_{d(EH-JH)}$	Delay time, $\overline{ENABLE}$ rise to ALATCH rise		130	160	210	ns
$t_{d(CH-EL)}$	Delay time, CLKOUT rise to $\overline{ENABLE}$ fall		-10	25	35	ns

† $t_{c(C)}$ is defined to be $2/f_{osc}$ and may be referred to as a machine state or simply a state.

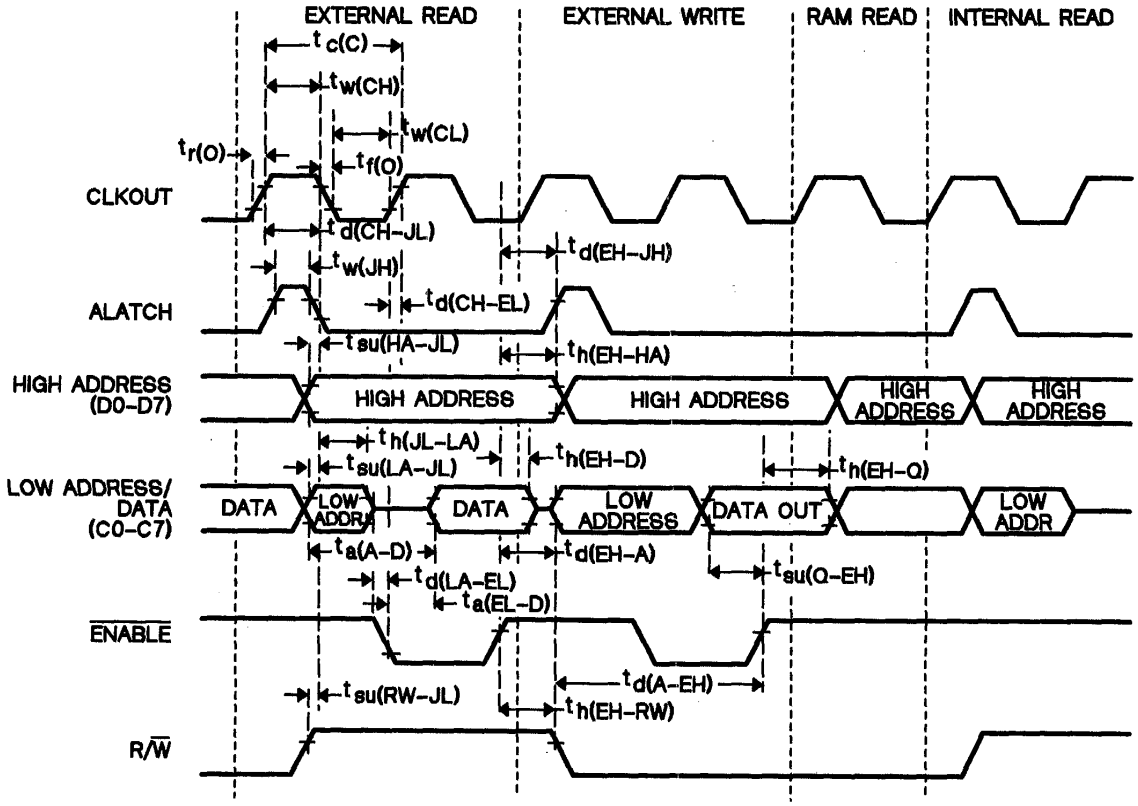


Figure 4-17. Read and Write Cycle Timing

4.3.1 Erasure

The TMS7742 is erased by exposing the chip to shortwave ultraviolet light that has a wavelength of 253.7 nanometers (2537 angstroms). The recommended minimum exposure dose (UV intensity × exposure time) is fifteen watt-seconds per square centimeter. The lamp should be located about 2.5 centimeters (1 inch) above the chip during erasure. After erasure, all bits are at a high level. Note that normal ambient light contains the correct wavelength fore erasure. Therefore, when using the TMS7742, the window should be covered with an opaque label.

Table 4-18. Switching Characteristics over Recommended Supply Voltage Range and Operating Free-Air Temperature Range

PARAMETER		TEST CONDITIONS†	MIN	MAX	UNITS
$t_{a(A)}$	Access time from address	CL = 100 pF,		1	µs
$t_{en(G)}$	Output enable time from $\overline{G}$	1 Series 74 TTL load,		350	ns
$t_{dis(G)}$‡	Output disable time from $\overline{G}$	$t_r \leq 20$ ns		350	ns
$t_{v(A)}$	Output data valid time after change of address, $\overline{E}$ or $\overline{G}$, whichever occurs first	$t_f \leq 20$ ns	0		ns

† Timing measurement reference levels for inputs and outputs are 0.8 V and 2 V.
‡ Value calculated from 0.5 V delta to measured output level.

Table 4-19. Recommended Conditions for Programming, TA = 25°C

		MIN	NOM	MAX	UNITS
$t_{w(E)}$	$\overline{E}$ pulse duration	9	10	11	ms
$t_{su(A)}$	Address setup time	2			µs
$t_{su(D)}$	Data setup time	2			µs
$t_{su(VPP)}$	V_{PP} setup time	2			µs
$t_{h(A)}$	Address hold time	0			µs
$t_{h(D)}$	Data hold time	2			µs
$t_{h(VPP)}$	V_{PP} hold time	2			µs
$t_{rec(PG)}$	V_{PP} recovery time	2			µs
$t_{r(PG)G}$	$\overline{G}$ rise time during programming	50			ns
t_{EHD}	Delay time, data valid after $\overline{E}$ flow			1	µs

Table 4-20. Programming Characteristics, TA = 25°C

PARAMETER		TEST CONDITIONS†	MIN	MAX	UNITS
$t_{dis(PR)}$	Output disable time		0	100	ns

† Timing measurement reference levels for inputs and outputs are 0.8 and 2.0 V.

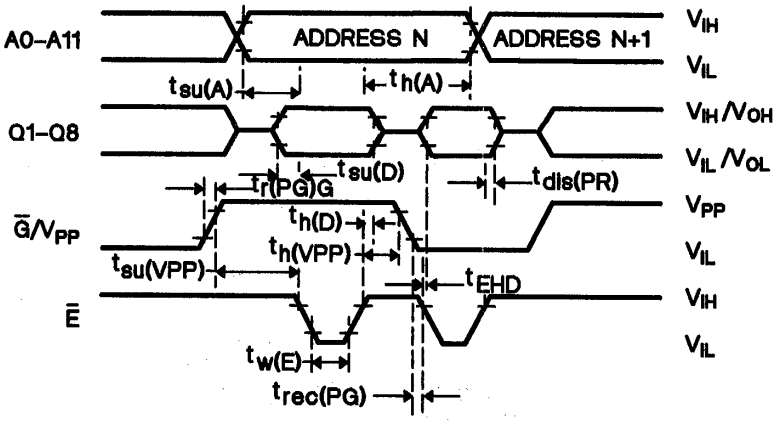


Figure 4-18. Program Cycle Timing

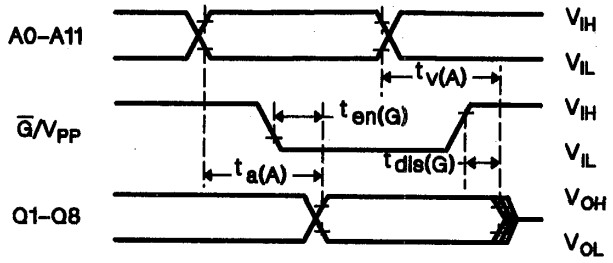
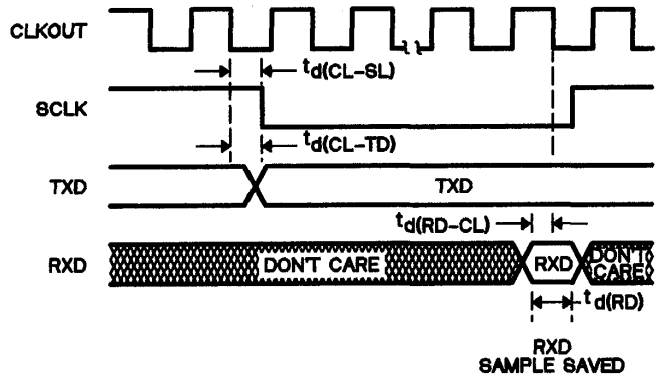


Figure 4-19. Read Cycle Timing

4.3.2 Serial Port Timing

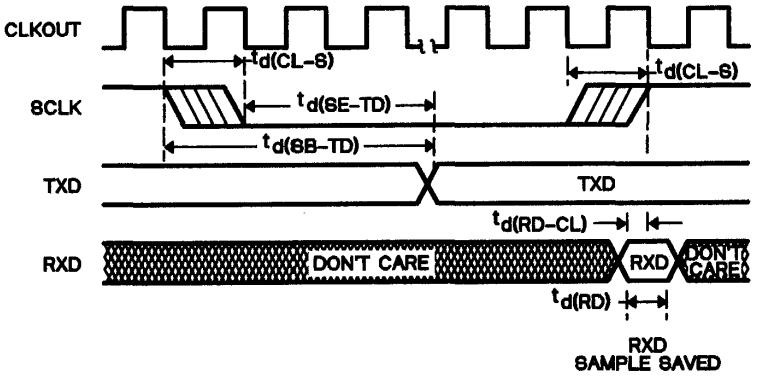
4.3.2.1 Internal Serial Clock



Notes: 1) The CLKOUT signal is not available in Single-Chip mode.
2) CLKOUT = $t_{c(C)}$.

PARAMETER		TYP	UNIT
$t_{d(CL-SL)}$	CLKOUT low to SCLK low	1/4 $t_{c(C)}$	ns
$t_{d(CL-TD)}$	CLKOUT low to new TXD data	1/4 $t_{c(C)}$	ns
$t_{d(RD-CL)}$	RXD data valid before CLKOUT low	1/4 $t_{c(C)}$	ns
$t_{d(RD)}$	RXD data valid time	1/2 $t_{c(C)}$	ns

4.3.2.2 External Serial Clock



Notes: 1) The CLKOUT signal is not available in Single-Chip mode.
2) CLKOUT = $t_{c(C)}$.
3) SCLK sampled; if SCLK = 1 then 0, fall transition found.
4) SCLK sampled; if SCLK = 0 then 1, rise transition found.

PARAMETER		TYP	UNIT
$t_{d(RD-CL)}$	RXD data valid before CLKOUT low	1/4 $t_{c(C)}$	ns
$t_{d(RD)}$	RXD data valid time	1/2 $t_{c(C)}$	ns
$t_{d(SB-TD)}$	Start of SCLK sample to new TXD data	3 1/4 $t_{c(C)}$	ns
$t_{d(SE-TD)}$	End of SCLK sample to new TXD data	2 1/4 $t_{c(C)}$	ns
$t_{d(CL-S)}$	Clockout low to SCLK transition	$t_{c(C)}$	ns

4.4 SE70P162 Specifications

Table 4-21. Absolute Maximum Ratings over Operating Free-Air Temperature Range (unless otherwise noted)

Supply voltage range, V_{CC}† ..... -0.3 V to 7 V
Input voltage range ..... -0.3 V to 7 V
Output voltage range ..... -.03 V to 7 V
Continuous power dissipation ..... 1.4 W
Maximum buffer current ..... ±10 mA
Storage temperature range ..... 0°C to 100°C

† Unless otherwise noted, all voltages are with respect to V_{SS}.

Caution:

Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions indicated in the "Recommended Operating Conditions" section of this specification is not implied. Exposure to absolute maximum rated conditions for extended periods may affect device reliability.

Table 4-22. Recommended Operating Conditions

			MIN	NOM	MAX	UNIT
V_{CC}	Supply voltage		4.5	5	5.5	V
V_{IH}	High-level input voltage	CLKIN	2.6			V
		All others	2.3			V
V_{IL}	Low-level input voltage	CLKIN			0.6	V
		All others			0.8	V
T_A	Operating free-air temperature		0		55	°C

Table 4-23. Electrical Characteristics over Full Range of Recommended Operating Conditions

PARAMETER			TEST CONDITIONS	MIN	TYP†	MAX	UNIT
I_I	Input current	A5,MC,$\overline{\text{RESET}}$, $\overline{\text{INT1}}$, $\overline{\text{INT3}}$, XTAL2	V_I = V_{SS} to V_{CC}		±2	±10	µA
		Ports C and D A0–A4, A6, A7	V_I = 0.4 V to V_{CC}		±10	±100	
V_{OH}	High-level output voltage		I_{OH} = -0.4 mA	2.4			V
V_{OL}	Low-level output voltage		I_{OL} = 2 mA			0.4	V
$t_{r(O)}$	Output rise time‡		See Figure 4-20		9	30	ns
$t_{f(O)}$	Output fall time‡		See Figure 4-20		10	35	ns
I_{CC}	Average supply current§		All outputs open		160	210	mA
$P_{D(av)}$	Average power dissipation		All outputs open		800	1155	mW

† All typical values are at V_{CC} = 5 V, T_A = 25°C.
‡ Rise and fall times are measured between the maximum low level and the minimum high level using the 10% and 90% points (see Figure 4-21).
§ Average supply current without piggyback EPROM device installed.

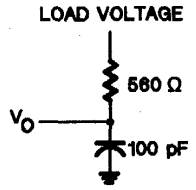


Figure 4-20. Output Loading Circuit for Test

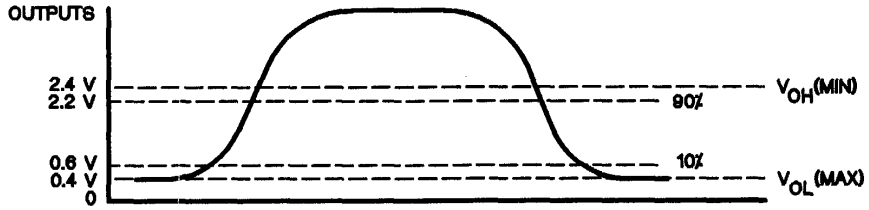


Figure 4-21. Measurement Points for Switching Characteristics

Table 4-24. Recommended Crystal/Clockin Operating Conditions over Full Operating Range

	PARAMETER	MIN	TYP	MAX	UNIT
f_{osc}	Crystal frequency	1.0		8.0	MHz
	CLKIN duty cycle		50		%
$t_{c(P)}$	Crystal cycle time	125		1000	ns
$t_{c(C)}$	Internal state cycle time	250		2000	ns
$t_{w(PH)}$	CLKIN pulse duration high	50			ns
$t_{w(PL)}$	CLKIN pulse duration low	50			ns
t_r	CLKIN rise time†			30	ns
t_f	CLKIN fall time†			30	ns
$t_{d(PH-CH)}$	CLKIN rise to CLKOUT rise delay		125	200	ns

† Rise and fall times are measured betwen the maximum low level and the minimum high level.

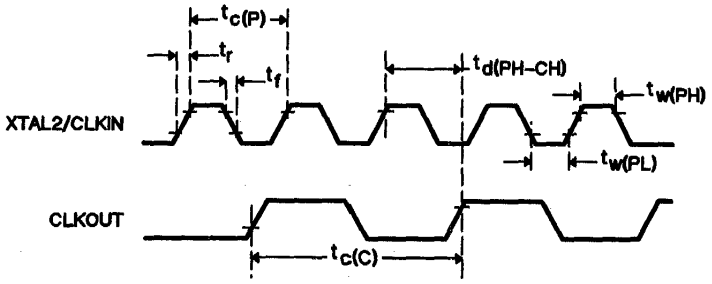


Figure 4-22. Clock Timing

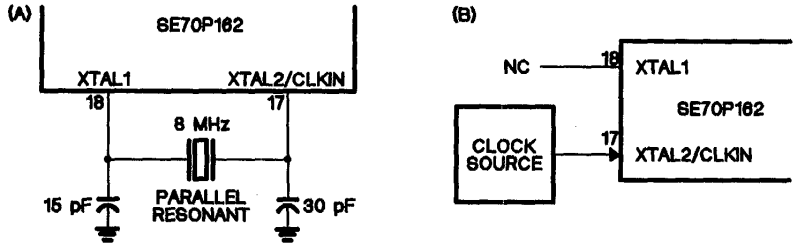


Figure 4-23. Recommended Clock Connections

Table 4-25. Memory Interface Timing

PARAMETER		MIN	MAX	UNIT
$t_{c(C)}$	CLKOUT cycle time†	250	2000	ns
$t_{w(CH)}$	CLKOUT high pulse duration	$0.5t_{c(C)}$-40	$0.5t_{c(C)}$+10	ns
$t_{w(CL)}$	CLKOUT low pulse duration	$0.5t_{c(C)}$-40	$0.5t_{c(C)}$+15	ns
$t_{d(CH-JL)}$	Delay time, CLKOUT rise to ALATCH fall	$0.5t_{c(C)}$-10	$0.5t_{c(C)}$+30	ns
$t_{w(JH)}$	ALATCH high pulse duration	$0.25t_{c(C)}$-15	$0.25t_{c(C)}$+30	ns
$t_{su(HA-JL)}$	Setup time, high address valid before ALATCH fall	$0.25t_{c(C)}$-40	$0.25t_{c(C)}$+45	ns
$t_{su(LA-JL)}$	Setup time, low address valid before ALATCH fall	$0.25t_{c(C)}$-40	$0.25t_{c(C)}$+15	ns
$t_{h(JL-LA)}$	Hold time, low address valid after ALATCH fall	$0.25t_{c(C)}$	$0.25t_{c(C)}$+45	ns
$t_{su(RW-JL)}$	Setup time, R/$\overline{W}$ valid before ALATCH fall	$0.25t_{c(C)}$-35	$0.25t_{c(C)}$+30	ns
$t_{h(EH-RW)}$	Hold time, R/$\overline{W}$ valid after $\overline{ENABLE}$ rise	$0.5t_{c(C)}$-40		ns
$t_{h(EH-HA)}$	Hold time, high address valid after $\overline{ENABLE}$ rise	$0.5t_{c(C)}$-50		ns
$t_{su(Q-EH)}$	Setup time, data output valid before $\overline{ENABLE}$ rise	$0.5t_{c(C)}$-45		ns
$t_{h(EH-Q)}$	Hold time, data output valid after $\overline{ENABLE}$ rise	$0.5t_{c(C)}$-45		ns
$t_{d(LA-EL)}$	Delay time, low address high impedance to $\overline{ENABLE}$ fall	$0.25t_{c(C)}$-45	$0.25t_{c(C)}$	ns
$t_{d(EH-A)}$	Delay time, $\overline{ENABLE}$ rise to next address drive	$0.5t_{c(C)}$-25		ns
$t_{a(EL-D)}$	Access time, data input valid after $\overline{ENABLE}$ fall	$0.75t_{c(C)}$-105		ns
$t_{a(A-D)}$	Access time, address valid to data input valid	$1.5t_{c(C)}$-115		ns
$t_{d(A-EH)}$	Delay time, address valid to $\overline{ENABLE}$ rise	$1.5t_{c(C)}$-80	$1.5t_{c(C)}$+30	ns
$t_{h(EH-D)}$	Hold time, data input valid after $\overline{ENABLE}$ rise	0		ns
$t_{d(EH-JH)}$	Delay time, $\overline{ENABLE}$ rise to ALATCH rise	$0.5t_{c(C)}$-25	$0.5t_{c(C)}$+10	ns
$t_{d(CH-EL)}$	Delay time, CLKOUT rise to $\overline{ENABLE}$ fall	-10	35	ns

† $t_{c(C)}$ is defined to be $2/f_{osc}$ and may be referred to as a machine state or simply a state.

Note: For memory interface timings at 8 MHz, see Table 4-11 on page 4-12.

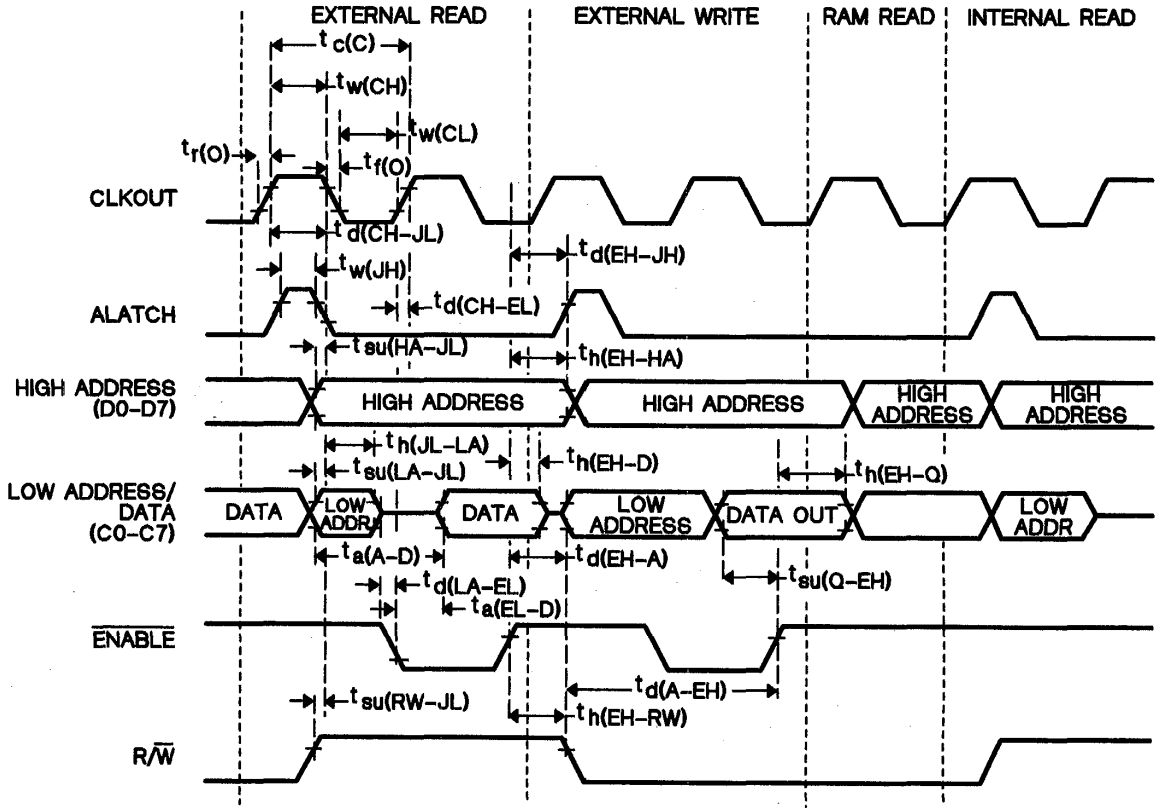
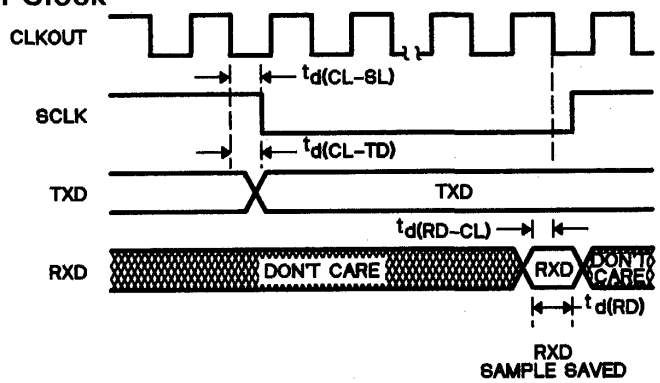


Figure 4-24. Read and Write Cycle Timings

4.4.1 Serial Port Timing

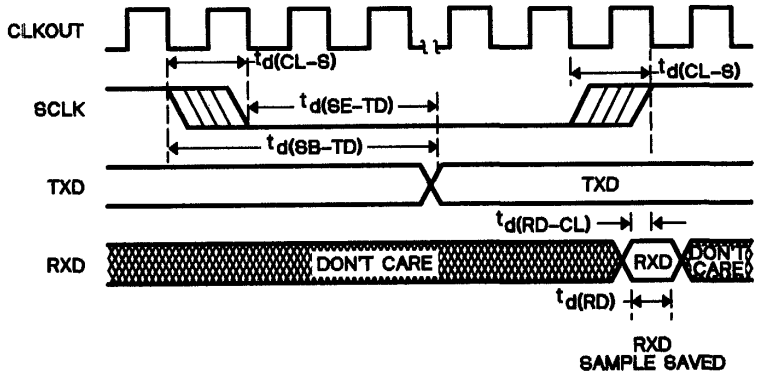
4.4.1.1 Internal Serial Clock



Notes: 1) The CLKOUT signal is not available in Single-Chip mode.
2) CLKOUT = $t_{c(C)}$.

PARAMETER		TYP	UNIT
$t_{d(CL-SL)}$	CLKOUT low to SCLK low	1/4 $t_{c(C)}$	ns
$t_{d(CL-TD)}$	CLKOUT low to new TXD data	1/4 $t_{c(C)}$	ns
$t_{d(RD-CL)}$	RXD data valid before CLKOUT low	1/4 $t_{c(C)}$	ns
$t_{d(RD)}$	RXD data valid time	1/2 $t_{c(C)}$	ns

4.4.1.2 External Serial Clock



Notes: 1) The CLKOUT signal is not available in Single-Chip mode.
2) CLKOUT = $t_{c(C)}$.
3) SCLK sampled; if SCLK = 1 then 0, fall transition found.
4) SCLK sampled; if SCLK = 0 then 1, rise transition found.

PARAMETER		TYP	UNIT
$t_{d(RD-CL)}$	RXD data valid before CLKOUT low	1/4 $t_{c(C)}$	ns
$t_{d(RD)}$	RXD data valid time	1/2 $t_{c(C)}$	ns
$t_{d(SB-TD)}$	Start of SCLK sample to new TXD data	3 1/4 $t_{c(C)}$	ns
$t_{d(SE-TD)}$	End of SCLK sample to new TXD data	2 1/4 $t_{c(C)}$	ns
$t_{d(CL-S)}$	Clockout low to SCLK transition	$t_{c(C)}$	ns

4.5 TMS70C00A, TMS70C20A, and TMS70C40A Specifications (Wide Voltage)

Table 4-26. Absolute Maximum Rating over Operating Free-Air Temperature Range (unless otherwise noted)

Supply voltage, V_{CC}† ..... -0.3 V to 7 V
All input voltages ..... -0.3 V to V_{CC} + 0.3 V
All output voltages ..... -0.3 V to V_{CC} + 0.3 V
Maximum I/O buffer current ..... ±10 mA
Storage temperature range ..... -55°C to 150°C
I_{CC}, I_{SS} current (maximum into pins 25 and 40) ..... ±60 mA

† Unless otherwise noted, all voltages are with respect to V_{SS}.

Caution:

Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the "Recommended Operating Conditions" section of this specification is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

Table 4-27. Recommended Operating Conditions

			MIN	NOM	MAX	UNIT
V_{CC}	Supply voltage		2.5		6.0	V
V_{IH}	High-level input voltage	XTAL2 pin, V_{CC} = 2.5 to 6 V	0.8V_{CC}			V
		All other pins, V_{CC} = 3 to 6 V	0.70V_{CC}			V
		All other pins, V_{CC} = 2.5 to 3 V	0.75V_{CC}			V
V_{IL}	Low-level input voltage	XTAL2 pin, V_{CC} = 2.5 to 6 V			0.2V_{CC}	V
		All other pins, V_{CC} = 2.5 to 6 V			0.3V_{CC}	V
T_A	Operating temperature range	Commercial (TMS70Cx0NL)	0		70	°C
		Industrial (TMS70Cx0NA)	-40		85	°C

Table 4-28. Electrical Characteristics over Full Range of Operating Conditions

PARAMETER		TEST CONDITIONS	MIN	TYP†	MAX	UNIT
I_I	Input leakage current	$V_{IN} = V_{SS}$ to V_{CC}		±0.1	±1	µA
C_I	Input capacitance			5		pF
V_{OH}	High-level output voltage‡	V_{CC} = 2.5 V, I_{OH} = -50 µA	2.25	2.4		V
		V_{CC} = 4.0 V, I_{OH} = -0.4 mA	3.2	3.6		V
		V_{CC} = 5.0 V, I_{OH} = -0.7 mA	3.9	4.5		V
		V_{CC} = 6.0 V, I_{OH} = -1.0 mA	4.6	5.4		V
V_{OL}	Low-level output voltage‡	V_{CC} = 2.5 V, I_{OL} = 0.4 mA		0.2	0.35	V
		V_{CC} = 4.0 V, I_{OL} = 1.6 mA		0.4	0.8	V
		V_{CC} = 5.0 V, I_{OL} = 2.5 mA		0.6	1.1	V
		V_{CC} = 6.0 V, I_{OH} = 3.4 mA		0.8	1.4	V
I_{OH}	Output source current	V_{CC} = 2.5 V, V_{OH} = 2.25 V	-0.05	-0.2		mA
		V_{CC} = 4.0 V, V_{OH} = 3.2 V	-0.4	-1.4		mA
		V_{CC} = 5.0 V, V_{OH} = 3.9 V	-0.7	-2.2		mA
		V_{CC} = 6.0 V, V_{OH} = 4.6 V	-1.0	-3.3		mA
I_{OL}	Output sink current	V_{CC} = 2.5 V, V_{OH} = 0.35 V	0.4	0.9		mA
		V_{CC} = 4.0 V, V_{OH} = 0.8 V	1.6	3.5		mA
		V_{CC} = 5.0 V, V_{OH} = 1.1 V	2.5	5.5		mA
		V_{CC} = 6.0 V, V_{OH} = 1.4 V	3.4	8.0		mA

† V_{CC} = 5 V, T_A = 25°C
‡ Output levels ensure 400 mV of noise margin over specified input levels.

Table 4-29. Supply Current Requirements

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
I_{CC}	Operating mode	f_{osc} = 6.0 MHz, V_{CC} = 5 V		9.0	14.4	mA
		f_{osc} = 3.0 MHz, V_{CC} = 5 V		4.5	7.2	mA
		f_{osc} = 0.5 MHz, V_{CC} = 5 V		0.8	1.2	mA
		f_{osc} = Z MHz, V_{CC} = 5 V		1.5	2.4	mA/MHz
		f_{osc} = 0.5 MHz, V_{CC} = 2.5 V		370	800	µA
I_{CC}	Wake-Up mode (timer active)	f_{osc} = 6.0 MHz, V_{CC} = 5 V		960	1920	µA
		f_{osc} = 3.0 MHz, V_{CC} = 5V		480	960	µA
		f_{osc} = 0.5 MHz, V_{CC} = 5 V		80	160	µA
		f_{osc} = Z MHz, V_{CC} = 5 V		160	320	µA/MHz
		f_{osc} = 0.5 MHz, V_{CC} = 2.5 V		40	80	µA
I_{CC}	Halt osc-on	f_{osc} = 6.0 MHz, V_{CC} = 5 V		480	980	µA
		f_{osc} = 3.0 MHz, V_{CC} = 5 V		240	500	µA
		f_{osc} = 0.5 MHz, V_{CC} = 5 V		45	100	µA
		f_{osc} = Z MHz V_{CC} = 5 V		See Note 2		µA
		f_{osc} = 0.5 MHz, V_{CC} = 2.5 V		25	60	µA
I_{CC}	Halt osc-off	V_{CC} = 2.5 to 6 V		1	10	µA

Notes: 1. All inputs = V_{CC} or V_{SS} (except XTAL2). All output pins are open.
2. Maximum current = 160(Z) + 20 µA

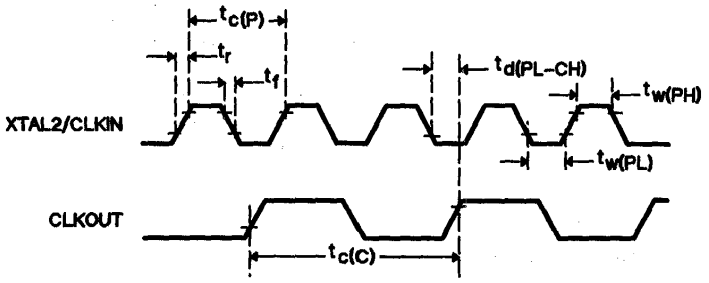


Figure 4-25. Clock Timing

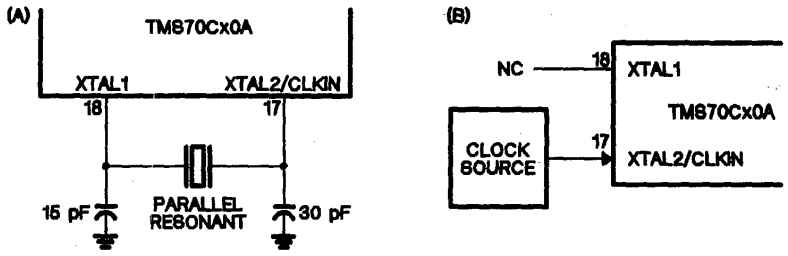


Figure 4-26. Recommended Clock Connections

Table 4-30. Recommended Crystal/Clockin Operating Conditions over Full Operating Range

PARAMETER		TEST CONDITIONS	MIN	TYP†	MAX	UNIT
f_{osc}	Crystal frequency	V_{CC} = 2.5 V	0.5		0.8	MHz
		V_{CC} = 4.0 V	0.5		4.0	MHz
		V_{CC} = 5.0 V	0.5		6.0	MHz
		V_{CC} = 6.0 V	0.5		6.5	MHz
	CLKIN duty cycle		45		55	%
$t_{c(P)}$	Crystal cycle time	V_{CC} = 2.5 V	1250		2000	ns
		V_{CC} = 4.0 V	250		2000	ns
		V_{CC} = 5.0 V	166		2000	ns
		V_{CC} = 6.0 V	153		2000	ns
$t_{c(C)}$	Internal state cycle time	V_{CC} = 2.5 V	2500		4000	ns
		V_{CC} = 4.0 V	500		4000	ns
		V_{CC} = 5.0 V	333		4000	ns
		V_{CC} = 6.0 V	306		4000	ns
$t_{w(PH)}$	CLKIN pulse duration high		70			ns
$t_{w(PL)}$	CLKIN pulse duration low		70			ns
t_r	CLKIN rise time				30	ns
t_f	CLKIN fall time				30	ns
$t_{d(PL-CH)}$	CLKIN fall to CLKOUT rise delay			110	250	ns

† V_{CC} = 5 V, T_A = 25°C

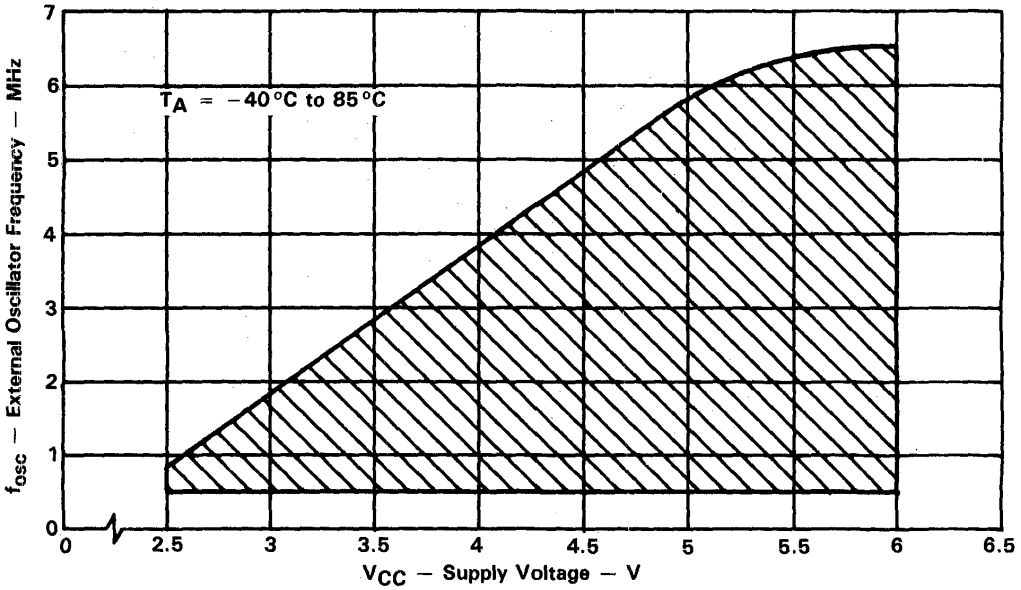


Figure 4-27. Operating Frequency Range

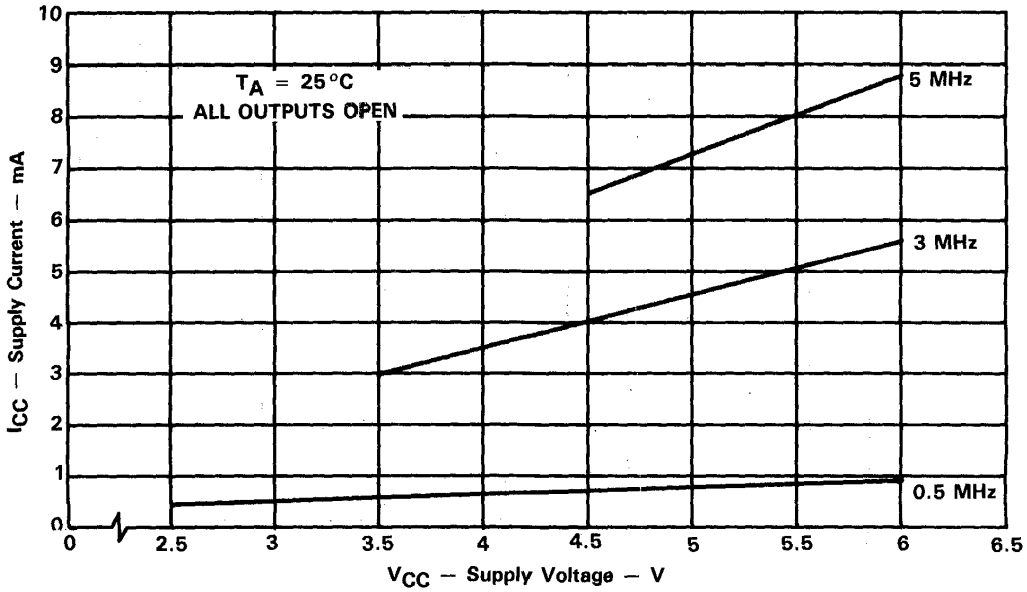


Figure 4-28. Typical Operating Current vs. Supply Voltage

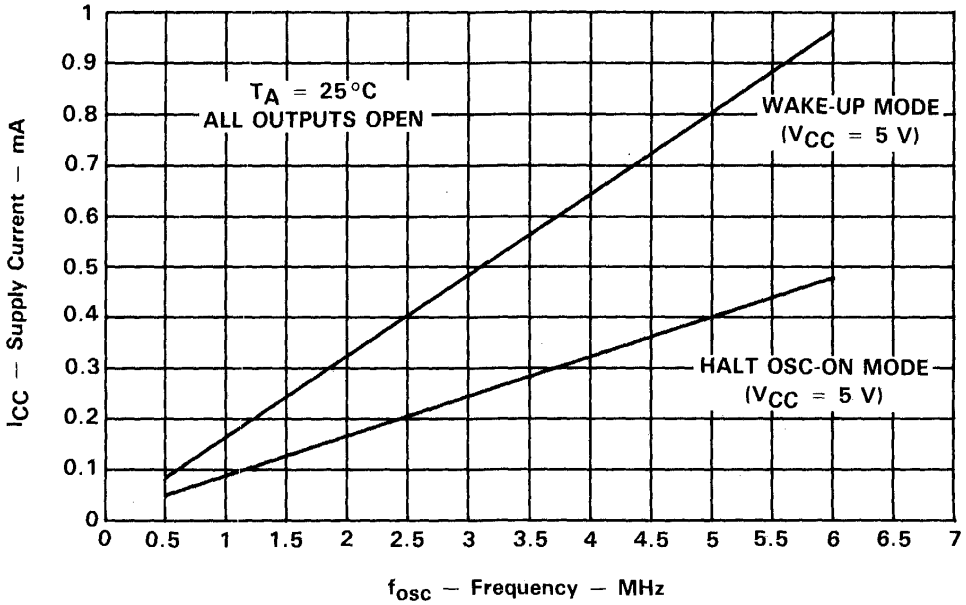


Figure 4-29. Typical Power-Down Current vs. Oscillator Frequency

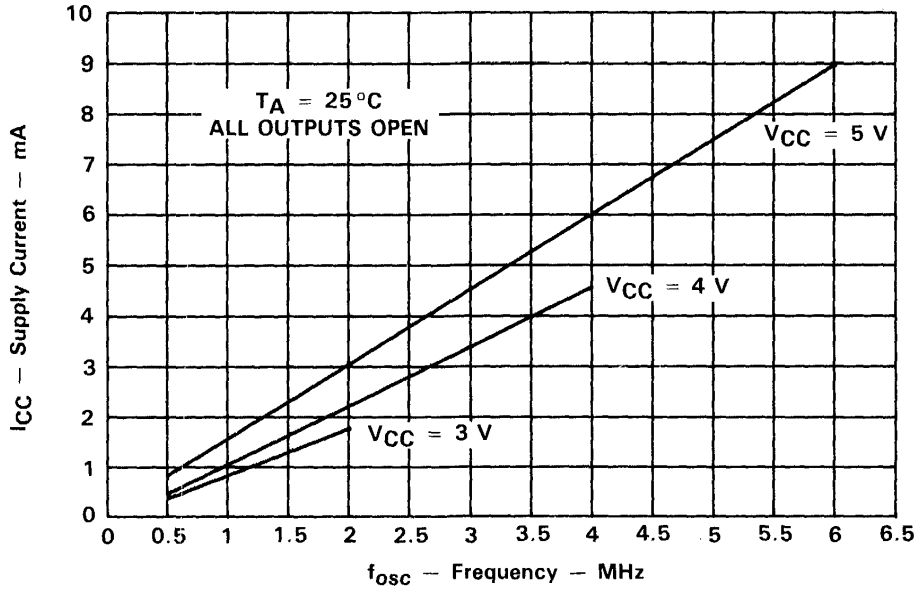


Figure 4-30. Typical Operating ICC vs. Oscillator Frequency

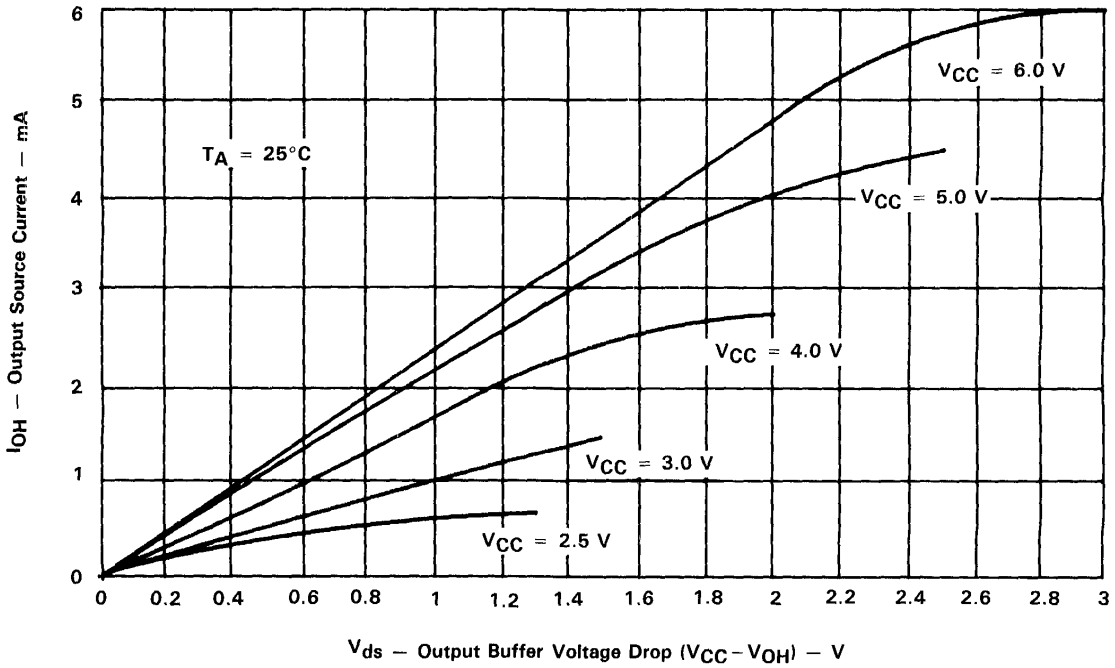


Figure 4-31. Typical Output Source Characteristics

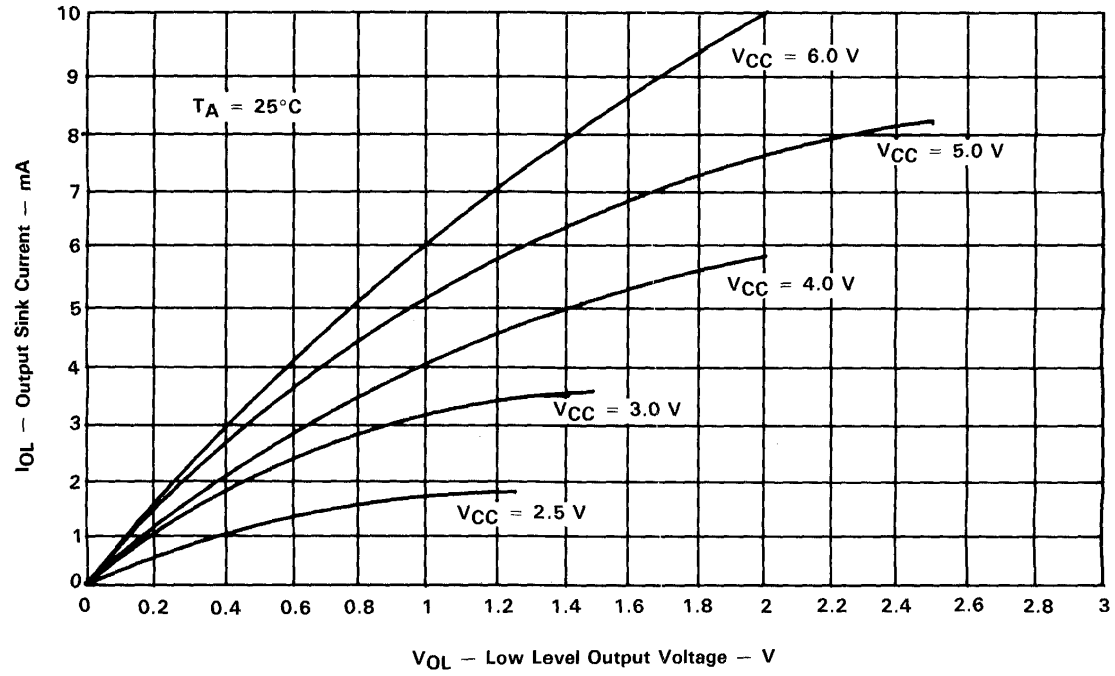


Figure 4-32. Typical Output Sink Characteristics

4.6 TMS70C00A, TMS70C20A, and TMS70C40A Specifications (5V ±10%)

Table 4-31. Absolute Maximum Rating over Operating Free-Air Temperature Range (unless otherwise noted)

Supply voltage, V_{CC}† ..... -0.3 V to 7 V
All input voltages ..... -0.3 V to V_{CC} + 0.3 V
All output voltages ..... -0.3 V to V_{CC} + 0.3 V
Maximum I/O buffer current ..... ±10 mA
Storage temperature range ..... -55°C to 150°C
I_{CC}, I_{SS} current (maximum into pins 25 and 40) ..... ±60 mA

† Unless otherwise noted, all voltages are with respect to V_{SS}.

Caution:

Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the "Recommended Operating Conditions" section of this specification is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

Table 4-32. Recommended Operating Conditions

			MIN	NOM	MAX	UNIT
V_{CC}	Supply voltage		4.5		5.5	V
V_{IH}	High-level input voltage	XTAL2 pin	$0.8V_{CC}$			V
		All other pins	$0.7V_{CC}$			V
V_{IL}	Low-level input voltage	XTAL2 pin			$0.2V_{CC}$	V
		All other pins			$0.3V_{CC}$	V
T_A	Operating temperature range	Commercial (TMS70Cx0NL)	0		70	°C
		Industrial (TMS70Cx0NA)	-40		85	°C

Table 4-33. Electrical Characteristics over Full Range of Operating Conditions

PARAMETER		TEST CONDITIONS	MIN	TYP†	MAX	UNIT
I_I	Input leakage current	$V_{IN} = V_{SS}$ to V_{CC}		±0.1	±1	µA
C_I	Input capacitance			5		pF
V_{OH}	High-level output voltage	I_{OH} = -0.3 mA	V_{CC}-0.5	4.7		V
V_{OL}	Low-level output voltage	I_{OL} = 1.4 mA		0.2	0.4	V
I_{OH}	High-level output source current	$V_{OH} = V_{CC}$ - 0.5 V	-0.3	-1.2		mA
		V_{OH} = 2.5 V min	-1.0	-3.0		mA
I_{OL}	Output sink current	V_{OL} = 0.4 V	1.4	2.0		mA

† V_{CC} = 5 V, T_A = 25°C

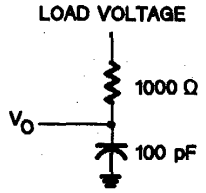


Figure 4-33. Output Loading Circuit for Test

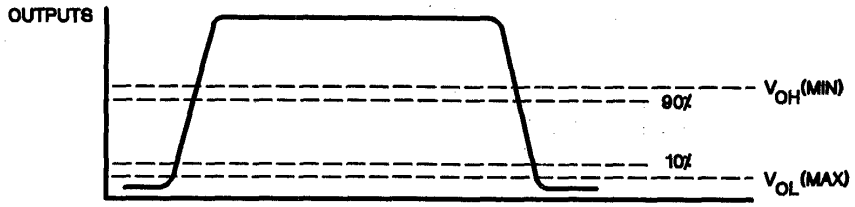


Figure 4-34. Measurement Points for Switching Characteristics

Table 4-34. AC Characteristics for I/O Ports

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNITS
t_r	I/O port output rise time	C_{load}= 15 pF,V_{CC} = 5 V		35	60	ns
t_f	I/O port output fall time	C_{load}= 15 pF,V_{CC} = 5 V		20	50	ns

Note: Rise and fall times are measured between the maximum low level and the minimum high level using the 10% and 90% points.

Table 4-35. Supply Current Requirements

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
I_{CC}	Operating mode	f_{osc} = 5.0 MHz		7.5	13.5	mA
		f_{osc} = 3.0 MHz		4.5	8.1	mA
		f_{osc} = 1.0 MHz		1.5	2.7	mA
		f_{osc} = Z MHz		1.5	2.7	mA/MHz
I_{CC}	Wake-Up mode (timer active)	f_{osc} = 5.0 MHz		800	1750	µA
		f_{osc} = 3.0 MHz		480	1050	µA
		f_{osc} = 1.0 MHz		160	350	µA
		f_{osc} = Z MHz		160	350	µA/MHz
I_{CC}	Halt osc-on	f_{osc} = 5.0 MHz		480	920	µA
		f_{osc} = 3.0 MHz		240	560	µA
		f_{osc} = 1.0 MHz		80	200	µA
		f_{osc} = Z MHz		See Note 2		µA
I_{CC}	Halt osc-off			1	10	µA

Notes: 1. All inputs = V_{CC} or V_{SS} (except XTAL2). All output pins are open.
2. Maximum current = 180(Z) + 20 µA.

Table 4-36. Recommended Crystal/Clockin Operating Conditions over Full Operating Range

PARAMETER		MIN	TYP†	MAX	UNIT
f_{osc}	Crystal frequency	0.5		5.0	MHz
	CLKIN duty cycle	45		55	%
$t_{c(P)}$	Crystal cycle time	200		2000	ns
$t_{c(C)}$	Internal state cycle time	400		4000	ns
$t_{w(PH)}$	CLKIN pulse duration high	90			ns
$t_{w(PL)}$	CLKIN pulse duration low	90			ns
t_r	CLKIN rise time			30	ns
t_f	CLKIN fall time			30	ns
$t_{d(PL-CH)}$	CLKIN fall to CLKOUT rise delay		140	250	ns

† V_{CC} = 5 V, T_A = 25°C

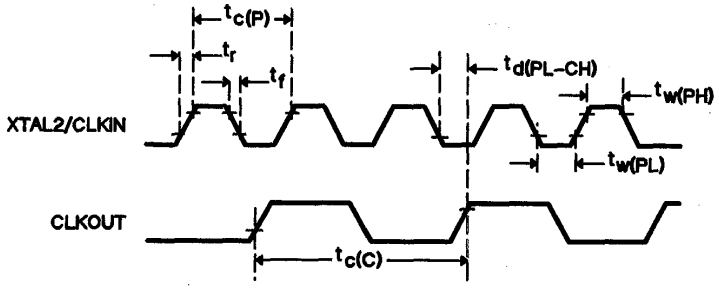


Figure 4-35. Clock Timing

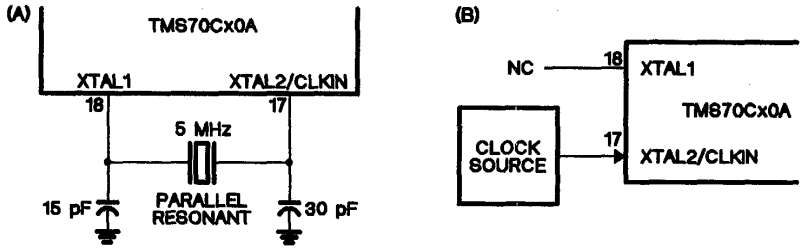


Figure 4-36. Recommended Clock Connections

Table 4-37. Memory Interface Timings†

PARAMETER		MIN	TYP	MAX	UNIT
$t_{c(C)}$	CLKOUT cycle time		$t_{c(C)}$		ns
$t_{w(CH)}$	CLKOUT high pulse duration	$0.5t_{c(C)}-90$		$0.5t_{c(C)}+90$	ns
$t_{w(CL)}$	CLKOUT low pulse duration	$0.5t_{c(C)}-90$		$0.5t_{c(C)}+90$	ns
$t_{d(CH-JL)}$	Delay time, CLKOUT rise to ALATCH fall	$0.75t_{c(C)}-50$			ns
$t_{w(JH)}$	ALATCH active duration	$0.5t_{c(C)}-15$			ns
$t_{su(HA-JL)}$	Setup time, high address valid before ALATCH fall	$0.5t_{c(C)}-100$			ns
$t_{su(LA-JL)}$	Setup time, low address valid before ALATCH fall	$0.5t_{c(C)}-100$			ns
$t_{h(JL-LA)}$	Hold time, low address hold after ALATCH fall	$0.5t_{c(C)}-60$			ns
$t_{su(RW-JL)}$	Setup time, R/$\overline{W}$ valid before ALATCH fall	$0.5t_{c(C)}-100$			ns
$t_{h(EH-RW)}$	Hold time, R/$\overline{W}$ after $\overline{ENABLE}$ rise	$0.25t_{c(C)}-60$			ns
$t_{h(EH-HA)}$	Hold time, high address valid after $\overline{ENABLE}$ rise	$0.25t_{c(C)}-60$			ns
$t_{d(Q-EH)}$	Delay time, data out valid before $\overline{ENABLE}$ rise	$0.75t_{c(C)}-70$			ns
$t_{h(EH-Q)}$	Hold time, data out valid after $\overline{ENABLE}$ rise	$0.25t_{c(C)}-30$			ns
$t_{d(EH-A)}$	Delay time, $\overline{ENABLE}$ rise to next address drive	$0.25t_{c(C)}-60$			ns
$t_{a(EL-D)}$	Access time, data in after $\overline{ENABLE}$ fall	$0.75t_{c(C)}-120$			ns
$t_{a(A-D)}$	Access time, data in from valid address	$1.5t_{c(C)}-200$			ns
$t_{d(A-EH)}$	Delay time, $\overline{ENABLE}$ high after valid address	$1.75t_{c(C)}-100$			ns
$t_{h(EH-D)}$	Hold time, data input valid after $\overline{ENABLE}$ rise	0			ns
$t_{d(CH-EL)}$	Delay time, CLKOUT rise to $\overline{ENABLE}$ fall	-10	35		ns

† V_{CC} = 4.5 to 5.5 V
CLKIN duty cycle = 50%

Table 4-38. Memory Interface Timings at 5 MHz†

PARAMETER		MIN	TYP	MAX	UNIT
$t_{c(C)}$	CLKOUT cycle time		400		ns
$t_{w(CH)}$	CLKOUT high pulse duration	110	200	290	ns
$t_{w(CL)}$	CLKOUT low pulse duration	110	200	290	ns
$t_{d(CH-JL)}$	Delay time, CLKOUT rise to ALATCH fall	250	300		ns
$t_{w(JH)}$	ALATCH active duration	185	200		ns
$t_{su(HA-JL)}$	Setup time, high address valid before ALATCH fall	100	200		ns
$t_{su(LA-JL)}$	Setup time, low address valid before ALATCH fall	100	200		ns
$t_{d(JL-LA)}$	Delay time, low address hold after ALATCH fall	140	200		ns
$t_{d(RW-JL)}$	Delay time, R/$\overline{W}$ valid before ALATCH fall	100	200		ns
$t_{h(EH-RW)}$	Hold time, R/$\overline{W}$ valid after $\overline{ENABLE}$ rise	40	100		ns
$t_{h(EH-HA)}$	Hold time, high address valid after $\overline{ENABLE}$ rise	40	100		ns
$t_{su(Q-EH)}$	Setup time, data out valid before $\overline{ENABLE}$ rise	230	300		ns
$t_{h(EH-Q)}$	Hold time, data out valid after $\overline{ENABLE}$ rise	70	100		ns
$t_{d(EH-A)}$	Delay time, $\overline{ENABLE}$ rise to next address drive	40	100		ns
$t_{d(EL-D)}$	Delay time, data in after $\overline{ENABLE}$ fall	180	300		ns
$t_{a(A-D)}$	Access time, data in from valid address	400	600		ns
$t_{d(A-EH)}$	Delay time, $\overline{ENABLE}$ high after address valid	600	700		ns
$t_{h(EH-D)}$	Hold time, data input valid after $\overline{ENABLE}$ rise	0			ns
$t_{d(CH-EL)}$	Delay time, CLKOUT rise to $\overline{ENABLE}$ fall	-10	35		ns

† V_{CC} = 4.5 to 5.5 V
CLKIN duty cycle = 50%

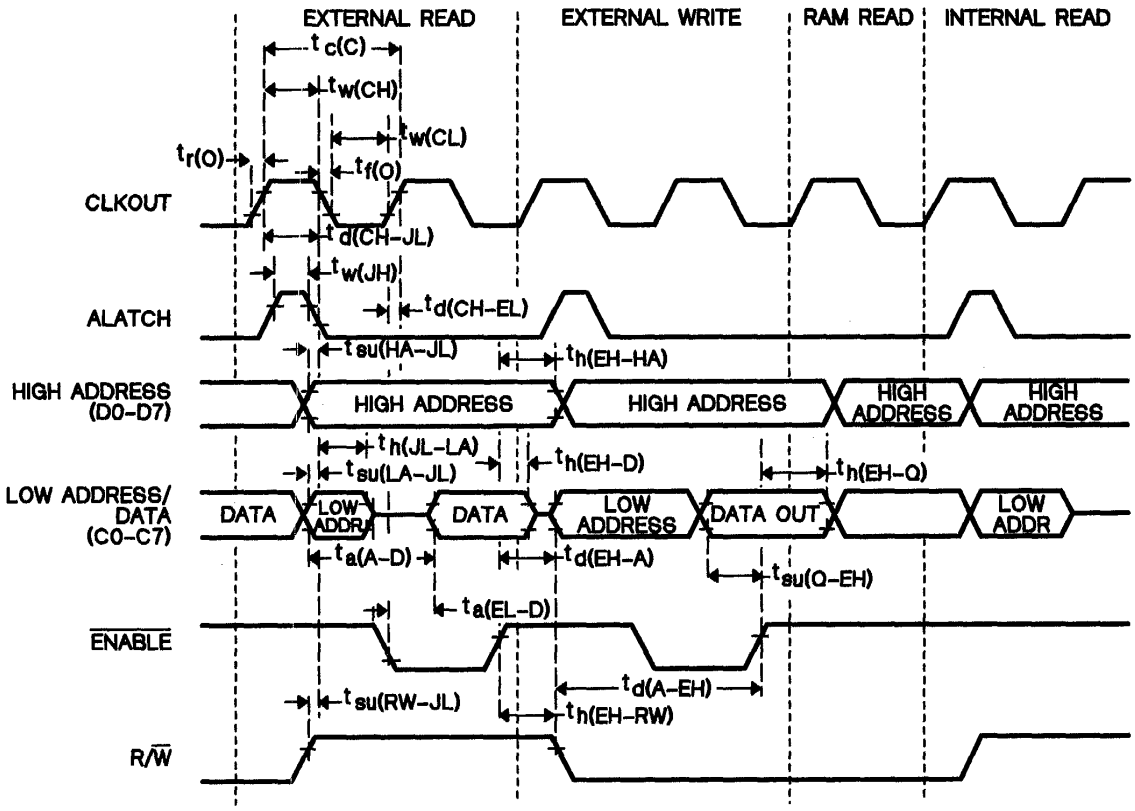


Figure 4-37. Read and Write Cycle Timing

4.7 TMS70C02 and TMS70C42 Specifications (Wide Voltage)

Table 4-39. Absolute Maximum Ratings over Operating Free-Air Temperature Range (unless otherwise noted)

Supply voltage range, V_{CC}† ..... -0.3 V to 7 V
Input voltage range ..... -0.3 V to V_{CC}+0.3 V
Output voltage range ..... -0.3 V to V_{CC}+0.3 V
Maximum I/O buffer current ..... ±10 mA
Storage temperature range ..... -55°C to 150°C
I_{CC}, I_{SS} (maximum into pin 25 or 40) ..... ±60 mA

† Unless otherwise noted, all voltages are with respect to V_{SS}.

Caution:

Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the "Recommended Operating Conditions" section of this specification is not implied. Exposure to absolute maximum rated conditions for extended periods may affect device reliability.

Table 4-40. Recommended Operating Conditions

			MIN	NOM	MAX	UNIT
V_{CC}	Supply voltage		2.5		6.0	V
V_{IH}	High-level input voltage	MC and XTAL2 pins, V_{CC} = 2.5 to 6 V	0.8V_{CC}			V
		All other input pins, V_{CC} = 3 to 6 V	0.70V_{CC}			V
		All other input pins, V_{CC} = 2.5 to 3 V	0.75V_{CC}			V
V_{IL}	Low-level input voltage	MC and XTAL2 pins, V_{CC} = 2.5 to 6 V			0.2V_{CC}	V
		All other input pins, V_{CC} = 2.5 to 6 V			0.3V_{CC}	V
T_A	Operating free-air temperature	Commercial (TMS70C42NL)	0		70	°C
		Industrial (TMS70C42NA)	-40		85	°C

Table 4-41. Electrical Characteristics over Full Range of Operating Conditions

PARAMETER		TEST CONDITIONS		MIN	TYP†	MAX	UNIT
I_I	Input current	MC pin, $V_{IN} = V_{SS}$ or V_{CC} All others, $V_{IN} = V_{SS}$ to V_{CC}			±0.1	±1	µA
C_I	Input capacitance				5		pF
V_{OH}	High-level output voltage‡	V_{CC} = 2.5 V,	I_{OH} = -50 µA	2.25	2.4		V
		V_{CC} = 4.0 V,	I_{OH} = -0.4 mA	3.2	3.6		V
		V_{CC} = 5.0 V,	I_{OH} = -0.7 mA	3.9	4.5		V
		V_{CC} = 6.0 V,	I_{OH} = -1.0 mA	4.6	5.4		V
V_{OL}	Low-level output voltage‡	V_{CC} = 2.5 V,	I_{OL} = 0.4 mA		0.2	0.35	V
		V_{CC} = 4.0 V,	I_{OL} = 1.6 mA		0.4	0.8	V
		V_{CC} = 5.0 V,	I_{OL} = 2.5 mA		0.6	1.1	V
		V_{CC} = 6.0 V,	I_{OL} = 3.4 mA		0.8	1.4	V
I_{OH}	Output source current	V_{CC} = 2.5 V,	V_{OH} = 2.25 V	-50	-200		µA
		V_{CC} = 4.0 V,	V_{OH} = 3.2 V	-0.4	-1.4		mA
		V_{CC} = 5.0 V,	V_{OH} = 3.9 V	-0.7	-2.2		mA
		V_{CC} = 6.0 V,	V_{OH} = 4.6 V	-1.0	-3.3		mA
I_{OL}	Output sink current	V_{CC} = 2.5 V,	V_{OH} = 0.35 V	0.4	0.9		mA
		V_{CC} = 4.0 V,	V_{OH} = 0.8 V	1.6	3.5		mA
		V_{CC} = 5.0 V,	V_{OH} = 1.1 V	2.5	5.5		mA
		V_{CC} = 6.0 V,	V_{OH} = 1.4 V	3.4	8.0		mA

† V_{CC} = 5 V, T_A = 25°C
‡ Output levels ensure 400 mV of noise margin over specified input levels.

Table 4-42. Supply Current Requirements

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
I_{CC}	Operating mode	f_{osc} = 7.0 MHz, V_{CC} = 5.0 V		17	24.5	mA
		f_{osc} = 3.0 MHz, V_{CC} = 5.0 V		7.2	10.5	mA
		f_{osc} = 0.5 MHz, V_{CC} = 5.0 V		1.2	1.8	mA
		f_{osc} = Z MHz, V_{CC} = 5.0 V		2.4	3.5	mA/MHz
		f_{osc} = 0.5 MHz, V_{CC} = 2.5 V		0.4	1.2	mA
I_{CC}	Wake-Up mode 1 (one timer and UART active)	f_{osc} = 7.0 MHz, V_{CC} = 5.0 V		2400	5600	µA
		f_{osc} = 3.0 MHz, V_{CC} = 5.0 V		1200	3300	µA
		f_{osc} = 0.5 MHz, V_{CC} = 5.0 V		250	800	µA
I_{CC}	Wake-Up mode 2 (one timer active, UART inactive)	f_{osc} = 7.0 MHz, V_{CC} = 5.0 V		960	3400	µA
		f_{osc} = 3.0 MHz, V_{CC} = 5.0 V		480	2000	µA
		f_{osc} = 0.5 MHz, V_{CC} = 5.0 V		140	550	µA
I_{CC}	Wake-Up mode 3 (UART active only)	f_{osc} = 7.0 MHz, V_{CC} = 5.0 V		1500	2400	µA
		f_{osc} = 3.0 MHz, V_{CC} = 5.0 V		800	1500	µA
		f_{osc} = 0.5 MHz, V_{CC} = 5.0 V		180	600	µA

Note: All inputs = V_{CC} or V_{SS} (except XTAL2). All output pins are open.

Table 4-43. Recommended Crystal/Clockin Operating Conditions over Full Operating Range

PARAMETER		TEST CONDITIONS	MIN	TYP†	MAX	UNIT
f_{osc}	Crystal frequency	V_{CC} = 2.5 V	0.5		0.8	MHz
		V_{CC} = 4.0 V	0.5		5.0	MHz
		V_{CC} = 5.0 V	0.5		7.0	MHz
		V_{CC} = 6.0 V	0.5		7.5	MHz
	CLKIN duty cycle		45		55	%
$t_{c(P)}$	Crystal cycle time	V_{CC} = 2.5 V	1250		2000	ns
		V_{CC} = 4.0 V	200		2000	ns
		V_{CC} = 5.0 V	143		2000	ns
		V_{CC} = 6.0 V	133		2000	ns
$t_{c(C)}$	Internal state cycle time	V_{CC} = 2.5 V	2500		4000	ns
		V_{CC} = 4.0 V	400		4000	ns
		V_{CC} = 5.0 V	286		4000	ns
		V_{CC} = 6.0 V	267		4000	ns
$t_{w(PH)}$	CLKIN pulse duration high		50			ns
$t_{w(PL)}$	CLKIN pulse duration low		50			ns
t_r	CLKIN rise time				30	ns
t_f	CLKIN fall time				30	ns
$t_{d(PL-CH)}$	CLKIN fall to CLKOUT rise delay			110	250	ns

† V_{CC} = 5 V, T_A = 25°C

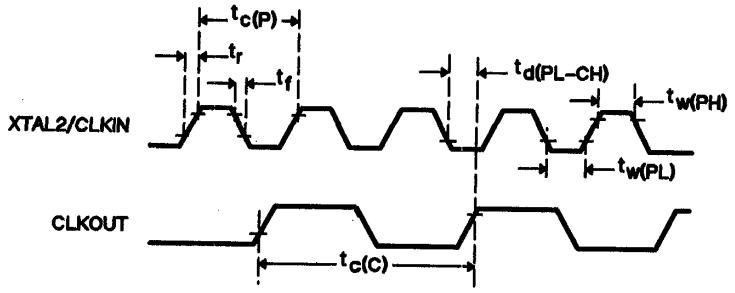


Figure 4-38. Clock Timing

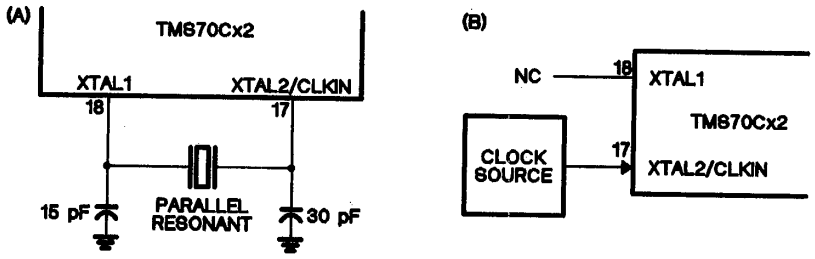


Figure 4-39. Recommended Clock Connections

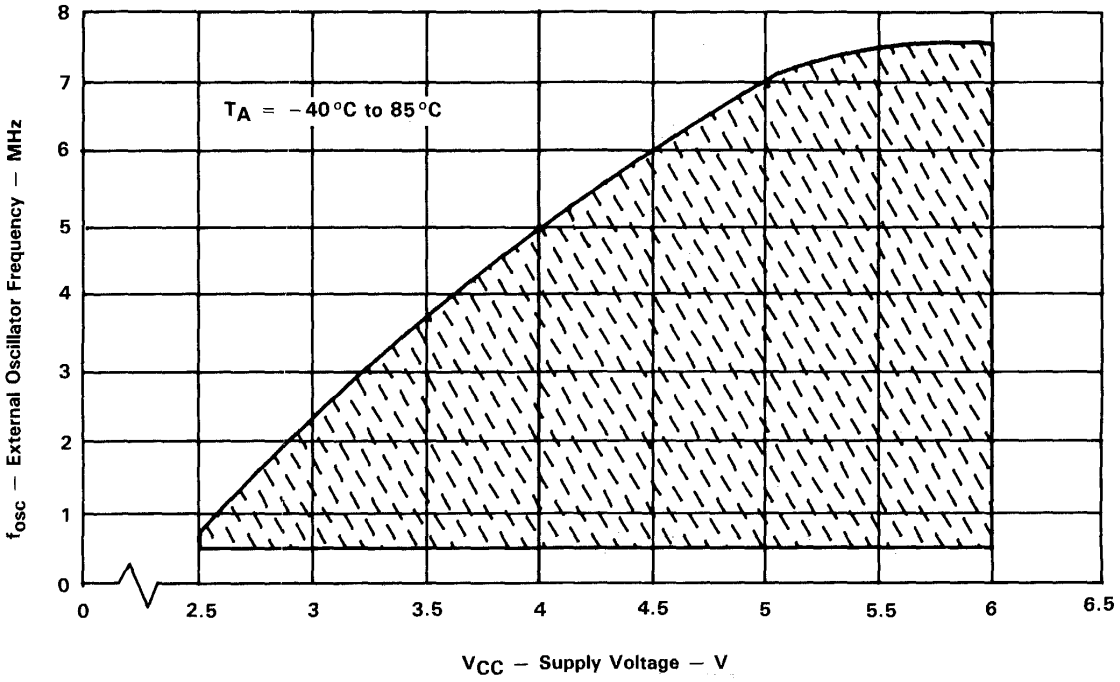


Figure 4-40. Operating Frequency Range

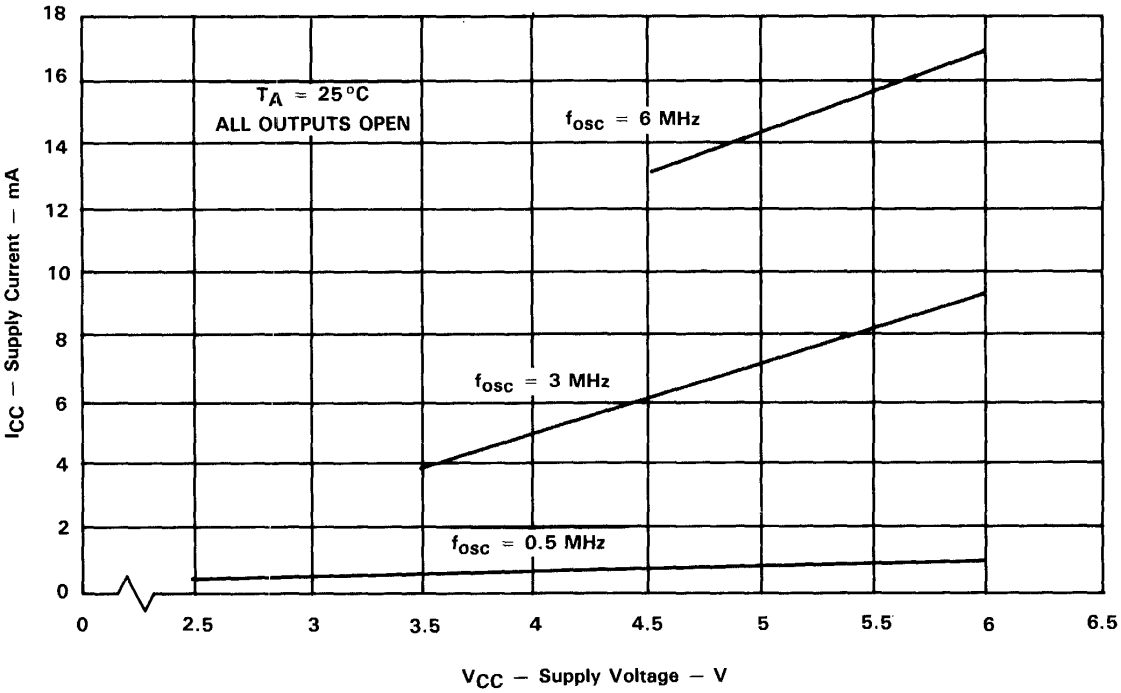


Figure 4-41. Typical Operating Current vs. Supply Voltage

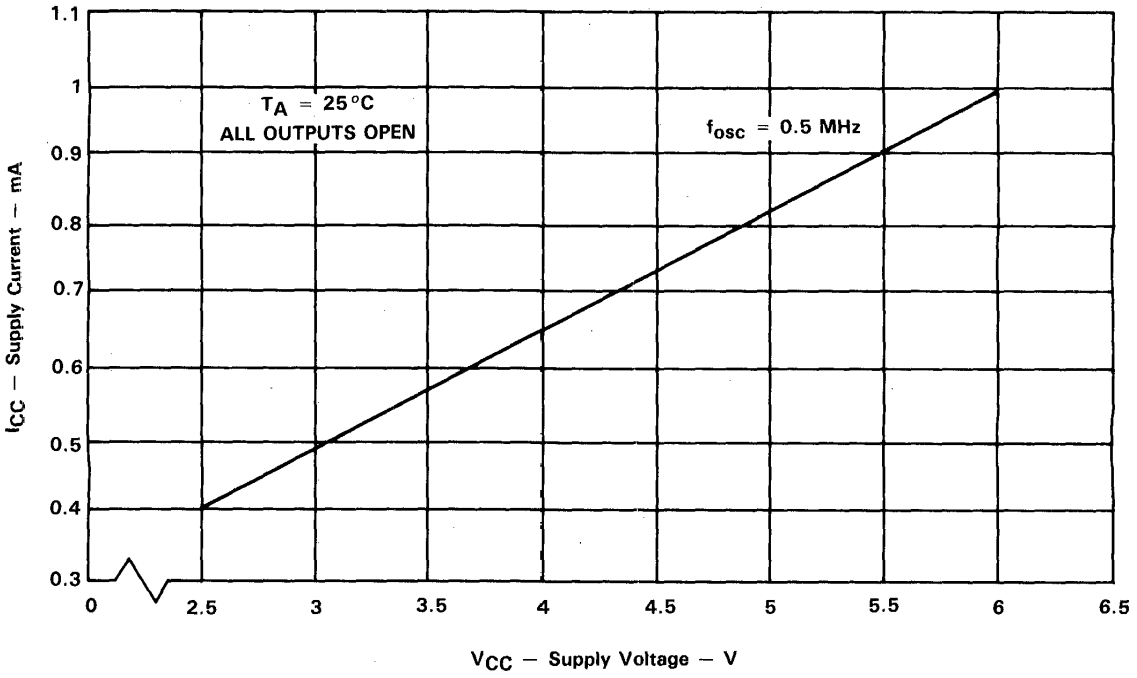


Figure 4-42. Typical Operating ICC vs. Oscillator Frequency

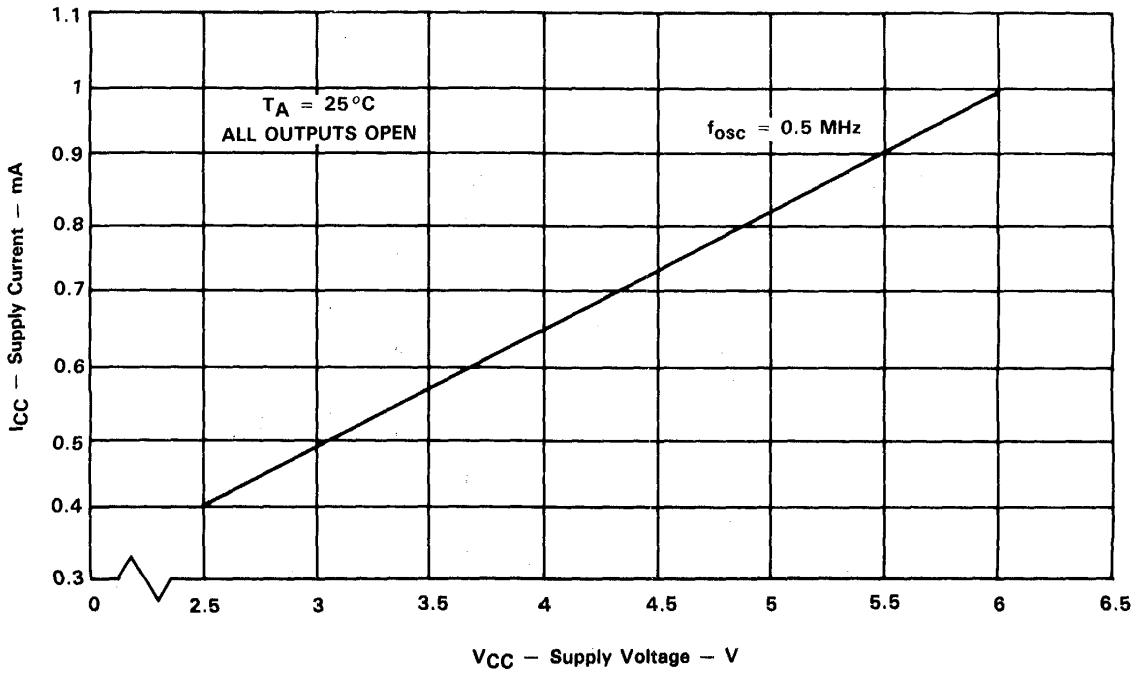


Figure 4-43. Typical Operating Current vs. Supply Voltage

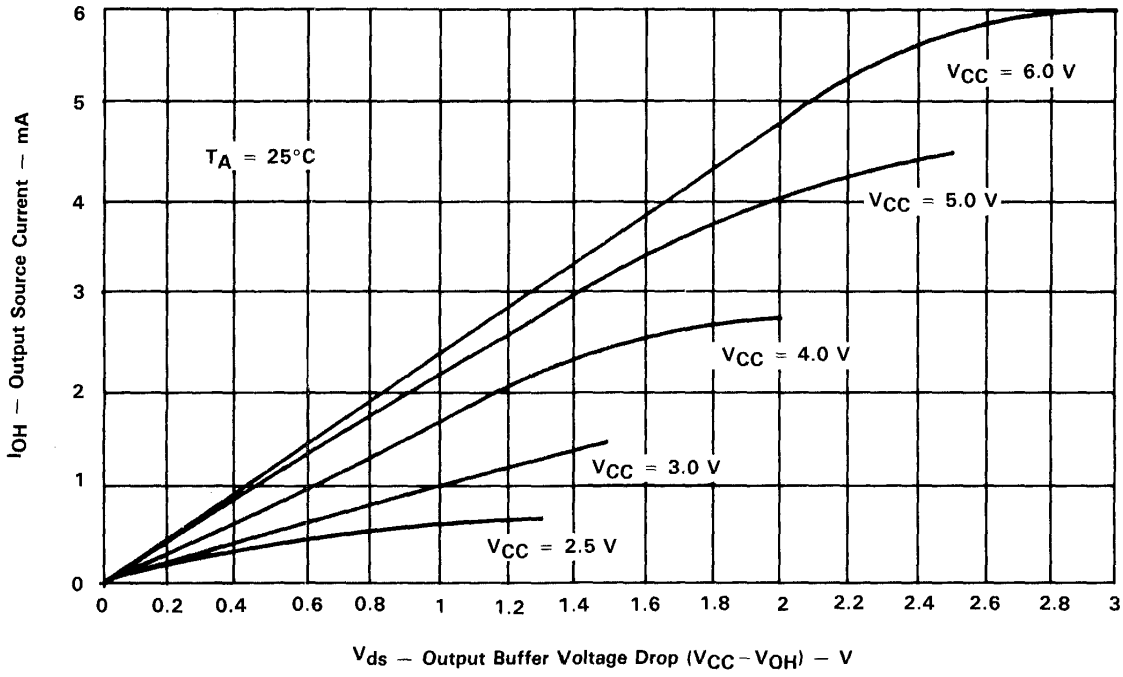


Figure 4-44. Typical Output Source Characteristics

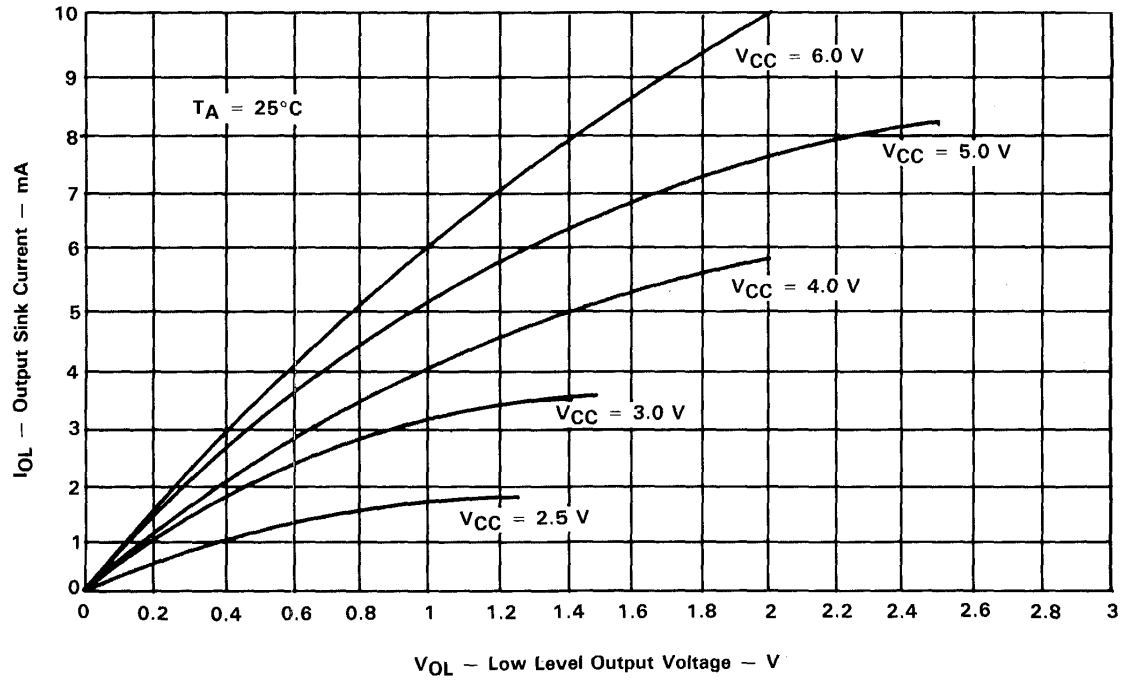
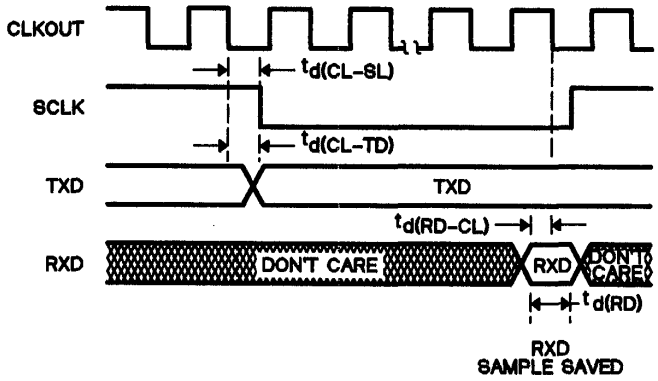


Figure 4-45. Typical Output Sink Characteristics

4.7.1 Serial Port Timing

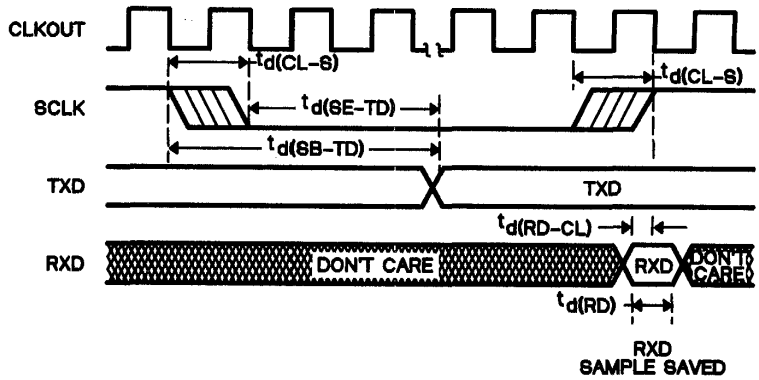
4.7.1.1 Internal Serial Clock



Notes: 1) The CLKOUT signal is not available in Single-Chip mode.
2) CLKOUT = $t_{c(C)}$.

PARAMETER		TYP	UNIT
$t_{d(CL-SL)}$	CLKOUT low to SCLK low	1/4 $t_{c(C)}$	ns
$t_{d(CL-TD)}$	CLKOUT low to new TXD data	1/4 $t_{c(C)}$	ns
$t_{d(RD-CL)}$	RXD data valid before CLKOUT low	1/4 $t_{c(C)}$	ns
$t_{d(RD)}$	RXD data valid time	1/2 $t_{c(C)}$	ns

4.7.1.2 External Serial Clock



Notes: 1) The CLKOUT signal is not available in Single-Chip mode.
2) CLKOUT = $t_{c(C)}$.
3) SCLK sampled; if SCLK = 1 then 0, fall transition found.
4) SCLK sampled; if SCLK = 0 then 1, rise transition found.

PARAMETER		TYP	UNIT
$t_{d(RD-CL)}$	RXD data valid before CLKOUT low	1/4 $t_{c(C)}$	ns
$t_{d(RD)}$	RXD data valid time	1/2 $t_{c(C)}$	ns
$t_{d(SB-TD)}$	Start of SCLK sample to new TXD data	3 1/4 $t_{c(C)}$	ns
$t_{d(SE-TD)}$	End of SCLK sample to new TXD data	2 1/4 $t_{c(C)}$	ns
$t_{d(CL-S)}$	Clockout low to SCLK transition	$t_{c(C)}$	ns

4.8 TMS70C02 and TMS70C42 Specifications (5V ±10%)

Table 4-44. Absolute Maximum Ratings over Operating Free-Air Temperature Range (unless otherwise noted)

Supply voltage range, V_{CC}† ..... -0.3 V to 7 V
Input voltage range ..... -0.3 V to V_{CC}+0.3 V
Output voltage range ..... -0.3 V to V_{CC}+0.3 V
Maximum I/O buffer current ..... ±10 mA
Storage temperature range ..... -55°C to 150°C
I_{CC}, I_{SS} (maximum into pin 25 or 40) ..... ±60 mA

† Unless otherwise noted, all voltages are with respect to V_{SS}.

Caution:

Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the "Recommended Operating Conditions" section of this specification is not implied. Exposure to absolute maximum rated conditions for extended periods may affect device reliability.

Table 4-45. Recommended Operating Conditions

			MIN	NOM	MAX	UNIT
V_{CC}	Supply voltage		4.5		5.5	V
V_{IH}	High-level input voltage	MC and XTAL2 pins	$0.8V_{CC}$			V
		All other input pins	$0.7V_{CC}$			V
V_{IL}	Low-level input voltage	MC and XTAL2 pins			$0.3V_{CC}$	V
		All other input pins			$0.2V_{CC}$	V
T_A	Operating temperature	Commercial (TMS70C42NL)	0		70	°C
		Industrial (TMS70C42NA)	-40		85	°C

Table 4-46. Electrical Characteristics over Full Range of Operating Conditions

PARAMETER		TEST CONDITIONS	MIN	TYP†	MAX	UNIT
I_I	Input leakage current	MC pin, $V_{IN} = V_{SS}$ or V_{CC} All others, $V_{IN} = V_{SS}$ to V_{CC}		±0.1	±1	µA
C_I	Input capacitance			5		pF
V_{OH}	High-level output voltage	I_{OH} = -0.3 mA	V_{CC}-0.5	4.7		V
V_{OL}	Low-level output voltage	I_{OL} = 1.4 mA		0.2	0.4	V
I_{OH}	High-level output source current	$V_{OH} = V_{CC}$ - 0.5 V	-0.3	-1.2		mA
		V_{OH} = 2.5 V min	-1.0	-3.0		mA
I_{OL}	Output sink current	V_{OL} = 0.4 V	1.4	2.0		mA

Table 4-47. AC Characteristics for Input/Output Ports†

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
$t_{r(IO)}$	I/O port output rise time	C_{load} = 15 pF, V_{CC} = 5 V		35	60	ns
$t_{f(IO)}$	I/O port output fall time	C_{load} = 15 pF, V_{CC} = 5 V		20	50	ns

† Rise and fall times are measured between the maximum low level and the miniumum high level using the 10% and 90% points.

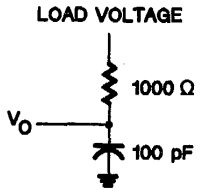


Figure 4-46. Output Loading Circuit for Test

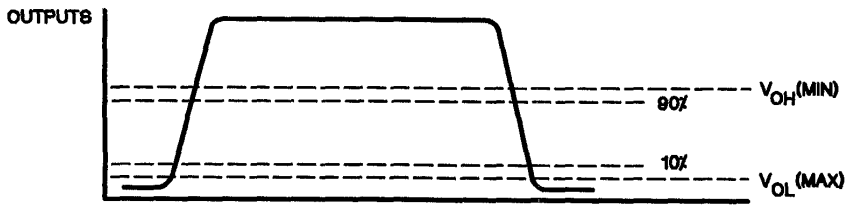


Figure 4-47. Measurement Points for Switching Characteristics

Table 4-48. Supply Current Requirements

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
I_{CC}	Supply current	f_{osc} = 6.0 MHz		15	24	mA
		f_{osc} = 3.0 MHz		7.2	12	mA
		f_{osc} = 1.0 MHz		2.4	4.0	mA
		f_{osc} = Z MHz		2.4	4.0	mA/MHz
I_{CC}	Wake-Up mode 1 (one timer and UART active)	f_{osc} = 6.0 MHz		2400	5400	µA
		f_{osc} = 3.0 MHz		1200	2900	µA
		f_{osc} = 1.0 MHz		650	1500	µA
I_{CC}	Wake-Up mode 2 (one timer active, UART inactive)	f_{osc} = 6.0 MHz		960	3200	µA
		f_{osc} = 3.0 MHz		480	1800	µA
		f_{osc} = 1.0 MHz		350	1000	µA
I_{CC}	Wake-Up mode 3 (UART active only)	f_{osc} = 6.0 MHz		1500	2200	µA
		f_{osc} = 3.0 MHz		800	1300	µA
		f_{osc} = 1.0 MHz		400	1100	µA

Note: All inputs = V_{CC} or V_{SS} (except XTAL2). All output pins are open.

Table 4-49. Recommended Crystal/Clockin Operating Conditions over Full Operating Range

PARAMETER		MIN	TYP†	MAX	UNIT
f_{osc}	Crystal frequency	0.5		6.0	MHz
	CLKIN duty cycle	45		55	%
$t_{c(P)}$	Crystal cycle time	167		2000	ns
$t_{c(C)}$	Internal state cycle time	333		4000	ns
$t_{w(PH)}$	CLKIN pulse duration high	70			ns
$t_{w(PL)}$	CLKIN pulse duration low	70			ns
t_r	CLKIN rise time			30	ns
t_f	CLKIN fall time			30	ns
$t_{d(PL-CH)}$	CLKIN fall to CLKOUT rise delay		110	250	ns

† V_{CC} = 5 V, T_A = 25°C

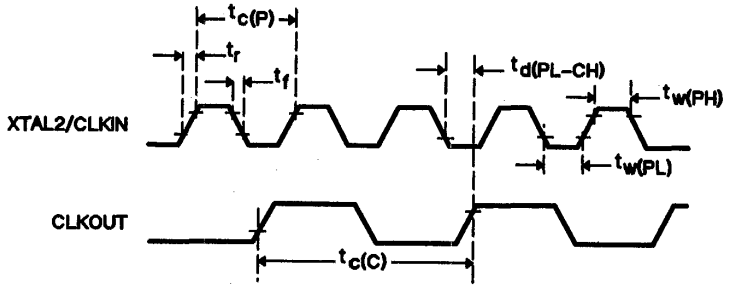


Figure 4-48. Clock Timing

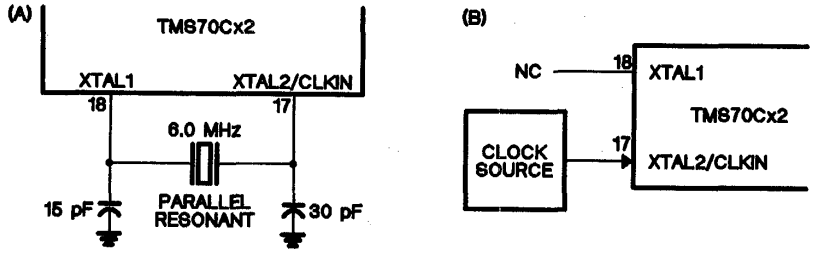


Figure 4-49. Recommended Clock Connections

Table 4-50. Memory Interface Timings†

	PARAMETER	MIN	TYP	MAX	UNIT
$t_{c(C)}$	CLKOUT cycle time	333		4000	ns
$t_{w(CH)}$	CLKOUT high pulse duration	$0.5t_{c(C)}$-90	$0.5t_{c(C)}$	$0.5t_{c(C)}$+90	ns
$t_{w(CL)}$	CLKOUT low pulse duration	$0.5t_{c(C)}$-90	$0.5t_{c(C)}$	$0.5t_{c(C)}$+90	ns
$t_{d(CH-JL)}$	Delay time, CLKOUT rise to ALATCH fall	$0.5t_{c(C)}$-50	$0.5t_{c(C)}$		ns
$t_{w(JH)}$	ALATCH high pulse duration	$0.25t_{c(C)}$-50	$0.25t_{c(C)}$		ns
$t_{su(HA-JL)}$	Setup time, high address valid before ALATCH fall	$0.25t_{c(C)}$-45	$0.25t_{c(C)}$		ns
$t_{su(LA-JL)}$	Setup time, low address valid before ALATCH fall	$0.25t_{c(C)}$-45	$0.25t_{c(C)}$		ns
$t_{d(JL-LA)}$	Delay time, low address valid after ALATCH fall	$0.5t_{c(C)}$-35	$0.5t_{c(C)}$		ns
$t_{su(RW-JL)}$	Setup time, R/$\overline{W}$ valid before ALATCH fall	$0.25t_{c(C)}$-40	$0.25t_{c(C)}$		ns
$t_{h(EH-RW)}$	Hold time, R/$\overline{W}$ valid after $\overline{ENABLE}$ rise	$0.5t_{c(C)}$-60	$0.5t_{c(C)}$		ns
$t_{h(EH-AH)}$	Hold time, high address valid after $\overline{ENABLE}$ rise	$0.5t_{c(C)}$-60	$0.5t_{c(C)}$		ns
$t_{su(Q-EH)}$	Setup time, data out valid before $\overline{ENABLE}$ rise	$0.5t_{c(C)}$-70	$0.5t_{c(C)}$		ns
$t_{h(EH-Q)}$	Hold time, data out valid after $\overline{ENABLE}$ rise	$0.5t_{c(C)}$-60	$0.5t_{c(C)}$		ns
$t_{d(LA-EL)}$	Delay time, low address HI-Z to $\overline{ENABLE}$ fall	$0.25t_{c(C)}$-55	$0.25t_{c(C)}$		ns
$t_{d(EH-A)}$	Delay time, $\overline{ENABLE}$ rise to next address drive	$0.5t_{c(C)}$-60	$0.5t_{c(C)}$		ns
$t_{d(EL-D)}$	Delay time, data in after $\overline{ENABLE}$ fall	$0.75t_{c(C)}$-160	$0.75t_{c(C)}$		ns
$t_{a(A-D)}$	Access time, data in from valid address	$1.5t_{c(C)}$-200	$1.5t_{c(C)}$-100		ns
$t_{d(A-EH)}$	Delay time, $\overline{ENABLE}$ high after address valid	$1.5t_{c(C)}$-50	$1.5t_{c(C)}$		ns
$t_{h(EH-D)}$	Hold time, Data input valid after $\overline{ENABLE}$ rise	0			ns
$t_{d(EH-JH)}$	Delay time, $\overline{ENABLE}$ rise to ALATCH rise	$0.5t_{c(C)}$-60	$0.5t_{c(C)}$		ns
$t_{d(CH-EL)}$	Delay time, CLKOUT rise to $\overline{ENABLE}$ fall		30		ns

† f_{osc} = 0.5 to 6.0 MHz
V_{CC} = 4.5 to 5.5 V
CLKIN duty cycle = 50%

Table 4-51. Memory Interface Timings at 6 MHz†

PARAMETER		MIN	TYP	MAX	UNIT
$t_{c(C)}$	CLKOUT cycle time		333		ns
$t_{w(CH)}$	CLKOUT high pulse duration	76	166	252	ns
$t_{w(CL)}$	CLKOUT low pulse duration	76	162	252	ns
$t_{d(CH-JL)}$	Delay time, CLKOUT rise to ALATCH fall	116	166		ns
$t_{w(JH)}$	ALATCH active duration	33	83		ns
$t_{su(AH-JL)}$	Setup time, high address valid before ALATCH fall	38	83		ns
$t_{su(LA-JL)}$	Setup time, low address valid before ALATCH fall	38	83		ns
$t_{d(JL-LA)}$	Delay time, low address hold after ALATCH fall	131	166		ns
$t_{d(RW-JL)}$	Delay time, R/$\overline{W}$ valid before ALATCH fall	43	83		ns
$t_{h(EH-RW)}$	Hold time, R/$\overline{W}$ valid after $\overline{ENABLE}$ rise	106	166		ns
$t_{h(EH-HA)}$	Hold time, high address valid after $\overline{ENABLE}$ rise	106	166		ns
$t_{su(Q-EH)}$	Setup time, data out valid before $\overline{ENABLE}$ rise	96	166		ns
$t_{h(EH-Q)}$	Hold time, data out valid after $\overline{ENABLE}$ rise	106	166		ns
$t_{d(LA-EL)}$	Delay time, low address HI-Z to $\overline{ENABLE}$ fall	38	83		ns
$t_{d(EH-A)}$	Delay time, $\overline{ENABLE}$ rise to next address drive	106	166		ns
$t_{d(EL-D)}$	Delay time, data in after $\overline{ENABLE}$ fall	90	250		ns
$t_{a(A-D)}$	Access time, data in from valid address	300	400		ns
$t_{d(A-EH)}$	Delay time, $\overline{ENABLE}$ high after address valid	450	500		ns
$t_{h(EH-D)}$	Hold time, data input valid after $\overline{ENABLE}$ rise	0			ns
$t_{d(EH-JH)}$	Delay time, $\overline{ENABLE}$ rise to ALATCH rise	106	166		ns
$t_{d(CH-EL)}$	Delay time, CLKOUT rise to $\overline{ENABLE}$ fall		30		ns

† V_{CC} = 4.5 to 5.5 V
CLKIN duty cycle = 50%

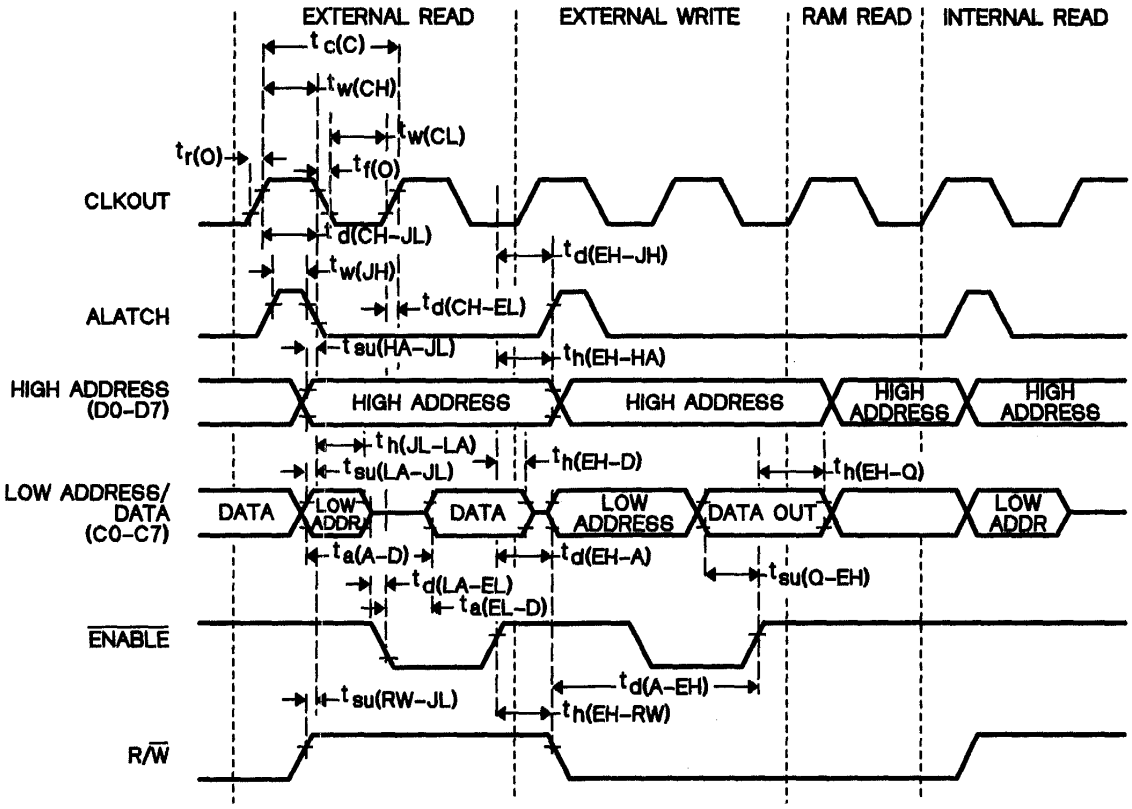
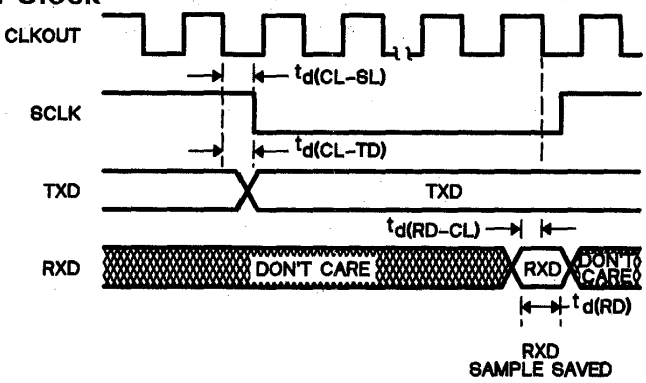


Figure 4-50. Read and Write Cycle Timing

4.8.1 Serial Port Timing

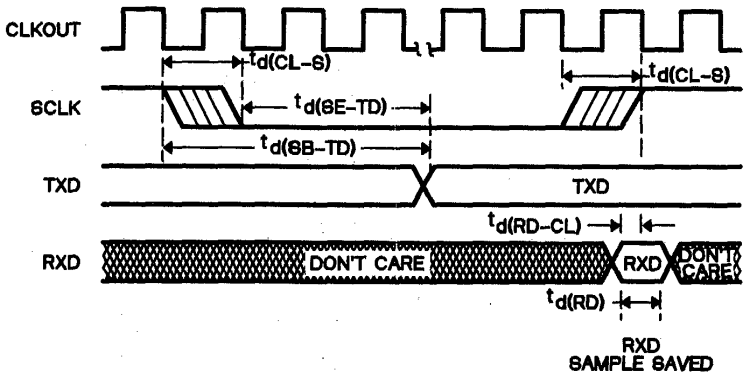
4.8.1.1 Internal Serial Clock



Notes: 1) The CLKOUT signal is not available in Single-Chip mode.
2) CLKOUT = $t_{c(C)}$.

	PARAMETER	TYP	UNIT
$t_{d(CL-SL)}$	CLKOUT low to SCLK low	1/4 $t_{c(C)}$	ns
$t_{d(CL-TD)}$	CLKOUT low to new TXD data	1/4 $t_{c(C)}$	ns
$t_{d(RD-CL)}$	RXD data valid before CLKOUT low	1/4 $t_{c(C)}$	ns
$t_{d(RD)}$	RXD data valid time	1/2 $t_{c(C)}$	ns

4.8.1.2 External Serial Clock



Notes: 1) The CLKOUT signal is not available in Single-Chip mode.
2) CLKOUT = $t_{c(C)}$.
3) SCLK sampled; if SCLK = 1 then 0, fall transition found.
4) SCLK sampled; if SCLK = 0 then 1, rise transition found.

	PARAMETER	TYP	UNIT
$t_{d(RD-CL)}$	RXD data valid before CLKOUT low	1/4 $t_{c(C)}$	ns
$t_{d(RD)}$	RXD data valid time	1/2 $t_{c(C)}$	ns
$t_{d(SB-TD)}$	Start of SCLK sample to new TXD data	3 1/4 $t_{c(C)}$	ns
$t_{d(SE-TD)}$	End of SCLK sample to new TXD data	2 1/4 $t_{c(C)}$	ns
$t_{d(CL-S)}$	Clockout low to SCLK transition	$t_{c(C)}$	ns

4.9 TMS77C82 (Advance Information)

Table 4-52. Absolute Maximum Ratings over Operating Free-Air Temperature Range (unless otherwise noted)

Supply voltage range, V_{CC}† ..... -0.3 V to 7 V
Input voltage range ..... -0.3 V to V_{CC}+0.3 V
Output voltage range ..... -0.3 V to V_{CC}+0.3 V
Maximum buffer sink current ..... ±10 mA
Storage temperature range ..... -55°C to 150°C
I_{CC}, I_{SS} (maximum into pin 25 or 40) ..... ±60 mA

† Unless otherwise noted, all voltages are with respect to V_{SS}.

Caution:

This is advance information on a new product in the sampling or preproduction phase of development. Characteristic data and other specifications are subject to change without notice.

Table 4-53. Recommended Operating Conditions

			MIN	NOM	MAX	UNIT
V_{CC}	Supply voltage		2.5		6.0	V
V_{IH}	High-level input voltage	MC and XTAL2 Pins V_{CC} = 2.5 to 6.0 V	0.8V_{CC}			V
		All other input pins V_{CC} = 3.0 to 6.0 V	0.70V_{CC}			V
		All other inputs V_{CC} = 2.5 to 3.0 V	0.75V_{CC}			V
V_{IL}	Low-level input voltage	MC and XTAL pins V_{CC} = 2.5 to 6.0 V			0.2V_{CC}	V
		All other inputs V_{CC} = 2.5 to 6.0 V			0.3V_{CC}	V
T_A	Operating temperature	Commercial	0		70	°C
		Industrial	-40		85	°C
f_{osc}	Oscillator frequency		.5		7.5	MHz

4.10 SE70CP160A Specifications

These specifications are for wide-voltage operation. For operation at 5 V ±10%, see Section 4.6. Be sure to use an EPROM that uses similar supply voltage specifications.

Table 4-54. Absolute Maximum Rating over Operating Free-Air Temperature Range (unless otherwise noted)

Supply voltage, V_{CC}† ..... -0.3 V to 7 V
All input voltages ..... -0.3 V to V_{CC} + 0.3 V
All output voltages ..... -0.3 V to V_{CC} + 0.3 V
Maximum I/O buffer current ..... ±10 mA
Storage temperature range ..... -55°C to 150°C
I_{CC}, I_{SS} current (maximum into pins 25 and 40) ..... ±60 mA

† Unless otherwise noted, all voltages are with respect to V_{SS}.

Caution:

Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the "Recommended Operating Conditions" section of this specification is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

Table 4-55. Recommended Operating Conditions

			MIN	NOM	MAX	UNIT
V_{CC}	Supply voltage		2.5		6.0	V
V_{IH}	High-level input voltage	XTAL2 pin, V_{CC} = 2.5 to 6 V	0.8V_{CC}			V
		All other pins, V_{CC} = 3 to 6 V	0.70V_{CC}			V
		All other pins, V_{CC} = 2.5 to 3 V	0.75V_{CC}			V
V_{IL}	Low-level input voltage	XTAL2 pin, V_{CC} = 2.5 to 6 V			0.2V_{CC}	V
		All other pins, V_{CC} = 2.5 to 6 V			0.3V_{CC}	V
T_A	Operating temperature range		0		55	°C

Table 4-56. Electrical Characteristics over Full Range of Operating Conditions

PARAMETER		TEST CONDITIONS	MIN	TYP†	MAX	UNIT
I_I	Input leakage current	V_{IN} = V_{SS} to V_{CC}		±0.1	±1	µA
C_I	Input capacitance			5		pF
V_{OH}	High-level output voltage‡	V_{CC} = 2.5 V, I_{OH} = -50 µA	2.25	2.4		V
		V_{CC} = 4.0 V, I_{OH} = -0.4 mA	3.2	3.6		V
		V_{CC} = 5.0 V, I_{OH} = -0.7 mA	3.9	4.5		V
		V_{CC} = 6.0 V, I_{OH} = -1.0 mA	4.6	5.4		V
V_{OL}	Low-level output voltage‡	V_{CC} = 2.5 V, I_{OL} = 0.4 mA		0.2	0.35	V
		V_{CC} = 4.0 V, I_{OL} = 1.6 mA		0.4	0.8	V
		V_{CC} = 5.0 V, I_{OL} = 2.5 mA		0.6	1.1	V
		V_{CC} = 6.0 V, I_{OL} = 3.4 mA		0.8	1.4	V
I_{OH}	Output source current	V_{CC} = 2.5 V, V_{OH} = 2.25 V	-0.05	-0.2		mA
		V_{CC} = 4.0 V, V_{OH} = 3.2 V	-0.4	-1.4		mA
		V_{CC} = 5.0 V, V_{OH} = 3.9 V	-0.7	-2.2		mA
		V_{CC} = 6.0 V, V_{OH} = 4.6 V	-1.0	-3.3		mA
I_{OL}	Output sink current	V_{CC} = 2.5 V, V_{OH} = 0.35 V	0.4	0.9		mA
		V_{CC} = 4.0 V, V_{OH} = 0.8 V	1.6	3.5		mA
		V_{CC} = 5.0 V, V_{OH} = 1.1 V	2.5	5.5		mA
		V_{CC} = 6.0 V, V_{OH} = 1.4 V	3.4	8.0		mA

† V_{CC} = 5 V, T_A = 25°C
‡ Output levels ensure 400 mV of noise margin over specified input levels.

Table 4-57. Supply Current Requirements

PARAMETER	TEST CONDITIONS	MIN	TYP	MAX	UNIT
I_{CC} Operating mode	f_{osc} = 6.0 MHz, V_{CC} = 5 V		9.0	14.4	mA
	f_{osc} = 3.0 MHz, V_{CC} = 5 V		4.5	7.2	mA
	f_{osc} = 0.5 MHz, V_{CC} = 5 V		0.8	1.2	mA
	f_{osc} = Z MHz, V_{CC} = 5 V		1.5	2.4	mA/MHz
	f_{osc} = 0.5 MHz, V_{CC} = 2.5 V		370	800	µA
I_{CC} Wake-Up mode (timer active)	f_{osc} = 6.0 MHz, V_{CC} = 5 V		960	1920	µA
	f_{osc} = 3.0 MHz, V_{CC} = 5V		480	960	µA
	f_{osc} = 0.5 MHz, V_{CC} = 5 V		80	160	µA
	f_{osc} = Z MHz, V_{CC} = 5 V		160	320	µA/MHz
	f_{osc} = 0.5 MHz, V_{CC} = 2.5 V		40	80	µA
I_{CC} Halt osc-on	f_{osc} = 6.0 MHz, V_{CC} = 5 V		480	980	µA
	f_{osc} = 3.0 MHz, V_{CC} = 5 V		240	500	µA
	f_{osc} = 0.5 MHz, V_{CC} = 5 V		45	100	µA
	f_{osc} = Z MHz V_{CC} = 5 V	See Note 2			µA
	f_{osc} = 0.5 MHz, V_{CC} = 2.5 V		25	60	µA
I_{CC} Halt osc-off	V_{CC} = 2.5 to 6 V		1	10	µA

fo on

Notes:
1. All inputs = V_{CC} or V_{SS} (except XTAL2). All output pins are open.
2. Maximum current = 160(Z) + 20 µA
3. I_{CC} applies to the supply current of the SE70CP160A without an EPROM device installed.

Table 4-58. Recommended Crystal/Clockin Operating Conditions over Full Operating Range

PARAMETER		TEST CONDITIONS	MIN	TYP†	MAX	UNIT
f_{osc}	Crystal frequency	V_{CC} = 2.5 V	0.5		0.8	MHz
		V_{CC} = 4.0 V	0.5		4.0	MHz
		V_{CC} = 5.0 V	0.5		6.0	MHz
		V_{CC} = 6.0 V	0.5		6.5	MHz
	CLKIN duty cycle		45		55	%
$t_{c(P)}$	Crystal cycle time	V_{CC} = 2.5 V	1250		2000	ns
		V_{CC} = 4.0 V	250		2000	ns
		V_{CC} = 5.0 V	166		2000	ns
		V_{CC} = 6.0 V	153		2000	ns
$t_{c(C)}$	Internal state cycle time	V_{CC} = 2.5 V	2500		4000	ns
		V_{CC} = 4.0 V	500		4000	ns
		V_{CC} = 5.0 V	333		4000	ns
		V_{CC} = 6.0 V	306		4000	ns
$t_{w(PH)}$	CLKIN pulse duration high		50			ns
$t_{w(PL)}$	CLKIN pulse duration low		50			ns
t_r	CLKIN rise time				30	ns
t_f	CLKIN fall time				30	ns
$t_{d(PL-CH)}$	CLKIN fall to CLKOUT rise delay			140	250	ns

† V_{CC} = 5 V, T_A = 25°C

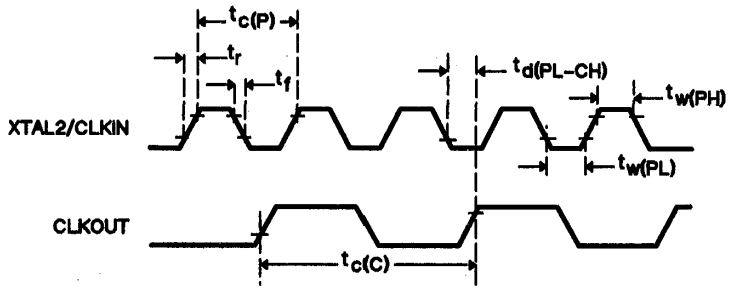


Figure 4-51. Clock Timing

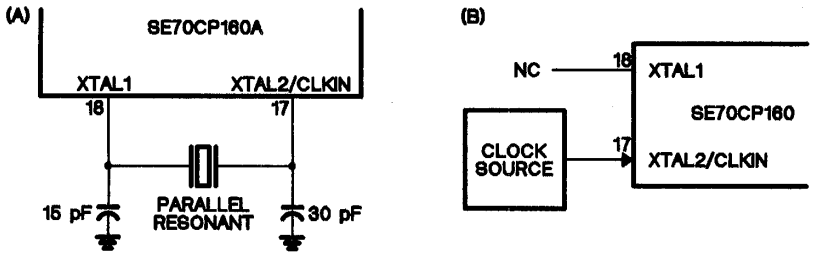


Figure 4-52. Recommended Clock Connections

4.11 SE70CP162 Specifications

These specifications are for wide-voltage operation. For operation at 5 V ±10%, see Section 4.8. Be sure to use an EPROM that uses similar supply voltage specifications.

Table 4-59. Absolute Maximum Ratings over Operating Free-Air Temperature Range (unless otherwise noted)

Supply voltage range, V_{CC}† ..... -0.3 V to 7 V
Input voltage range ..... -0.3 V to V_{CC}+0.3 V
Output voltage range ..... -0.3 V to V_{CC}+0.3 V
Maximum I/O buffer current ..... ±10 mA
Storage temperature range ..... -55°C to 150°C
I_{CC}, I_{SS} (maximum into pin 25 or 40) ..... ±60 mA

† Unless otherwise noted, all voltages are with respect to V_{SS}.

Caution:

Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the "Recommended Operating Conditions" section of this specification is not implied. Exposure to absolute maximum rated conditions for extended periods may affect device reliability.

Table 4-60. Recommended Operating Conditions

			MIN	NOM	MAX	UNIT
V_{CC}	Supply voltage		2.5		6.0	V
V_{IH}	High-level input voltage	MC and XTAL2 pins, V_{CC} = 2.5 to 6 V	0.8V_{CC}			V
		All other input pins, V_{CC} = 3 to 6 V	0.70V_{CC}			V
		All other input pins, V_{CC} = 2.5 to 3 V	0.75V_{CC}			V
V_{IL}	Low-level input voltage	MC and XTAL2 pins, V_{CC} = 2.5 to 6 V			0.2V_{CC}	V
		All other input pins, V_{CC} = 2.5 to 6 V			0.3V_{CC}	V
T_A	Operating free-air temperature		0		55	°C

Table 4-61. Electrical Characteristics over Full Range of Operating Conditions

PARAMETER		TEST CONDITIONS		MIN	TYP†	MAX	UNIT
I_I	Input current	MC pin, $V_{IN} = V_{SS}$ or V_{CC} All others, $V_{IN} = V_{SS}$ to V_{CC}			±0.1	±1	µA
C_I	Input capacitance				5		pF
V_{OH}	High-level output voltage‡	V_{CC} = 2.5 V,	I_{OH} = -50 µA	2.25	2.4		V
		V_{CC} = 4.0 V,	I_{OH} = -0.4 mA	3.2	3.6		V
		V_{CC} = 5.0 V,	I_{OH} = -0.7 mA	3.9	4.5		V
		V_{CC} = 6.0 V,	I_{OH} = -1.0 mA	4.6	5.4		V
V_{OL}	Low-level output voltage‡	V_{CC} = 2.5 V,	I_{OL} = 0.4 mA		0.2	0.35	V
		V_{CC} = 4.0 V,	I_{OL} = 1.6 mA		0.4	0.8	V
		V_{CC} = 5.0 V,	I_{OL} = 2.5 mA		0.6	1.1	V
		V_{CC} = 6.0 V,	I_{OL} = 3.4 mA		0.8	1.4	V
I_{OH}	Output source current	V_{CC} = 2.5 V,	V_{OH} = 2.25 V	-50	-200		µA
		V_{CC} = 4.0 V,	V_{OH} = 3.2 V	-0.4	-1.4		mA
		V_{CC} = 5.0 V,	V_{OH} = 3.9 V	-0.7	-2.2		mA
		V_{CC} = 6.0 V,	V_{OH} = 4.6 V	-1.0	-3.3		mA
I_{OL}	Output sink current	V_{CC} = 2.5 V,	V_{OH} = 0.35 V	0.4	0.9		mA
		V_{CC} = 4.0 V,	V_{OH} = 0.8 V	1.6	3.5		mA
		V_{CC} = 5.0 V,	V_{OH} = 1.1 V	2.5	5.5		mA
		V_{CC} = 6.0 V,	V_{OH} = 1.4 V	3.4	8.0		mA

† V_{CC} = 5 V, T_A = 25°C
‡ Output levels ensure 400 mV of noise margin over specified input levels.

Table 4-62. Supply Current Requirements

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
I_{CC}	Operating mode	f_{osc} = 7.0 MHz, V_{CC} = 5.0 V		17	24.5	mA
		f_{osc} = 3.0 MHz, V_{CC} = 5.0 V		7.2	10.5	mA
		f_{osc} = 0.5 MHz, V_{CC} = 5.0 V		1.2	1.8	mA
		f_{osc} = Z MHz, V_{CC} = 5.0 V		2.4	3.5	mA/MHz
		f_{osc} = 0.5 MHz, V_{CC} = 2.5 V		0.4	1.2	mA
I_{CC}	Wake-Up mode 1 (one timer and UART active)	f_{osc} = 7.0 MHz, V_{CC} = 5.0 V		2400	5600	µA
		f_{osc} = 3.0 MHz, V_{CC} = 5.0 V		1200	3300	µA
		f_{osc} = 0.5 MHz, V_{CC} = 5.0 V		250	800	µA
I_{CC}	Wake-Up mode 2 (one timer active, UART inactive)	f_{osc} = 7.0 MHz, V_{CC} = 5.0 V		960	3400	µA
		f_{osc} = 3.0 MHz, V_{CC} = 5.0 V		480	2000	µA
		f_{osc} = 0.5 MHz, V_{CC} = 5.0 V		140	550	µA
I_{CC}	Wake-Up mode 3 (UART active only)	f_{osc} = 7.0 MHz, V_{CC} = 5.0 V		1500	2400	µA
		f_{osc} = 3.0 MHz, V_{CC} = 5.0 V		800	1500	µA
		f_{osc} = 0.5 MHz, V_{CC} = 5.0 V		180	600	µA

Notes:
1. All inputs = V_{CC} or V_{SS} (except XTAL2). All output pins are open.
2. I_{CC} applies to the supply current of the SE70CP162 without an EPROM device installed.

Table 4-63. Recommended Crystal/Clockin Operating Conditions over Full Operating Range

PARAMETER		TEST CONDITIONS	MIN	TYP†	MAX	UNIT
f_{osc}	Crystal frequency	V_{CC} = 2.5 V	0.5		0.8	MHz
		V_{CC} = 4.0 V	0.5		5.0	MHz
		V_{CC} = 5.0 V	0.5		7.0	MHz
		V_{CC} = 6.0 V	0.5		7.5	MHz
	CLKIN duty cycle		45		55	%
$t_{c(P)}$	Crystal cycle time	V_{CC} = 2.5 V	1250		2000	ns
		V_{CC} = 4.0 V	200		2000	ns
		V_{CC} = 5.0 V	143		2000	ns
		V_{CC} = 6.0 V	133		2000	ns
$t_{c(C)}$	Internal state cycle time	V_{CC} = 2.5 V	2500		4000	ns
		V_{CC} = 4.0 V	400		4000	ns
		V_{CC} = 5.0 V	286		4000	ns
		V_{CC} = 6.0 V	267		4000	ns
$t_{w(PH)}$	CLKIN pulse duration high		50			ns
$t_{w(PL)}$	CLKIN pulse duration low		50			ns
t_r	CLKIN rise time				30	ns
t_f	CLKIN fall time				30	ns
$t_{d(PL-CH)}$	CLKIN fall to CLKOUT rise delay			110	250	ns

† V_{CC} = 5 V, T_A = 25°C

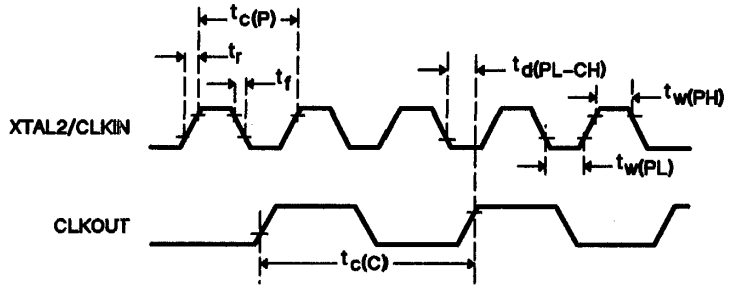


Figure 4-53. Clock Timing

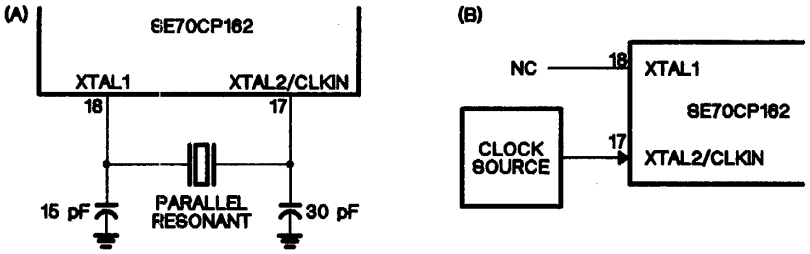
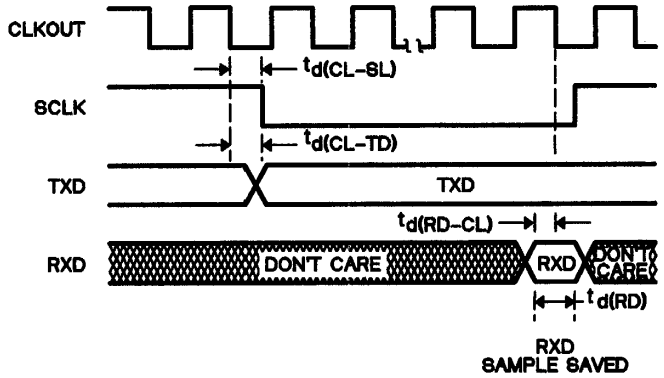


Figure 4-54. Recommended Clock Connections

4.11.1 Serial Port Timing

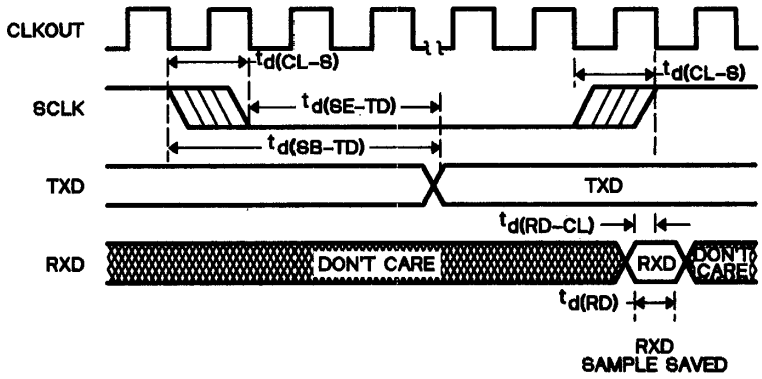
4.11.1.1 Internal Serial Clock



Notes: 1) The CLKOUT signal is not available in Single-Chip mode.
2) CLKOUT = $t_{c(C)}$.

PARAMETER		TYP	UNIT
$t_{d(CL-SL)}$	CLKOUT low to SCLK low	1/4 $t_{c(C)}$	ns
$t_{d(CL-TD)}$	CLKOUT low to new TXD data	1/4 $t_{c(C)}$	ns
$t_{d(RD-CL)}$	RXD data valid before CLKOUT low	1/4 $t_{c(C)}$	ns
$t_{d(RD)}$	RXD data valid time	1/2 $t_{c(C)}$	ns

4.11.1.2 External Serial Clock



Notes: 1) The CLKOUT signal is not available in Single-Chip mode.
2) CLKOUT = $t_{c(C)}$.
3) SCLK sampled; if SCLK = 1 then 0, fall transition found.
4) SCLK sampled; if SCLK = 0 then 1, rise transition found.

PARAMETER		TYP	UNIT
$t_{d(RD-CL)}$	RXD data valid before CLKOUT low	1/4 $t_{c(C)}$	ns
$t_{d(RD)}$	RXD data valid time	1/2 $t_{c(C)}$	ns
$t_{d(SB-TD)}$	Start of SCLK sample to new TXD data	3 1/4 $t_{c(C)}$	ns
$t_{d(SE-TD)}$	End of SCLK sample to new TXD data	2 1/4 $t_{c(C)}$	ns
$t_{d(CL-S)}$	Clockout low to SCLK transition	$t_{c(C)}$	ns

8. Macro Language

The TMS7000 Macro Assembler supports a macro definition language. Macro definitions allow you to create your own "commands." This is especially useful when a program executes a particular task several times. A macro definition contains source statements that are associated with a unique macro name. When the macro name is used as an opcode in a program source statement (referred to as a *macro call*), the macro definition's predefined source statements are substituted for the macro call statement.

This section discusses the following topics:

8.1 Defining Macros

A macro definition begins with a source statement like this:

```
<MACNAME>  $MACRO [<parm1>,<parm2>...] [<comment>]
```

where:

<MACNAME>	Names the macro; it may contain a maximum of six alpha-numeric characters. It is placed in the source statement's label field.
$MACRO	Identifies this source statement as the first line of a macro definition; it appears in the opcode field.
<parms>	Parameters passed to the macro when called (not all macros will have parameters); they appear in the operand field.
<comment>	Optional.

There are three methods for defining macros:

1) Macros can be defined in the **source file** where they are used. Macros must be defined before they are called; it is good practice to place all the definitions at the top of the file. This provides easy reference to all the definitions because they are in one location.

2) Macros can also be defined in external files. These files are simply text files, like the assembler source file. Only one macro may be defined per external file. These external macro definition files are collected to form a **macro library**.

3) All macros can be placed in one file without the source program, and then the COPY directive can be used to include the macro file in the source program.

8.1.1 Using Macro Libraries

When a macro is called, the assembler searches several places for its definition. Let's assume that the directory file 'VOLUME.DIRECTORY.MACLIB' contains a library of macro definitions. The MLIB directive tells the assembler that a macro library exists. The MLIB directive syntax is:

```
MLIB  'VOLUME.DIRECTORY.MACLIB'
```

The quoted string names the macro library. (This string represents a directory name in the host operating system format.)

This library contains a definition for a macro named CPXADD. Assume that an assembly language source program contains the following macro call:

```
LABEL   CPXADD CX1,CX2
```

The assembler uses the following search order to find the macro definition:

1) The **in-memory macro table** is the first place searched. CPXADD will be in the macro table if:

   a) It was previously defined in the assembler source file or
   b) It has already been read from a macro file.

2) If CPXADD is not found in the macro table, the assembler searches the normal **assembler opcode/directive table**. If found there, the opcode will be assembled as a normal machine instruction.

3) If the definition is not in the opcode/directive table, **the macro name is appended to the macro library name**.

   If more than one MLIB directive was encountered, the assembler searches the most recently defined library first, then the library defined before that, and so on.

If the file is found, the macro definition is copied into the assembler's macro file (in a compressed format), and an entry is made in the macro table for later use.

The search order prevents a macro defined in a library from automatically redefining a machine instruction because the assembler searches the opcode table before the libraries. This can be circumvented in two ways:

1) Define the macro in the source program or

2) Include another file in the macro library called an MLIST (macro list).

An MLIST file is a text file that contains the names of the opcodes and currently defined macros that are redefined by macros in the library.

A typical MLIST file might be constructed as follows; note that there is only one definition per line and each statement begins in column one.

```
file named  <MLIB directory name>.MLIST
record 1    ADD          (opcode)
record 2    LACK         (opcode)
record 3    MOV          (opcode)
record 4    FSUB         (macro)
eof                   (MLIST)
```

The MLIST is read (if provided) when the MLIB directive is processed. If a name found there matches a currently defined opcode or a name in the macro table, the matching entry is removed from its table. This forces a search of the libraries, since the name will not be found elsewhere. The following message is printed when a name is found that matches an opcode:

```
' ****  OPCODES REDEFINED'
```

The message appears after the printing of the MLIB statement. A similar message:

```
' **** MACROS REDEFINED'
```

appears when currently defined macros are redefined. If you do not want an opcode or macro to be redefined, you must delete the appropriate records from the MLIST file.

The name of a macro in a file should be the same as the file name, or the macros are not used efficiently. If the file named CPXADD contains a definition line such as

```
CPXMUL   $MACRO   MR, MD
```

the macro CPXMUL is entered into the macro table, and the next call to CPXADD will be undefined and re-entered into the macro table as CPXMUL.

8.1.1.1 Using Macro Libraries on MS/PC-DOS Systems

The following program segment suggests a method for using macro libraries on an MS/PC-DOS system.

```
      MLIB 'E:'        The pathname must be a drive name
      ADD  R3,R4       Typical assembly code
      MOV  R6,R9              .
       .    .                 .
       .    .                 .
*
      XMAC             First macro call
*
      NOP
*
      YMAC             Another macro call
*
      NOP
      END
```

The assembler searches the drive specified by the MLIB directive for a file with the same name as the macro. The macro name cannot have an extension. Only one macro is allowed per file.

The assembler searches the current MS/PC-DOS directory structure for the drive specified in the MLIB directive. A possible example of macro library use is:

- Store all macros on the **A** drive in a directory named MACROS.
- Store the TMS7000 assembler on the **E** drive (or any drive other than A) in a directory named PROGRAMS. The assembler program name is XASM7.EXE.
- Store the source program on the **E** drive in a directory named ASSEMBLY. The source program name is CODE.ASM. It includes this directive statement:

  ```
  MLIB 'A:'
  ```

- Issue a path statement that includes the program directory:

  ```
  PATH E:\;E:\MSDOS;E:\PROGRAMS
  ```

- The following batch file will assemble the program:

```
E:                          Insure execution from drive E:
CD A:\MACROS                Change A: drive's directory
CD E:\ASSEMBLY              Change E: drive's directory
XASM7 CODE.ASM;             Assemble the file CODE.ASM
```

8.1.2 Sample Macros

Assume that a symbol representing a memory address, ADR, is set in a source file:

```
ADR     EQU     >F000
```

This is a simple example of a macro definition that increments ADR:

```
INCADR  $MACRO
        LDA     @ADR
        INC     A
        STA     @ADR
        $END
```

where:

INCADR	Names a macro, INCADR.
$MACRO	Identifies the beginning of the macro definition.
LDA @ADR INC A STA @ADR	Are model statements that are substituted into the source program when the macro is called. A model statement "models" an assembler language statement. Such a statement is (or will form after macro substitution) a legal language statement.
$END	Identifies the end of the macro definition.

The macro INCADR can now be used in the source program as often as necessary. Call the macro by entering the following line into the source file:

```
INCADR
```

The macro assembler replaces this line with the macro definition:

```
LDA     @ADR
INC     A
STA     @ADR
```

INCADR is limited because the macro can only be used with a single memory location, ADR. The following macro uses parameters and is more flexible. It can be used with any memory location.

```
INC     $MACRO  M
        LDA     @:M.S:
        INC     A
        STA     @:M.S:
        $END
```

where:

M	Is a *macro parameter.* It is replaced by the actual parameter when the macro is called.
M.S	Is the string component of this variable (the symbol representation of the variable).

For example, the line:

```
INC     Y
```

will be replaced by:

```
LDA     @Y
INC     A
STA     @Y
```

but

```
INC     DATA4
```

will be replaced by:

```
LDA     @DATA4
INC     A
STA     @DATA4
```

8.2 Strings, Constants, and Operators

Macro language literal **strings** are identical to the character strings used by TMS7000 assembly language. The strings contain one or more characters enclosed in single quotes.

Examples of valid strings are:

'ONE'
' ' (a blank)

Macro language **constants** are defined in the same manner as assembly language constants.

Examples of valid constants are:

>9F3C
$ (current PC value)

Arithmetic operators can be used in operands. Functions of +, -, * (multiply), and / (divide) can be used to generate operand values. Examples using arithmetic operators are:

```
LABEL   EQU   $+4  (current PC value + 4)
```

Relational operators can also be used. Relational operators compare the values of two variables or constants and return the answer of TRUE or FALSE. The relational operators are:

=	Equal
>	Greater than
<	Less than
#=	Not equal

Examples using relational operators are:

```
$IF  A.V>3     Process succeeding block if value
               component of variable A is >3.
$IF  B.L#=A.L  Process succeeding block if length
               component of variable B is not equal
               to length component of variable A.
```

The macro assembler also allows the use of **Boolean operators**, which perform the desired operation and return either TRUE or FALSE. The Boolean operators are:

&	AND
++	OR
--	NOT

An example using the Boolean operators is:

```
$IF --((A.V>3)&(B.L#=A.L))
```

Macro symbol components can be concatenated with literal strings, model statement characters, and other macro variables. Concatenation is indicated by writing character strings side by side with string mode references.

8.3 Variables

Macro definitions can include **variables** which are represented in the same manner as symbols in the assembler symbol table (AST). Macro variables can have a maximum length of two characters. Examples of valid variables are:

VA
P4
SC
F2
A

> **Note:**
>
> Macro variables are strictly local, available only to the macro which defines them. Symbols in the assembler symbol table can only be accessed through symbol components.

Macro variables can be defined in two ways:

1) As parameters defined by the $MACRO statement, and
2) In $ASG statements (see the $ASG verb).

The macro translator maintains a macro symbol table (MST) similar to the AST. Each MST entry contains the variable/parameter and its string, value, length, and attribute components. The macro expander module places parameters in the MST when macro calls are processed and places variables in the MST when it processes $ASG statements.

8.3.1 Parameters

Parameters are variables that are declared in the $MACRO definition statement. The parameter declaration sequence corresponds to the sequence of the operands in the macro call statement. During macro expansion, the parameters receive the values of the macro call operands. Examples of $MACRO statements with parameters are:

```
LABEL   $MACRO  A,B3

NAME    $MACRO  O,RC,AM
```

8.3.2 Macro Variable Components

There are four types of variable/parameter components:

1) The **string component** of an MST entry contains a character string assigned to the macro variable/parameter by the macro expander.
2) The **value component** of an MST entry contains:
   a) The *binary equivalent* of the string component, if the string component is an *integer*.
   b) The *value of the symbol*, if the string component is a *symbol* in the AST.
   c) The *length of the list*, if the parameter is an *operand list*.
3) The **length component** contains the number of characters in the string component.
4) The **attribute component** of the MST is a bit vector. The bits correspond to the attributes of the variable or parameter.

The following statement defines a macro with parameters X and NUM:

```
ADDK     $MACRO   X,NUM
```

The following statement calls the ADDK macro:

```
         ADDK     VAR1,3
```

The MST now contains entries for parameters X and NUM and their associated components:

Parameter X:

String Component	Is the character string VAR1.
Attribute Component	Indicates that the parameter is supplied in a macro call (keyword $PCALL).
Length Component	Is 4.

Parameter NUM:

String Component	Is the character 3.
Value Component	Is 3 also, expressed as a 16-bit binary number.
Length Component	Is 1.
Attribute Component	Indicates that the parameter is supplied in the macro call (keyword $PCALL).

Each component of a macro variable can be accessed individually in either **binary** or **string mode**:

- In *binary mode*, the referenced macro variable component is treated as a signed 16-bit integer. Binary mode is accessed by writing the variable name and component. A reference to the string component of a macro variable in binary mode is the 16-bit integer value of the ASCII representation of the first two characters of the string. For example, the binary mode value of the string component of X, in the preceding example, is >5641, which is the ASCII representation for VA.

- *String mode* access of macro variable components is signified by enclosing the variable in a pair of colon characters (:). For example,

```
:X:
```

> **Note:**
>
> Colons are always used in pairs to enclose a variable name. If a variable component qualifier is used, the pair of colons enclose the entire qualified name.

8.3.3 Variable Qualifiers

Table 8-1 lists the names used to indicate variable/parameter components. The variable name is followed by a period (.) and the single letter qualifier.

Table 8-1. Variable Qualifiers

QUALIFIER	MEANING
S	The string component of the variable
A	The attribute component of the variable
V	The value component of the variable
L	The length component of the variable

The following examples show qualified variables for the macro call:

```
ADDK    VAR1,3
```

which was defined by the following statement:

```
ADDK    $MACRO  X,NUM
```

X.S Is the string component (binary mode) of variable VAR1. X.S equals the binary equivalent for VA, or >5641. If string mode is indicated, as in :X.S:, the string component is the character string VAR1.

X.A Is the attribute component of variable VAR1. This component is accessed by using logical operators and keywords as described in Table 8-2, Table 8-3, and Table 8-4.

X.V Is the value component of variable VAR1.

X.L Is the length component of variable VAR1; in this case, it is equal to the character string 4.

Unqualified variables (except those in $ASG statements) refer to the variable's string component. These two strings are equivalent:

:CT.S: WAY Variable CT qualified; string component = WAY.

:CT: WAY Variable CT unqualified; string component = WAY.

> **Note:**
>
> Binary references to macro variables in model statements **must** be qualified.

8.3.4 Symbol Components

Entries in the assembler symbol table have symbol components. To access symbol components in a macro, the symbol must be assigned to the string component of a macro variable by an $ASG statement. The additional qualifiers shown in Table 8-2 are used with macro variables to access the AST symbol's components.

Table 8-2. Variable Qualifiers for Symbol Components

QUALIFIER	MEANING
SS	String component of a symbol that is the string component of a variable.
SV	Value component of a symbol that is the string component of a variable.
SA	Attribute component of a symbol that is the string component of a variable.
SL	Length component of a symbol that is the string component of a variable.

The following examples show qualified variables that specify symbol components of variable string components. Assume that the following statement appears in the source program:

```
MASK    EQU     >FF
```

This statement appears in a macro definition:

```
        $ASG    V1.S TO MASK
```

V1.SS Is the string component of the symbol MASK. This is null unless a macro instruction has caused a string to be associated with it by using a $ASG statement.

V1.SV Is the value component of the symbol MASK (>FF). In the string mode, :V1.SV: equals the character string 255.

V1.SA Is the attribute component of the symbol MASK. This component may be accessed by using logical operators and keywords.

V1.SL Is the length component of the symbol MASK. If a string has been assigned to MASK, then V1.SL is the length of that string.

Concatenation is especially useful when a previously defined string is augmented with additional characters. Assume that CT.S represents the string ONE.

```
:CT.S:' WAY'          produces the string      'ONE WAY'
```

If CT.S represented the character string TWO, the result of the concatenation in the example would be TWO WAY. Strings and qualified variables can be concatenated as required. Components of variables that are represented by a binary value (e.g., CT.V and CT.L) are converted to their ASCII decimal equivalent before concatenation. For example:

```
:CT.S' WAY ':CT.L:      expands into            ONE WAY 3
```

since the length component of the variable CT is three.

8.4 Keywords

Keywords identify assembler symbol and macro parameter attribute components. Each keyword represents a bit position in a word that contains all of the symbol or parameter attribute components. Keywords can be used with logical operators and attribute components to test or set a specific attribute of a symbol or parameter. The following paragraphs describe how keywords are used with symbols and parameters.

8.4.1 Symbol Attribute Component Keywords

Table 8-3 lists keywords that are used with a logical operator and the symbol attribute component (.SA) to test or set the corresponding attribute component in the AST.

Table 8-3. Symbol Attribute Keywords

KEYWORD	MEANING
$REL	Symbol is relocatable
$REF	Symbol is an operand of an REF directive
$DEF	Symbol is an operand of a DEF directive
$STR	Symbol has been assigned a component string
$MAC	Symbol is defined as a macro name
$UNDF	Symbol is not defined

Note: Using these attributes in conditional assembly (with the $IF verb) may lead to pass conflict errors if the symbol is not defined before the macro is called.

Assume that the next statement is an assembler program source statement and the second statement appears in a macro definition:

```
MASK    EQU     >FF

        $ASG    V1.S TO MASK
```

The next line ANDs symbol MASK's attribute component with a flag corresponding to the keyword $STR.

```
        V1.SA&$STR
```

This expression is TRUE when MASK's contents are not null; otherwise, the expression is FALSE.

The next example shows ORs symbol MASK's attribute component with the flag corresponding to the keyword $REL.

```
    V1.SA++$REL
```

8.4.2 Parameter Attribute Keywords

Table 8-4 lists keywords that are used with a logical operator and the macro symbol attribute component to test or set the corresponding attribute in the MST attribute component. Use these attribute keywords to test or set attribute components of all variables in the MST.

Table 8-4. Parameter Attribute Keywords

KEYWORD	MEANING
$PCALL	Parameter appears as a macro-instruction operand
$POPL	Parameter is an operand list; the value component contains the number of operands in the list
$PSYM	Parameter is a symbolic memory address †

† A symbolic memory address is recognized when the variable is preceded by an @ character.

The following expressions use parameter attribute component keywords:

P6.A&$PCALL AND variable P6's attribute component with the flag corresponding to keyword $PCALL. The expression is TRUE when variable P6 is a parameter supplied in a macro call, otherwise the expression is FALSE.

RA.A++$PSYM OR variable RA's attribute component with the flag corresponding to keyword $PSYM.

8.5 Assigning Values to Parameters

Macro definitions expand macro calls (statements that have the macro name as an opcode).

Macro definition syntax is:

```
<macro name> $MACRO [<parm>][,<parm>] [<comment>]
```

Macro call syntax is:

```
<macro name> [<operand/list>],[<operand/list>] [<comment>]
```

When a macro call is processed, the macro expander associates the first parameter in the $MACRO statement with the first operand or operand list in the macro call, the second parameter with the second operand or operand list, and so on.

Each operand may be any assembler expression or address type, or a quote-enclosed character string. An operand list is a group of operands enclosed in parentheses and separated by commas (when two or more operands are in list). An operand list is processed as a set, after the outer parentheses are removed, during macro expansion. Operands (or operand lists) may be nested in parentheses in the macro call for use within macro definitions.

The following $MACRO statement defines two parameters.

```
ONE     $MACRO  P1,P2
```

The corresponding macro call

```
        ONE     PAR1,PAR2
```

associates PAR1 with P1 and PAR2 with P2. However, a call such as:

```
        ONE     PAR1,(PAR21,PAR22)
```

associates PAR1 with P1 and the list PAR21,PAR22 with P2.

Now :P2: or :P2.S: can be used as a pair of operands in a model statement.

The $PCALL attribute is set for each parameter that receives a value. When the $MACRO statement defines more parameters than the number of operands in the macro call, the $PCALL attribute is not set for the excess parameters. The $PCALL attribute is also not set if an operand is "null"; i.e., the call line has two commas adjacent or an operand list of zero operands. Expansion of the macro can be controlled by the number of operands by using the $PCALL attribute and $IF statement. For example, the following macro definition and macro call

```
AMAC    $MACRO  P1,P2,P3
        AMAC    AB1,AB2
```

sets $PCALL for parameters P1 and P2 but not for P3. Similarly,

```
        AMAC    XY,,XY3
```

sets $PCALL for P1 and P3 but not for P2.

When the macro instruction has more operands than the number of parameters in the $MACRO statement, the excess operands are combined with the operand or operand list corresponding to the last parameter to form an operand list (or a longer operand list). In the macro statements below, the operands of the two macro calls would be assigned to the parameters in the same ways:

```
(1)
ONE       EQU       9
TWO       EQU       43
THREE     EQU       86
FIX       $MACRO    P1,P2              Define Macro FIX
            .
            .
            .
          FIX       ONE,TWO,THREE      Call Macro FIX
          FIX       ONE,(TWO,THREE)    Call Macro FIX
(2)
A         EQU       7
B         EQU       15
C         DATA      17
D         DATA      63
E         EQU       95
F         EQU       47
G         EQU       58
H         EQU       101
I         EQU       119
PARM      $MACRO    P1,P2,P3,P4,P5,P6,P7,P8,P9
            .
            .
            .
          PARM      @A,,B,(),C,(D),E,(G,(H,I))
```

Parameter assignments:

P1.S = A
P1.A = $PCALL
P1.L = 1
P1.V = 7

P2.S = (no string)
P2.A = (all false)
P2.L = 0
P2.V = 0

P3.S = B
P3.A = $PCALL
P3.L = 1
P3.V = 15

P4.S = (no string)
P4.A = $POPL
P4.L = 0
P4.V = 0

P5.S = C
P5.A = $PCALL
P5.L = 1
P5.V = 17

P6.S = D
P6.A = $PCALL,$POPL
P6.L = 1
P6.V = 1

P7.S = E
P7.A = $PCALL
P7.L = 1
P7.V = 95

P8.S = G,(H,I)
P8.A = $PCALL,$POPL
P8.L = 7
P8.V = 2

P9.S = (no string)
P9.A = 0 (all false)
P9.L = 0
P9.V = 0

8.6 Verbs

The macro language supports seven verbs that are used in macro language statements. Table 8-4 lists the seven verbs. Any statement in a macro definition that does not contain a macro language verb in the operation field is processed as a model statement.

Table 8-5. Macro Language Verb Summary

VERB	DESCRIPTION
$MACRO	Marks beginning of macro definition
$VAR	Declares variables for macro definitions
$ASG	Assigns values to variable components
$IF	Provides conditional processing
$ELSE	Begins an alternate block in a conditional process
$ENDIF	Terminates conditional processing
$END	Marks the end of a macro definition

Syntax <macro name> $MACRO [<parm>][,<parm>] [<comment>]

Description The $MACRO verb begins a macro definition. It must be the first statement in the definition. $MACRO assigns a name to the macro and declares the macro parameters.

The **macro name** contains one to six alphanumeric characters; the first must be a letter. Each **<parm>** is a parameter for the definition as described in Section 8.3.1. The operand field may contain as many parameters as the size of the field allows and must contain all parameters used in the macro definition. The **comment field** can only be used if there are parameters.

The macro definition is used to expand macro calls (statements that have the macro name as an opcode). The macro name specifies the macro definition to be used. When a macro call is processed, the macro expander associates the first parameter in the $MACRO statement with the first operand or operand list in the macro call, the second parameter with the second operand or operand list, and so on.

Example

```
ONE     $MACRO  P1,P2
```

specifies two parameters. A call such as

```
        ONE     PAR1,PAR2
```

associates PAR1 with P1 and PAR2 with P2.

> **Note:**
>
> A macro definition supercedes previous macro definitions and opcodes with the same name. Symbolic operands which appear in a macro call are treated as symbolic operands in opcodes; if they are not defined with the program in which they appear, they will be listed as undefined symbols.

Syntax

```
$VAR  <var>[,<var>]  [<comment>]
```

Description

The $VAR statement declares the variables for a macro definition. $VAR is required only if the macro definition contains one or more variables that are not parameters. More than one $VAR statement may be included; each $VAR statement may declare more than one variable. Each <var> in the operand is a variable as previously described (see Section 8.3).

The $VAR statement does not assign values to any components of the variables. $VAR statements may appear anywhere in the macro definition to which they apply, provided each variable is declared before the first statement that uses the variable. Placing $VAR statements immediately following the $MACRO statement is recommended.

Example

```
$VAR  A,CT,V3  Three variables for a macro
```

This example declares variables A, CT, and V3; A, CT, and V3 must not have been declared as parameters.

$ASG Assign Values to Variable Components Verb $ASG

Syntax

$ASG <expression/string> TO <var> [<comment>]

Description

The $ASG statement assigns values to variable components. Variables that are not parameters do not have values for any components until values are assigned using $ASG statements. Variable components with previously assigned values may be assigned new values with $ASG statements.

The expression operand may be any expression valid to the assembler and may contain binary mode variable references and the keywords in Table 8-3 and Table 8-4.

> **Note:**
>
> The binary mode value of a string component or symbol string component used in an expression is the binary value of the first two characters of the string. Thus, if GP.S has the string LAST, the value used for GP.S is an expression in the <string> hexadecimal number >4C41 which is the ASCII representation for LA.

A string may be one or more characters enclosed in single quotes, or the concatenation of such a literal string with the string mode value of a qualified variable. The <var> may be either an unqualified variable or a qualified variable.

When the operands are both unqualified variables, all components are transferred to target variables. When the destination variable is qualified, only the specified component receives the corresponding component of the expression or string. An exception to this is when a string is assigned to the string component of a variable or symbol, the length component of that variable or symbol is set to the number of characters in the assigned string. If the attribute component of the destination variable is to be changed, only those attributes which can be tested using keywords are changed. Other attributes maintained by the macro assembler may or may not be changed as appropriate.

> **Note:**
>
> A qualified variable that specifies the length component is illegal as a destination in a $ASG statement and will **not** set the length component.

Examples

Assume that variables P3, V3,and CT were previously declared as parameters ($MACRO statement) or variables ($VAR statement).

```
*     Assign all the components of variable P3 to
*     variable V3.
    $ASG    P3 TO V3
*
*     Concatenate string 'ES' to the string com-
*     ponent of variable P3, and set the string
*     component to the result. Also, add 2 to
*     the value of the new length component.
    $ASG   :P3.S:'ES' TO P3.S
*
*     Set the flag in the attribute component
*     of variable CT to indicate the symbolic
*     address attribute.
    $ASG A++PSYM TO CT.A
```

The $ASG statement may be used to modify symbol components as shown in the following examples. Assume that P3.V = 6 and P3.S = SUB.

```
*     Assign 'TEN' as the string component of
*     variable G. When 'TEN' is a symbol in the
*     AST, this statement allows the use of in-
*     direct component qualifiers to modify the
*     components of symbol TEN.
    $ASG 'TEN' TO G.S
*
*     Set the value component of the symbol in
*     the string component of variable G to the
*     value component of variable P3. In this
*     case, the value component of TEN is set to 6.
    $ASG P3.V TO G.SV
*
*     Concatenate string 'A', the string compo-
*     nent of variable P3, and string 'S' and
*     place the result in the indirect string
*     component of the same symbol. Thus, the
*     string component of TEN is ASUBS and the
*     length component is 5.
    $ASG 'A':P3.S:'S' TO G.SS
```

> **Note:**
>
> Keywords in an $ASG statement **must** be used with a Boolean operator and an attribute component of a variable in the source field. The attribute component must come first.

Syntax

$IF <expression> [<comment>]

Description

The $IF statement provides conditional processing in a macro definition.

An $IF statement is followed by a block of macro language statements terminated by an $ELSE statement or an $ENDIF statement. When the $ELSE statement is used, it is followed by another block of macro language statements terminated by an $ENDIF statement. When the expression in the $IF statement has a nonzero value (or evaluated as TRUE), the block of statements following the $IF statement is processed. When the expression in the $IF statement has a zero value (or evaluated as FALSE), the block of statements following the $IF statement is skipped. When the $ELSE statement is used and the expression in the $IF statement has a nonzero value, the block of statements following the $ELSE statement and terminated by the $ENDIF statement is skipped. Thus, the condition of the $IF statement may determine whether or not a block of statements is processed, or which of two blocks of statements is processed. A block may consist of zero or more statements. The <expression> may be any expression as defined for the $ASG statement and may include qualified variables and keywords. The expression defines the condition for the $IF statement.

> **Note:**
>
> The $IF expression is always evaluated in binary mode. Specifically, the relational operations (<,>,=,#=) operate only on the binary mode values of macro variables. Boolean operators may be nested. In addition, $IF blocks may be nested, at most, 44 levels deep.

Example

These examples show conditional processing in macro definitions:

```
  .
  .
$IF  KY.SV     Process the statements of BLOCK
  .            A when the indirect value com-
  .            ponent of the variable KY con-
  .            tains a non-zero value.
  .    BLOCK A Process the statements of BLOCK
  .            B when the component contains
$ELSE          zero after processing either
  .            block of statements. Continue
  .    BLOCK B processing the statement fol-
  .            lowing the $ENDIF statement.
$ENDIF
  .
```

```
.
$IF  --(T.A&$PCALL)
 .                  Process the statements of BLOCK
 .                  A when the attribute component
 .                  of parameter T indicates that
 .   BLOCK A        parameter T was not supplied in
 .                  the macro instruction. If para-
 .                  meter T was supplied, do not
 .                  process the statements of BLOCK
$ENDIF              A. Continue processing at the
 .                  statement following the $ENDIF
 .                  statements in either case.
 .
$IF  T.L=5          Process the statements of BLOCK
 .                  A when the length component of
 .                  variable T is equal to 5, do not
 .                  process the statements of BLOCK
 .   BLOCK A        A. Continue processing at the
$ENDIF              statement following the $ENDIF.
```

Alternate Conditional Block Verb

Syntax

$ELSE [<comment>]

Description

The $ELSE statement begins an alternate block to be processed if the preceding $IF expression was false.

Syntax

$ENDIF [<comment>]

Description

The $ENDIF statement terminates the conditional processing initiated by an $IF statement in a macro definition.

Syntax $END [<macro name>][<comment>]

Description The $END statement ends a macro definition. When executed, the $END statement terminates the processing of the macro definition. The <macro name> parameter is optional.

Example

```
$END FIX  Terminates the definition of macro FIX.
```

8.7 Model Statements

Most macro definitions contain **model statements**. A model statement is, or produces, an assembly language statement. Model statements are composed of the usual assembly language statement elements and can include qualified variable components (string mode only). The source statement produced must be a legal assembly language statement.

The following examples show model statements:

```
MOV     %6,R12
```

This model statement is itself an assembly language source statement that contains a machine instruction.

```
:P7.S:    MPY    :P2.S:,R8 :V4.S:
```

This model statement begins with the string component of variable P7. Three blanks, MPY, and three more blanks are concatenated to the string. The string component of variable P2 is concatenated to the result, to which R8 and three blanks are concatenated. A final concatenation places the string component of variable V4 in the model statement. This produces an assembly language instruction in which the label, comment and part of the operand fields are supplied as string components.

```
:MS.S:
```

This model statement is the string component of variable MS. Preceding statements in the macro definition must place a valid assembly language source statement in the string component to prevent assembly errors.

> **Note:**
>
> Conditional assembly directives may not appear as operations in a model statement. Comments supplied in model statements may not contain periods (.) since the macro assembler scans comments in the same way as model statements and improper use of punctuation may cause syntax errors.

8.8 Macro Examples

Macros may simply substitute a machine instruction for a macro instruction, or they may include conditional processing, access the assembler symbol table, and employ recursion. Several examples of macro definitions are described in the following paragraphs.

8.8.1 Macro ID

Example macro ID is a macro with a default value. The macro supplies two DATA directives to the source program. It consists of nine macro language statements, four of which are model statements.

```
ID      $MACRO  WS,PC       Defines ID with parameters WS and PC
        DATA    :WS.S:      Model statement - places a DATA direc-
                            tive with the string of the first pa-
                            rameter as the operand in the source
                            program.
        $IF     PC.A&$PCALL Tests for presence of parameter PC
        DATA    :PC.S:,15   Model statement - places a DATA direc-
                            tive in the source program. The first
                            operand is the string of the second
                            parameter, and the second operand is
                            15. This statement is processed if the
                            second parameter is present.
        $ELSE               Start of alternate portion of defi-
                            nition.
        DATA    START,15    Model statement - places a DATA direc-
                            tive in the source program. The first
                            operand is label START, and the second
                            operand is 15. This statement is pro-
                            cessed if the second parameter is
                            omitted.
START   EQU     $           Model statement - places a label START
                            in the source program. This statement
                            is processed if the second parameter
                            is omitted.
        $ENDIF              End of conditional processing.
        $END                End of macro.
```

The macro call syntax is:

```
[<LABEL>] ID <address>[,<address>] [<comment>]
```

The addresses may be expressions or symbols.

A sample ID call would be:

```
        ID      WORK1,BEGIN
```

This would be replaced with the following source code:

```
        DATA    WORK1
        DATA    BEGIN,15
```

If only one operand is supplied, the macro instruction could be coded as follows:

```
          ID     WORK2
```

This would produce the following source code:

```
          DATA   WORK2
          DATA   START,15
START     EQU    $
```

This form of the macro instruction imposes two restrictions on the source program:

1) The source program may not use the label START and
2) May not call macro ID more than once.

Problems with labels supplied in macros may be prevented by reserving certain characters for use in macro-generated labels. A macro definition may maintain a count of the number of times it is called and use this count in each label generated by the macro.

8.8.2 Macro GENCMT

This example shows how to implement both those comments which appear in the macro definition only and those which appear in the macro expansion. When this macro is called, the statement in line six generates a comment.

```
 0001                        IDT    'GENCMT'
 0002              GENCMT    $MACRO
 0003                        $VAR V
 0004              * This is a macro definition comment     *
 0005                        $ASG '*' TO V.S
 0006              :V.S: This is a macro expansion comment *
 0007                        $END
 0008                        GENCMT
 0001              * This is a macro expansion comment      *
 0009 0000 0000             DATA 0,1
      0002 0001
 0010                        GENCMT
 0001              * This is a macro expansion comment      *
 0011                        GENCMT
 0001              * This is a macro expansion comment      *
 0012 0004 0004             DATA 4
 0013                        END
NO ERRORS, NO WARNINGS
```

8.8.3 Macro FACT

This example shows the recursive use of macros. FACT produces the assembly code necessary to calculate the factorial of N, and store that value at data memory address LOC. Macro FACT accomplishes this by calling FACT1, which calls itself recursively.

```
FACT   $MACRO N,LOC
       $IF    N.V<2
       MOV    %1,A           * 1% = 0% =1
       STA    @:LOC:
       $ELSE
       MOV    %:N.V:,A       * N greater than/equal 2,
       STA    @:LOC:         * so store N at LOC
       $ASG   N.V-1 TO N.V   * Decrement N
       FACT1  :N.V:,:LOC:    * Do Factorial of N-1
       $ENDIF
       $END
*
FACT1  $MACRO M,AREA
       $IF    M.V>1
       LDA    @:AREA:        * Multiply factorial so far
       MPY    %:M.V:,A       * by current position
       MOV    B,A
       STA    @:AREA:        * Save result
       $ASG   M.V-1 TO M.V   * Decrement position
       FACT1  :M.V:,:AREA:   * Recursively calls itself
       $ENDIF
       $END
```

8.8.4 Macro PULSE

This is a set of macros in which the name describes an addressing mode expected by the macro. The example assigns Register A to a port, Register B to a port, and an immediate value to a port. These macros can be useful in programming I/O routines.

```
PULSEA  $MACRO  PX
        ORP     A,:PX.S:
        $END
*
PULSEB  $MACRO  PX
        ORP     B,:PX.S:
        $END
*
PULSEI  $MACRO  I,PX
        ORP     %:I.S:,:PX.S:
        $END
```

8.9 Macro Error Messages

Table 8-6 lists and defines the Macro error messages which may be generated.

Table 8-6. Macro Error Messages

MACRO ERROR MESSAGE	DESCRIPTION
MACRO LINE TOO LONG	In a macro definition, macro directive lines may only be 58 characters long, and model statements, when fully expanded, may only be 60 characters long.
LONG MACRO VARIABLE QUALIFIER	Macro variable qualifiers may only be one or two characters in length.
TOO MANY MANY VARIABLES	The total number of macro parameters, variables and labels in one macro definition may not exceed 128.
INVALID MACRO QUALIFIER	The only valid macro qualifiers are: S,V, L, A, SS, SV, SL and SA.
VARIABLE ALREADY DEFINED	A macro variable cannot be redefined within a macro.
IF LEVEL EXCEEDED	The maximum nesting level of $IF directives is 44.
MACRO ASSEMBLER	The Macro Assembler has detected an internal PROGRAM ERROR error. These can be caused by incorrect syntax.

9. Design Aids

This section contains sample TMS7000 applications to aid you in system development.

9.1 Microprocessor Interface Example

Figure 9-1 illustrates a method for interfacing a TMS70x2 microcomputer to external memory devices such as EPROM and RAM. This interface is designed to operate at the TMS70x2's maximum operating frequency (8 MHz). Any combination of ROM, RAM or other peripheral devices could be added into the circuit and enabled by the other $\overline{SEL}$ pins, provided that their timing requirements allow them to be interfaced to the TMS70x2.

In this circuit, the Mode Control pin (MC) is tied to V_{CC}, placing the TMS70x2 in Microprocessor mode. All 16 addressing bits on Ports C and D are available in Microprocessor mode. The on-chip ROM is disabled in this mode, and its address space is available externally. For more information on port and mode operation see Section 3.

Note the following features in this sample circuit:

- Port A and the lower nibble of Port B operate the same as in the Single-Chip mode.
- The memory control signals are brought out on the upper nibble of Port B.
- Port C becomes the multiplexed least significant 8-bit address bus (A7–A0) and full 8-bit data bus.
- Port D becomes the most significant 8-bit address bus (A15–A8).
- The least significant 8 bits of the 16-bit address bus (A7–A0) are latched into the SN74AS373 (U2) by the ALATCH signal during read/write memory cycles.
- A full address decode is accomplished with the SN74AS138 (U3). Eight memory select lines ($\overline{SEL7}$–$\overline{SEL0}$) are generated by U3 and are each individually activated on an 8K-byte address block. Table 9-1 lists the address range decoded by each select pin.

Table 9-1. Memory Address Decode

PIN	ADDRESS RANGE
$\overline{SEL7}$	>E000 to >FFFF
$\overline{SEL6}$	>C000 to >DFFF
$\overline{SEL5}$	>A000 to >BFFF
$\overline{SEL4}$	>8000 to >9FFF
$\overline{SEL3}$	>6000 to >7FFF
$\overline{SEL2}$	>4000 to >5FFF
$\overline{SEL1}$	>2000 to >3FFF
$\overline{SEL0}$	>0000 to >1FFF

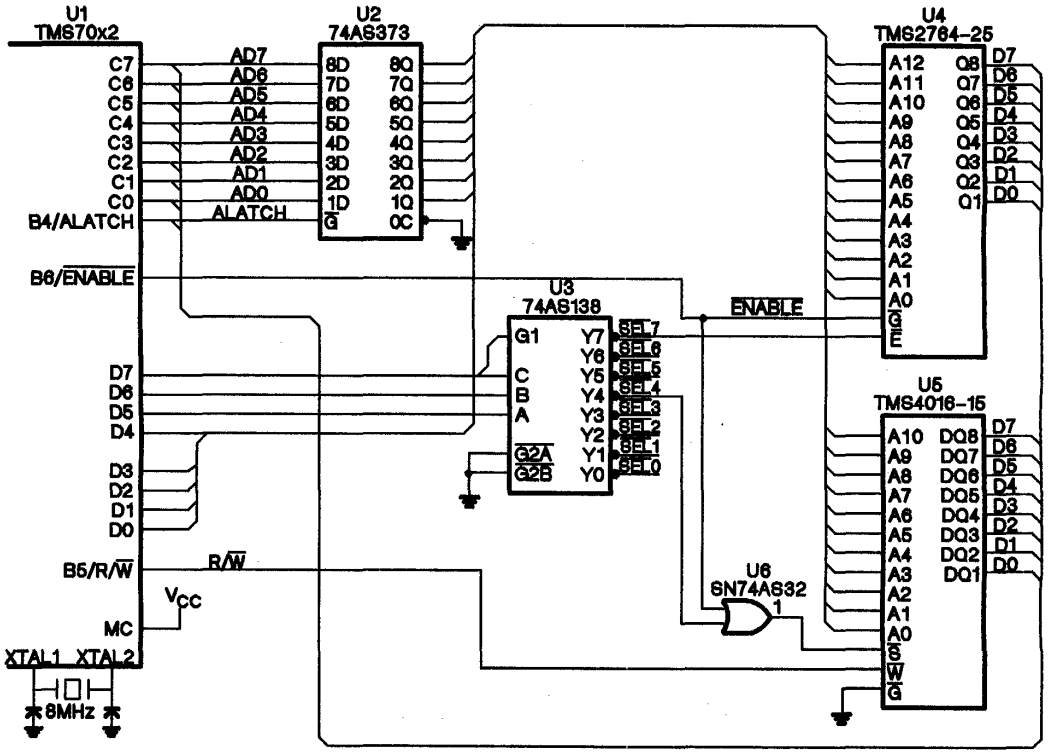


Figure 9-1. TMS70x2 Microprocessor Interface Sample Circuit

The devices used in this circuit are:

U1 **TMS70X2** - 8-bit microcomputer with UART.

U2 **SN74AS373** - The AS version of the 373 is used in this circuit, allowing use of the less expensive TMS2764-25 EPROM chip (U4) instead of the TMS2764-20 EPROM chip.

U3 **SN74AS138** - Like U2, the AS version of the 138 allows use of less expensive EPROMs.

U4 **TMS2764-25** - This EPROM chip is the slowest (least expensive) device that can be used in this circuit because the timing requirement [$T_{a(A-D)}$] for the TMS70x2 is 260 ns. The propogation delay through U2 is 6 ns, so only 254 ns remain for the EPROM chip to use. Therefore, the TMS2764-25 with its 250 ns access time [$T_{a(A)}$] was selected.

U5 **TMS4016-15** - This is the slowest RAM chip that can be used in this circuit because the timing requirement [$T_{a(EL-D)}$] for the TMS70x2 is 82 ns. The propogation delay through U6 is 5.8 ns, so only 76.2 ns remain for U5 to use. Therefore, the TMS4016-15 with its 75-ns delay time was selected.

U6 **SN74AS32** - The AS version of this chip allows use of the less expensive TMS4016-15 RAM instead of the TMS4016-12 RAM.

9.1.1 Read Cycle Timing

The TMS70x2 requires a minimum ***address-to-data access time*** [$t_{a(A-D)}$] of 260 ns at 8 MHz. $t_{a(A-D)}$ for the TMS2764-25 in this circuit is:

$$\begin{aligned} \text{Access time (260 ns)} &\geq t_{phl[U2]} + t_{a(A)[U4]} \\ &\geq 6 + 250 \\ 260 \text{ ns} &\geq 256 \text{ ns} \end{aligned}$$

$t_{a(A-D)}$ for the TMS4016-15 in this circuit is:

$$\begin{aligned} \text{Access time (260 ns)} &\geq t_{phl[U2]} + t_{a(A)[U5]} \\ &\geq 6 + 150 \\ 260 \text{ ns} &\geq 156 \text{ ns} \end{aligned}$$

The TMS70x2 parameter used to calculate $t_{a(A-D)}$ will also be used to calculate ***chip-select-to-data access time***. $t_{a(E)}$ for the TMS2764-25 in this circuit is:

$$\begin{aligned} \text{Access time (260 ns)} &\geq t_{phl[U3]} + t_{a(E)[U4]} \\ &\geq 6 + 250 \\ 260 \text{ ns} &\geq 256 \text{ ns} \end{aligned}$$

Since the chip select to the TMS4016-12 is gated with the ENABLE signal, use the access time $T_{a(EL-D)}$ to calculate the ***chip-select-to-data time***. $t_{a(S)}$ for the TMS4016-15 in this circuit is:

$$\begin{aligned} \text{Access time (82 ns)} &\geq t_{plh[U6]} + t_{a(S)[U5]} \\ &\geq 5.8 + 75 \\ 82 \text{ ns} &\geq 80.8 \text{ ns} \end{aligned}$$

The TMS70x2 requires a minimum ***$\overline{\text{ENABLE}}$-rise-to-data-disable time*** of 100 ns at 8 MHz. The minimum requirement for the TMS2764-25 in this circuit is:

$$\begin{aligned} \text{Disable time (100 ns)} &\geq t_{dis(G)[U4]} \\ 100 \text{ ns} &\geq 85 \text{ ns} \end{aligned}$$

The requirement for the TMS4016-15 in this circuit is:

$$\begin{aligned} \text{Disable time (100 ns)} &\geq t_{dis(S)[U5]} + t_{phl[U6]} \\ &\geq 50 + 5.8 \\ 100 \text{ ns} &\geq 55.8 \text{ ns} \end{aligned}$$

9.1.2 Write Cycle Timing for Microprocessor Mode

The TMS70x2 requires a minimum ***data-output-valid time*** ($T_{d(EH-A)}$) of 80 ns at 8 MHz.

Since $\overline{S}$ is gated to the $\overline{\text{ENABLE}}$ line, the $\overline{\text{ENABLE}}$ signal can be used calculate the data-output requirement for the TMS4016-15.

$$\begin{aligned} \text{Output valid (80 ns)} &\geq t_{phl[U6]} + t_{h(D)[U5]} \\ &\geq 5.8 + 10 \\ 80 \text{ ns} &\geq 15.8 \text{ ns} \end{aligned}$$

Table 9-2. Memory Interface Timing

PARAMETER		MIN	MAX	UNIT
$t_{c(C)}$	CLKOUT cycle time†	250	2000	ns
$t_{w(CH)}$	CLKOUT high pulse duration	$0.5t_{c(C)}-40$	$0.5t_{c(C)}+10$	ns
$t_{w(CL)}$	CLKOUT low pulse duration	$0.5t_{c(C)}-40$	$0.5t_{c(C)}+15$	ns
$t_{d(CH-JL)}$	Delay time, CLKOUT rise to ALATCH fall	$0.5t_{c(C)}-10$	$0.5t_{c(C)}+30$	ns
$t_{w(JH)}$	ALATCH high pulse duration	$0.25t_{c(C)}-15$	$0.25t_{c(C)}+30$	ns
$t_{su(HA-JL)}$	Setup time, high address valid before ALATCH fall	$0.25t_{c(C)}-40$	$0.25t_{c(C)}+45$	ns
$t_{su(LA-JL)}$	Setup time, low address valid before ALATCH fall	$0.25t_{c(C)}-40$	$0.25t_{c(C)}+15$	ns
$t_{h(JL-LA)}$	Hold time, low address valid after ALATCH fall	$0.25t_{c(C)}$	$0.25t_{c(C)}+45$	ns
$t_{su(RW-JL)}$	Setup time, R/$\overline{W}$ valid before ALATCH fall	$0.25t_{c(C)}-35$	$0.25t_{c(C)}+30$	ns
$t_{h(EH-RW)}$	Hold time, R/$\overline{W}$ valid after $\overline{ENABLE}$ rise	$0.5t_{c(C)}-40$		ns
$t_{h(EH-HA)}$	Hold time, high address valid after $\overline{ENABLE}$ rise	$0.5t_{c(C)}-50$		ns
$t_{su(Q-EH)}$	Setup time, data output valid before $\overline{ENABLE}$ rise	$0.5t_{c(C)}-45$		ns
$t_{h(EH-Q)}$	Hold time, data output valid after $\overline{ENABLE}$ rise	$0.5t_{c(C)}-45$		ns
$t_{d(LA-EL)}$	Delay time, low address high impedance to $\overline{ENABLE}$ fall	$0.25t_{c(C)}-45$	$0.25t_{c(C)}$	ns
$t_{d(EH-A)}$	Delay time, $\overline{ENABLE}$ rise to next address drive	$0.5t_{c(C)}-25$		ns
$t_{a(EL-D)}$	Access time, data input valid after $\overline{ENABLE}$ rise	$0.75t_{c(C)}-105$		ns
$t_{a(A-D)}$	Access time, address valid to data input valid	$1.5t_{c(C)}-115$		ns
$t_{d(A-EH)}$	Delay time, address valid to $\overline{ENABLE}$ rise	$1.5t_{c(C)}-80$	$1.5t_{c(C)}+30$	ns
$t_{h(EH-D)}$	Hold time, data input valid after $\overline{ENABLE}$ rise	0		ns
$t_{d(EH-JH)}$	Delay time, $\overline{ENABLE}$ rise to ALATCH rise	$0.5t_{c(C)}-25$	$0.5t_{c(C)}+10$	ns
$t_{d(CH-EL)}$	Delay time, CLKOUT rise to $\overline{ENABLE}$ fall	-10	35	ns

† $t_{c(C)}$ is defined to be $2/f_{osc}$ and may be referred to as a machine state or simply a state.

Table 9-3. TMS4016-15 Timing Characteristics

PARAMETER		MIN	MAX	UNIT
$t_{a(A)}$	Access time from address		150	ns
$t_{a(\overline{S})}$	Access time from chip select low		75	ns
$t_{dis(\overline{S})}$	Output disable time after chip select high		50	ns
$t_{h(A)}$	Address hold time	0		ns
$t_{su(D)}$	Data setup time	60		ns
$t_{h(D)}$	Data hold time	10		ns

Table 9-4. TMS2764-25 Timing Characteristics

PARAMETER		MIN	MAX	UNIT
$t_{a(A)}$	Access time from address		250	ns
$t_{a(E)}$	Access time from $\overline{E}$		250	ns
$t_{en(G)}$	Output enable time from $\overline{G}$	100		ns
$t_{dis(G)}$	Output disable from $\overline{G}$	0	85	ns

Table 9-5. SN74AS373, SN74AS138, and SN74AS32 Propogation Delay Times

PARAMETER		MIN	MAX	UNIT
t_{pd}	Propogation delay, SN74AS373		6	ns
t_{pd}	Propogation delay, SN74AS138		6	ns
t_{pd}	Propogation delay, SN74AS32		5.8	ns

9.2 Programming the TMS7742

The TMS7742 is an EPROM version of the TMS7042. It can be programmed using these devices:

- Standard PROM programmer (see Section 9.2.1, page 9-7)
- TMS7000 Evaluation Module (see Section 9.2.2, page 9-8)
- TMS7000 XDS Emulator (see Section 9.2.3, page 9-9)

The TMS7742 can emulate the TMS7020, TMS7040, and TMS7042:

- **TMS7020 and TMS7040 Emulation:**

  The TMS7742 can emulate the TMS7020 and TMS7040 in all operating modes. It does not directly emulate edge- and level-sensitive interrupts, but does emulate level-sensitive only interrupts.

- **TMS7042 Emulation:**

  The TMS7742 can directly emulate the TMS7042 in all operating modes at up to 5 MHz operation.

Table 9-6 shows the pin conditions required for operating in the various modes. Note that the $\overline{\text{RESET}}$ and XTAL2 pins must be held low to enter EPROM mode.

Table 9-6. Mode Select Conditions for the TMS7742

MODE SELECT		SINGLE-CHIP	PERIPH.-EXPANSION	FULL-EXPANSION	MICRO-PROCESSOR	EPROM PROG. MODE	EPROM VERIFY MODE
I/O Control register	Bit 7	0	0	1	X	X	X
	Bit 6	0	1	0	X	X	X
Mode Control pin (MC)		V_{SS}	V_{SS}	V_{SS}	V_{CC}	V_{PP}	V_{SS}
$\overline{\text{RESET}}$ pin		V_{CC}	V_{CC}	V_{CC}	V_{CC}	V_{SS}	V_{SS}
XTAL2 pin		N/A	N/A	N/A	N/A	V_{SS}	V_{SS}

Notes: 1. X = don't care
2. N/A = not applicable

9.2.1 Programming the TMS7742 Using a PROM Programmer

A PROM programmer can be used to program the TMS7742 in a manner similar to programming a TMS2732A EPROM. A 40-to-24-pin conversion socket is required and $\overline{\text{RESET}}$ and XTAL2 must be grounded. Some PROM programmers implement current-limiting circuitry to sense correct EPROM placements. The TMS7742 can draw a maximum of 250 mA during programming; if your PROM programmer produces an EPROM placement error, you must supply an external +5 V ±10% power supply to the TMS7742. Figure 9-2 shows the connections for the 40-to-24-pin socket.

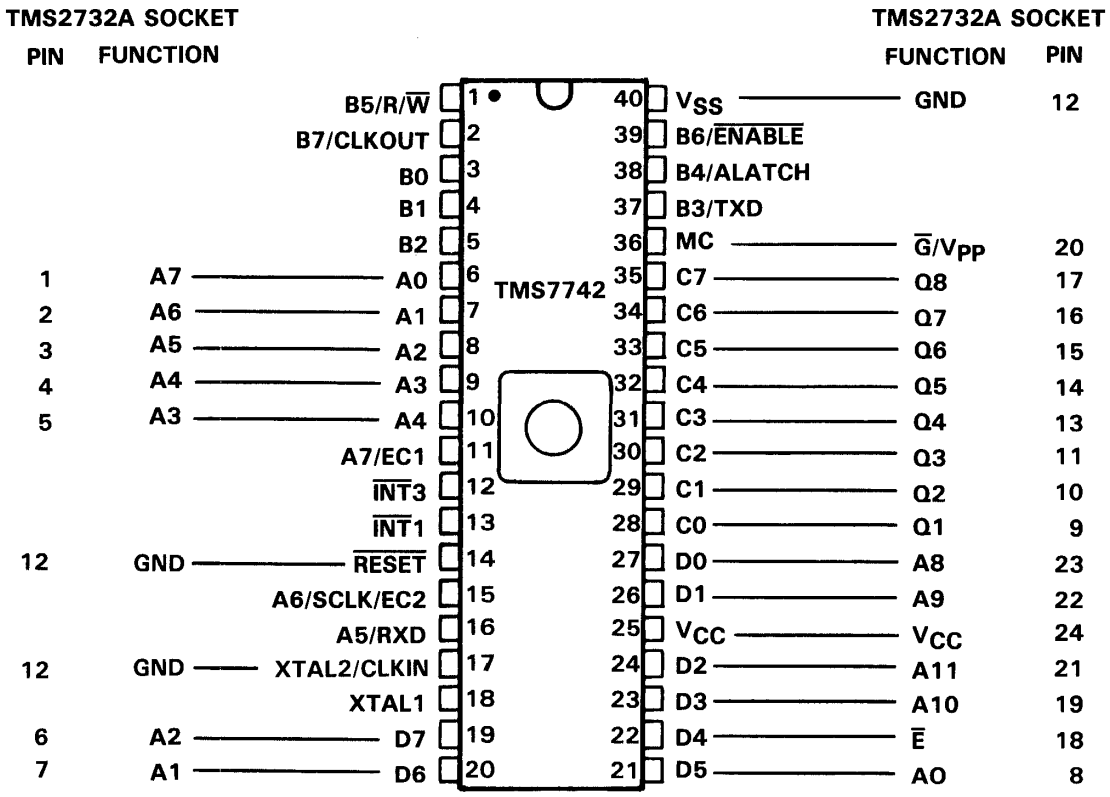


Figure 9-2. PROM Programmer 40-to-24-Pin Conversion Socket

Use the following sample procedure to program the TMS7742 on a PROM programmer:

1) Insert the TMS7742 into the conversion socket.

2) Place the conversion socket (with the TMS7742) into the 24-pin socket on the PROM programmer.

3) Program and verify the contents of the TMS7742 in the same manner as any standard EPROM.

9.2.2 Programming the TMS7742 Using the TMS7000 Evaluation Module

The RTC/EVM7000 (TMS7000 Evaluation Module) can be used to program the TMS7742. A 40-to-28-pin conversion socket is required and $\overline{\text{RESET}}$ and XTAL2 must be grounded. Figure 9-3 shows the connections for the 40-to-24-pin socket.

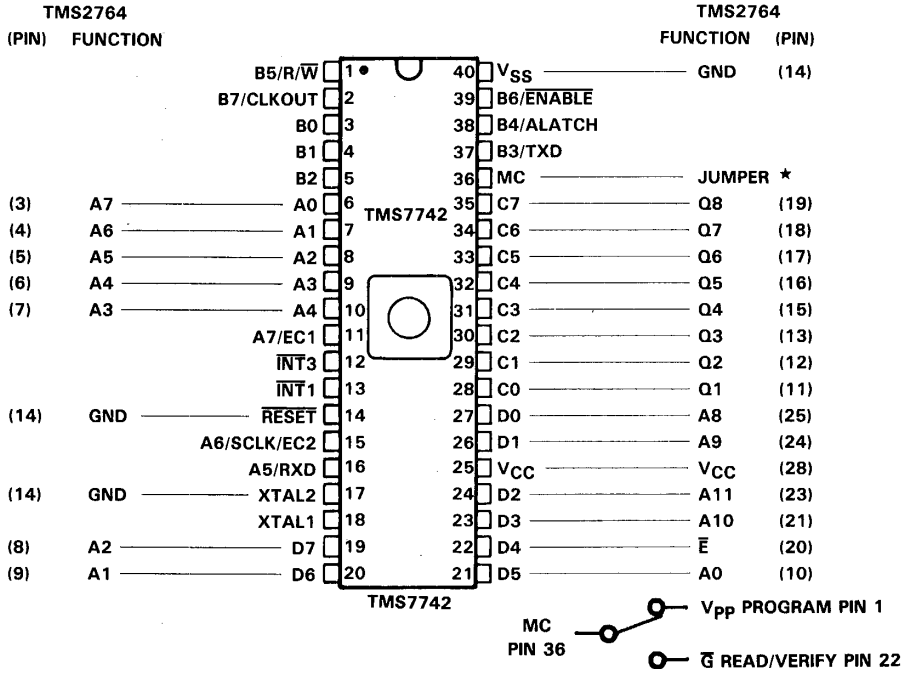


Figure 9-3. RTC/EVM7000 40-to-28-Pin Conversion Socket

Use the following procedure to program the TMS7742 on an RTC/EVM7000:

1) Verify that the TMS7742 is erased (all >FFs).

   a) Set the switch between pin 36 on the TMS7742 and pin 22 on the conversion socket (read/verify position).
   b) Enter: ?VE 0 FFF 2 <CR>

2) Program the TMS7742. Note that the program to be loaded into the TMS7742 must reside in EVM memory beginning at address >F006 or above.

   a) Set the switch between pin 36 on the TMS7742 and pin 1 on the conversion socket (program position).
   b) Enter: ?PE 0 FFF F000 2 <CR>

3) Compare the TMS7742 EPROM to EVM memory to verify that they are identical.

   a) Set the switch between pin 36 on the TMS7742 and pin 22 on the conversion socket (read/verify position).
   b) Enter: ?CE 0 FFF F000 2 <CR>

9.2.3 Programming the TMS7000 using the TMS7000 XDS

The TMS7742 can be programmed using the TMS7000 XDS, the driver program, and an interface board. Figure 9-4 shows the schematic for the interface board and Figure 9-5 contains the driver program.

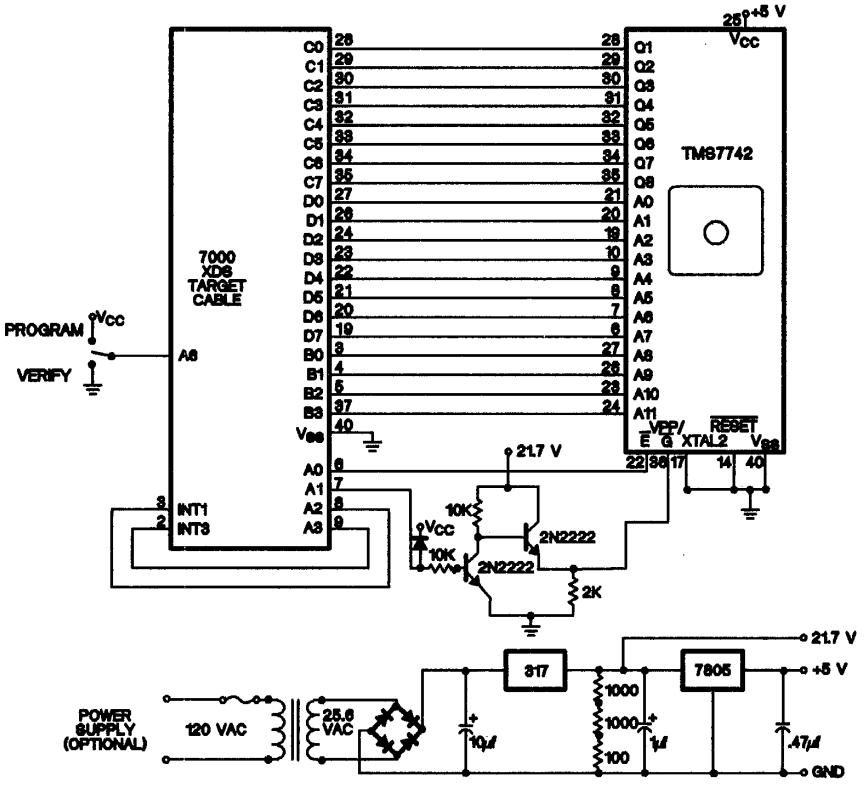


Figure 9-4. Interface Circuit for Programming the TMS7742 with the TMS7000 XDS

```
        IDT   'EPROM'
*
*  This program checks to see if the TMS7742 is blank,
*  then programs and verifies the EPROM byte by byte.
*  The program can also verify that the contents of
*  XDS memory are identical to the TMS7742.
*
***** Register File *********************************
*
ADDRES  EQU   R5            Current address
COUNT   EQU   R7            Number of bytes to program
COUNT2  EQU   R8
*
***** Peripheral File *******************************
*
INTROL  EQU   P0            Interrupt control
PORTA   EQU   P4
PORTB   EQU   P6
PORTC   EQU   P8
PORTD   EQU   P10
ADDR    EQU   P5
DDDR    EQU   P11
CDDR    EQU   P9
*
***** Control Constants for Port A ******************
*
*  A0 = E-
*  A1 = G-/VPP 21V
*  A2 = INT1-  light 2
*  A3 = INT3-  light 3
* 32VE-
ENOT    EQU   ?00000001     E-
VPP21   EQU   ?00000010     21V to VPP/G
ERR1    EQU   ?00001011     Not blank error
ERR2    EQU   ?00000111     Not programming correctly error
ERR3    EQU   ?00000011     Failed verify test
READ1   EQU   ?00001110     Read setup
READ2   EQU   ?00001111     Release read setup
*
*****************************************************
        AORG  >E000
START   DINT
        MOVP  %>FF,ADDR     Outputs
        MOVP  %>FF,DDDR     Outputs
        MOVP  %>00,CDDR     Inputs
        MOVP  %0,INTROL     Full-Expansion mode, no ints
        MOVP  %178,P2       Timer latch
        BTJZP %>40,PORTA,VERIFY  Verify or program?
BLANK   MOVD  %>FFFF,ADDRES Check memory for all blanks
        MOVD  %>FFF,COUNT   Put in counts and pointers
LOOPBL  CALL  @READ         Read memory
        CMP   %>FF,A        Is it blank? (>FF = blank)
        JNZ   ERROR1        If no, error
        DECD  ADDRES        Next address
        DECD  COUNT
        JC    LOOPBL        End of routine?
```

Figure 9-5. Driver Program for Programming the TMS7742 with the TMS7000 XDS

```
PROGRM  MOVD  %>FFFF,ADDRES Program EPROM
        MOVD  %>FFF,COUNT   Put in counters
LOOPPR  LDA   *ADDRES       Get data from XDS memory
        MOV   A,B
        CALL  @WRITE        Program one address
        DECD  ADDRES        Next address
        DECD  COUNT
        JC    LOOPPR        End of routine?
*
VERIFY  MOVD  %>FFFF,ADDRES Check memory for all blanks
        MOVD  %>FFF,COUNT   Put in counters and pointers
LOOPVE  CALL  @READ         Read EPROM
        MOV   A,B
        LDA   *ADDRES       Get original data
        CMP   B,A           Does EPROM compare to original?
        JNZ   ERROR3        If no, error
        DECD  ADDRES        Next address
        DECD  COUNT
        JC    LOOPVE        End of routine?
**************************************************
ERROR1  MOVP  %ERR1,PORTA   Fail blank - light 2
        JMP   STOP2
ERROR2  MOVP  %ERR2,PORTA   Fail programming - light 3
        JMP   STOP2
ERROR3  MOVP  %ERR3,PORTA   Fail verify - lights 2 and 3
STOP2   IDLE
        JMP   STOP2
**************************************************
READ    CALL  @SETUP        Put address on bus
READB   MOVP  %>00,CDDR     Port C = inputs
        MOVP  %READ1,PORTA  Turn on enable
        MOVP  PORTC,A       Read data
        MOVP  %READ2,PORTA  Turn off enable
*
WRITE   CALL  @SETUP        Put address on bus
        MOVP  B,PORTC       Put data on bus
        MOV   %3,COUNT2     Initialize counter
*
PULSE   MOVP  %>FF,CDDR     Port C = outputs
        ANDP  %#VPP21,PORTA Turn on VPP
        ANDP  %#ENOT,PORTA  Turn on E-
*
        MOVP  %>80+31,P3    Start timer
HERE2   BTJZP %8,INTROL,HERE2 Wait for timer countout
        MOVP  %?00101010,INTROL Clear timer flag
        DJNZ  COUNT2,HERE2  Wait a total of 55 ms
*
        ORP   %ENOT,PORTA   Turn off E-
        ORP   %VPP21,PORTA  Turn off VPP
        CALL  @READB        Read EPROM
        CMP   A,B           Compare to actual data
        JNE   ERROR2        If not equal pulse again
        RETS                then turn on light 3
*
```

Figure 9-5. Driver Program for Programming the TMS7742 with the TMS7000 XDS (Concluded)

Use the following procedure to program the TMS7742 using the TMS7000 XDS Emulator. To avoid the possiblity of leaving +21 V on V_{PP}, do not stop the program until the IDLE light is on.

1) Enter: INIT(3,0,0,0)

2) Enter: ROM=E000

3) Set the switch on interface board to *program*.

4) Download object code into XDS memory (>F000->FFFF).

5) Download the driver program into XDS memory (this will not affect the present program at memory locations >F000->FFFF).

6) Use the MR command to set the following values:

   PC = >E000,
   ST = >00,
   SP = >60

7) Enter: P5=FF, P4=FF (This clears the programming voltages on the socket.)

8) Insert the target cable into socket A.

9) Insert the TMS7742 into socket B.

10) Enter: RUN (Light 4 should go on.)

11) The program will take approximately four minutes to complete; light 1 will go on when the program is complete.

12) If an error was encountered, light 2 and/or light 3 will be lit. Examine addresses >04 and >05 for the error location. Register A contains EPROM data, and Register B contains the original data.

13) Remove the TMS7742.

If an error condition is found, then the indicator lights on the XDS front panel will show the pattern for the error. Table 9-7 shows the status conditions indicated by the front panel lights.

Table 9-7. Error Patterns for XDS

XDS LIGHTS				STATUS
1	2	3	4	
0	0	0	0	Program is not running
0	0	0	1	Program is running
1	0	0	1	Program is finished, no errors
1	0	1	1	Programming error
1	1	0	1	EPROM was not blank
1	1	1	1	Verify error

Light 1 Program is in IDLE state
Light 2 $\overline{\text{INT1}}$
Light 3 $\overline{\text{INT3}}$
Light 4 Processor is running

To verify the TMS7742 EPROM memory against the XDS memory, set the switch on the interface board to *verify* and follow the programming procedure. As a precaution, do not connect the +21.7-V power supply.

9.2.4 TMS7742 Erasure

The TMS7742 can be erase by exposing the chip to shortwave ultraviolet light that has a wavelength of 253.7 nanometers (2537 angstroms). The recommended minimum exposure dose (UV intensity × exposure time) is 15 watt-seconds per square centimeter. The lamp should be located about 2.5 centimeters (1 inch) above the chip during erasure. After erasure, all bits are at a high level. Note that normal ambient light contains the correct wavelength for erasure; therefore, when using the TMS7742 the window should be covered with an opaque label.

9.3 Serial Communication with the TMS7000 Family

This section discusses using the TMS7000 for serial communication with a UART (Universal Asynchronous Receiver Transmitter). It describes implementing the UART function in software using any TMS7000 device and with the on-chip serial port using the TMS7042.

9.3.1 Communication Formats

The TMS7000 family handles three basic formats of serial communication - Asynchronous, Isosynchronous and Serial I/O. The first two require framing bits to be added to the data, allowing the receiver to properly detect incoming data. The last two require an addition serial clock to synchronize the data. This UART routine uses Asyncronous communications; all the formats are discussed in detail in Section 3.

In **Asynchronous format**, as shown in Figure 9-6, each character to be transmitted is preceded by a Start framing bit and followed by a Parity bit (if parity is enabled), then one or more Stop framing bits.

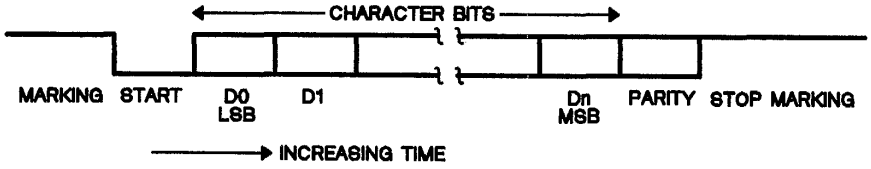


Figure 9-6. Asynchronous Communication Format

The **Start** bit is a logical 0, or **space**. It notifies the receiver to start assembling a character and allows the receiver to synchronize itself with the transmitter.

A **Parity** bit is an additional bit added to a character for error checking. The Parity bit is set to 0 or 1 in order to make the number of 1s in the character (including the Parity bit) even or odd depending on whether even or odd parity is selected.

The **Stop** bit is a logical 1 or **mark**. One or more **Stop** bits are added to the end of the character to ensure that the **Start** bit of the next character will cause a transition on the communication line.

The connections for both the software and on-chip hardware UARTs are identical. Both use A5/RX for the incomming data and B3/TX for outgoing data. The connections are shown in Figure 9-7. The TMS7000 outputs a TTL-level signal which must be converted to ±12 volts for RS-232-C compatibility. The 75188 and 75189 devices are used for this purpose.

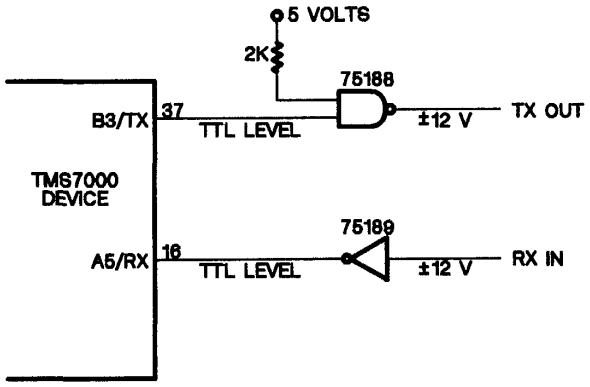


Figure 9-7. I/O Interface

9.3.2 Software UART (All TMS7000 Devices)

This software UART routine will run on any TMS7000 family microcomputer. It requires the use of one timer to produce a consistent baud rate without requiring full use of the program's time. This UART will run mainly in the Interrupt-2 routine, allowing the main program to run independently of the UART.

The timer is configured so that the interrupts arrive every half bit. This is because the receiver section must find the start bit as soon as possible, but it must also test the following bits at the middle of the bit. Testing at the edge of a bit time would produce data errors. Figure 9-8 shows the start bit detection.

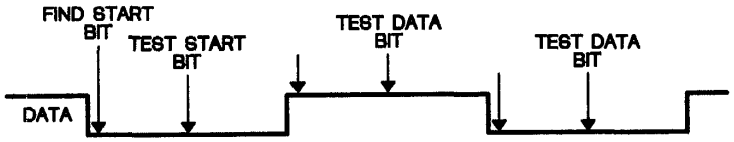


Figure 9-8. Start Bit Detection

The software, which consists of a receiver routine and a transmitter routine, runs mainly during the Interrupt-2 routine. Both routines maintain a progressive State Counter, which will have one of the following values to indicate its condition:

State 0 The receive portion is in this state until a low Start bit is detected.

State 1 This state begins a half bit later and tests for a valid Start bit.

State 2 and
State 3 The 8 character bits are built in states 2 and 3.

State 4 and

State 5 The Parity bit is received in states 4 and 5. If the parity does not agree with the parity of the input byte then a bit is set to indicate a parity error.

State 6 and State 7 These states look for the Stop bit. If the stop bit is not found, then another bit is set to indicate a framing error. The complete character is then placed in the RXSTOR register and a bit is set to indicate to the main program that a character is ready to be read. The main program must clear the parity and framing error bits.

The transmitter routine operates similarly to the receiver routine, using a separate State Counter to record its condition. The transmitter routine skips every other interrupt because the routine can be entered every full bit instead of every half bit, as in the receiver routine. The transmitter sends out bytes stored in a table. This table can be in either ROM or RAM and the table ends with a >FF to signify the end of string. Parity is calculated for both the receiver and transmitter by exclusive ORing the data bits together to produce even parity for the string.

9.3.2.1 Software UART Enhancements

If it is not necessary for the transmitter and receiver to run simultanously, then several inprovements can be implemented.

- The transmitter's baud rate can easily be doubled by interrupting every full bit instead of every half bit.
- The receiver can be improved by connecting the RX-in line to RX and to an Interrupt pin ($\overline{INT1}$ or $\overline{INT3}$). When the Start bit is detected, the program enters the external interrupt routine. This interrupt routine must start the timer to count out one-half bit and also disable the interrupt. When the half-bit interrupt occurs, the timer must be reset and restarted to produce a full-bit interrupt; this would occur in the middle of the data bits.
- The parity can be selected by testing an even/odd bit and setting the initial parity register (TXPAR, RXPAR) to the correct value. Currently, the registers are cleared for every new byte, producing even parity.
- An extra stop bit could be added by using a test bit and repeating States 6 and 7 if the bit is set.
- Additional RS-232-C signals could be added to the program to interface to more complex equipment.

9.3.2.2 Software UART Routines

```
*
                IDT     'SWUART'
                OPTION  XREF,SYMLST
          *
          *  This program simultaneously transmits and
          *  receives RS-232-C format data.
          *  Maximum baud rate = 4800 at 8 Mhz.
          *
          *  Transmitt pin out = B3
          *  Receiver  pin in  = A5
          *
          ***** REGISTER  FILE *********************************
          *
          *       UARTREGISTERS
          *
0002      STATER  EQU R2   The state of the current receive data
0003      STATET  EQU R3   The state of the current transmit data
0004      RXBUF   EQU R4   Build the input byte here
0005      RXCNT   EQU R5   The number of bits left to receive
0006      RXSTOR  EQU R6   Pick up the finished input word here
0007      RXPAR   EQU R7   Bit 0=parity (7 other bits free)
0008      TXCNT   EQU R8   The number of bits left to transmit
0009      TXTABL  EQU R9   Address offset from String beginning
000A      TXBUF   EQU R10  Shift the out word from here.
000B      TXPAR   EQU R11  Bit 0 = parity ( 7 other bits free)
000C      BITS    EQU R12  Bit0= Transmit routine now or next INT
          *                  Bit1= Transmitter active now
          *                  Bit2= Receiver contains word now
          *                  Bit3= Framing error ( bad stop bit)
          *                  Bit4= Finished with the string output
          *                  Bit5= Parity error
          *
          ***** PERIPHERAL FILE ******************************
          *      PERIPHERAL PORTS AND REGISTERS
0000      IOCNTL EQU  P0   Interrupt control
0002      TIMERL EQU  P2   Timer latch value
0003      TIMERC EQU  P3   Timer control
0004      PORTA  EQU  P4   Port A data
0006      PORTB  EQU  P6   Port B data
0005      ADDR   EQU  P5   Port A Data Direction register
```

```
                        *
                        ********  CONTROL CONSTANTS FOR PORT A  ***********
                        *
                        *                                        BAUD RATE
                        * CRYSTAL                    300   600   1200   2400   4800
                        * 5 MHz    Latch             129    63    129     64     32
                        *          Prescale            3     3      0      0      0
                        * 8 MHz    Latch             207   207    207    103     51
                        *          Prescale            3     1      0      0      0
                        *
                        *
                        *                                    CRYSTAL FREQ
                        *          (L+1)*(PS+1)= ------------------
                        *                        (BAUDRATE * 2) * 16
                        *
00CF                    BAUD1   EQU     207         Value for the timer latch
0083                    BAUD2   EQU     >80+3       Value for the timer control
                        *                           register
0001                    BIT0    EQU     1           Various bit constants to
0002                    BIT1    EQU     2           make code more readable
0004                    BIT2    EQU     4
0008                    BIT3    EQU     8
0010                    BIT4    EQU     16
0020                    BIT5    EQU     32
0040                    BIT6    EQU     64
0080                    BIT7    EQU     128
                        *
                        ***************************************************
                        *
F806                                    AORG  >F806
                        *
F806 06                 START   DINT                Disable all interrupts
F807 52 FD                      MOV     %>FD,B      Set index to clear out
F809 B5                         CLR     A           all of RAM
F80A AB 0001            CLEAR   STA     @1(B)       Store 0s into all of RAM
F80D CA FB                      DJNZ    B,CLEAR     Loop until RAM is all 0s
F80F 52 60                      MOV     %>60,B      Set stack pointer
F811 0D                         LDSP
                        *       MOV     %BIT1,BITS  Active transmittter and
                        *                           initialize receiver
F812 A2 2E 00                   MOVP    %?00101110,IOCNTL  Enable Timer INT
F815 A2 00 04                   MOVP    %?00000000,PORTA   Clear Port A
F818 A2 00 05                   MOVP    %?00000000,ADDR    Init. A5 for input
F81B A2 08 06                   MOVP    %?00001000,PORTB   Initialize Port B
F81E A2 CF 02                   MOVP    %BAUD1,TIMERL      Put the baud rate
F821 A2 83 03                   MOVP    %BAUD2,TIMERC      into the timer
                        *                                  latch and timer
                        *                                  control
F824 05                         EINT                       Start looking for
                        *                                  interrupts
                        *
F825 01                 LOOP    IDLE                Wait for timer interrupt
F826 E0 FD                      JMP     LOOP        or execute main program
                        *                           here
```

```
                        *
                        ************************************************
                        *
                        *       TIMER 1 INTERRUPT
                        *
     28                 INTER2  EQU   $          Start of timer interrupt
F828 B8                        PUSH  A          Store registers
F829 C8                        PUSH  B
F82A 32 02                     MOV   STATER,B   Get current receiver
                        *                       state
F82C CF                        RLC   B          Double in preparation
                        *                       for jump
F82D AE F844                   CALL  @JUMPR(B)  Go perform receiver
                        *                       tasks
                        *
F830 77 02 0C 0D               BTJZ  %BIT1,BITS,OUT Is a word being
                        *                           transmitted?
F834 75 01 0C                  XOR   %BIT0,BITS     Do only every
                        *                           other interrupt
F837 77 01 0C 06               BTJZ  %BIT0,BITS,OUT Transmit 1/2 the
                        *                           time
F83B 32 03                     MOV   STATET,B   Move transmit state to
                        *                       index
F83D CF                        RLC   B
F83E AE F8A5                   CALL  @JUMPT(B)  Go to proper state of
                        *                       routine
                        *
F841 C9                 OUT    POP   B
F842 B9                        POP   A          Restore the registers
F843 0B                        RETI             Exit the routine
                        *
                        *       RECEIVER JUMP TABLE
                        *
F844 E0 0E              JUMPR  JMP   STATE0     Check for start bit
F846 E0 13                     JMP   STATE1     Check for half a start
                        *                       bit
F848 E0 21                     JMP   STATE2     Bit boundry, wait for
                        *                       1/2 bit
F84A E0 22                     JMP   STATE3     Test input for data
F84C E0 1D                     JMP   STATE4     Parity bit boundary
F84E E0 34                     JMP   STATE5     Check parity bit
F850 E0 19                     JMP   STATE6     Stop bit boundary
F852 E0 47                     JMP   STATE7     Check middle of the stop
                        *                       bit
                        *                       Is the receive line low?
F854 A6 20 04 02 STATE0 BTJOP %BIT5,PORTA,ISPACE
F858 D3 02                     INC   STATER     If so, new start bit,
F85A 0A                 ISPACE RETS             go to next state,
                        *                       if not, do nothing
                        *                       Check for false starts
F85B A7 20 04 03 STATE1 BTJZP %BIT5,PORTA,ISTART
F85F D5 02                     CLR   STATER     Clear state if false
                        *                       start
F861 0A                        RETS
```

```
F862 72 08 05     ISTART  MOV   %8,RXCNT  Number of bits to
                  *                       receive
                  *                       Initialize parity
F865 73 FE 07             AND   %#BIT0,RXPAR
F868 D3 02                INC   STATER    Go to State 2
F86A 0A                   RETS
                  *
F86B              STATE2  EQU   $         States 2,4 and 6 are
F86B              STATE4  EQU   $         identical in operation
F86B D3 02        STATE6  INC   STATER    Half bit, go to next
                  *                       state
F86D 0A                   RETS
                  *
                  *                       Input new bit
F86E A7 20 04 01  STATE3  BTJZP %BIT5,PORTA,BITLOW
F872 07                   SETC            A 1 was found
F873 DD 04        BITLOW  RRC   RXBUF     Build the input word
                  *                       here
                  *                       Build up even parity
F875 45 04 07             XOR   RXBUF,RXPAR
F878 D2 02                DEC   STATER    Goto half state
                  *                       Is entire byte in?
F87A DA 05 06             DJNZ  RXCNT,OUTP3
                  *       Store byte in storage register
F87D 42 04 06             MOV   RXBUF,RXSTOR
F880 72 04 02             MOV   %4,STATER Go to State 4
F883 0A           OUTP3   RETS
                  *
                  *       Check for even parity (use BTJZ for
                  *       odd parity)
F884 76 01 07 09  STATE5  BTJO  %BIT0,RXPAR,IS1
                  *                       Out if both parities 0?
F888 A7 20 04 09  IS0     BTJZP %BIT5,PORTA,OUTPAR
F88C 74 20 0C     BADPAR  OR    %BIT5,BITS Bit 5= Parity error
F88F E0 04                JMP   OUTPAR
                  *       Continue if parities both =1
F891 A7 20 04 F7  IS1     BTJZP %BIT5,PORTA,BADPAR
F895 D3 02        OUTPAR  INC   STATER     Reset State Counter
F897 74 04 0C             OR    %BIT2,BITS Set 'Word ready' bit
F89A 0A                   RETS
                  *
                  *                       Stop bit = 1?
F89B A6 20 04 03  STATE7  BTJOP %BIT5,PORTA,ISSTOP
F89F 74 08 0C             OR    %BIT3,BITS Bit 3= Framing error
F8A2 D5 02        ISSTOP  CLR   STATER     Reset State Counter
F8A4 0A                   RETS
                  ************************************************
                  *       TRANSMITTER SECTION
                  *
                  *       TRANSMITTER JUMP TABLE
                  *
F8A5 E0 08        JUMPT   JMP   STATEA    Start outputting string
F8A7 E0 0B                JMP   STATEB    Output start bit
F8A9 E0 15                JMP   STATEC    Output data bits
F8AB E0 2A                JMP   STATED    Output parity bit
F8AD E0 37                JMP   STATEE    Output stop bit
```

```
                        *
F8AF D5 09              STATEA  CLR   TXTABL     Initialize table pointer
F8B1 8E F8F1                    CALL  @FIRST     Load the first byte into
                        *                        buffer
                        *                        Send out a Start bit
F8B4 A3 F7 06           STATEB  ANDP  %#BIT3,PORTB
F8B7 72 08 08                   MOV   %8,TXCNT   8 bits per character
                        *                        Initialize parity to 0
F8BA 73 FE 0B                   AND   %#BIT0,TXPAR
F8BD D3 03                      INC   STATET     Go to the next state
F8BF 0A                         RETS
                        *
F8C0 45 0A 0B           STATEC  XOR   TXBUF,TXPAR Build up Parity bit
                        *                        Send a 1 or a 0?
F8C3 77 01 0A 05                BTJZ  %BIT0,TXBUF,TRANS0
F8C7 A4 08 06                   ORP   %BIT3,PORTB Output a 1 bit
F8CA E0 03                      JMP   NXTBIT
F8CC A3 F7 06           TRANS0  ANDP  %#BIT3,PORTB Output a 0 bit
F8CF DC 0A              NXTBIT  RR    TXBUF       Point to the next bit
F8D1 DA 08 02                   DJNZ  TXCNT,OUTC  Are all 8 bits done
                        *                        yet?
F8D4 D3 03                      INC   STATET      Output stop bits next
F8D6 0A                 OUTC    RETS
                        *
                        *       Output even parity (use BTJO for
                        *       odd parity)
F8D7 77 01 0B 05        STATED  BTJZ  %BIT0,TXPAR,PARTY0
F8DB A4 08 06                   ORP   %BIT3,PORTB Output a 1 bit
F8DE E0 03                      JMP   OUTD
F8E0 A3 F7 06           PARTY0  ANDP  %#BIT3,PORTB Output a 0 bit
F8E3 D3 03              OUTD    INC   STATET      Output stop bit next
F8E5 0A                         RETS
                        *
F8E6 A4 08 06           STATEE  ORP   %BIT3,PORTB Send out a stop bit
F8E9 72 01 03                   MOV   %1,STATET   Send out start bit
                        *                        next
F8EC 74 01 0C                   OR    %BIT0,BITS  Go to TX routine every
                        *                        other interrupt
                        *
F8EF D3 09                      INC   TXTABL      Point to next byte
                        *                        from table
F8F1 32 09              FIRST   MOV   TXTABL,B    Setup output table
                        *                        pointer
F8F3 AA F908                    LDA   @STRING(B)  Get value from table
F8F6 72 01 03                   MOV   %1,STATET   Output Start bit next
F8F9 2D FF                      CMP   %>FF,A      FF = end of string
F8FB E6 08                      JNE   NEWTX       Jump if not end of
                        *                        string
F8FD 74 10 0C                   OR    %BIT4,BITS  End of text string,
                        *                        set bit
F900 73 FD 0C                   AND   %#BIT1,BITS Turn off transmitter
F903 D2 03                      DEC   STATET      Start at beginning
                        *                        next time
F905 D0 0A              NEWTX   MOV   A,TXBUF     Store new byte
F907 0A                         RETS
```

```
                        ***********************************************
                        *
                        *       This text string could be in RAM or ROM
F908 41 42 43 44 S      TEXT    'ABCDEFGHIJKLMNOPQRSTVUWXYZ1234567890'
F90C 45 46 47 48
F910 49 4A 4B 4C
F914 4D 4E 4F 50
F918 51 52 53 54
F91C 56 55 57 58
F920 59 5A 31 32
F924 33 34 35 36
F928 37 38 39 30
F92C FF                 BYTE    >FF     End of string byte
                        *
F92D                    INTER1  EQU     $       External interrupts
                        *                       vectors
F92D 0B                 INTER3  RETI            Not used in this program
                        *
FFF8                    AORG    >FFFE-(3*2)
FFF8 2D F828            DATA    INTER3,INTER2,INTER1,START
FFFC 2D F806
                        END
NO ERRORS, NO WARNINGS
```

9.3.3 Hardware UART (TMS70x2)

The main portions of the serial port are the receiver (RX), transmitter (TX), and timer (T3). The complete functional definition of the serial port is configured by the user program. A set of control words must first be sent out to configure the serial port. For more information about the serial port, see Section 3.

The serial port is controlled and accessed through registers in the Peripheral File. The registers associated with the serial port are shown in Table 9-8.

Table 9-8. Serial Port Control Registers

REGISTER		NAME	TYPE	FUNCTION
TMS70Cx2	TMS70x2			
P20	P17	SMODE	FIRST WRITE	Serial Port Mode
P21	P17	SCTL0	READ/WRITE†	Serial Port Control 0
P22	P17	SSTAT	READ	Serial Port Status
P23	P20	T3DATA	READ/WRITE	Timer 3 Data
P24	P21	SCTL1	READ/WRITE	Serial Port Control 1
P25	P22	RXBUF	READ	Receiver Buffer
P26	P23	TXBUF	WRITE	Transmission Buffer

† Write only for TMS70x2 devices

The hardware serial port program is divided into three sections:

1) The initialization section
2) The transmitter section
3) The receiver section

The transmitter and the receiver sections are in the serial-port interrupt service routine. The main body of the program follows the initialization section and runs between interrupts.

9.3.3.1 Initialization

The program first initializes all registers, starting with the interrupt control registers IOCNT0 and IOCNT1. The stack pointer is set and output ports A and B are initialized.

Next, the serial port registers are set up. The first write operation to PF location P17 immediately following a reset accesses the SMODE register. All subsequent writes to P17 access the control register SCTL0. If the condition of P17 is unknown, then writing a single 0 to P17 will cause the register to be SCTL0. The program can then reset the serial port by writing a 1 to the UR bit in SCTL0.

Finally, the serial port timer is started and the interrupts are enabled. The processor then waits for the timer interrupt to service the serial port. Faster baud rates allow less time for the main program to run, since it only runs between the interrupts.

INT4 is dedicated to the serial port. Three sources can generate an interrupt through INT4: the transmitter (TX), the receiver (RX), and Timer 3 (T3). The serial port can be driven by Timer 3 or external baud rate generator. The Timer 3 interrupt function is usually disabled when using the UART because the timer will interrupt 16 times for every bit or about 160 times per byte. In this HWUART program, the T3 interrupt is disabled and the internal Timer 3 is chosen as the serial clock.

9.3.3.2 Transmitter

When the program enters the serial port interrupt routine, it determines if the transmitter or receiver caused the interrupt. If the interrupt occurred because the transmitter is empty, then the program takes the next byte in the transmitter table and places it in the transmitter buffer. The first byte of the transmitter data contains the total number of bytes in the string. If the index is zero, the program places this byte count into the index register instead of transmitting it. This is an alternate method to the software UART's example of ending the string with a unique character.

9.3.3.3 Receiver

If the receiver causes an interrupt and no errors exist, then the program takes the value in the receiver buffer and places it into a receiver table. After the character is placed into the table, the character counter at the beginning of the table is updated. The main program must take this data and reset the character count before the RAM buffer becomes full. This is an alternate method to the software UART's example of putting the value in a register and setting a flag for the main program.

9.3.3.4 Error Conditions

If the program detects an error condition in the serial port Status Register, then the program sets a bit in RAM for the main program body to detect. When the main program detects this error bit, it looks at SSTAT to determine the cause of the error and takes action (if necessary). The main program may cause the byte to be retransmitted, if necessary.

9.3.3.5 Baud Rates

The baud rate generated by Timer 3 is user-programmable and is determined by the value of the 2-bit prescaler and the 8-bit timer reload register. The serial port discussion in Section 3 provides a table of common baud-rate values.

9.3.3.6 RS-232-C Interface

The RS-232-C interface consists of SN75188 line drivers and SN75189A line receivers as shown Figure 9-7. This is the same interface circuit used in the software example. Port A5 (input) of the TMS70x2 is used for all data receptions, and Port B3 (output) is used for all data transmissions.

9.3.3.7 Hardware UART Routines

```
                    IDT     'HWUART'
                *
                * This program uses the onboard UART to simulta-
                * neously transmit and receive characters.
                * Characters for transmitting are placed starting
                * at TTABLE with the first byte equal to the
                * string byte count. The received bytes are
                * stored in the table RTABLE with the beginning
                * byte equal to the characters received.
                *-----------------------------------------------
                * Peripheral Register Definition  TMS7042
                *-----------------------------------------------
0000            IOCNT0  EQU   P0   Interrupts and mode control
0004            PORTA   EQU   P4   Port A - UART input
0005            ADDR    EQU   P5   Port A direction
0006            PORTB   EQU   P6   Port B - UART output
0010            IOCNT1  EQU   P16  Interrupt 4,5 control
0011            SMODE   EQU   P17  Serial port mode
0011            SCTL0   EQU   P17  Serial port control 0
0011            SSTAT   EQU   P17  Serial port control status
0014            T3DATA  EQU   P20  Timer 3 data
0015            SCTL1   EQU   P21  Serial port control 1
0016            RXBUF   EQU   P22  Receiver buffer
0017            TXBUF   EQU   P23  Transmitter buffer
```

```
                    *--------------------------------------------------
                    * Register Definition
                    *--------------------------------------------------
0005                POINTR EQU   R5    Pointer into receiver table
0006                POINTT EQU   R6    Number of bytes ready to send
0007                POINTC EQU   R7    Transmitter chars send so far
0008                BITS   EQU   R8    Store random conditional bits
                    *                  here
001E                RTABLE EQU   030   Beginning of receiver table
                    *
0001                BIT0   EQU   1     Bit constants to make code more
0002                BIT1   EQU   2     readable
0004                BIT2   EQU   4
0008                BIT3   EQU   8
0010                BIT4   EQU   16
0020                BIT5   EQU   32
0040                BIT6   EQU   64
0080                BIT7   EQU   128
                    *--------------------------------------------------
                    *
F006                       AORG  >F006
F006 06             START  DINT                 Disable interrupts
                    *                          (precaution)
F007 A2 2A 00              MOVP  %>2A,IOCNT0    Single chip, clear INT
                    *                          flags
                    *                          Disable I1, I2, I3
F00A A2 03 10              MOVP  %>03,IOCNT1    Clear INT4 flag and
                    *                          enable INT4
F00D 52 60                 MOV   %>60,B
F00F 0D                    LDSP                 Initialize stack pointer
F010 A2 FB 05              MOVP  %#BIT2,ADDR    Set A2 = input others
                    *                          are output
F013 A2 08 06              MOVP  %BIT3,PORTB    Enable TX by setting
                    *                          B3 = 1
F016 A2 00 11              MOVP  %>00,P17       Make sure P17 points to
                    *                          SCTL0
F019 A2 40 11              MOVP  %BIT6,SCTL0    Reset the UART via the
                    *                          UR bit
F01C A2 7E 11              MOVP  %?01111110,SMODE
                    *                          One stop bit, communi-
                    *                          cations mode, even
                    *                          parity, 8 bits,
                    *                          Asynchronous mode,
                    *                          Motorola
F01F A2 15 11              MOVP  %>15,SCTL0     Clear the serial port
                    *                          reset bit
```

```
                    *       Clear all error flags and enable
                    *       the transmitter and receiver
F022 A2 00 15               MOVP  %>00,SCTL1  Make sure the start bit
                    *                         is off
F025 A2 C0 15               MOVP  %>C0,SCTL1  Use internal CLK, reset
                    *                         T3FLAG
                    *                         Disable T3 interrupt
                    *                         and set PS= 0
F028 A2 67 14               MOVP  %103,T3DATA Set timer at 1200 baud
                    *                         (5 MHz)
F02B 05                     EINT              Enable maskable
                    *                         interrupt
F02C D5 07          SETUP   CLR   POINTC      Clear bytes transmitted
                    *                         count
F02E D5 06                  CLR   POINTT      Clear bytes to transmit
F030 D5 05                  CLR   POINTR      Clear bytes received
                    *                         count
                    *
                    *-----------------------------------------------
                    *** Main body of program goes here
                    *
                    *** Main body finds and corrects serial port
                    *   error conditions by checking Bit 0 of
                    *   'BITS' and SSTAT.
                    *-----------------------------------------------
                    *   INTERRUPT 4 SERVICE ROUTINE
                    *-----------------------------------------------
F032 A6 38 11 02INTER4 BTJOP %>38,SSTAT,ERROR
                    *                         Was there an error?
F036 E0 03                  JMP   SAVEIT
F038 74 01 08       ERROR   OR    %BIT0,BITS  Set  an  error  bit  for
          the
                    *                         main program to find
                    *                         and continue
F03B B8             SAVEIT  PUSH  A           Save register A
F03C C8                     PUSH  B
F03D A7 02 11 12            BTJZP %BIT1,SSTAT,TXOUT
                    *       Did receiver cause interrupt?
                    *
F041 D3 05          RXCV    INC   POINTR      Get receiver table
                    *                         ponter
F043 7D 1E 05               CMP   %30,POINTR  Is receiver table full
                    *                         yet?
F046 E3 0B                  JHS   TXOUT       get out of routine
                    *                         if so
F048 32 05                  MOV   POINTR,B    Get index value
F04A 80 16          SKIP1   MOVP  RXBUF,A     Put received character
                    *                         in Register A
F04C AB 001E                STA   @RTABLE(B)  Put value into table
F04F 62                     MOV   B,A         Store the new character
                    *                         count
F050 8B 001E                STA   @RTABLE     Put count at location
                    *                         0 in table and exit
```

```
F053 A7 01 11 16TXOUT  BTJZP %BIT0,SSTAT,OUTI4
                       *      Did XMIT cause interrupt?
                       *
F057 4D 06 07    XMIT  CMP    POINTT,POINTC
                       *                        Is the table finished?
F05A E3 11             JHS    OUTI4             Jump if finished
F05C D3 07             INC    POINTC            Point to the next index
F05E 32 07             MOV    POINTC,B          Get transmit table
                       *                        pointer
F060 AA F070     SKIP0 LDA    @TTABLE(B)        Load value from TX
                       *                        table
F063 5D 00             CMP    %0,B              Is this the byte count?
F065 E6 04             JNE    OUTPUT            If not, output the byte
F067 D0 06             MOV    A,POINTT          If so, put into pointer
F069 E0 02             JMP    OUTI4
F06B 82 17       OUTPUT MOVP  A,TXBUF           Put data into
                       *                        transmitter
                       *
F06D C9          OUTI4 POP    B                 Restore registers
F06E B9                POP    A
F06F 0B                RETI                     Return to main program
                 *
F070 1A          TTABLE BYTE  26                Text can be either in
                 *                              ROM or RAM registers
F071 41 42 43 44       TEXT   'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
F075 45 46 47 48
F079 49 4A 4B 4C
F07D 4D 4E 4F 50
F081 51 52 53 54
F085 55 56 57 58
F089 59 5A
                 *
FFF6                   AORG   -(4+1)*2           Set up 4 vectors
                 *                               =interrupts
FFF6 F032 F006         DATA   INTER4,START,START,START,START
FFFA F006 F006
FFFE F006
                       END
NO ERRORS, NO WARNINGS
```

9.4 The Status Register

The Status Register contains four status bits that provide conditional execution for a variety of arithmetic and logical tasks. The carry (C), negative (N), zero (Z), and interrupt enable (I) flags occupy bits 7–4 of the Status Register. The C, N, and Z bits are affected by most instructions. The global interrupt enable (I) bit is affected by the EINT, DINT, and POP ST instructions.

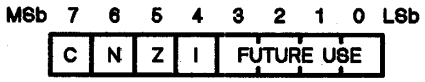


Figure 9-9. Status Register

Section 9.4.1 describes the way in which the compare instructions can be used to create the necessary status conditions for either a logical-type (unsigned) or arithmetic-type (signed) jump instruction. In Section 9.4.2 describes the effects of addition and subtraction on the Status Register for both signed and unsigned systems. Finally, Section 9.4.3 describes how SWAP and the rotation instructions (RR, RRC, RL, and RLC) can be used to clear, set, shift, or test the various status bits as required.

9.4.1 Compare and Jump Instructions

The compare instructions, CMP and CMPA, affect the C, N, and Z bits in the Status Register by subtracting a source operand (S) from a destination operand (d). Destination and source may be misnomers in this case, because the result of (d) - (s) is not stored; however, the status bits are set according to the result of the subtraction.

C Serves as a "no-borrow" bit. If (d) is greater than or equal to (s), then there is no borrow and C is set to 1. C is set to 0 if (d) is less than (s).

N Is set to the same value as the MSb of the result. For 2's complement (signed) systems, N = 1 indicates a negative number, and N = 0 indicates a positive number.

Z Is set to 1 if the source is equal to the destination [(d) = (s)].

The CMP instruction uses the contents of a register (Rn) as the destination operand, and either an immediate operand or the contents of another Rn as the source operand. The CMPA instruction uses the contents of Register A as the destination operand and one of the extended addressing modes (Direct, Register File Indirect, or Indexed) generates the source operand. Table 9-9 illustrates the limits of both signed and unsigned systems by listing the status bits affected for various source and destination operands substituted into the (d) - (s) expression.

Table 9-9. Compare Instruction Examples: Status Bit Values

SOURCE	DESTINATION	D-S	C	N	Z	INSTRUCTIONS THAT WILL JUMP					
FF	00	01	0	0	0	JL	JNC	JNE	JNZ	JP	JPZ
00	FF	FF	1	1	0	JHS	JC	JNE	JNZ	JN	
00	7F	7F	1	0	0	JHS	JC	JNE	JNZ	JP	JPZ
81	00	7F	0	0	0	JL	JNC	JNE	JNZ	JP	JPZ
00	81	81	1	1	0	JHS	JC	JNE	JNZ	JN	
80	00	80	0	1	0	JL	JNC	JNE	JNZ	JN	
00	80	80	1	1	0	JHS	JC	JNE	JNZ	JN	
7F	80	01	1	0	0	JHS	JC	JNE	JNZ	JP	JPZ
80	7F	FF	0	1	0	JL	JNC	JNE	JNZ	JN	
7F	7F	00	1	0	1	JHS	JC	JEQ	JZ	JPZ	
7F	00	81	0	1	0	JL	JNC	JNE	JNZ	JN	

Since the compare instructions do not alter the source and destination operands, these instructions can be executed before a conditional jump instruction to test for a particular relationship between the source and destination operands. Table 9-10 lists the necessary status bit conditions for each of the conditional jump instructions.

Table 9-10. Status Bit Values for Conditional Jump Instructions

MNEMONIC	INSTRUCTION	CONDITION ON WHICH JUMP IS TAKEN	STATUS BIT VALUES FOR JUMP:		
			C	N	Z
JC/JHS	Jump If Carry/Jump If Higher or Same	(d) unsigned >= (s)	1	X	X
JNC/JL	Jump If No Carry/Jump If Lower	(d)unsigned<(s)	0	X	X
JZ/JEQ	Jump If Zero/Jump If Equal	(d) = (s)	X	X	1
JNZ/JNE	Jump If Non-zero/Jump If Not Equal	(d) ≠ (s)	X	X	0
JP	Jump If Positive	(d) - (s) = pos #	X	0	0
JN	Jump If Negative	(d) - (s) = neg #	X	1	X
JPZ	Jump If Positive or Zero	(d) - (s) = pos # or 0	X	0	1

Note: X = Don't Care

9.4.2 Addition and Subtraction Instructions

The TMS7000 instruction set supports both single and multi-precision addition and subtraction for either binary or BCD, signed (2's complement) or unsigned data.

The following example illustrates 32-bit addition with the ADD and ADC instructions:

```
ADD   R30,R120
ADC   R29,R119
ADC   R28,R118
ADC   R27,R117
```

Since no initial carry-in is desired, the first instruction is ADD. The ADC instruction is then executed three times in succession to transfer the carry through all 32 bits.

The following example illustrates 24-bit subtraction with the SUB and SBB instructions:

```
SUB   R4,R127
SBB   R3,R126
SBB   R2,R125
```

Since no initial borrow-in is desired, the first instruction is SUB. The SBB instruction is then executed twice in succession to achieve the 24-bit result.

9.4.3 Swap and Rotation Instructions

Figure 9-10 illustrates the rotation operations performed by the four rotation instructions Rotate Right (RR), Rotate Right Through Carry (RRC), Rotate Left (RL), and Rotate Left Through Carry (RLC), and the SWAP instruction. SWAP executes the equivalent of four consecutive RL instructions, setting the C bit in the Status Register equal to bit 4 of the original operand or bit 0 (LSb) of the result.

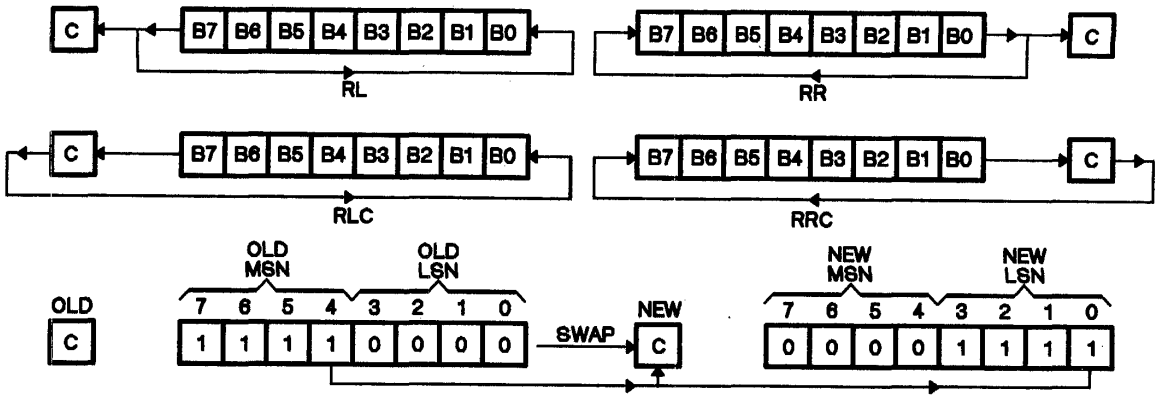


Figure 9-10. Swap and Rotation Operations

9.5 Stack Operations

The stack is located in RAM and can be tailored to your needs. One powerful application of the stack is the establishment of tables. For example, Figure 9-11 illustrates a dispatch table with an interpretive program counter (IPC). An IPC is used in some high level languages, such as Pascal, to give the proper execution sequence. The IPC can be contained in any register; it points to an interpretive pseudo code (pcode) byte that in turn specifies one of 256 dispatch routines. The overall effect of this function is that a program can execute one of a large number of different routines depending on a single value stored in a register. Two separate 256-byte sections are required for the high and low address bytes of each dispatch routine. The first entry of each section (ROV0) corresponds to pcode=0, and the second entry (ROV1) to PCODE=1, etc.

```
IPC     EQU   R3                Interpretive Program Counter
        LDA   *IPC              Get the input code, range=0-255
        DECD  IPC               Point to the next input code
        MOV   A,B               PCODE Index Register
        LDA   @DTABLE(B)        Lookup Address MSB
        PUSH  A                 Put MSB on stack
        LDA   @DTABLE+256(B)    Lookup Address LSB
        PUSH  A                 Put LSB on stack
        RETS                    Jump to the Address on the stack
DTABLE  BYTE  ROV0/256          Beginning of MSB table
        BYTE  ROV1/256
        .
        BYTE  ROV255/256
*                               LSB table starts here
        BYTE  ROV0              Warning messages may appear here,
        BYTE  ROV1              but they don't affect results
        .
        BYTE  ROV255-(ROV255/256*256)  No warning message here
```

Figure 9-11. A Dispatch Table with an Interpretive Program Counter (IPC)

Note that the assembler expressions have 16-bit values. For those instructions requiring an 8-bit operand, the expression is truncated to the least significant 8 bits. This may produce a warning message, but the value will be correct. Thus, the following instructions place byte values >AA, >55, and >55 at memory locations >8000, >8001, and >8002, respectively:

```
        AA55  LABEL  EQU   >AA55
8000                 AORG  >8000
8000    AA55         DATA  LABEL      16-bit word
              *                      LSB only
8002    55           BYTE  LABEL-(LABEL/256*256)
```

The most significant byte (MSB) of an expression can be obtained by dividing the value by 256 (2^8) as shown below:

```
        AA55  LABEL  EQU   >AA55
8000                 AORG  >8000
8000    AA55         DATA  LABEL
8002    AA           BYTE  LABEL/256    MSB only
```

9.6 Subroutine Instructions

Two instructions, CALL and TRAP, can invoke subroutines. TRAP is a one byte subroutine call. Both instructions save the current value of the Program Counter (PC) on the stack before transferring control to the subroutine. Since the return address is stored on the stack, subroutines can be easily nested. The two instructions differ only in the way in which the subroutine address is determined and in the amount of program memory required for execution.

The CALL instruction uses the Extended Addressing modes (Direct, Register File Indirect, and Indexed) to specify the subroutine address. This permits simple calls with a fully specified address as well as more complex calls with a calculated address. Of the two types of instructions, the CALL instruction requires more program memory than the TRAP instructions. For example:

```
CALL  @BITTEST
```

requires three bytes of memory - one byte for the opcode and two bytes for the subroutine address. If the subroutine is called six times, 18 bytes are necessary to implement the CALLs. The equivalent task for the TRAP instruction requires only 8 bytes for six successive uses of the same TRAP, since only the opcode byte is necessary after the first use. Six of these 8 bytes are the TRAP opcodes and the other two bytes are the trap vector. The first use of the TRAP instruction requires one opcode byte plus the two bytes of the subroutine address which are located in the Trap Table. Each subsequent use requires only one more byte, compared to three bytes for each CALL. All the trap vectors are stored at the end of memory with the most significant byte of the trap subroutine stored in the lower numbered location. The exact address where the trap vector (which is the trap subroutine address) is stored is derived from the following formula.

LSB of Address which contains the TRAP subroutine address = >FFFF - 2 x N where N is the TRAP number.

MSB of address = LSB - 1

The TRAP instructions (TRAPs 4–23) provide the most efficient means of invoking subroutines. Figure 9-12 shows a subroutine call generated by a TRAP instruction.

```
                .
                .                (Main Program)
          TRAP  4
                .                (More Main Program)
                .
          BR    MAINPR
                .
                .
BITTEST   EQU   $
                .
                .                (Subroutine Body)
          RETS
                .
                .
          AORG  >FFF6 Trap 4 vector
          DATA  BITTEST
                .
                .
```

Figure 9-12. Example of a Subroutine Call by Means of a TRAP Instruction

The Return from Subroutine (RETS) instruction should be executed to pop the PC from the stack and restore program control to the instruction immediately following the CALL or TRAP instruction.

9.7 Multiplication and Shifting

The MPY instruction performs an 8-bit by 8-bit multiply and stores the 16-bit result in Registers A and B. The most significant byte (MSB) of the result is in Register A, and the least significant byte (LSB) is in Register B. The MPY instruction can also be used to perform multi-bit right or left shifts by using an immediate operand as the multiplier. For example:

```
MPY   %8,B
```

The preceding example multiplies the value of Register B by 8. After the instruction executes, Register B contains the previous value left-shifted three bits ($2^3 = 8$) with no fill bits. Register A contains the previous value's most significant three bits which produces a value equivalent to shifting the previous value right five bits (8 - 3 = 5) with no fill bits. Using this method, it is possible to shift any 8-bit value left or right up to 8 bits. In many cases this is faster than the rotate instructions and almost always takes less program bytes. If the word only needs to be shifted one or two places then the rotate instructions may take less execution time. Table 9-11 lists the number of bits right- or left-shifted for a range of immediate multipliers.

Table 9-11. Multi-Bit Right or Left Shifts by Immediate Multiply

IMMEDIATE MULTIPLIER	BITS RIGHT SHIFTED	BITS LEFT SHIFTED
2	7	1
4	6	2
8	5	3
16	4	4
32	3	5
64	2	6
128	1	7

Multi-precision multiplications can be easily executed by breaking the multiplier and the multiplicand into scaled 8-bit quantities, as shown in the examples at the end of this section.

9.8 The Branch Instruction

The branch instruction (BR) unconditionally transfers program control to any desired location in the 64K byte memory space. BR supports direct, indexed, and indirect addressing:

- Direct addressing is used for simple GOTO programming.
- Indexed addressing allows table branches. This indexed branch technique is similar to the Pascal CASE statement. Program control is transferred to location CASE0 if the input is 0, to CASE1 if it is a 1, etc. This transferring method can implement up to 85 different cases. In the example below, indexed addressing is used to access a relative branch table:

```
JTABLE MOVP P4,A            Get data from A port
*                           (value < 85)
       ADD  A,B             Add twice to triple value
       ADD  A,B             Multiply it by 3
*                           (BR is 3 bytes long)
       BR   @CTABLE(B)      Branch according to the
*                           A port value * 2
        .
*
CTABLE BR   @CASE0          If P4 = 0 do this branch
       BR   @CASE1          If P4 = 1 do this branch
       BR   @CASE2          If P4 = 2 do this branch
*
```

- The branch instruction can also be used with indirect addressing in order to branch to a computed address. For example, suppose that a computed branch address has been constructed in R19 and R20. The desired program control transfer is made by:

```
BR   *R20
```

9.9 Interrupts

The number of interrupts and the hardware configuration for a TMS7000 family device are specified in Sections 2 and 3. The TMS7020, for example, has three interrupts in addition to RESET.

RESET and the interrupts are vectored through predetermined memory locations. RESET uses the TRAP 0 vector which is stored at memory locations >FFFE and >FFFF. The interrupts also use the TRAP vector table with $\overline{\text{INT1}}$ using the TRAP 1 vector, etc. Thus, the TRAP 2 instruction involves the same code as the interrupt INT2.

The interrupts differ from the TRAPs; they push the Status Register value on the stack, clear the interrupt enable bit in the Status Register, and reset the corresponding interrupt flag bit. Thus the EINT instruction must be used if nested interrupts are desired. The return from interrupt (RETI) instruction restores the Status Register and the Program Counter, re-enabling interrupts.

Many interrupt service routines alter the status of key registers such as Registers A and B. These routines should use the stack to restore the *machine state* to the desired value. For example, the following interrupt routine performs an I/O driven table look-up. Registers A and B are used, but their values are saved and restored:

```
INT  PUSH   A           Store Registers A and B on stack
     PUSH   B
     MOVP   P4,B        Get input from Port A
     LDA    @LOOKUP(B)  Do a table lookup to get new value
     MOVP   A,P6        Output new value on Port B
     POP    B           Restore Registers A and B in the
     POP    A           reverse order that they were put
*                       on
     RETI               Back to main program
```

All interrupts are usually disabled during an interrupt service routine. If it is necessary for an interrupt to occur while the processor is servicing another interrupt, then the global interrupt enable bit should be set to 1 by the interrupt service routine. The number of interrupts that can be serviced at any one time is determined by the size of the stack, which is also the internal RAM size (the stack resides in the Register File). Since other registers and data will most probably share the same space, the stack size is usually much less. When nesting interrupts, great care must be taken to avoid corrupting the data in the registers used by the most recent routine. If $\overline{\text{INT1}}$ interrupts an ongoing INT1 service routine, then the registers used by the INT1 routine are used in two different contexts. If provisions are not made for these situations, such as disabling all interrupts at critical times, then data errors will occur.

Sometimes a program contains distinct portions that require different responses to the same interrupt call. Since the interrupt vector is always set in nonchangeable ROM, another method must be used to change the vector for each part. One method for accomplishing this is to store a second vector in a RAM register pair and allow the first instruction in the interrupt routine execute an indirect branch on that register.

```
* Program to demonstrate multiple interrupt service
* routine locations.
* Main Program
*
       MOVD  %SERVIC,R127   Put INT1 service routine
       EINT                 address in register
       IDLE                 Turn on and wait for
*                           interrupts
       MOVD  %SERVI2,R127   Change INT1 routine to
*                           SERVI2
         .
         .
*
* First Interrupt 1 Service Routine
SERVIC PUSH  A              Beginning of the INT1
       PUSH  B              service routine for
*                           this part of the program
         .
         .
*
* Second Interrupt 1 Service Routine
SERVI2 PUSH  A              Start of another interrupt
       DEC   R4             1 service routine
         .
         .
*
INT1   BR    *R127          The entire INT1 service
*                           routine tranfers control
*                           to the address which is
*                           in R127 and R126
*
* Interrupt vector table at end of memory
       AORG  >FFFC
       DATA  INT1           Address of Interrupt 1
*                           service routine
       DATA  >F806          Reset vector start of
*                           program
```

9.10 Write-Only Registers

Certain TMS70xx peripheral registers are write-only registers, which means that the program cannot directly ascertain the contents of the register. Table 9-12 lists write-only registers.

Table 9-12. Write-Only Registers

REGISTER	LOCATION	FUNCTION	REGISTER	LOCATION	FUNCTION
IOCNT0	P0	Current mode	IOCNT1	P16	Interrupts
T1DATA	P2	Timer 1 latch	T1CTL	P3	Timer 1 control
T2DATA	P18	Timer 2 latch	T2CTL	P19	Timer 2 control
T3DATA	P20	Timer 3 latch	SCTL0	P17	Serial port
SMODE	P17	Serial port	TXBUF	P23	Transmit buffer

Problems may arise using some instructions with these write-only registers because most have a separate read-only function at the same address. An error may occur when you execute an instruction that reads the register, modifies the value and then writes back to the register. These instructions are ANDP, ORP, XORP. For instance, the program cannot turn on the timer by ORing a 1 to the timer Start bit, because the instruction will read the capture latch, set the MSb to 1, and then write this value to the timer control register. Unfortunately, this will change the prescaler and the timer may wait forever for a nonexistent external clock source.

The solution to this problem involves **image registers** which store the contents of a write-only register. An image register is a RAM register set aside to contain the value of a particular register. Whenever the write-only PF register must be changed, the program first fetches its image register, changes it, and then writes the image register to the peripheral register. This way, the image register always contains the value of the peripheral register. The following code using an image register could be used to turn on the timer start bit.

```
        OR    %>80,T1CTLI      Turn on start bit of
*                              timer control
        MOV   T1CTLI,A
        MOVP  A,T1CTL          Move the image register
*                              to the Peripheral File
```

9.11 Sample Routines

The following sections contain sample routines to show the various ways the TMS7000 handles common software tasks. Actual programs usually contain a combination of simple routines such as these along with custom routines tailored to the applications.

9.11.1 Clear RAM

This routine clears all the internal RAM registers. It can be used at the beginning of a program to initialize the RAM to a known value.

Register	Function
A	Holds the initialization value
B	Index into the RAM

```
        AORG   >F006
*
CLEAR   MOV    %126,B     Number of register to clear - 2
        CLR    A          Load the initialization value of
*                         zero
LOOP    STA    @2(B)      Clear the location indexed by
*                         B + 2
        DJNZ   B,LOOP     Loop until all RAM is cleared
```

9.11.2 RAM Self Test

This routine performs a simple alternating 0/1 test on the RAM. The RAM is tested by writing a >AA, >55 pattern to the entire RAM and then checking the RAM for this pattern. The inverted pattern is then written to RAM and re-checked. Finally, the entire RAM is cleared. If an error is found, a bit is set in a flag register.

Register	Before	After No Error	After Error
A	XX	0	?
B	XX	0	?
Rn	XX	0	?
FLAG	XX	0	Bit 0 = 1

Passing data: None
Registers affected: All
Ending data: All registers = 0
Bit 0 in FLAG = 1 if error was found

```
************************************************************
       MOV   %>55,A    Start RAM fill with >55
FILLR  MOV   %>FD,B    Set RAM start address - 2
*                      (don't change register A or B)
FILL1  STA   @2(B)     Fill RAM with AA 55 pattern
       RR    A         Change from 55 to AA to 55
       DJNZ  B,FILL1   Fill the entire RAM with this
*                      pattern
       RR    A         Change to beginning number
       MOV   %>FD,B    Refresh index
*
COMPAR CMPA  @2(B)     Check for errors
       JNE   ERROR     Exit if the values don't match
       RR    A         Change from 55 to AA to 55
       DJNZ  B,COMPAR  Check the entire RAM
*
       TSTA            Is Reg A now 55, AA or 00?
       JN    FILLR     =AA, change to opposite pattern
       JZ    EXIT      =00, finished now get out
FILL0  CLR   A         =55, clear the RAM now
       JMP   FILLR     Repeat the fill and check
*                      routine
ERROR  OR    %1,FLAG   Set bit 0 in the flag
*                      register
EXIT   EQU   $         Continue program here
```

9.11.3 ROM Checksum

This routine checks the integrity of the ROM by performing a checksum on the entire ROM. All ROM bytes from >F008 to >FFFF are added together in a 16-bit word. This sum is checked against the value at the beginning of the ROM (>F006,>F007). If the values don't match, then an error has occured and a bit is set in a register.

Register	Before	After No Error	After Error
A	XX	??	??
B	XX	??	??
R2	XX	CHKSUM MSB	CHKSUM MSB
R3	XX	CHKSUM LSB	CHKSUM LSB
R4	XX	>F0	>F0
R5	XX	>07	>07
R6	XX	>FF	>FF
R7	XX	>FF	>FF
FLAG	XX	Bit 1 = 0	Bit 1 = 1

```
*************************************************************
        AORG  >F006
        DATA  CHECKSUM Put correct checksum into ROM
*       ...            Other initialization program
*                      here
ROMCHK  MOVD  %>FFFF,R5Starting address (end of memory)
        MOVD  %>FF7,R7 Number of bytes to add + 1
        MOVD  %>0,R3   Reset summing register
*
ADDLOP  LDA   *R5      Get ROM byte
        ADD   A,R3     Add to 16-bit sum
        ADC   %0,R2
        DECD  R5       Point to next address
        DECD  R7       Decrement byte counter
        JC    ADDLOP   Continue until byte count goes
*                      past 0
        LDA   @>F007   Compare LSB stored to LSB sum
        CMP   A,R3
        JNE   ERROR    Set error bit if different
        LDA   @>F006   Compare MSB stored to MSB sum
        CMP   A,R2
        JEQ   EXIT     Set error bit if different
ERROR   OR    %2,FLAG  Set bit 1 in the Flag register
EXIT    EQU   $        Continue program here
*
```

9.11.4 Binary-to-BCD Conversion

This program converts a 16-bit binary word to a packed 6-nibble value.

Register	Before	After
A	XXXX	BCD MSB
B	XXXX	BCD
R2	XXXX	BCD LSB
R3	BINARY MSB	ZERO
R4	BINARY LSB	ZERO
R5	XXXX	ZERO

```
        AORG   >F006
*
BN2BCD  CLR    A         Prepare answer registers
        CLR    B
        CLR    R2
        MOV    %16,R5    Move loop count to register
LOOP    RLC    R4        Shift higher binary bit out
        RLC    R3        Carry contains higher bit
        DAC    R2,R2
        DAC    B,B       Double the number then add the
*                        binary bit
        DAC    A,A       Binary bit (a 1 in carry on 1st
*                        time is doubled 16 times).
        DJNZ   R4,LOOP   Do this 16 times, once for each
*                        bit
        RETS
```

9.11.5 BCD-to-Binary Conversion

Register	Before	After
A	XXXX	BCD MSB
B	XXXX	BCD
R2	XXXX	BCD LSB
R3	BINARY MSB	ZERO
R4	BINARY LSB	ZERO
R5	XXXX	ZERO

```
        AORG   >F006
*
BCD2BN  MOV    A,R2      Store word in R2
        AND    %>F0,A    Isolate MSB
        SWAP   A         Move to LSB position
        CMP    %10,A     Is it a valid BCD digit?
        JHS    ERROR     Goto error routine if not
        MPY    %10,A     Multiply MSB by 10, results
*                        at A,B in binary
        AND    %>0F,R2   Isolate LSB
        CMP    %10,A     Is it a valid BCD digit?
        JHS    ERROR
        ADD    R2,B      Add LSB to binary MSB to finish
*                        conversion
ERROR   RETS
        END
```

9.11.6 BCD String Addition

The following subroutine uses the addition instructions to add two multi-digit numbers together. Each of the numbers is a packed BCD string of less than 256 bytes (512 digits) stored at memory locations STR1 and STR2. This routine adds the two strings together and places the result in STR2. The strings must be stored with the most significant byte in the lowest numbered register. The TMS7000 family instruction set favors storing all numbers and addresses with the most significant byte in the lower numbered location.

Register	Before	After	Function
A	XXXX	????	Accumulator
B	XXXX	0	Length of string
R2	XXXX	????	Temporary save register
STR1	XXXX	no change	BCD string
STR2	XXXX	STR1+STR2	Target string, 6 bytes max

```
*      Decimal Addition Subroutine
*      Stack must have 3 available bytes.
*      On output: STR2 = STR1 + STR2
*
*
ADDBCD  CLRC                Clear carry bit
        PUSH    ST          Save status of stack
LOOP    LDA     @STR1-1(B)  Load current byte
        MOV     A,R2        Save it in R2
        LDA     @STR2-1(B)  Load next byte of STR2
        POP     ST          Restore carry from last add
        DAC     R2,A        Add decimal bytes
        PUSH    ST          Save the carry from this
*                           add
        STA     @STR2-1(B)  Store result
        DJNZ    B,LOOP      Loop until done
        POP     ST          Restore stack to starting
*                           position
        RETS                Back to calling routine
*
```

Notice the use of the Indexed Addressing mode to reference the bytes of the decimal strings. Notice also the need to push the status register between decimal additions, to save the decimal carry bit. Register B is used to keep count of the number of bytes that have been added.

9.11.7 Fast Parity

This routine presents a quick way to determine the parity of a byte. By exclusiving ORing all the bits of the byte together, a single bit will be derived which is the even parity of the word. When exclusive ORing, an even number of 1s will combine to form a 0, leaving either an odd 1 or 0 bit. This routine keeps splitting the byte in half and exclusive ORing the two halves.

Register	Before	After	Function
A	Target	????	Passing byte from program
B	XXXX	????	Length of string
Carry	XXXX	Parity	Status bit, result to calling routine

```
*************************************************************
* STEP 1                                                    SUBROUTINE
* Byte bits   7654 3210                                     TO FIND
*          XOR     7654 [MSN above]                        EVEN PARITY
*            ==========
*             xxxx ABCD
* STEP 2                  ------>  AB CD
*                                 XOR   AB [MS bits above]
*                                     =====
*                                     xx ab
* STEP 3                                  --->  a b
*                                           XOR  a   [MS bit]
*                                              ====
*                                              x P   {answer }
*
*************************************************************
*
PARITY MOV    B,A    Duplicate the target byte
       SWAP   A      Line up the MS nibble with the LS nibble
       XOR    B,A    Exclusive OR the nibbles to get a nibble
*                    answer
       MOV    A,B    Duplicate the nibble answer
       RR     A      Line up bits 0, 1 of the answer to bits
       RR     A      2, 3 of the answer
       XOR    B,A    XOR to get a new 2-bit answer
       MOV    A,B    Duplicate this 2-bit answer
       RR     A      Line up bit 0 with bit 1
       XOR    B,A    XOR to get final even parity answer
       RR     A      Rotate answer into the carry bit and bit 7
       RETS          Carry = 0 = even # of 1s
*                    Carry = 1 = odd # of 1s
*                    Use JC, JN or JNC JPZ in next executed
*                    instruction
```

9.11.8 Overflow and Underflow

An exclusive OR of the C and N bits ANDed with the exclusive OR of the MSbs of the operands can be used as a check for an overflow or underflow for **subtraction** in a signed system (if (C XOR N) AND (MSb1 XOR MSb2) = 1 then out of range).

When **adding** two signed numbers, the test for an out-of-range condition is similar to the subtraction method. When an exclusive OR of the C and N bits ANDed with the inverse of the exclusive OR of the MSbs of the two operands equals one then an overflow or underflow has occurred (if (C XOR N) AND (NOT(MSb1 XOR MSb2)) = 1 then out of range).

Register	Before	After	Function
A	XXXX	????	
OPRND1	XXXX	OPRND1	
OPRND2	XXXX	OPRD2-OPRD1	Subtraction results

```
* Routine to check for signed underflow or overflow
* If (C XOR N) AND (MSb1 XOR MSb2) = 1 then out of range
*
        MOV   OPRND1,A
        XOR   OPRND2,A       Get XOR of the MSbs
        SUB   OPRND1,OPRND2  Subtract 2 signed numbers
        JN    ISNEG
NOTNEG  JNC   NOERR          N = 0
        JMP   CXORN1         C XOR N = 1, First part of
*                            equation is true
ISNEG   JC    NOERR          N=1
CXORN1  TSTA                 C XOR N = 1; set flags for
*                            MSb1 XOR MSb2
        JPZ   NOERR          If (N XOR C) AND (MSb1 XOR
*                            MSB2) = 1 then out of range.
*                            For addition change this
*                            instruction to JN NOERR
OUTRNG  ...                  Out of range; underflow or
*                            overflow
*
NOERR   ...                  No underflow or overflow
```

9.11.9 Bubble Sort

This routine will sort up to 256 bytes using the bubble sort method. Longer tables could be sorted using the Indirect Addressing mode.

Register	Function
A	Temporary storage register
B	Index into the table
R2	Holds flag to indicate a byte swap has been made

```
        AORG   >F006
*
FLAG    EQU    R2              'Swap has been made' flag
*
SORT    CLR    FLAG            Reset swap flag
        MOV    %149,B          150 bytes to be sorted
LOOP1   LDA    @TABLE(B)       Look at entry in table
        CMPA   @TABLE-1(B)     Look at next lower byte
        JL     LOOP2           If lower skip to next value
        INC    FLAG            Entry is not lower, set swap
*                              flag
        PUSH   A               Store upper byte
        LDA    @TABLE-1(B)     Take lower byte
        STA    @TABLE(B)       Put where upper was
        POP    A               Get the old upper byte
        STA    @TABLE-1(B)     Put where the lower byte was
LOOP2   DJNZ   B,LOOP1         Loop until all the table is
*                              looked at
        BTJO   %>FF,FLAG,SORT
* If swap was made, then resweep table
* If no swap was made, then table is done
```

9.11.10 Table Search

Table searches are efficiently performed by using the CMPA (Compare Register A Extended) instruction. In the following example, a 150 byte table is searched for a match with a 6-byte string:

Register	Before	After	Function
A	XXXX	????	
B	XXXX	????	
R2	XXXX	????	Table length
TABLE	XXXX	no change	Long string in table
STRING	XXXX	no change	Target string, 6 bytes max

```
*
SEARCH  MOV   %150+1,R2      Table length = 150 bytes
LOOP1   MOV   %6,B           String length = 6 bytes
LOOP2   XCHB  R2             Swap pointers, long string
*                            in B
        DEC   B              Table end? If so, no match
*                            found
        JZ    NOFIND
        LDA   @TABLE-1(B)    Load test character
        XCHB  R2             Swap pointers, string
*                            pointer in B
        CMPA  @STRING-1(B)   Match?
        JNE   LOOP1          If not, reset string
*                            pointer else test
        DJNZ  B,LOOP2        next character
MATCH   EQU   $              Match found
*
NOFIND  EQU   $              No match found
```

The Indexed Addressing mode is used in this example and has the capability to search a 256-byte string, if needed. Register B alternates between a pointer into the 6-byte test string and a pointer into the longer table string.

9.11.11 16-Bit Address Stack Operations

This routine performs 16-bit stack operations using the 1-byte TRAP instruction for pushing and popping. It uses macros to make code more readable. All values pass through Register A.

Register	Function
A	Passing register for routines
R2	Indirect pointer MSB
R3	Indirect pointer LSB

```
*  Define Macro PUSH16 as a trap instruction
*
PUSH16  $MACRO
        TRAP    6
        $END
*
*  Define Trap 7 to be the POP16 operation
*
POP16   $MACRO
        TRAP    7
        $END
*
TRAP6   INC     R3          PUSH16
        ADC     %0,R2       Increment the indirect pointer
        STA     *R3         Push Register A
        RETS
*
TRAP7   LDA     *R3         POP16   Pop into Register A
        DECD    R3          decrement the indirect pointer
        RETS
*
*
        AORG    >FFF0       Set up Trap and Interrupt
*                           vectors
        DATA TRAP7,TRAP6,INT5,INT4,INT3,INT2,INT1,RESET
        END
*
* Examples of use
*
        MOVD    %>1234,R3Initialize the 16-bit stack
*                           pointer
        MOV     %DATA,A     Load Register A
        PUSH16              Use the macro to push A onto
*                           the stack
        POP16               Return a value from the stack.
        MOV     A,TEMP      Move the value to a temporary
*                           register
```

9.11.12 16-by-16 (32-Bit) Multiplication

This example multiplies the 16-bit value in register pair R2,R3 by the value in register pair R4,R5. The results are stored in R6, R7, R8, R9, and Registers A and B are altered.

```
*
*
*         16-BIT MPY:                XH     XL        X VALUE
*                              X     YH     YL        Y VALUE
*                              ----------------
*                                   XLYLm  XLYL1     1 = LSB
*                             XHYLm  XHYL1           m = MSB
*                             XLYHm  XLYH1
*                   +  XHYHm  XHYH1
*                   -------------------------
*                      RSLT3  RSLT2  RSLT1  RSLT0
*
XH          EQU     R2          Higher operand of X
XL          EQU     R3          Lower  operand of X
YH          EQU     R4          Higher operand of Y
YL          EQU     R5          Lower  operand of Y
RSLT3       EQU     R6          Msb of the final result
RSLT2       EQU     R7
RSLT1       EQU     R8
RSLT0       EQU     R9          LSB of the final result
*
MPY32       CLR     RSLT2       Clear the present value
            CLR     RSLT3
            MPY     XL,YL       Multiply LSBs
            MOV     B,RSLT0     Store LSB in result register 0
            MOV     A,RSLT1     Store MSB in result register 1
            MPY     XH,YL       Get XHYL
            ADD     R1,RSLT1    Add to existing result  XLYL
            ADC     R0,RSLT2    Add carry if present
            MPY     XL,YH       Multiply to get  XLYH
            ADD     R1,RSLT1    Add to existing result XLYL+XHYL
            ADC     R0,RSLT2    Add to existing results and carry
            ADC     %0,RSLT3    Add if carry present
            MPY     XH,YH       Multiply MSBs
            ADD     R1,RSLT2    Add once again to the result reg
            ADC     R0,RSLT3    Do the final add to the result reg
```

9.11.13 Binary Division, Example 1

This program divides a 16-bit dividend by an 8-bit divisor giving a 8-bit quotient and an 8-bit remainder. All numbers are unsigned positive numbers. The dividend's MSB must be less than the divisor to ensure an 8-bit quotient.

Dividend: 0-7FFF
Divisor: 1-255
Quotient: 0-255

Register	Before	After
A	DIVIDEND MSB	REMAINDER
B	DIVIDEND LSB	QUOTIENT
R2	DIVISOR	DIVISOR
R3	XXXX	ZERO

```
        AORG  >F006
*
BINDVD  MOV   %8,R3     Set loop counter to 8
DVDLP   RLC   B         Multiply dividend by 2
        RLC   A
        JNC   SKIP1     * These * steps are not needed
        SUB   R2,A      * if the dividend is limited
        SETC            * to 15 bits
        JMP   DIVEND    *
SKIP1   CMP   R2,A      Is MSB of dividend > divisor
        JNC   DIVEND
SUBIT   SUB   R2,A      If so dividend=dividend
*                       - divisor
*                       C=1 gets folded into next
*                       rotate
DIVEND  DJNZ  R3,DVDLP  Next bit, is the divide done.
        RLC   B         Finish the last rotate
```

9.11.14 Binary Division, Example 2

This program divides a 16-bit dividend by an 8-bit divisor, producing a 16-bit quotient and an 8-bit remainder. All numbers are unsigned positive numbers. The dividend's MSB can be larger than divisor.

Dividend: 0-FFFF
Divisor: 0-255
Quotient: 0-255

$$8\overline{)16}^{\text{16 r8}}$$

Register	Before	After
A	XXXX	REMAINDER
B	DIVISOR	DIVISOR
R2	DIVIDEND MSB	QUOTIENT MSB
R3	DIVIDEND LSB	QUOTIENT LSB
R4	XXXX	ZERO

```
        AORG   >F006
*
BINDVD  MOV    %16,R4      Set loop counter to 16 (8+8)
        CLR    A           Initialize result register
DVDLP   RLC    R3          Multiply dividend by 2
        RLC    R2
        RLC    A
        JNC    SKIP1       * These * steps are not needed
        SUB    B,A         * if the dividend is limited
        SETC               * to 15 bits
        JMP    DIVEND      *
SKIP1   CMP    B,A         Is MSB of dividend > divisor
        JNC    DIVEND
        SUB    B,A         If so dividend=dividend
*                          - divisor
*                          C=1 gets folded into next
*                          rotate
DIVEND  DJNZ   R4,DVDLP    Next bit, is the divide done?
        RLC    R3          Finish the last rotate
```

9.11.15 Binary Division, Example 3

This program divides a 16-bit dividend by an 16-bit divisor, producing a 16-bit quotient and a 16-bit remainder. All numbers are unsigned positive numbers. The dividend's MSB can be larger than divisor.

Dividend: 0-FFFF
Divisor: 0-FFFF
Quotient: 0-FFFF

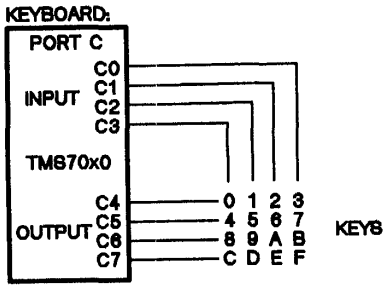
$$\begin{array}{r} 16\ r16 \\ \hline 16\overline{)16} \end{array}$$

Register	Before	After
A	XXXX	REMAINDER MSB
B	XXXX	REMAINDER LSB
R2	DIVIDEND MSB	QUOTIENT MSB
R3	DIVIDEND LSB	QUOTIENT LSB
R4	DIVISOR MSB	DIVISOR MSB
R5	DIVISOR LSB	DIVISOR LSB
R6	XXXX	ZERO

```
        AORG  >F006
*
BINDVD  MOV   %24,R6   Set loop counter to 24
*                      (16 + 8)
        CLR   A        Initialize result register
        CLR   B
DIVLOP  RLC   R3       Multiply dividend by 2
        RLC   R2
        RLC   B
        RLC   A
        JNC   SKIP1    Check for possible error
        SUB   R5,B     condition that results
        SBB   R4,A     when a 1 is shifed past
*                      the most significant bit
        SETC           Correct by subtracting out
*                      the divisor
        JMP   DIVEND
*
SKIP1   CMP   R4,A     Is MSB+LSB of dividend >
*                      divisor
        JNC   DIVEND
        JNE   MSBNE    Are MSBs equal?
        CMP   R5,R3    If so, compare LSBs
        JNC   DIVEND
*
MSBNE   SUB   R5,B     If borrow, dividend=divi-
*                      dend - divisor
        SBB   R4,A     C=1 get folded into next
*                      rotate
*                      Next bit, is the divide
*                      done?
DIVEND  DJNZ  R6,DIVLOP
        RLC   R3       Finish the last rotate
        RLC   R2
```

9.11.16 Keyboard Scan

This routine reads a 16-key keyboard, returns the hex digit of the key, and debounces the key to avoid noise. A 'valid key' flag is set when a new key is found.



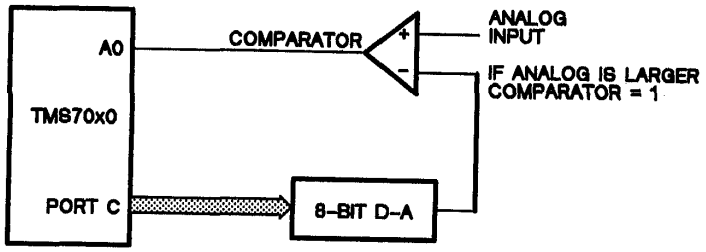
Register	Before	After No Key	After New Key	Function
A	XXXX	0	COLUMN	Temporary
B	XXXX	0	ROW	Temporary
R2	XXXX	16	KEY #	Temp store for Key value
R3	OLD KEY	>FF	KEY #	Holds Key pressed now
R4	DEBOUNCE	0	0	Debounce counter, old key or new
R5	GENERAL BITS	?xxxxxxx0	?xxxxxxx1	One bit of register is 1 if new key

```
        AORG >F006
*
CDDR    EQU     P9
PORTC   EQU     P8
*
GETKEY  MOV     %8,B            Initialize row pointer
        CLR     R2
        MOVP    %>F0,CDDR       Set Data direction register 4 output,
*                               4 input
LOOP    RLC     B               Select next row
        JC      NOKEY           Last row ? if so no key was found
        ADD     %4,R2           Add number of keys/row to key
*                               accumulator
        MOVP    B,PORTC         Activate row
        MOVP    PORTC,A         Read columns
        MOVP    %0,PORTC        Clear row
        AND     %>F,A           Isolate column data
        JZ      LOOP            If no keys found then check next row
KEYLSB  DEC     R2              Decrement column offset
        RRC     A               Find  column
        JNC     KEYLSB          If not column then, try again
*
NEWKEY  CMP     R2,R3           Is the new key the same as the old key
        JEQ     DEBONS          If it is then debounce it
        MOV     R2,R3           Brand new key, Move it to current key
*                               value
        MOV     %16,R4          Set up debounce count
```

```
DEBONS  CMP     %2,R4           Is the debounce count 1 or 0 ?
        JL      GOODKY
        DJNZ    R4,GETKEY       If greater than 1 then debounce is
*                               not finished, go read key again
*
GOODKY  JZ      NOTNEW          If debounce count =0 then key was here
*                                last time
        DEC     R4              If it was one this is a new valid key,
*                               make old key
        OR      %1,R5           Set new key flag in BIT register, the
NOTNEW  RETS                    calling routine uses this flag
*
NOKEY   MOV     %>FF,R3         No key was found, set key value to
*                               unique value
        RETS
```

9.11.17 8-Bit Analog-to-Digital Converter

This routine converts an analog signal to a digital value using a digital-to-analog converter and a comparator.

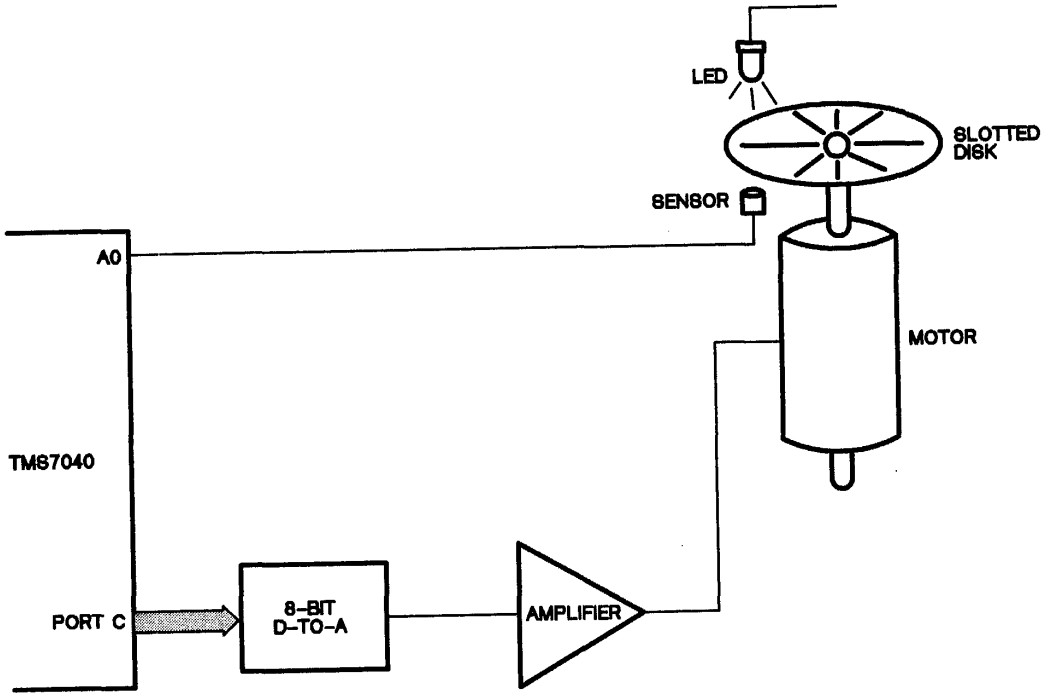


Register	Before	After	Function
A	XXXX	ANALOG VALUE	Final digital value
B	XXXX	ZERO	Trial and error test value

```
ATOD    MOV     %>80,B          Starting value for binary search
        CLR     A               Initialize  value
        MOVP    %>FF,P9         Port C is all outputs
*
LOOP    OR      B,A             Set the next bit in Test value
        MOVP    A,P8            Send it to the D-A converter
        BTJOP   %1,P4,ABIGER    Is this value Less than the analog
*                               value?
ASMALL  XOR     B,A             If Analog value is smaller, decrease
*                               test value
ABIGER  RRC     B               If Bigger go to next bit in test value
        JNC     LOOP            If not at the end, then go test the
*                               next bit
*
FINISH  RETS
```

9.11.18 Motor Speed Controller

This routine keeps the speed of a motor constant. A pulse proportional to the speed of the motor comes from a sensor next to a slotted disk on the motor. The motor is controlled by a variable voltage generated by a D-A converter. Some mechanical considerations are necessary for an actual system.



Register	Before	After	Function
A	DATA	NO CHANGE	Temporary register
PULSE1	PULSE MSB	0	Holds MSB of pulse length value
SPEED	SPEED	NEW SPEED	Holds current Voltage value for D-A
STEP	STEP SIZE	NEW SIZE	How much the voltage is changed per cycle
SPEED1	SPEED MSB	NO CHANGE	The desired time between the slots as measured by the
SPEED2	SPEED LSB	NO CHANGE	timer (1=MSB, 2=LSB)

```
        AORG    >F006
PULSE1  EQU     R4      MSB of 16-bit pulse length counter
SPEED   EQU     R5      Current voltage output to motor
STEP    EQU     R6      Change output voltage by this amount
SPEED1  EQU     R7      MSB of 16-bit speed reference
SPEED2  EQU     R8      LSB of 16-bit speed reference
BITS    EQU     R9      General purpose register for bits
INCR1   EQU     2       Step size for coarse adjustment
INCR2   EQU     4       Step size for fine adjustment
```

```
MCNTL   MOVP    %>FF,P2      Initialize the timer value
        MOVP    %>80+32,P3   Initialize the prescaler and start
*                            timer
        MOVP    %>3E,P0      Clear interrupts, enable I2, I3
        EINT                 The interrupts are now enabled
*
*   Main program body here
*
INT2    BTJZP   %>20,P0,OK   Interrupt 2 routine, check for pending
*                            INT 3
        BTJOP   %>80,P3,OK   Check Capture Latch value for recent
*                            change
        JMP     INT3         If P3 is pending and CL just under-
*                            flowed then INT3 came first,
*                            go directly to INT3
OK      INC     PULSE1       Increment the MSB counter for the
*                            pulse length
        JNC     NOERR        If overflow there was an error
*                            (Motor too slow)
ERROR1  OR      %01,BITS     Set an error bit for the main
*                            routine to find
NOERR   RETI
*
INT3    MOVP    %>80+32,P3   Restart the timer at beginning
        PUSH    A            Save register
        MOV     %INCR1,STEP  Coarse adjustment step size for
*                            voltage change
        CMP     SPEED1,PULSE1
*                            Compare desired speed to measured
*                            speed (MSB)
        JEQ     TESTLS       If the same then compare LSBs
TESTSP  JL      GOSLOW       Does motor need to go faster or slower
GOFAST  ADD     STEP,SPEED   If faster, increase voltage to motor
OUTPUT  MOV     SPEED,A      Move new voltage value to D-A
        MOVP    A,P8
SAME    POP     A            Restore register
        CLR     PULSE1       Clear MSB of pulse length
        RETI
*
GOSLOW  SUB     STEP,SPEED   Decrease the motor voltage
        JMP     OUTPUT       Output voltage value
*
TESTLS  MOVP    P3,A         Get LSB of pulse length from capture
*                            latch
        INV     A            Since it counts from FF to 00, invert
*                            value
        CMP     SPEED2,A     Compare desired speed to measured
*                            speed (LSB)
        JEQ     SAME         If the same do nothing
        MOV     %INCR2,STEP  Fine adjustment step size for voltage
*                            change
        JMP     TESTSP       Set new speed according to LSB values
```

10. Development Support

Texas Instruments provides extensive development support for the TMS7000 family. TMS7000 software support is referred to as CrossWare, and includes a macro assembler and a link editor. Appendix G contains instructions for installing the TMS7000 CrossWare.

- The TMS7000 Assembler translates TMS7000 assembly language instructions and directives into object code. Sections 5 and 6 discuss the assembler and the TMS7000 assembly language instructions.
- When several components of a source program are assembled individually, the TMS7000 Link Editor links together the object code produced by these program modules to form one complete executable program. Section 7 discusses the link editor.

TMS7000 in-circuit development tools include:

- The XDS (Extended Development System) emulator, which provides realtime in-circuit emulation of the TMS7000 devices in all modes.
- The TMS7000 Evaluation Module (EVM), a single-board development system that emulates the TMS7000 devices in Single-Chip mode.
- Several prototyping units, including the TMS7742, SE70P162, SE70CP160, SE70CP162, and TMS77C82[00].

These tools allow a designer to evaluate the TMS7000's performance, benchmark time-critical code, and determine the feasibility of using a TMS7000 in a specific application. The TMS7000 CrossWare translates programs into modules that can be executed on the XDS emulator or EVM. This section discusses key features of the hardware development tools; extensive XDS and EVM documentation is available (the preface contains literature numbers).

Section **Page**

---

4 Advance Information

10.1 The XDS Emulator

The TMS7000 XDS/22[5] (Extended Development Support) emulator is a self-contained system that provides full-speed in-circuit emulation. Key features include:

- Host-independent development system
- Supports the TMS70x0, TMS70Cx0, TMS70x2, and TMS70Cx2 devices in Single-Chip and Expansion modes
- Realtime hardware breakpoint/trace/time capabilities
- Execution of programs from target memory
- Three EIA ports allow communication with peripherals
- Several possible system configurations, including standalone, host-computer, and multiprocessor configurations

The host-independent configuration shown in Figure 10-1, combined with a complete set of development and debugging tools, allows you to select the TMS7000 processor best suited to your application. Since the same set of tools emulates each processor, you only need to learn the basic development format once.

XDS cross-assemblers and host interfaces are available for the following systems:

- IBM PC, TI PC running MS/PC-DOS
- DEC VAX 11 running VMS
- IBM 370, 3033, 43xx running MVS or CMS
- TI DX10

XDS hardware includes a chassis, power supply, and a three-board set consisting of an emulator, communications board, and a breakpoint/trace/time board.

---

5 XDS is a registered trademark for Texas Instruments Incorporated. All rights are reserved.

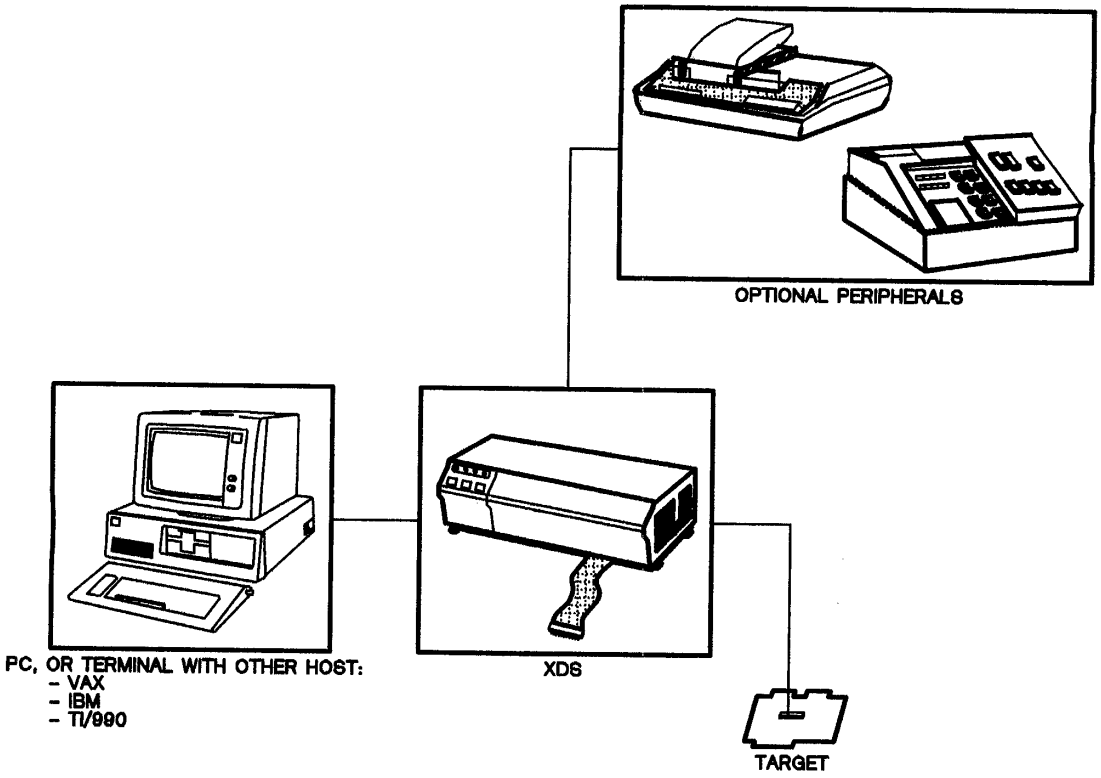


Figure 10-1. Typical XDS Configuration

10.1.1 Software Development

Software written and developed on a host computer can be downloaded to the XDS/22 emulator memory space via a standard RS-232 EIA link. The XDS monitor is located in the firmware onboard the emulator. A powerful set of commands provide complete control of the emulator functions and the target system, enhancing development and testing of target hardware and software. The XDS monitor commands include an assembler that permits almost any system to be used as an intelligent terminal and prepare the source text for assembly by the XDS emulator. Table 10-1 lists the TMS7000 XDS/22 commands.

Table 10-1. TMS7000 XDS/22 Commands

REGISTER COMMANDS		TRACE COMMANDS	
A	Display or set Register 0	DT	Display trace
B	Display or set Register 1	FT	Find trace sample
C	Display or set carry bit	IT	Inspect trace
DR	Display registers	SOR†	Set opcode range
I	Display or set STINT bit	TR†	Set trace qualifiers
MR	Modify registers	TRIX†	Trace on extended IAQ
N	Display or set negative bit	TRM†	Trace memory select
PC	Display or set Program Counter	UPLOAD/DOWNLOAD COMMANDS	
Pnn	Display or set register nn	DL	Download to emulator
ROM	Display or set ROM pointer	IHC	Initialize host control chars
SP	Display or set Stack Pointer	IPORT	Initialize EIA ports
ST	Display or set status bit	UL	Upload to host
RUN COMMANDS		STATUS COMMANDS	
CRUN	Continue run	DHS	Display halt status
GHALT	Group halt (MP mode)	DPS	Display processor status
GRUN	Group run	DTS	Display trace status
RTR	Reset target and run	ID	Display foreground EMU banner
RUN	Start program execution	INIT	Initialize emulator
SRR	Software reset and run	IPC	Initialize peripheral control
SS	Single-step execution	RESTART	Restart emulator
STOP	Stop execution (ARM mode)	/n	Display status of emulator n
THALT	Total halt (MP mode)	#n	Select emulator from chain
TRUN	Total run (MP mode)		
BTT COMMANDS		INTERNAL COMMANDS	
BTT	Set B/T/T conditions	$CLK	Set clock divider
DBTT	Display B/T/T parameters	$INT	Modify interrupt
DTIME	Display time	$UART	Modify UART type
IBTT	Initialize B/T/T		
XTIME	Analyze timing		

† These commands are only valid when the B/T/T board is installed.

Table 10-1. TMS7000 XDS/22 Commands (Concluded)

BREAKPOINT COMMANDS		MEMORY COMMANDS	
BP†	Set hardware breakpoint conditions	BLK	Remap memory block
BPM†	Set BP cond. on memory access	DM	Display program memory
CASB	Clear all software breakpoints	EXP	Remap expansion memory
CSB	Clear a software breakpoint	FILL	Fill memory with data
DSB	Display all software breakpoints	FIND	Find data in memory
SIB	Set internal breakpoint	IM	Display or set memory
SSB	Set a software breakpoint	MM	Modify program memory
MISCELLANEOUS COMMANDS		**MODE COMMANDS**	
COPY	Copy memory	ARM	Initialize alternate run mode
DV	Display value	BGND	Initialize background mode
HELP	Display command menu	DIAG	Initialize diagnostic mode
ICC	Initialize cursor control	DISARM	Disable alternate run mode
LOAD	Load command defaults from memory	HOST	Initialize host mode
LOG	Turn logging device on or off	IMD	Initialize MP mode
MESG	Send message (diag. mode)	IMP	Initialize MP mode
RCC	Reset cursor controls	QDIAG	Quit diagnostic mode
SAVE	Save command defaults into memory		
SNAP	Set up snapshot display		
XA	Execute assembler		
XRA	Execute reverse assembler		

† These commands are only valid when the B/T/T board is installed.

The XDS timing capabilities allow you to store trace samples that contain realtime timing stamps. Trace samples, like breakpoints, may be selectively chosen on desired memory and I/O cycles, allowing such software measurements as:

- Program/memory activity
- Module execution duration
- Intermodule execution duration
- Module usage

Using the hardware and software breakpoint commands and the trace function, a complete record of events can be examined. You can select a range of memory addresses and I/O addresses to set valid breakpoints. The breakpoint/trace/time (B/T/T) board allows you to set breakpoints on any memory cycle - memory read, memory write, or instruction acquisition. For I/O operations, the B/T/T board can breakpoint on any I/O read or I/O write, if the I/O address qualifications are met. A 2047-sample trace buffer provides a history of execution before or after the breakpoint. Trace samples are stored in the trace memory and can be read back after execution has been halted. Memory and I/O cycles can also be traced.

This cycle of using the host computer and the XDS/22 for testing provides a quick, efficient method for target system development. After debugging is complete, EPROMs can be programmed using the host computer's PROM programming capabilities.

10.1.2 XDS Memory Map

The XDS memory map for the TMS7000 family is extremely flexible. The emulator contains 64K bytes of RAM to support the entire address space of the TMS7000 devices. This 64K-byte memory space can be used to emulate on-chip ROM and external memory in the target application. Memory is allocated in 256-byte blocks, X blocks as on-chip ROM and Y blocks as off-chip memory, where 256(X+Y) → 64K bytes. Memory can be arranged in any practical configuration desired, allowing system-level debug rather than just software or hardware debug.

10.1.3 Communication Capabilities

The XDS unit can communicate with a host computer, terminal, PROM programmer, or printer through four EIA RS-232-C links. Communication functions include:

- Downloading of data files from an external devices (external host, PROM programmer, or terminal) to emulator memory.
- Downloading of data to a PROM programmer or logging device.
- Terminal-to-host communication via passthrough mode.
- Transmission of data from emulator memory to a PROM programmer or logging device.
- Uploading of data files from the emulator to an external device (external host, PROM programmer, or terminal).

10.1.4 System Configurations

The TMS7000 XDS/22 can operate in one of four modes:

- **Standalone mode** is the minimum configuration, requiring only the XDS and your terminal.
- The XDS is best suited for use with a host computer and terminal in **Host-Computer mode**. This allows you to write programs using a familiar editor and then download them to the XDS. When debugging is complete, you can upload the code and store it on the host system.
- **PC-Based mode** is a variation of the host-computer mode – the host system is a single-user system such as a PC. The XDS supports host uploads/downloads over a single port, allowing a PC to function as both a terminal and a host. This configuration requires a terminal-emulation software package such Crosstalk XVI by Microstuf.
- An increasing number of designs use multiple microprocessor systems. In **Multiprocessor mode**, the XDS supports debugging of up to nine XDS stations linked together in a daisy-chained fashion. The XDS system is connected to the host computer via the RS-232 port of the last

XDS workstation. A single CRT interface can control all of the workstations. Each workstation may be used individually or the workstations can be grouped or subgrouped to synchronize control over the entire target system.

10.1.5 Breakpoint, Trace, and Timing Functions

The breakpoint/trace/time (B/T/T) board allows you to set a hardware interrupt or breakpoint that halts emulator execution. Breakpoints can be set on I/O and/or memory operations with three simple monitor commands. You can select a range of memory addresses and I/O addresses for valid breakpoints, or select two separate memory addresses or two separate I/O addresses. The B/T/T board can breakpoint on any memory cycle – read, write, or instruction acquisition. For I/O operations, the B/T/T board can breakpoint on any I/O read or write if the I/O address qualifications are met.

The trace function provides a history of execution prior to the breakpoint. It is used to analyze a set of signals based on addresses and commands. Trace samples are stored in trace memory and can be read back after execution has been halted. Both memory and I/O cycles can be traced, including memory read, memory write, and instruction acquisitions or all memory cycles, and I/O read, I/O write, or any I/O cycle.

The trace memory can hold 2047 words by 48 bits of trace samples. You are given the option of how many of these 2047 samples to take, or to keep wrapping around in trace memory, writing over the oldest trace sample with the newest trace sample.

The B/T/T board also contains a cable which allows easy interfacing to logic analyzers. This interface provides may useful system signals not available through a target connector.

10.1.6 Physical Specifications

The XDS/22 emulator is a table-top sized unit, suitable for most work surfaces. The XDS/22 has an air inlet on each side of the unit and an air exhaust port on the rear of the unit. A minimum of five inches clearance must be maintained between the XDS and neighboring equipment on the sides *and* rear for proper air flow. Listed below are the dimension and clearance requirements.

DIMENSIONS
Width = 17.0 Inches (43.2 CM) Depth = 16.5 Inches (41.9 CM) Height = 7.4 Inches (18.8 CM) Target Cable = 18.0 Inches (46.0 CM)
CLEARANCE REQUIREMENTS
Sides : 5 Inches Minimum (15.2 CM) Back : 5 Inches Minimum (15.2 CM) Top : None Required Front : None Required

10.2 Evaluation Modules

The TMS7000 Evaluation Module (EVM) provides hands-on hardware evaluation of TMS7000 devices. This single-board unit can function as limited feature, standalone development system. Key features include:

- Realtime in-circuit emulation
- Text editor
- Assembler
- Debug monitor
- Onboard EPROM programming utility
- Upload/download capabilities
- Single-step execution capabilities
- Audio-cassette interface

The RTC/EVM7000 emulates the TMS7000 Single-Chip mode; TMS7000 expansion modes are not supported. There are two versions of the evaluation module for the TMS7000 family:

1) RTC/EVM7000N-1 for NMOS devices
2) RTC/EVM7000C-1 for CMOS devices

The EVM is equipped with eight 8K-byte sockets for the entire 64K-byte address space of the TMS7000. 16K bytes of the EPROM are devoted to the resident firmware. User RAM can be expanded in 8K-byte increments, from 16K bytes to 32K bytes. During assembly and debug operations, the EVM RAM can be configured to emulate all TMS7000 family members; for the emulation of the 2K-ROM and 4K-ROM versions, it allows assembly of text files directly from RAM. A wire-wrapped development area, with all required signals provided and labeled, is available for additional logic.

The EVM crystal frequency can be modified to fit the needs of the target system.

10.2.1 System Configurations

Several system configurations are possible:

- **Standalone Mode** - is the minimum configuration. The onboard text editor is used for creating TMS7000 assembly language text files. The audio cassette tape interface, which has limited directory and file search capability, is used for mass storage.

- **Host-Computer Mode** - provides a more productive environment. The host is used to develop and save the text files. The files may then be assembled using the TMS7000 CrossWare, or they can be downloaded to the EVM for assembly by the onboard assembler. The EVM has two EIA RS-232 ports to support this and other possible configurations.

- **PC-Based mode** is a variation of the host-computer mode which allows you to use a PC as both a terminal and a host. This requires a terminal-emulation package such as Microstuf's Crosstalk XVI.

10.2.2 Communications

The EVM firmware supports three ports for loading and dumping data (text, object code) for storage and/or display. Port 1 and Port 2 conform to EIA RS-232-C standards and support baud rates ranging from 110 to 9600 BPI. Port 3 is the audio tape interface.

10.2.3 Software Development

The EVM firmware resides in 16K bytes of EPROM and is divided into three functional areas:

- Debug monitor and EPROM programmer
- Assembler
- Text editor

The text editor is line oriented and provides basic character editing capabilities. Files can also be created using a host computer and downloaded to the EVM. CrossWare or the resident EVM assembler can be used to produce object code. Table 10-2 lists the TMS7000 EVM debug monitor commands.

Table 10-2. TMS7000 EVM Commands

MODIFY/DISPLAY REGISTER COMMANDS		GENERAL UTILITIES	
CP	Clear processor status	AR	Signed hexadecimal arithmetic
DP	Display processor status	CL	Display/modify cursor-left
MA	Display/modify Register A	CU	Display/modify cursor-up
MB	Display/modify Register B	DC	Display hex-byte conversion
MM	Display/modify memory	DV	Display/modify device type
MP	Display/modify Peripheral File	HC	Hex-Decimal word conversion
MR	Display/modify Register File	HE	Help
HS	Display/modify software handshake	MS/PC/ SR/SP	Display/modify PC, ST, and SP
MEMORY LOAD/DUMP COMMANDS		**GENERAL MEMORY/REGISTER MANIPULATION COMMANDS**	
DS	Display/save machine state	DM	Display memory
LM	Load memory, 7000 format	FB	Find byte in memory
LS	Load machine state	FM	Fill memory
LT	Load memory, Tektronix format	FR	Fill Register File
SM	Save memory, 7000 format	IO	Display I/O status
St	Save memory, Tektronix format	MV	Move memory
EIA SUPPORT COMMAND		NP	Fill Memory with NOPs
BR	Display/modify baud rate	**AUDIO TAPE COMMANDS**	
TEXT EDITOR SUPPORT COMMAND		DR	Audio tape directory
XE	Execute text editor	MO	Enable cassette motor

Table 10-2. TMS7000 EVM Commands (Concluded)

ASSEMBLER SUPPORT COMMANDS		EPROM PROGRAMMER COMMANDS	
AT	Display assembler label table	CE	Compare EPROM
XA	Execute assembler	PE	Program EPROM
XL	Execute line-by-line assembler	RE	Read EPROM
XP	Execute patch assembler	VE	Verify EPROM
PROGRAM SUPPORT COMMANDS			
BT	Set breakpoints on trap	LA	Show address of line
B1	Set breakpoint 1	LL	List line(s) from editor
B2	Set breakpoint 2	LN	Show line at address
CB	Clear breakpoints	L1	Set breakpoint 1 by line number
CT	Clear breakpoint on trap	L2	Set breakpoint 2 by line number
C1	Clear breakpoint 1	RT	Reset target processor
C2	Clear breakpoint 2	RU	Execute program without breakpoints
DB	Display breakpoints	SS	Single-step program
DT	Display breakpoint on trap	TC	Configure single-step trace
EF	Execute program with fixed display	TR	Display line trace
ET	Execute program with bpts/trace	TS	Single-step program with trace
EX	Execute program with breakpoints	T0	Load Program Counter with Trap 0 vector
FS	Single-step with fixed display		

10.2.4 EPROM Programming Utility

The EVM is equipped to program TMS2764, TMS27C64, TMS27128, and TMS27C128 EPROMs and the TMS7742 EPROM microcomputer. The ability to program EPROMs greatly reduces evaluation and development time. These devices use a 28-pin programming socket.

10.3 Prototyping Support

The SE70P162, SE70CP160, SE70CP162, TMS7742, and the TMS77C82[6] are protyping components that Texas Instruments offers to support form-factor emulation of a TMS7000 target processor. The SE devices are also referred to as piggybacks.

10.3.1 TMS7742 Description

The TMS7742 is an on-chip EPROM version of the 8-bit TMS7042 microcomputer. The TMS7742 can be used to emulate the TMS7020, TMS7040, and the TMS7042 microcomputers.

10.3.1.1 TMS7020 and TMS7040 Emulation

The TMS7742 can emulate the TMS7020/40 in all operating modes. If operated in a memory-expansion mode, the enhanced timing interface signals of the TMS7742 will seem transparent to any memory-expansion interface logic required for the TMS7020/40. The only feature of the TMS7020/40 that the TMS7742 cannot directly emulate is the edge- and level-sensitive interrupts. If level-sensitive interrupts are desired, external circuitry is required to allow the TMS7742 to sense level interrupts. If level-sensitive interrupts are not desired, the TMS7742 can emulate the TMS7020/40 with no alterations to the system hardware or software.

10.3.1.2 TMS7042 Emulation

The TMS7742 can directly emulate the TMS7042 up to 5 MHz without any hardware or software modifications. Above 5 MHz (5 MHz to 8 MHz), the SE70P162 provides direct emulation.

10.3.2 SE70P162 Description

The SE70P162 is the piggyback-EPROM prototyping device for the TMS7000 NMOS family of microcomputers. The SE70P162 can be used to emulate the TMS7020, TMS7040, and the TMS7042 microcomputers, with the same limitations as the TMS7742. However, the SE70P162 can operate at a maximum frequency of 8 MHz, enabling it to emulate the TMS7042 over the full operating range of the device.

10.3.3 SE70CP160 Description

The SE70CP160 is a CMOS piggyback-EPROM prototyping device. It emulates the TMS70C20 and TMS70C40 microcomputers.

10.3.4 SE70CP162 Description

The SE70CP162 is a CMOS piggyback-EPROM prototyping device. It emulates the TMS70C42.

10.3.5 TMS77C82 (Advance Information)

The TMS77C82[6] is an 8K on-chip EPROM version of the 8-bit TMS70C42 microcomputer. The TMS77C82 supports prototyping for the TMS70C42.

---

6 Advance Information

11. Independent Support

The TMS7000 family of single-chip microcomputers is supported by product offerings from a number of independent vendors. These support products take many forms, including cross-assemblers that run on small systems, second sources for the TMS7000 components, and PROM programming manufacturers that support TMS7000 EPROM programming.

This section discusses a number of tools that enhance the support provided by Texas Instruments. This does not constitute product endorsement by Texas Instruments; it is merely an attempt to aid product awareness. The products listed here are representative of independent vendor supplied products. This information is not intended to be an all-inclusive list.

Section **Page**

11.1 Allen Ashley - CP/M-Based Support Tools

Allen Ashley supports cross-assemblers for the TMS7000 family which allow any CP/M[7] system to serve as a development station for single-chip microcomputers and microprocessors.

The SYSTEM-TMS7 is a total software package, complete with documentation and utilities, for developing TMS7000 code on a CP/M-based small microprocessor system. The following computers are supported:

- IBM PC
- Morrow Micro Decision
- TRS-80 (TRSDOS) Mod III
- Osborne I
- Kaypro II
- North Star - CP/M
- Micropolis Mod II
- Xerox 820
- Standard 8" CP/M format (SSSD)

With minor exceptions, the SYSTEM-TMS7 assembler features instruction mnemonics and syntax as defined by Texas Instruments. The SYSTEM-TMS7 includes the ASMB interactive assembler/editor, the MAKRO macro assembler, the EDIT text editor, a cross reference generator, and offloading facilities.

The ASMB editor/assembler is intended for the creation, modification and test of program modules. ASMB includes a simple assembler, a line editor, and the facilities for saving and retrieving files from disk. Source code for ASMB is maintained in memory to eliminate the requirement for a separate edit cycle. The source language is assembled into object code directly into RAM for immediate testing. Program errors can be caught, repaired and re-assembled in seconds with ASMB. Validated program modules developed with ASMB can be saved on disk for input to the more powerful MAKRO disk assembler.

The MAKRO assembler includes full macro and conditional assembly features, as well as the ability to link a series of source files together during a single assembly. MAKRO reads the source code from disk and writes object code back to disk; all available memory is free for symbol tables and macro expansion. MAKRO is the vehicle by which the modules developed under ASMB can be collected together into a single program. MAKRO treats the disk as an extension of memory, and source files exceeding available memory size can be assembled.

---

7 CP/M is a registered trademark for Digital Research, Incorporated. All rights are reserved.

Independent Support - Allen Ashley

EDIT is a full-spectrum, string-oriented text editor which includes all the features required to create or modify source programs for the MAKRO assembler. Source programs on an input disk file are paged into a dynamic memory buffer, modified and written out to the output disk file. Commands include block move or delete, string search or change, and disk file merge. A single command reformats the line-oriented source file created under ASMB to the free-form source input of MAKRO.

Programs created with the development systems must be offloaded to the target processor. Facilities are provided to implement the offload as a direct transfer from memory, via a byte stream over a CPU port, or via COM or HEX files. An off loader for HEX files is provided. Direct support for off loading to the XDS line of TI support tools is included.

For more information, contact:

Allen Ashley, Inc.
395 Sierra Madre Villa
Pasadena, Ca. 91107

(818) 793-5748

11.2 Cybernetic Micro Systems - IBM-PC Crossware and TMS7000 Simulator

- IBM-PC Crossware

  Cybernetic Micro Systems' combination cross-assembler and EPROM programming board enables designers to develop assembly language programs for the TI TMS7000 family on an IBM PC. The CYS-7000 cross-assembler supports all of the TMS7000 family assembly language mnemonics, but eliminates support for macroroutines and relocatable object code.

  The software assembles instructions at a rate of 450 lines per minute. For EPROM programming needs, Cybernetic Micro Systems' CYP-27XX EPROM programming board can be connected to the PC's serial port and is able to program most 16- to 256-kbit EPROMs and 16-kbit EEPROMs.

  The entire development package consists of one diskette and programming board. The software runs on an IBM PC under PC-DOS 2.0. Source programs can be generated by any standard PC editor. Versions of this cross-assembler are also available from Cybernetic Micro Systems for the TI Professional Computer.

- TMS7000 Simulator

  The Cybernetic Micro Systems Sim7000 Simulator executes code for the TMS7000 family microcomputer on the IBM-PC type personal computer. The simulator allows TMS7000 programs to be debugged before execution on an emulator or piggyback chip. Sim7000 can simulate all the hardware functions of the TMS7000 family, including the serial port devices. The Sim7000 provides numerous features that assist the designer in debugging TMS7000 code, including symbolic execution, traps and breakpoints, access to memory spaces, and flow graph generation. This package is designed to work with the Cybernetic CYS-7000 cross assembler described above.

  The Sim7000 offers a display which is separated into various windows for easy viewing. These window provide the following information:

  Code window Shows lines for the source code

  Register window Display current state of the device

  Memory window Displays a portion of differents memory spaces.

  Stack window Lists the contents of the Stack

  Flow window The control flow with various options is shown.

  Help window Describes a command

  Command window Shows the current command with prompting

For more information, contact:

Cybernetic Micro Systems
P.O. Box 3000
San Gregorio, CA 94074

(415) 726-3000

11.3 Software Development Systems, Inc. - UNIX™ Based Cross-Development Tools

Uniware™[8] is an independent software package that supports any UNIX™-based host processor, with a cross-assembler available to support any of the TI TMS7000 devices as a target microprocessor. Uniware's software support includes a macro preprocessor that performs macro, textual variable substitutions and looping constructs on TMS7000 assembly language code. A link editor assigns load addresses to object modules, conditionally links in library modules and resolves symbolic references between modules. An extensive collection of utilities that include listing generators, object code format translators, and down loaders are also available.

Host processors supported by Uniware include:

- AT&T 3B
- Apollo
- DEC VAX (all)
- Heurikon
- Hewlett-Packard 9000
- Masscomp
- NCR Tower
- Plexus
- Suxi Microsystems
- Sequent
- Zilog System 8000
- Gould Power Systems
- Pyramid
- IBM PC/AT under Xenix
- IBM PC/AT and compatibles under DOS

For more information, contact:

Software Development Systems, Inc.
Uniware Cross-Development Tools
3110 Woodcreek Drive
Downers Grove, IL 60515

(312) 971-8170

8 UNIWARE is a trademark of Nuvatec, Inc. UNIX is a trademark of AT&T.

11.4 SEEQ - Self-Adaptive EEROM

The SEEQ[9] 72710 is a full-function single-chip microcomputer, fabricated in N-channel silicon-gate technology, which contains a 1K-by-8 5V nonvolatile electrically-erasable (EEROM) program memory. The program memory can be erased and programmed via the processor itself during normal program execution or can be programmed under control as if it were a standard 5V EEROM memory component. The EEROM can easily be expanded off-chip using the processor's Full-Expansion mode. External EEROM can be programmed with the same instruction used to alter on-chip EEROM.

A security lock mechanism is implemented in EEROM memory which allows your program to inhibit external access to its proprietary program code. Once activated, this lock can be reset only by an external EEROM block-clear operation, which erases the entire program memory contents.

As with other SEEQ EEROM devices, the 72710 has DiTrace[9] and Silicon Signature[9] features to facilitate production testing tracking. Each device is encoded with detailed processing and testing results which are stored in a special EEROM memory as it passes through the manufacturing cycle. Also stored is an unalterable identification code which contains information such as mask revision and EEROM programming parameters.

An EEROM member of the TMS7000 family is desirable because a single-chip microcomputer with non-volatile program memory that can be altered under process control allows the design of low cost products with many new features:

- Self adaptive code for machines that learn as they perform their tasks.
- In-circuit reprogrammability to eliminate product disassembly for firmware updates.
- Remote reprogrammability to eliminate service calls for firmware updates.
- Internally stored product history including factory test results, product configuration, revision level, and service records.
- Stored initialization parameters to eliminate front panel switches and automatically configure product for one or many users.
- Product usage and error logging to simplify maintenance and pinpoint product failure modes.
- Code and data security to protect proprietary programs and confidential data.

For more information, contact:

SEEQ Technology Incorporated
1849 Fortune Drive
San Jose, California 95131

(408) 942-1990

---

9 SEEQ, DiTrace, and Silicon Signature are registered trademarks for SEEQ Technology Incorporated. All rights are reserved.

11.5 Microcomputer Control - Multi-tasking Operating System

Microcomputer Control provides operating system support for all TMS7000 devices.

MICRO/OS provides a standard integrated software environment for managing tasks, time, and interrupts. Software design engineers are relieved of many time-consuming and error-prone activities involved in developing a reliable and flexible realtime control system. Control functions such as keypad scanning and display driving can be developed as independent tasks. Each task can be assigned its own priority and execution schedule. Built-in interrupt management allows tasks to be assigned to any interrupt source; preemption and context switching to the assigned task are performed automatically. *No additional program code is required.*

Task management is based on application-task priorities and the readiness state of tasks. At any given moment, the highest priority "ready task" is given full control of hardware resources. Hardware interrupts, time delays, and other tasks can make a task ready.

Time management allows independent parallel time delays to be active for each application task. Time delays are used to implement periodic functions such as keypad scanning and display updates. All time delays are based upon a user-specified System Time Unit. Built-in management of an on-chip timer removes an additional hardware or software requirements.

Interrupt management, an error-prone area in any control system, is reduced to its basic essential, the assignment of a task to a specific hardware interrupt. Built-in interrupt handling automatically reaides the assigned task, and blocks lower priority interrupts until the task is completed.

For more information, contact:

Microcomputer Control
P.O. Box 275
Hopewell, New Jersey 08525

(609) 466-1751

11.6 Hewlett-Packard - HP64000 Microcomputer Development System

The Hewlett-Packard HP64000 microcomputer development system is a real-time user-definable system which can be configured to support the TMS7000 family of microcomputers.

This user-definable system consists of the following devices which can be configured specifically for the TMS7000 family devices:

- HP642745 - User-definable emulator
- HP648515 - User-definable assembler/linker
- HP64856AF - User-definable inverse assembler
- HP64851B - User-definable interface

For more information, call the nearest Hewlett-Packard sales office listed in the telephone white pages. Ask for the Electronic Instrument department. You may also write to:

Hewlett-Packard
P.O. Box 617
Colorado Springs, Colorado 80901

In Colorado, call 590-3340 (collect)
Nationwide, call 1-800-447-3282

11.7 EPROM Microcomputer Support

The following third-party companies support programming of TMS7000 EPROM microcomputers.

- **Data I/O Corporation**

  10525 Willows Road N.E.
  P.O. Box 97046
  Redmond, Washington 98073-9746

  (206) 881-6444

- **PROMAC**

  Adams MacDonald Enterprises, Inc.
  2999 Monterey/Salinas Highway
  Monterey, California 93940

  (408) 373-3607

  Products include the PROMAC 2, PROMAC 15, and PROMAC 16.

- **Advanced Microcomputer Systems, Inc.**

  2780 S.W. 14th Street
  Pomano Beach, Florida 33069

  (305) 975-9515

  Products include the AMS2000 (IBM-PC compatible PC board) and the PROM 2000-8 (Personality box for the TMS7742).

- **Logical Devices, Inc.**

  1321-E N.W. 65th Place
  Fort Lauderdale, Florida 33309

  (305) 974-0967

  Products include the PROMPRO-XP with PM77 Adaptor and the PROMPRO-8x with PM77 Adaptor.

12. Customer Information

Topics covered in this section include:

12.1 Mask ROM Prototype and Production Flow

The TMS7000 family of masked-ROM microcomputers are semi-custom devices. The ROM is tailored to the customer's application requirements. The semi-custom nature of these devices requires a standard, defined interface between the customer and the factory in the production of TMS7000 devices with on-chip ROM. Figure 12-1 shows this standard prototype/production flow for customer ROM receipt.

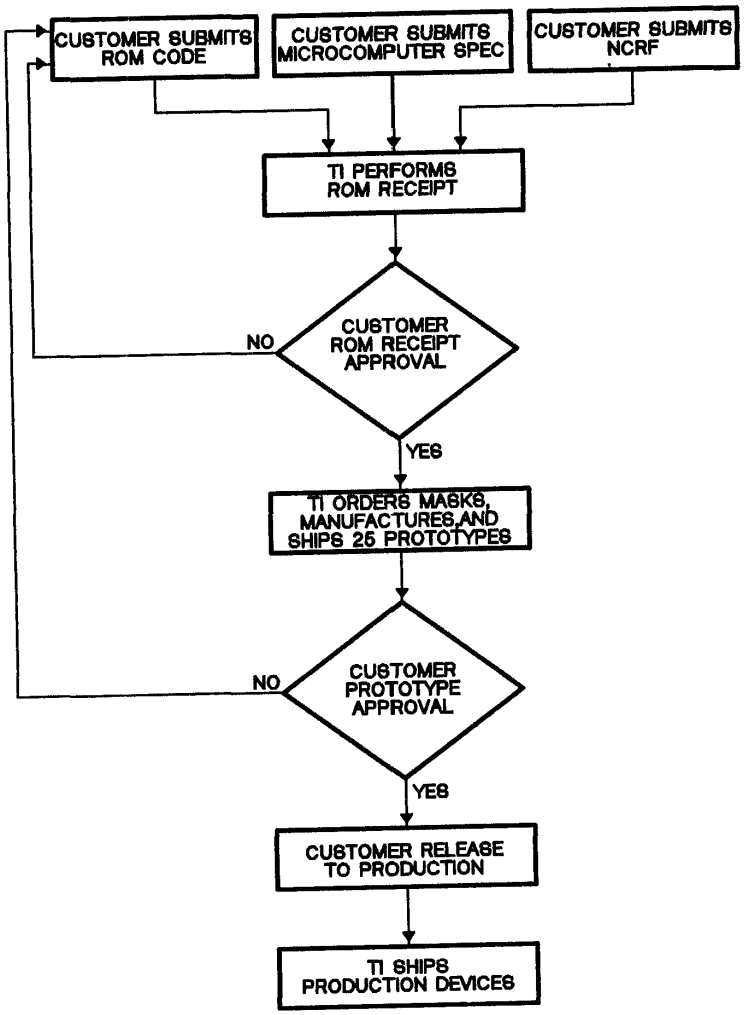


Figure 12-1. Prototype and Production Flow

1) Customer Required Information

For TI to accept the receipt of a customer ROM algorithm, each of the following three items must be received by the TI factory:

a) The customer completes and submits a New Code Release Form (NCRF - available from TI Field Sales Office) describing the custom features of the device (e.g., customer information, prototype and production quantities and dates, any exceptions to standard electrical specifications, customer part numbers and symbolization, package type, etc.).

b) If non-standard specifications are requested on the NCRF then the customer submits a copy of the specification for the microcomputer in their system, including the functional description and electrical specification (including absolute maximum ratings, recommended operating conditions, and timing values).

c) When the customer has completed code development and after verification of this code with the development system, the standard TMS7000 tagged object code is submitted to the TI factory on an acceptable media for processing. These include:

- EPROM devices (currently supported: TI2516, IN2716, TMS2732, TMS2764, and TMS27128)
- MS-DOS formatted disk compatible with IBM or TI PC
- Electronic ROM transfer: PC-to-PC via Xmodem protocol or Microstuf's Crosstalk XVI protocol
- Bulk Data Transfer from a Texas Instruments Regional Technology Center (RTC) to the TI Wilcrest facility to the DX990.
- Double-sided, double density floppy disks formatted by the TMAM9000 AMPLUS operating system.

The completed NCRF, customer specification (if required), and ROM code should be given to the Field Sales Office or sent to:

Texas Instruments Microcomputer Division
P.O. Box 1443, MS 6435
9901 S. Wilcrest
Houston, TX 77099
ATTN: TMS7000 Marketing Manager - ROM Receipt

2) TI Performs ROM Receipt

Code review and ROM receipt is performed on the customer's code and a unique manufacturing ROM code number is assigned to the customer's algorithm. All future correspondence should indicate this number. The ROM receipt procedure reads the ROM code information, processes it, reproduces the customer's ROM object code on the same media on which it was received, and returns the processed and the original code to the customer for verification of correct ROM receipt.

3) Customer ROM Receipt Approval

The customer then verifies that the ROM code received and processed by TI is correct and that no information was misinterpreted in the transfer. The customer must then return written confirmation of correct ROM receipt verification or re-submit the code for processing. This written confirmation of verification constitutes the contractual agreement for creation of the custom mask and manufacture of ROM verification prototype units.

4) TI Orders Masks, Manufacturing, and Ships 25 Prototypes

TI generates the prototype photomasks, processes, manufactures, and tests 25 microcomputer prototypes containing the customer's ROM pattern for shipment to the customer for ROM code verification. These microcomputer devices have been made using the custom mask but are for the purposes of ROM verification only. For expediency, the prototype devices are tested only at room temperature (25°C). **Texas Instruments recommends that prototype devices not be used in production systems.** Prototype devices are symbolized with a **P** preceding the manufacturing ROM code number (eg., PC13827N) to differentiate them from production devices.

5) Customer Prototype Approval

The customer verifies the operation of these prototypes in the system and responds with written customer prototype approval or disapproval. This written customer prototype approval constitutes the contractual agreement to initiate volume microcomputer production using the verified prototype ROM code.

6) Customer Release to Production

With customer algorithm approval, the ROM code is released to production and TI will begin shipment of production devices according to customer's final specification and order requirements.

Two lead times are quoted in reference to the preceding flow:

- Prototype lead time - elapsed time from the receipt of written ROM receipt verification to the delivery of 25 prototype devices.
- Production lead time - elapsed time from the receipt of written customer prototype approval to delivery of production devices.

For the latest TMS7000 family lead times, contact the nearest TI field sales office.

12.1.1 Reserved ROM Locations

All TMS7000 family devices with on-chip mask ROM reserve the first six bytes of the ROM space for TI use and therefore should not be used in the customer's software algorithm. For applications targeted for on-chip mask ROM production, the customer must remember to reserve this space during the development stage when using the XDS emulator, the EVM board, the TMS7742, piggyback emulators (SE70P162, SE70CP160, SE70CP162), or a TMS7000 family member without on-chip ROM. Table 12-1 lists the valid ROM starting addresses for the mask-ROM devices.

Table 12-1. Valid ROM Start Addresses

MEMBER	ROM SPACE	VALID START ADDRESS
TMS7020,70C20	2K bytes	>F806
TMS7040, TMS7042 TMS70C40, TMS70C42	4K bytes	>F006

12.1.2 Manufacturing Mask Options

The TMS7000 family supports two mask-programmed options, the oscillator input option (CMOS only) and the clock divide-by option (TMS7020 and TMS7040 only). These options are selected at the time of mask manufacture and therefore cannot be changed by software or hardware once the device has been manufactured. Selection of these mask options are designated by the customer in the New Code Release Form (NCRF) when ordering TMS7000 family members win on-chip mask ROM. TMS7000 family members without on-chip mask ROM have this designation as part of their standard part number symbolization.

The oscillator input option defines the type of external clock source connected to the oscillator inputs. The crystal input option identifies that the external clock source will be either a crystal, ceramic resonator, or another approximately 50% duty cycle external clock. The R-C input option identifies that an external R-C network will be connected to the oscillator terminals. The R-C option provides a simple and economical oscillator for uses where frequency tolerance is not a concern, and significantly reduces the low-power mode current requirements for all CMOS devices. The R-C option is supported only on the CMOS devices (TMS70C00, TMS70C20, TMS70C40, TMS70C02, and TMS70C42). All NMOS processors have the crystal option defined as the only form of oscillator option.

The clock divide-by option defines the internal oscillator divide-by for converting the external oscillator frequency, f_{osc}, to the internal machine cycle frequency. The ÷2 clock option defines that the internal machine cycle will be external oscillator frequency divided by two (for example, an 5 MHz external crystal would generate an internal machine cycle of 2.5 MHz). The ÷4 clock option defines that the internal machine cycle frequency will be the external oscillator frequency divided by four (for example, a 10 MHz external crystal would generate an internal machine frequency of 2.5 MHz). Table 12-2 defines the clock divide-by option supported by each family member.

Table 12-2. Clock Divide Options

CLOCK DIVIDE-BY	FAMILY MEMBERS
÷2	**NMOS** TMS7000, TMS7020, TMS7040, TMS7002, TMS7042, TMS7742, SE70P162 **CMOS** TMS70C00, TMS70C20, TMS70C40, TMS70C02, TMS70C42, SE70CP160, SE70CP162
÷4	**NMOS** TMS7000, TMS7020, TMS7040

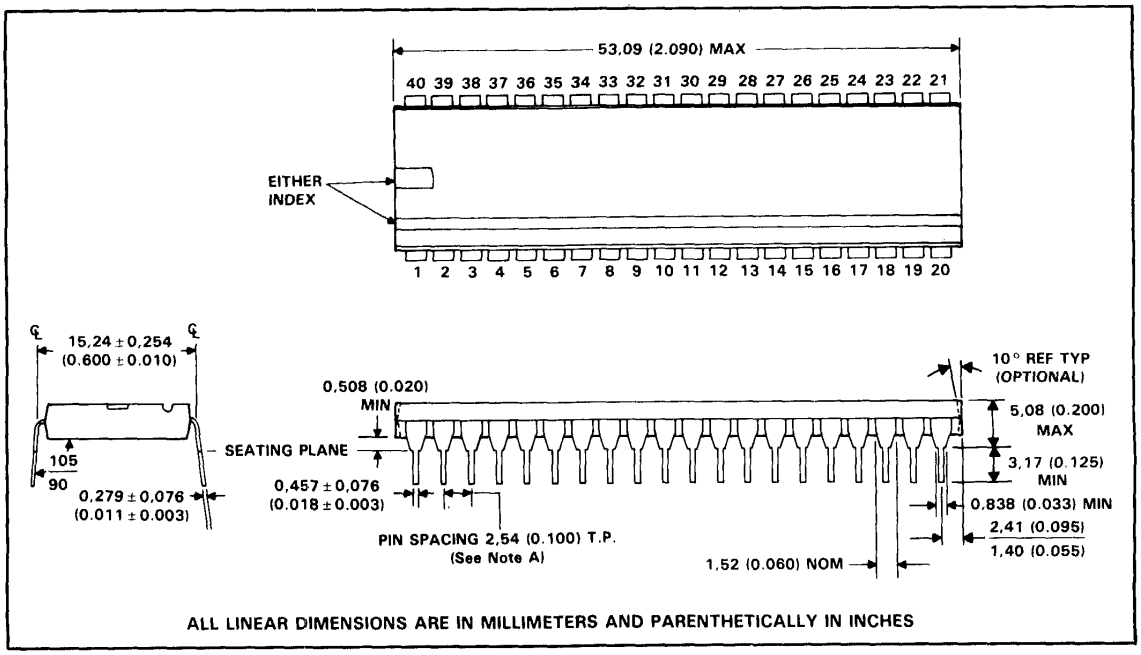
12.2 Mechanical Package Information

The TMS7000 microcomputer family devices are packaged in four package types according to the type of material and outline used for the package: plastic dual-inline package (DIP), plastic leaded chip carrier (PLCC), ceramic sidebraze package, and ceramic sidebraze piggyback package. Package types are designated in the device symbolization by the suffix on the customer's ROM code number for devices manufactured with customer ROM code (eg., C12799N) and by the suffix of the standard device number for devices without on-chip ROM. Table 12-3 indicates the package type, suffix indicator, and family members supported on that package type.

Table 12-3. Package Types

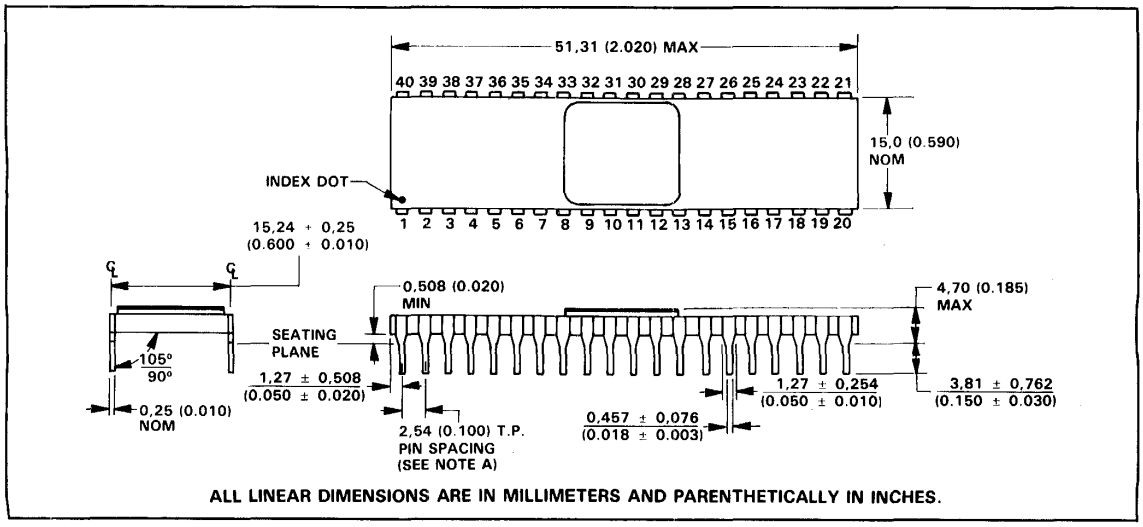
PACKAGE TYPE	SUFFIX INDICATOR	FAMILY MEMBERS	
40-pin plastic DIP (100-mil pin spacing)	N	**NMOS**	TMS7000, TMS7020, TMS7040 TMS7002, TMS7042
		CMOS	TMS70C00, TMS70C20, TMS70C40 TMS70C02, TMS70C42
40-pin ceramic sidebraze (100-mil pin spacing)	JD JD	**NMOS** **CMOS**	TMS7742 TMS77C82†
40-pin ceramic piggyback (100-mil pin spacing)	JD JD	**NMOS** **CMOS**	SE70P162 SE70CP160, SE70CP162
44-pin PLCC (50-mil pin spacing)	FN	**CMOS**	TMS70C00, TMS70C20, TMS70C40 TMS70C02, TMS70C42

† Advance Information



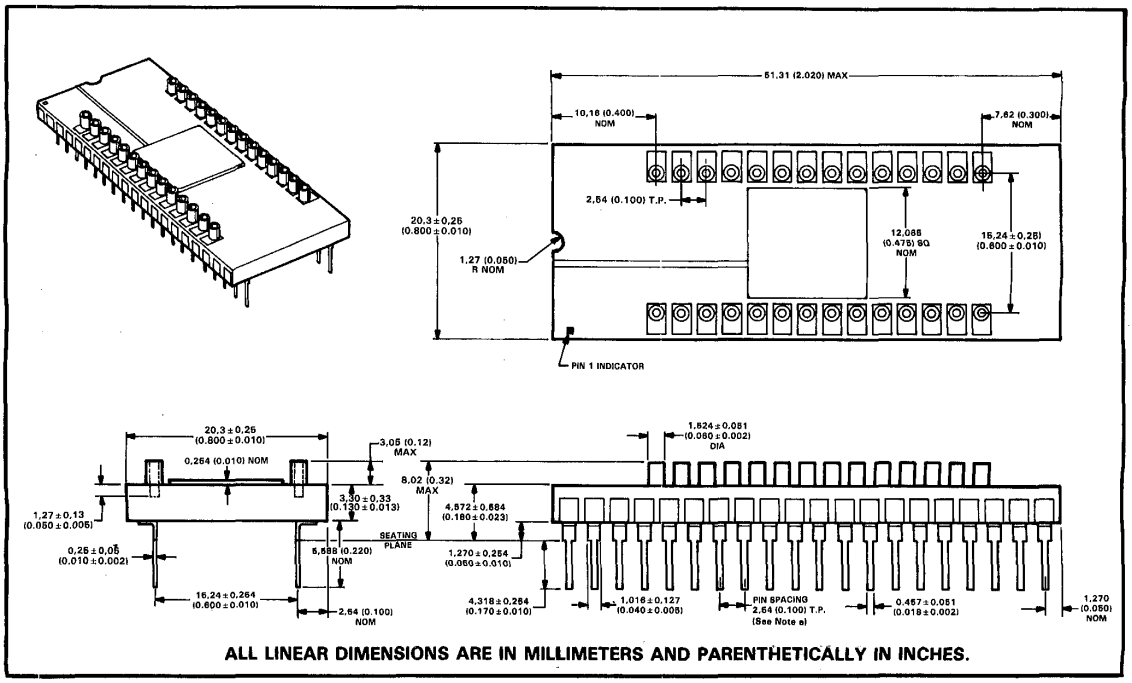
NOTE A: Each pin centerline is located within 0,254 (0.010) of its true longitudinal position.

Figure 12-2. 40-Pin Plastic Package, 100-MIL Pin Spacing (N Package Suffix)



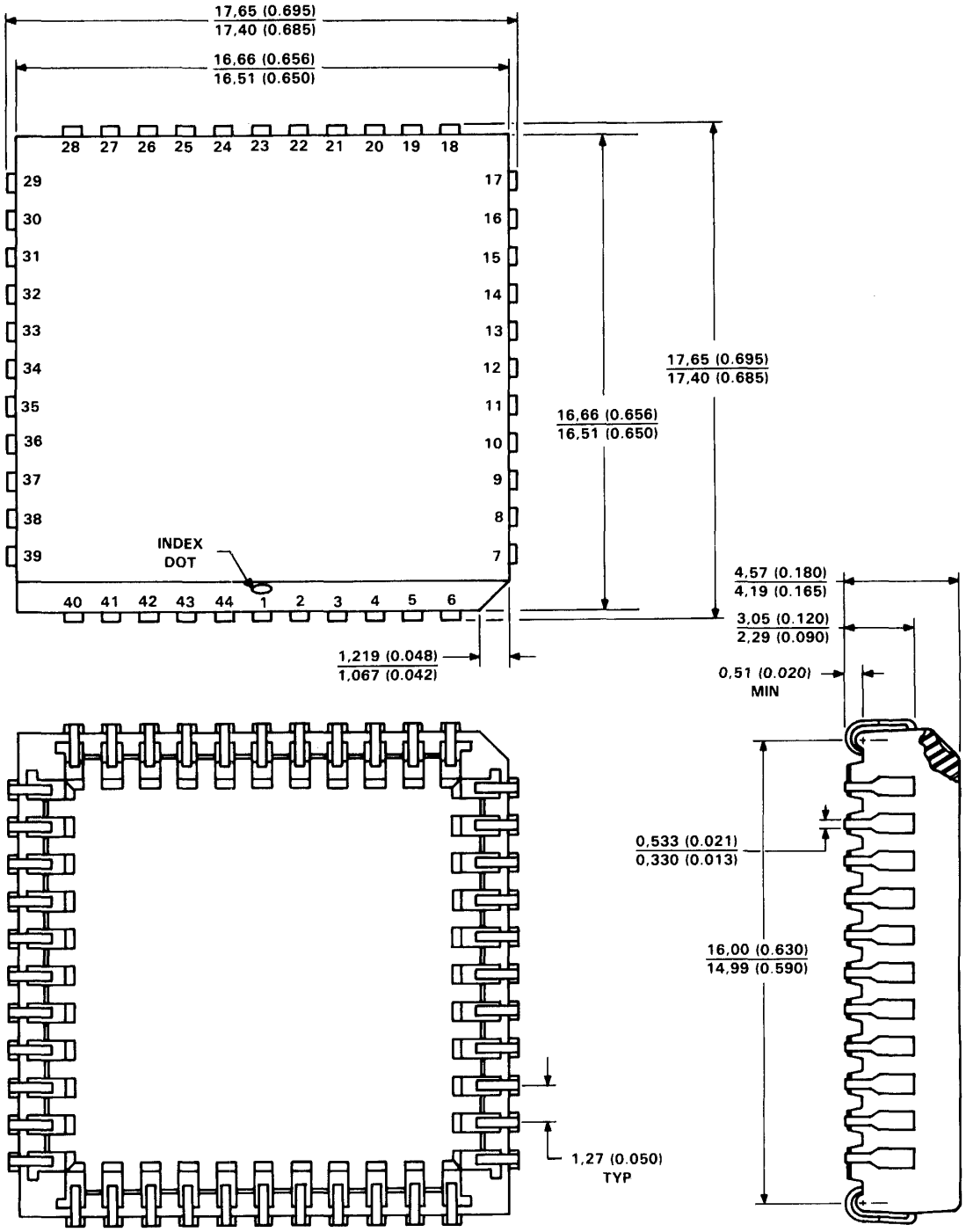
NOTE A: Each pin centerline is located within 0,254 (0.010) of its true longitudinal position.

Figure 12-3. 40-Pin Ceramic Package, 100-MIL Pin Spacing (Type JD Package Suffix)



NOTE A: Each pin centerline is located within 0,254 (0.010) of its true longitudinal position.

Figure 12-4. 40-Pin Ceramic Piggyback Package, 100-MIL Pin Spacing (Type JD Package Suffix)



ALL DIMENSIONS ARE IN MILLIMETERS AND PARENTHETICALLY IN INCHES

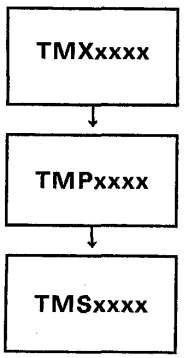
Figure 12-5. 44-Pin Plastic-Leaded Chip Carrier Package

12.3 TMS7000 Family Numbering and Symbol Conventions

12.3.1 Device Prefix Designators

To provide expeditious system evaluations by customers during the product development cycle, Texas Instruments assigns a prefix designator with four options: TMS, TMP, TMX, and SE.

TMX, TMP, and TMS are representative of the evolutionary stages of product development from engineering prototypes through fully qualified production devices. Figure 12-6 depicts this evolutionary development flowchart. Production devices shipped by Texas Instruments have the TMS designator signifying that they have demonstrated the high standards of Texas Instruments quality and reliability.



Experimental devices that are not representative of the final device's electrical specifications and has not completed reliability verification.

Final silicon die that conforms to the device's electrical specifications but has not completed quality and reliablity verification.

Fully qualified production devices.

Figure 12-6. Development Flowchart

TMX devices are shipped against the following disclaimer:

1) Experimental product and its reliability has not been characterized.
2) Product is sold "as is".
3) Product is not warranted to be exemplary of final production version if or when released by Texas Instruments.

TMP devices are shipped against the following disclaimer:

1) Customer understands that the product purchased hereunder has not been fully characterized and the expectation of reliability cannot be defined; therefore, Texas Instruments standard warranty refers only to the device's specifications.
2) No warranty of merchantability or fitness is expressed or implied.

TMS devices have been fully characterized and the quality and reliability of the device has been fully demonstrated. Texas Instruments' standard warranty applies.

The SE prefix designation is given to the system evaluator devices used for prototyping purposes. This designation applies only to the piggyback prototype members of the TMS7000 family (the NMOS SE70P162 and the CMOS SE70CP160 and SE70CP162 devices). SE devices are shipped against the following disclaimer:

System evaluators and development tools are for use only in a prototype environment and their reliability has not been characterized.

12.3.2 Device Numbering Convention

Figure 12-7 illustrates the numbering and symbol nomenclature for the TMS7000 family.

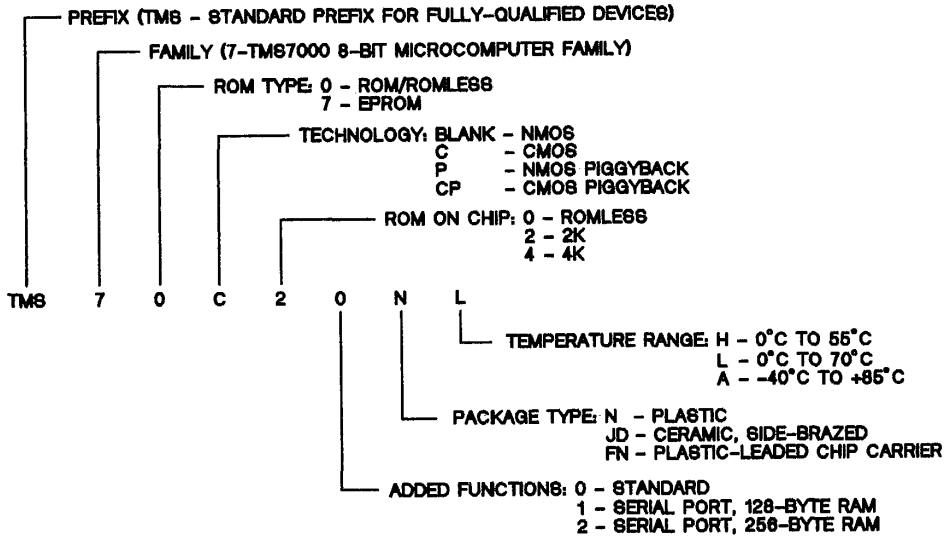


Figure 12-7. TMS7000 Family Nomenclature

12.3.3 Device Symbols

The TMS7000 family members can be divided into two categories for description of symbols, with the distinction being made on the presence (or absence) of on-chip ROM.

12.3.3.1 TMS7000 Family Members with On-Chip ROM

TMS7000 family members with on-chip ROM are semicustom devices where the ROM is mask programmed according to the customer's requirements. These devices follow the prototyping and production flow outlined in Section 12.3. Since they are semicustom devices, they receive a unique identification.

There are two types of symbolization for TMS7000 family members with on-chip ROM:

1) TI standard symbolization and
2) TI standard symbolization with customer part number.

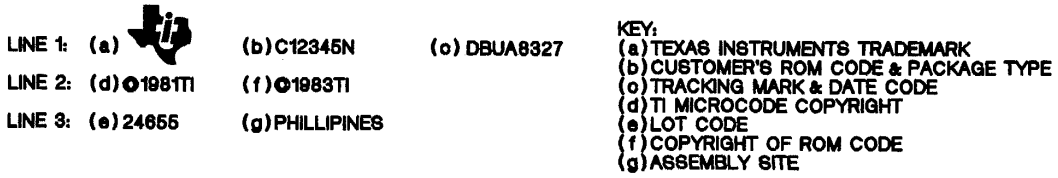


Figure 12-8. TI Standard Symbolization

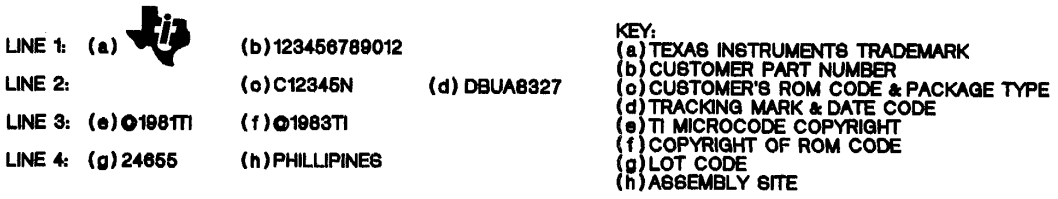


Figure 12-9. TI Standard Symbolization with Customer Part Number

12.3.3.2 TMS7000 Family Members without On-Chip ROM

TMS7000 family members without on-chip ROM are standard device types, and therefore have a standard identification. Examples of TMS7000 family members without on-chip ROM include:

TMS7000NL-2	TMS7002NL
TMS7000NL-4	TMS70C02NA

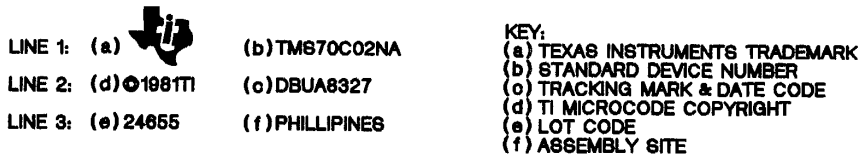


Figure 12-10. TI Standard Symbolization for Devices without On-Chip ROM

12.4 Development Support Tools Ordering Information

12.4.1 TMS7000 Macro Assembler/Linker

PART NUMBER	DESCRIPTION	OPERATING SYSTEM	MEDIUM
TMDS7040810-02	TI/IBM PC	PC/MS-DOS	5 1/4" floppy
TMDS7040123-06	TI 990	DX10	T50 hard disk
TMDS7040123-08	TI 990	DX10	1600 BPI mag tape
TMDS7040123-10	TI 990	DX10	DS10 hard disk
TMDS7040123-22	TI 990	DX10	CD1400 hard disk
TMDS7040210-08	DEC VAX	VMS	1600 BPI mag tape
TMDS7040310-08	IBM Mainframe	MVS	1600 BPI mag tape
TMDS7040320-08	IBM Mainframe	CMS	1600 BPI mag tape

12.4.2 TMS7000 XDS Emulators

PART NUMBER	XDS MODEL #
TMDS7062210	Model 22

TMDS7062210 XDS Upgrade Kit:

PART NUMBER
TMDS7068210

12.4.3 TMS7000 Evaluation Modules

PART NUMBER	DEVICES SUPPORTED
RTC/EVM7000N-1	TMS7020, TMS7040
RTC/EVM7000C-1	TMS70C20, TMS70C40, TMS70C42

5. The TMS7000 Assembler

TMS7000 Assembly Language instructions are mnemonic operation codes (or mnemonics) that correspond directly to binary machine instructions. An assembly language program (source program) must be converted to a machine language program (object program) by a process called *assembling* before a computer can execute it. Assembling converts the mnemonics to binary values and associates those values with binary addresses, creating machine language instructions. Assembler directives, discussed in Section 5.5, control this process, place data in the object program, and assign values to the symbols used in the object program.

TMS7000 assembly language is processed by a two-pass Macro Assembler that executes on a host computer. During the first pass the assembler:

1) Maintains the Location Counter,
2) Builds a symbol table, and
3) Produces a copy of the source code.

During the second pass the assembler:

1) Reads the copy of the source code and
2) Assembles the object code using the opcodes and symbol table produced during the first pass.

This section discusses the following topics:

Section **Page**

5.1 Source Statement Format

An assembly language source program consists of source statements that may contain assembler directives, machine instructions, pseudo-instructions, or comments. Source statements may contain four ordered fields – label, command, operand, and comment. Source statements that have an asterisk (*) in the first character position are comments and do not affect the assembly.

The syntax for source statements other than comment lines is:

```
[<label>]  <mnemonic>  [<operand>]  [<comment>]
```

where:

- The label and comments fields are optional.
- One or more blank spaces must separate each field.
- A statement must start with either a label or a blank space.

Note that square brackets ([*and*]) *indicate an optional entry.*

Figure 5-1 illustrates one method of entering source statements. Labels begin in column 1, opcodes in column 8, operands in column 14, and comments in column 26. The assembler produces the three left hand numbers. The first is the statement number, the second shows the program address, and the third shows the data value.

```
EXAMPLE  TMS7000 FAMILY MACRO ASSEMBLER

PAGE 0001
                        *---------------------------------*
                        * EXAMPLE OF SOURCE PROGRAM INPUT *
                        *---------------------------------*
                                      IDT   'EXAMPLE'
                                      CLR   B
                              LABEL1  MOVP  P4,A
                                      BTJZ  %01,A,LABEL1
                                      END
NO ERRORS, NO WARNINGS
```

```
0001
0002
0003
0005
0005
0006  0000  C5
0007  0001  80
      0002  04
0008  0003  67
      0004  FC
0009
```

Figure 5-1. Source Statement Format

5.1.1 Label Field

The label field is optional for machine instructions and for many assembler directives. If it is not used, the first character position must contain a blank. The label begins in the first character position of the source statement and extends to the first blank. It contains a symbol of up to 6 alphanumeric characters; the first character must be a letter.

A source statement that contains only a label field is a valid statement. It assigns the current value of the location counter to the label, which is equivalent to the following directive statement:

```
<label>  EQU  $
```

5.1.2 Command Field

The command field begins after the blank that terminates the label field. It is terminated by one or more blanks and may not extend past the right margin. If the label is omitted, the command can start in the second character position. The command field can contain one of the following opcodes:

- Machine-instruction mnemonic
- User-defined instruction
- Assembler directive

5.1.3 Operand Field

The operand field begins following the blank that ends the command field. It may not extend past the right margin of the source record. The operand field may contain one or more constants or expressions (described in Section 5.2 and Section 5.4) separated by commas. It is terminated by one or more blanks.

5.1.4 Comment Field

The comment field begins after the blank that terminates the operand field (or the blank that terminates the command field, if there are no operands). The comment field can extend to the end of the source record, if required, and can contain any ASCII character including blanks. The comment field contents (up to the end of the input record) are listed in the assembly source listing but do not affect the assembly.

5.2 Constants

The assembler recognizes five types of constants, each internally maintained as a 16-bit quantity:

- Decimal integer constants
- Binary integer constants
- Hexadecimal integer constants
- Character constants
- Assembly-time constants

5.2.1 Decimal Integer Constants

Decimal integer constants are written as strings of decimal digits, ranging from -32,768 to +65,535. Positive decimal integer constants in the range 32,768 to 65,535 are considered negative when interpreted by functions needing 2's complement values.

These are valid decimal constants:

1000	Constant equal to 1000 or >3E8
-32768	Constant equal to -32768 or >8000
25	Constant equal to 25 or >19
65535	Constant equal to 65535 to >FFFF

5.2.2 Binary Integer Constants

Binary integer constants are written as strings of up to 16 binary digits (0/1) preceded by a question mark (?). If less than 16 digits are specified, the assembler right justifies the bits.

These are valid binary constants:

?00010011	Constant equal to 19 or >13
?0111111111111111	Constant equal to 32767 or >7FFF
?11110	Constant equal to 30 or >001E

5.2.3 Hexadecimal Integer Constants

Hexadecimal integer constants are written as strings of up to four hexadecimal digits preceded by a greater than sign (>). Hexadecimal digits include the decimal values '0' through '9' and the letters 'A' through 'F'.

These are valid hexadecimal constants:

>78	Constant equal to 120
>F	Constant equal to 15
>37AC	Constant equal to 14252

5.2.4 Character Constants

Character constants are written as strings of one or two alphabetic characters enclosed in single quotes. Two consecutive single quotes are required to represent a single quote in a character constant. The characters are represented internally as 8-bit ASCII characters. A character constant consisting of only two single quotes (no letter) is valid and is assigned the value >0000.

These are valid character constants:

'AB'	Represented internally as >4142
'C'	Represented internally as >43 or >0043
'N'	Represented internally as >4E or >004E
''''D'	Represented internally as >2744

5.2.5 Assembly-Time Constants

Assembly-time constants are symbols assigned values by an EQU directive (see the EQU directive). The symbol value is determined at assembly time. It is considered to be absolute or relocatable according to the relocatability of the expression, not according to the relocatability of the Location Counter value. Absolute value symbols may be assigned values with expressions using any of the above constant types.

5.3 Symbols

Symbols are used in the label field and the operand field. A symbol is a string of alphanumeric characters (A–Z, 0–9, and $). The first character in a symbol must be A–Z or $. No character may be blank. When more than six characters are used in a symbol, the assembler prints all the characters, but only recognizes the first six characters during processing (the assembler also prints a symbol truncation warning). Therefore, the first six characters of a symbol should be unique. User-defined symbols are valid only during the assembly in which they are defined.

Symbols used in the label field become symbolic addresses. They are associated with locations in the program and must not be used in the label field of other statements. Mnemonic opcodes and assembler directive names may be used as valid user-defined symbols in the label field.

Symbols used in the operand field must be defined in the assembly, usually by appearing in the label field of a statement or in the operand field of a REF or SREF directive.

These are examples of valid symbols:

```
START
ADD
OPERATION
```

Each of these symbols will be assigned the value of the location where it appears in the label field. Note that the symbol OPERATION will be truncated to OPERAT.

5.3.1 Predefined Symbols

The dollar sign ($), register (Rn), and port (Pn) symbols are predefined. The dollar sign represents the current value of the location counter. Register and port symbols are in the form Rn and Pn, respectively, where n is a constant in the range 0–255. All registers and peripheral file addresses should be defined before they are used in instructions.

These are examples of valid predefined symbols:

$	The current location
R0	Register 0
P22	Peripheral Register 22

The symbol ST (Status Register) is reserved and may not be re-defined.

5.3.2 Terms

Terms are used in the operand field of machine instructions and assembler directives. A term may be a binary, character, decimal or hexadecimal constant, an absolute assembly-time constant or a label having an absolute value.

5.3.3 Character Strings

Several assembler directives require character strings as operands. A character string is a string of characters enclosed in single quotes. Single quotes *within* a character string are represented by two consecutive single quotes. The maximum length of a string is defined for each directive that requires a character string. The characters are represented internally as 8-bit ASCII characters.

These are valid character strings:

'SAMPLE PROGRAM' Defines a 14-character string, SAMPLE PROGRAM

'PLAN ''C''' Defines an 8-character string, PLAN 'C'

'OPERATOR MESSAGE : PRESS START SWITCH'
Defines a 37-character string, OPERATOR MESSAGE : PRESS START SWITCH

5.4 Expressions

Expressions are used in the operand fields of assembler directives and machine instructions. An expression is a constant or symbol, a series of constants or symbols, or a series of constants and symbols separated by arithmetic operators. Each constant or symbol may be preceded by a unary minus sign (-), a unary plus sign (+), or the unary invert symbol (#). The # symbol causes the value of the logical complement of the following constant or symbol to be used. An expression may not contain embedded blanks. Symbols defined as external references may be operands of arithmetic instructions within certain limits, as described in Section 5.4.1.

5.4.1 Arithmetic Operators in Expressions

The arithmetic operators used in expressions are:

+	Addition
-	Subtraction
*	Multiplication
/	Signed division
#	Logical not (inversion)

When the assembler evaluates an expression, it first negates symbols or constants preceded by a minus (-) sign and then performs arithmetic operations from left to right. The assembler does not assign precedence to any operation other than unary plus or unary minus. All operations are integer operations; any fractions produced by division are truncated.

For example, the expression 4+5*2 is evaluated as 18, not 14. The expression 7+1/2 is evaluated as 4; the expression 1/2+7 is evaluated as 7 (note truncation).

The assembler checks for overflow conditions when arithmetic operations are performed. It issues a warning message when an overflow occurs or when the sign of the result is not as expected in respect to the operands and the operation performed. Examples where a "VALUE TRUNCATED" message is given are:

```
-2*>4000        >FFFE+2        -1*>8001

>8000*2         ->8000-1       -2*>8000
```

When the immediate value is greater than >7F and you precede the value with %#, signifying immediate and unary negation operations, the assembler correctly calculates the value but issues an error message. Ignore the EXPRESSION OUT OF BOUNDS error message. (Note that this problem has been fixed in version 2.3 of the assembler.) The following example illustrates this condition.

```
TEST      TMS7000 MACRO ASSEMBER

PAGE 0001
  0001              *
  0002              *   DX-10 X-SUPPORT TEST SOFTWARE
  0003              *
  0004                      IDT     'TEST'
  0005 F000                 AORG    >F000
  0006 F000   52            MOV     %>10,B
       F001   10
  0007 F002   0D            LDSP
  0008 F003   01            IDLE
  0009 F004   28            ADD     %#>40,A
       F005   BF
  0010 F006   28            ADD     %#>7F,A
       F007   80
  0011 F008   28            ADD     %#>80,A
       F009   7F
  ********EXPRESSION OUT OF BOUNDS
  0012                      END
  0001 ERROR, 0000 WARNINGS, LAST ERROR AT 0011
```

5.4.2 Logical Operands in Expressions

If a pound sign (#) precedes a number or an expression it is complemented. All other arithmetic operations have precedence over the logical not (#) operation, except where modified by parentheses.

5.4.3 Parentheses in Expressions

Use parentheses to alter the order of expression evaluation. Parenthetical expressions can be nested up to eight levels. The portion of an expression within the innermost parentheses is evaluated first, then the next innermost pair is evaluated, etc. When all parenthetical phrases have been evaluated, the expression is evaluated from left to right. Evaluation of parenthetical phrases at the same nesting level may be considered to be simultaneous.

This expression is evaluated as follows:

LAB1+((4+3)*7)

1) Add 4 to 3
2) Multiply 7 by 7
3) Add the value of LAB1 to 49

5.4.4 Well-Defined Expressions

Some assembler directives require well-defined expressions in operand fields. Well-defined expressions contain only symbols or assembly-time constants that are defined before they are encountered in the expression. The evaluation of a well-defined expression must be absolute. A well-defined expression must not contain a character constant.

5.4.5 Relocatable Symbols in Expressions

An expression that contains a relocatable symbol or relocatable constant immediately following a multiplication or division operator is illegal. When the result of evaluating an expression up to a multiplication or division operator is relocatable, the expression is illegal.

If the current value of an expression is relocatable with respect to one relocatable section, a symbol of another section may not be included until the value of the expression becomes absolute. Some examples of relocatable symbols used in expressions are:

BLUE+1 The sum of the value of symbol BLUE plus one.
GREEN-4 The result of subtracting four from the value of symbol GREEN.
2*16+RED The sum of the value of symbol RED plus the product of two and 16.
440/2-RED The result of dividing 440 by two and subtracting the value of symbol RED from the quotient. RED must be absolute.

Table 5-1 defines the relocatability of the result for each type of operator.

Table 5-1. Results of Operations on Absolute and Relocatable Items in Expressions

A	B	A+B	A-B	A×B	A/B
ABS	ABS	ABS	ABS	ABS	ABS(B<>0)
ABS	RELOC	RELOC	illegal	†	illegal
RELOC	ABS	RELOC	RELOC	‡	§
RELOC	RELOC	illegal	¶	illegal	illegal

† Illegal unless A equals zero or one. If A is one, the result is relocatable. If A is zero, the result is an absolute zero.
‡ Illegal unless B equals zero or one. If B is one, the result is relocatable. If B is zero, the result is an absolute zero.
§ Illegal unless B equals one. If B equals one, the result is relocatable.
¶ Illegal unless A and B are in the same relocatable segment. If A and B are in the same section, the result is absolute.

5.4.6 Externally Defined Symbols in Expressions

Externally defined symbols (defined in REF and SREF directives) are allowed in expressions under the following conditions:

1) Only one externally referenced symbol may be used in an expression.

2) The character preceding the referenced symbol must be a plus sign, a blank, or a comma (the @ sign is not considered). The portion of the expression preceding the symbol, if any, must be added to the symbol.

3) The portion of the expression following the referenced symbol must not include multiplication, division, or logical operations on the symbol (as for a relocatable symbol described in Section 5.4.5).

4) The remainder of the expression following the referenced symbol must be absolute.

The assembler limits the total number of external referenced symbols to 255 per module. Modules using more than 255 external symbols must be broken into smaller modules for assembly and linked using the link editor.

5.5 Assembler Directives

Assembler directives control the assembly process. This section discusses the various categories of directives supported by the TMS7000 Assembler and defines the directives in alphabetical order.

Directives that Affect the Location Counter

As the assembler reads program source statements it increments its Location Counter. The Location Counter contents correspond to the memory locations assigned to the resulting object code. Twelve directives, listed in Table 5-2 on page 5-13, affect the Location Counter. BES and BSS advance the Location Counter to provide a block of program memory for the object code. The EVEN directive ensures an even address word boundary. The remaining nine directives initialize the Location Counter and define its value as relocatable, absolute, or dummy.

Directives in this category include:

- AORG
- BES
- BSS
- CEND
- CSEG
- DEND
- DORG
- DSEG
- EVEN
- PEND
- PSEG
- RORG

Directives that Affect Assembler Output

Directives that affect assembler output are mainly used to improve program useability. The IDT directive supplies a program identifier; the five other directives affect the source listing.

- IDT
- LIST
- OPTION
- PAGE
- TITL
- UNL

Directives that Initialize Constants

These directives assign values to successive bytes or words of the object code (BYTE, DATA), place text characters in object code for display purposes (TEXT), or initialize constants to be used during the assembly (EQU).

- BYTE
- DATA
- EQU
- TEXT

Directives for Linking Programs

The Link Editor resolves externally referenced symbols and definitions. These directives help the Link Editor by identifying symbols and definitions that may be used or defined by another program module. This allows separate program modules to be assembled separately and integrated into an executable program.

- DEF
- LOAD
- REF
- SREF

Miscellaneous Directives

This category includes those assembler directives not applicable to the other categories:

- COPY
- END
- MLIB

Table 5-2. Summary of Assembler Directives

DIRECTIVES THAT AFFECT THE LOCATION COUNTER		
MNEMONIC	**DIRECTIVE**	**SYNTAX**
AORG	Absolute origin	[<label>] AORG [<wd-exp> [<comment>]]
BES	Block ending with symbol	[<label>] BES <wd-exp> [<comment>]
BSS	Block starting with symbol	[<label>] BSS <wd-exp> [<comment>]
CEND	Common segment end	[<label>] CEND [<comment>]
CSEG	Common segment	[<label>] CSEG ['<string>' [<comment>]]
DEND	Data segment end	[<label>] DEND [<comment>]
DORG	Dummy origin	[<label>] DORG [<exp> [<comment>]]
DSEG	Data segment	[<label>] DSEG [<comment>]
EVEN	Even boundary	[<label>] EVEN [<comment>]
PEND	Program segment end	[<label>] PEND [<comment>]
PSEG	Program segment	[<label>] PSEG [<comment>]
RORG	Relocatable origin	[<label>] RORG [<exp> [<comment>]]
DIRECTIVES THAT AFFECT ASSEMBLER OUTPUT		
MNEMONIC	**DIRECTIVE**	**SYNTAX**
IDT	Program identifier	[<label>] IDT '<string>' [<comment>]
LIST	Restart source listing	[<label>] LIST [<comment>]
OPTION	Output options	[<label>] OPTION <option list> [<comment>]
PAGE	Page eject	[<label>] PAGE [<comment>]
TITL	Page title	[<label>] TITL '<string>' [<comment>]
UNL	Stop source listing	[<label>] UNL [<comment>]
DIRECTIVES THAT INITIALIZE CONSTANTS		
MNEMONIC	**DIRECTIVE**	**SYNTAX**
BYTE	Initialize byte	[<label>] BYTE <exp>[,<exp>] [<comment>]
DATA	Initialize word	[<label>] DATA <exp>[,<exp>] [<comment>]
EQU	Define assembly-time constant	[<label>] EQU <exp> [<comment>]
TEXT	Initialize text	[<label>] TEXT [-]'<string>' [<comment>]
DIRECTIVES FOR LINKING PROGRAMS		
MNEMONIC	**DIRECTIVE**	**SYNTAX**
DEF	External definition	[<label>] DEF <symbol>[,<symbol>] [<comment>]
LOAD	Force load	[<label>] LOAD <symbol>[,<symbol>] [<comment>]
REF	External reference	[<label>] REF <symbol>[,<symbol>] [<comment>]
SREF	Secondary external reference	[<label>] SREF <symbol>[,<symbol>] [<comment>]
MISCELLANEOUS DIRECTIVES		
MNEMONIC	**DIRECTIVE**	**SYNTAX**
COPY	Copy source file	[<label>] COPY <filename> [<comment>]
END	Program end	[<label>] END [<symbol> [<comment>]]
MLIB	Define macro library	[<label>] MLIB '<pathname>' [<comment>]

Syntax [<label>] AORG [<wd-exp> [<comment>]]

Fields

Label Optional; if used, the label is assigned the same value that AORG places in the Location Counter.

Operand Optional; if used, the operand field must contain a well-defined expression (<wd-exp>).

Comment Optional; may only be used with the operand field.

Description AORG loads the Location Counter with the first address of a segment of absolute code. This address is usually specified by the operand. If no operand is used, the value in the Location Counter equals the length of all preceding absolute code. When no AORG directive is entered, the object program does not include absolute addresses.

Example 1

```
        AORG  >1000+X
```

Symbol X must be absolute and previously defined. If X has a value of 6, the Location Counter is set to >1006. If a label had been included, it would have been assigned the value >1006.

Example 2 Avoid using AORG in object modules which will be linked. Linking a module that contains an AORG directive may produce an *Illegal immediate tag encountered* error at link time. Use the PSEG, CSEG, and DSEG directives instead to identify the locations in the source code. Use the PROGRAM, COMMON, and DATA commands in the link control file to define the locations.

The link control file will look similar to this example:

```
TASK    MYPROG
PROGRAM >F006  Program starting point (PSEG)
DATA    >FFD0  Trap and vector table stg pt (DSEG)
COMMON         Additional starting location (CSEG)
INCLUDE FILE1
INCLUDE FILE2
END
```

This example will work if the program contains no more than three AORGs.

Syntax [<label>] BES <wd-exp> [<comment>]

Fields

Label Optional; if used, the label is assigned the value of the location following the block.

Operand Contains a well-defined expression that represents the number of bytes to be added to the Location Counter.

Comment Optional

Description BES increments the Location Counter by the operand value.

Example

```
BUFF2   BES  >10
```

A 16-byte buffer is reserved. If the Location Counter had contained >100 when the directive was processed, BUFF2 would have been assigned the value >110.

BSS Block Starting with Symbol Directive BSS

Syntax [<label>] BSS <wd-exp> [<comment>]

Fields

Label Optional; if used, a label is assigned the value of the location of the first byte in the block.

Operand Contains a well-defined expression that represents the number of bytes to be added to the Location Counter.

Comment Optional

Description BSS increments the Location Counter by the operand value.

Avoid using the BSS directive for defining register names. Using BSS in this manner may produce a *Pass1/Pass2 operand conflict* error at assembly time. Use the EQU directive for defining register names.

Example

```
BUFF1   BSS  80  Card input buffer
```

An 80-byte buffer is reserved starting at location BUFF1.

Syntax [<label>] BYTE <exp>[,<exp>] [<comment>]

Fields

Label Optional; if used, the label is assigned the location where the assembler places the first byte.

Operand Contains one or more expressions separated by commas. These expressions cannot contain external references. The assembler evaluates each expression and places the value in a byte as an 8-bit number. If truncation is required, the assembler prints a truncation warning message and puts the 8 LSbs of the value in the byte.

Comment Optional

Description BYTE places one or more values in one or more successive bytes of memory.

Example

```
KONS BYTE >F+1,-1,'D'-'=',0,'AB'-'AA'
```

This example initializes five bytes, starting with a byte at location KONS. The contents of the resulting bytes are 00010000, 11111111, 00000111, 00000000, and 00000001.

Syntax [<label>] CEND [<comment>]

Fields

Label Optional; if used, the label is assigned the value of the Location Counter before modification.

Operand Not used

Comment Optional

Description

CEND terminates the definition of a block of common-relocatable code by placing a value in the Location Counter and defining succeeding locations as program-relocatable. The Location Counter is set to one of the following values:

- The maximum value the Location Counter has ever attained by assembling any preceding block of program-relocatable code.
- Zero, if no program-relocatable code was previously assembled.

If encountered in data- or program-relocatable code, this directive functions as a DEND or PEND. CEND is invalid when used in absolute code.

Syntax [<label>] COPY <filename> [<comment>]

Fields

Label Optional

Operand Names a file that source statements are read from. The file name may be:

- An access name recognized by the operating system
- A synonym form of an access name

Comment Optional

Description COPY changes the source input for the assembler. A COPY directive may be placed in a file being copied. Nested copying of files can be performed by placing a COPY directive in a file being copied. The assembler limits such nesting to eight levels; the host operating system may place additional restrictions on nesting capabilities.

Example

```
        COPY    SFILE
```

This example causes the assembler to take its source statements from a file SFILE. At the end-of-file for SFILE, the assembler resumes processing source statements from the file or device previous to the COPY directive.

Syntax [<label>] CSEG ['<string>'[,<exp>] [<comment>]]

Fields

Label Optional; if used, the label is assigned the value placed in the Location Counter.

Operand Optional (see preceding Description).

Comment Optional; may only be used with the operand field.

Description

CSEG begins or continues a common-relocatable segment (relocatable with respect to a common segment) at the address in the Location Counter. If the operand is not used, the CSEG directive defines the beginning of (or continuation of) the blank common segment of the program.

When used, the operand field contains a character string of up to six characters enclosed in quotes. (The assembler truncates strings that are longer than six characters and prints a truncation error message.) If this string did not previously appear as the operand of a CSEG directive, the assembler:

1) Associates a new relocation section number with the operand,
2) Sets the Location Counter to zero, and
3) Defines succeeding locations as relocatable with respect to the new relocatable section.

If the operand string was previously used in a CSEG, the succeeding code represents a continuation of the particular common segment associated with the operand. The Location Counter is restored to the maximum value attained during the previous assembly of any portion of that particular common segment. The second operand, <exp>, specifies the memory alignment for the beginning of the Section.

Common-relocatable code is normally terminated by a CEND directive, but can also be terminated by the PSEG, DSEG, AORG, and END directives. The CEND and PSEG directives define succeeding locations as program-relocatable. The DSEG and AORG directives terminate the common segment by beginning a data or an absolute segment. The END directive terminates the common segment and the program.

The CSEG directive permits construction and definition of independently relocatable segments of data that several programs can access or reference at execution time. Information placed in the object code by the assembler permits the link editor to relocate all common segments independently and make appropriate adjustments to all addresses that reference locations within common segments. Locations within a common segment may be referenced by several different programs if each program contains a CSEG directive with the same operand or no operand.

Example

```
COM1A   CSEG    'ONE'
          .
          .
<Common-relocatable section, type 'ONE'>
          .
          .
        CEND
          .
COM2A   CSEG    'TWO'
          .
          .
<Common-relocatable section, type 'TWO'>
          .
          .
COM2B   CEND
          .
COM1C   CSEG   'ONE'
          .
          .
<Common-relocatable section, type 'ONE'>
          .
          .
COM1B   CEND
          .
COM1L   DATA COM1B-COM1A  LENGTH OF SEGMENT 'ONE'
COM2L   DATA COM2B-COM2A  LENGTH OF SEGMENT 'TWO'
```

The three blocks of code between the CSEG and CEND directives are common-relocatable.

The first and third blocks are relocatable with respect to one common relocation type; the second is relocatable with respect to another. The first and third blocks comprise the common segment 'ONE'; the value of the symbol COM1L is the length in bytes of this segment.

The symbol COM2A is the symbolic address of the first word of the first word of common segment 'TWO'; COM2B is the common-relocatable (type 'TWO') byte address of the location following the segment. (Note that the symbols COM2B and COM1C are of different relocation types and possibly different values.) The value of the symbol COM2L is the length in bytes of common segment 'TWO'.

Syntax [<label>] DATA <exp>[,<exp>] [<comment>]

Fields

Label Optional; if used, the label is assigned the location where the assembler places the first word.

Operand Contains one or more expressions separated by commas. The assembler evaluates each expression and places the value in a word as a 16-bit number. Words are stored most significant byte first, i.e., at the lower address.

Comment Optional

Description DATA places one or more values in one or more successive 2-byte words of memory.

Example 1

```
KONS1 DATA 3200,1+'AB',-'AF',>F4A0,'A'
```

This example initializes five words, starting with a word at location KONS1. The contents of the resulting words are >0C80, >4143, >BEBA, >F4A0, and >0041.

Example 2 In a DATA directive statement with an operand of multiple fields, the assembled value of the location counter symbol ($) will **not** be correctly calculated if the $ is not in the first field (i.e., a correct value will be calculated for $ if it is in the first field of the DATA statement.) The following example shows both cases. This example is for assembler revision 2.1.

```
TEST        7000 FAMILY MACRO ASSEMBLER  DX2.1  83.074  15:23:38
                7/25/84
                                                          PAGE 0001
0001                ****************************************
0002                * This is an example which produces    *
0003                * correct values for $.                *
0004                ****************************************
0005  0000
0006                IDT   'TEST $'
0007  0000  0009    DATA  9
0008  0002  0008    DATA  8
0009  0004  0004'   DATA  $          CORRECT VALUE FOR $
0010  0006  0006'   DATA  $,8,9      CORRECT VALUE FOR $
      0008  0008
      000A  0009
0011  000C  000F'   DATA  $+3,7+1    CORRECT VALUE FOR $
      000E  0008
0012  0010  0008    DATA  7+1
0013  0012  0015'   DATA  $+3        CORRECT VALUE FOR $
0014  0014
0015                ****************************************
0016                * This is an example which produces    *
0017                * incorrect values for $               *
0018                ****************************************
0019  0014
0020  0014  0009    DATA  9,8,$      INCORRECT VALUE FOR $
      0016  0008
      0018  0014'
0021  001A  0008    DATA  7+1,$+3    INCORRECT VALUE FOR $
      001C  001D'
0022                END
NO ERRORS, NO WARNINGS
```

Syntax [<label>] DEF <symbol>[,<symbol>] [<comment>]

Fields

Label Optional; if used, the label assumes the current value of the Location Counter.

Operand Contains one or more symbols, separated by commas, to be defined in the program being assembled.

Comment Optional

Description DEF makes one or more symbols available to other programs for reference. All symbols used in the DEF statement must be defined in the same module.

Example

```
        DEF  ENTER,ANS
```

This example causes the assembler to include symbols ENTER and ANS in the object code; these symbols are available to other programs.

Syntax [<data>] DEND [<comment>]

Fields

Label Optional; if used, the label is assigned the value of the Location Counter before modification.

Operand Not used

Comment Optional

Description

DEND terminates a block of data-relocatable code and defines succeeding locations as program-relocatable. One of two values is placed in the Location Counter:

1) The maximum value attained by the Location Counter as a result of assembling the preceding block of program-relocatable code

2) Zero, if no program-relocatable code was previously assembled

If encountered in common-relocatable or program-relocatable code, DEND functions as a CEND or PEND, and the assembler issues a warning message. Like CEND and PEND, DEND is invalid in absolute code.

Syntax [<label>] DORG [<exp> [<comment>]]

Fields

Label Optional; when used, the label is assigned the same value that is placed in the Location Counter.

Operand Optional; when used, it contains an expression <exp> that can be either absolute or relocatable. Any symbol in the expression must have been previously defined.

When the operand field is absolute, the Location Counter is assigned the absolute value. When the operand is relocatable, the Location Counter is assigned the relocatable value and the same relocation type as the operand. When this occurs, space is reserved in the section that has that relocation type.

Comment Optional

Description

DORG loads the Location Counter with the beginning address of a dummy block or section. This address is specified by the operand. The assembler does not generate code for a dummy section, but operates normally in all other respects. The symbols that describe the dummy section layout are available when the remainder of the program is assembled.

Example 1

```
DORG  0
```

The assembler assigns values relative to the start of the dummy section to the labels within the dummy section. This example is appropriate for defining a data structure. The executable portion of the module (following the RORG directive) should use the labels of the dummy section as relative addresses. In this manner, the data is available to the procedure regardless of the memory area into which the data is loaded.

Example 2

```
RORG  0
  .
  .
  .     (code as desired)
  .
DORG  $
  .
  .
  .     (data segment)
  .
END
  .
```

This is appropriate for the executable portion (procedure division) of a procedure that is common to more than one task. The code corresponding to the dummy section must be assembled in another program module. In this manner, separate data portions (dummy sections) are available to the procedure portion.

The DORG directive may also be used with data-relocatable or common-relocatable operands to specify dummy data or common segments.

Example 3

```
        CSEG   'COM1'
        DORG   $      "$" has a common-relocatable
                       value
          .
          .
          .
          .
LAB1    DATA   $
MASK    DATA   >F000
          .
          .
          .
          .
        CEND
```

In this example, no object code is generated to initialize the common segment COM1, but space is reserved and all common-relocatable labels describing the structure of the common block (including LAB1 and MASK) are available for use throughout the program.

Syntax [<label>] DSEG [<comment>]

Fields

Label Optional; if used, the label is assigned the data-relocatable value placed in the Location Counter.

Operand Not used

Comment Optional

Description

DSEG begins a block of data-relocatable code at the address in the Location Counter. Data-relocatable blocks comprise the data segment of a program. The data segment can be relocated independently of the program segment at link-edit time. This separates modifiable data from executable code.

A data-relocatable block is normally terminated by a DEND directive. It can also be terminated by a PSEG, CSEG, AORG, or END directive. The PSEG and DEND directives identify succeeding locations as program-relocatable. The CSEG and AORG directives terminate the data segment by beginning a common or an absolute segment, respectively. The END directive terminates the data segment and the program.

The Location Counter is initially set to zero.

Example

```
RAM     DSEG            Start of data area
  .
  .
  .
  .
  <Data-relocatable code>
  .
  .
  .
  .
ERAM    DEND

LRAM    EQU    ERAM-RAM
```

The block of code between the DSEG and DEND directives is data-relocatable. RAM is the symbolic address of the first word of this block; ERAM is the data-relocatable byte address of the location following the code block. The value of the symbol LRAM is the length in bytes of the block.

Syntax [<label>] END [<symbol> [<comment>]]

Fields

Label Optional; if used, the label assumes the current value of the Location Counter.

Operand Optional; when used, the operand contains a program-relocatable or absolute symbol that specifies the program entry-point. If the operand is not used, no entry point is placed in the object code.

Comment Optional; may only be used with the operand field.

Description END terminates the assembly. It should be **the last source statement of a program**. Any source statements following the END directive are considered part of the next assembly.

Example

```
        END   START
```

This example terminates program assembly. The assembler also places the value of START in the object code as an entry point.

EQU Define Assembly-Time Constant Directive EQU

Syntax <label> EQU <exp> [<comment>]

Fields

Label A symbol that will be assigned the operand's value.

Operand An expression whose value is assigned to the label.

Comment Optional

Description EQU assigns a value to a symbol.

> **Note:**
>
> <exp> may not contain a REF'd symbol or forward references.

Example 1

```
SUM     EQU     R5
```

This example assigns an absolute value to the symbol SUM, making SUM available to use as a register address. A register should always be defined before it is used.

Example 2

```
TIME    EQU     HOURS
```

This example assigns the value of the previously defined symbol HOURS to the symbol TIME. When HOURS appears in the label field of a machine instruction in a relocatable block of the program, the value is a relocatable value. The two symbols may be used interchangeably. Symbols in the operand field must be previously defined.

Even Boundary Directive

Syntax [<label>] EVEN [<comment>]

Fields

Label Optional; if used, the label is assigned the value in the Location Counter after the directive is processed.

Operand Not used

Comment Optional

Description EVEN places the Location Counter on the next word boundary (even byte address). When the Location Counter is already on an even boundary, the Location Counter is not altered.

Example

```
WRF1    EVEN
```

Assures that the Location Counter contains an even boundary address and assigns the Location Counter address to label WRF1.

Syntax [<label>] IDT '<string>' [<comment>]

Fields

Label Optional; if used, the label is assigned the current value of the Location Counter.

Operand Contains the module name <string>, a character string of up to eight characters enclosed in single quotes. The assembler truncates strings that are longer than eight characters and prints a truncation error message.

Comment Optional

Description IDT assigns a name to the object module produced.

Example

```
IDT  'CONVERT'
```

This example assigns the name CONVERT to the module being assembled. The module name is printed in the source listing as the operand of the IDT directive and appears in the page heading of the source listing. The module name is also placed in the object code and is used by the link editor for automatic entry-point resolution. A routine whose entry point is to be automatically resolved by the link editor must be declared as the 'string' on the IDT statement for that module. The entry point must also be REF'd in this case.

> **Note:**
>
> Although the Assembler accepts lowercase letters and special characters within the quotes, ROM loaders (for example) will not. Therefore, only uppercase letters and numerals are recommended.

Restart Source Listing Directive

Syntax [<label>] LIST [<comment>]

Fields

Label Optional; if used, the label assumes the current value of the Location Counter.

Operand Not used

Comment Optional; if used, the assembler does not print the comment.

Description LIST restores printing of the source listing after it was cancelled by a UNL directive. This directive is not printed in the source listing, but the line counter increments.

Syntax [<label>] LOAD <symbol>[,<symbol>] [<comment>]

Fields

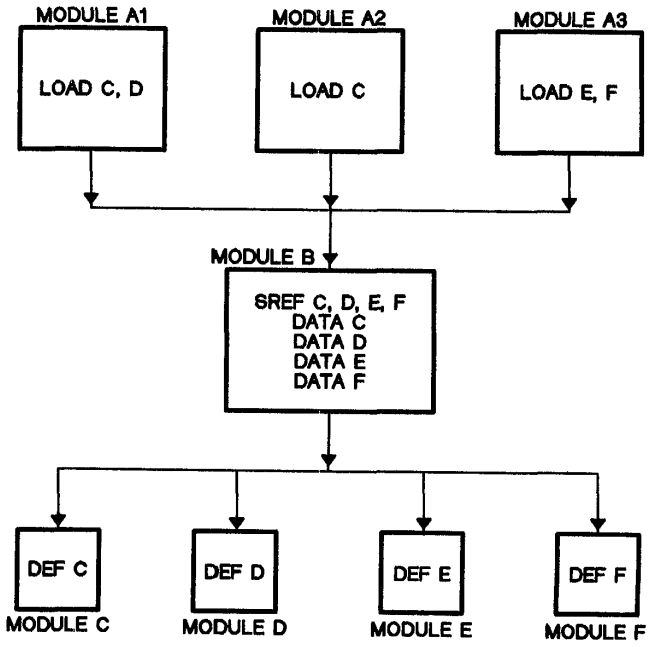
Label Optional

Operand Contains one or more symbols, separated by commas, to be used in the operand field of a subsequent source statement.

Comment Optional

Description The LOAD directive is like a REF, but the symbol does not need to be used in the module containing the LOAD. The symbol used in the LOAD must be defined in some other module. LOADs are used with SREFs. If one-to-one matching of LOAD and DEF symbols does not occur, then unresolved references will occur during link editing.

Example



- Module A1 uses a branch table in module B to obtain one module C, D, E, or F.
- Module A1 knows which of module C, D, E, and F it requires.
- Module B has an SREF for C, D, E, and F.
- Module C has a DEF for C.
- Module D has a DEF for D.
- Module E has a DEF for E.
- Module F has a DEF for F.
- Module A1 has a LOAD for the modules C and D it needs.
- Module A2 has a LOAD for the module C it needs.
- Module A3 has a LOAD for the modules E and F it needs.

The LOAD and SREF directives permit module B to be written to handle a highly involved case and still be linked together without unnecessary modules since A1 only has LOAD directives for the modules it needs.

When a link edit is performed, automatic symbol resolutions will pull in the modules appearing in the LOAD directives.

If the link control file included A1 and A2, modules C and D would be pulled in while modules E and F would not be pulled in. If the link control file included A3, modules E and F would be pulled in while modules C and D would not be pulled in. If the link control file included A2, module C would be pulled in while modules D, E, and F would not be pulled in.

Syntax [<label>] MLIB '<pathname>' [<comment>]

Fields

Label Optional; if used, the label assumes the current value of the Location Counter.

Operand Contains the pathname, a character string of up to 48 characters enclosed in single quotes. Longer strings produce truncation error messages.

Comment Optional

Description

The MLIB directive provides the assembler with the name of a library containing macro definitions. The operand is a directory pathname (constructed according to the host operating system conventions) enclosed in single quotes (see IDT and TITL directives). This directive is defined only for hosts that support libraries on hard disks.

> **Note:**
>
> Neither the assembler nor its runtime support have access to the operating system's synonym table, and so cannot expand pathnames. The use of synonyms prevents finding any macros in that library.

Example 1

```
MLIB 'MYVOLUME.MACDIR.CMPXMACS.NEWMACS'
MLIB 'USER32.BIGPROJ.MYTASK.MACROS'
```

This example causes the macro function, when the program finds a macro call SUBMAC (not previously defined), to search first for a file named `USER32.BIGPROJ.MYTASK.MACROS.SUBMAC`, and then if that file isn't found, to search for a file named `MYVOLUME.MACDIR.CMPXMACS.NEWMACS.SUBMAC`, in that order.

On a VAX/VMS system, a pathname would be specified as follows:

```
MLIB 'DRCO:[MOORE.ASM32]'
```

Example 2

The following program segment illustrates macro library use for an MS/PC-DOS system.

```
MLIB 'E:'   Pathname must be a drive name
  .         Typical assembly code
  .
XMAC        First macro call
  .
YMAC        Another macro call
  .
END
```

The assembler will search the drive specified by the MLIB directive for a file with the same name as the macro. The macro name cannot have an extension. Only one macro is allowed per file.

Syntax OPTION <option-list>

Fields

Label Not used

Operand <option-list> (see preceding Description)

Comment Not used

Description OPTION selects several options for the assembler listing output. The <option-list> operand is a list of keywords separated by commas. Each keyword selects one of the following listing features:

BUNLST:	Limit the listing of BYTE directives to one line
DUNLST:	Limit the listing of DATA directives to one line
TUNLST:	Limit the listing of TEXT directives to one line
FUNLST:	Turn off all unlist options
XREF:	Produce a symbol cross-reference listing
NOLIST:	Inhibit all listing output (this overrides the LIST directive)
SYMLST:	Produce a symbol listing in the object file, no symbols are put in the listing file

Syntax [<page>] PAGE [<comment>]

Fields

Label Optional; if used, the label assumes the current value of the Location Counter.

Operand Not used

Comment Optional; if used, the assembler does not print the comment.

Description PAGE prints the source program listing on a new page. The PAGE directive is not printed in the source listing, but the line counter increments.

Example

```
PAGE
```

The assembler begins a new page of the source listing. The next source statement is the first statement listed on the new page. Using the PAGE directive to separate source listing into logical divisions improves program documentation.

Syntax [<label>] PEND [<comment>]

Fields

Label Optional; if used, the label is assigned the value of the Location Counter before modification.

Operand Not used

Comment Optional

Description The PEND directive is the program-segment counterpart of the DEND and CEND directives. It begins a section of program-relocatable code at the address in the Location Counter. The value placed in the Location Counter is the maximum value it attained by assembling all preceding program-relocatable code. It is invalid when used in absolute code.

Syntax [<label>] PSEG [<comment>]

Fields

Label Optional; if used, the label is assigned the value placed in the Location Counter.

Operand Optional

Comment Optional

Description PSEG begins a program-relocatable segment at the address in the Location Counter. The Location Counter is set to one of the following values:

- The maximum value the Location Counter has attained by assembling any preceding block of program-relocatable code.
- Zero, if no program-relocatable code was previously assembled.

The PSEG directive is the program-segment counterpart of the DSEG and CSEG directives. Together, the three directives provide a consistent method of defining the various types of relocatable segments. The following sequences of directives are functionally equivalent.

SEQUENCE 1

```
DSEG
 .
 .
 .
<Data-relocatable code>
 .
 .
 .
DEND
CSEG
 .
 .
 .
<Common-relocatable code>
 .
 .
 .
CEND
PSEG
 .
 .
 .
<Program-relocatable code>
 .
 .
PEND
 .
END
```

SEQUENCE 2

```
DSEG
 .
 .
 .
<Data-relocatable code>
 .
 .
 .
 .
CSEG
 .
 .
 .
<Common-relocatable code>
 .
 .
 .
 .
PSEG
 .
 .
 .
<Program-relocatable code>
 .
 .
 .
 .
END
```

Syntax [<label>] REF <symbol>[,<symbol>] [<comment>]

Fields

Label Optional; if used, the label assumes the current value of the Location Counter.

Operand Contains one or more symbols, separated by commas, to be used in the operand field of a subsequent source statement.

Comment Optional

Description REF provides access to one or more symbols defined in other programs. If a symbol is listed in the REF statement, then a corresponding symbol must also be present in a DEF statement in another source module. If the symbol is not defined in another module, then an error occurs at link edit time. The system generates a summary list of all unresolved references.

Example

```
REF  ARG1,ARG2
```

This example causes the assembler to include symbols ARG1 and ARG2 in the object code so that the corresponding addresses may be obtained from other programs.

Syntax [<label>] RORG [<exp> [<comment>]]

Fields

Label Optional; if used, the label is assigned the same value that is placed in the Location Counter.

Operand Optional; when used, the operand must be a relocatable expression (<exp>). It can only contain previously defined symbols.

Comment Optional; may only be used with the operand field.

Description

RORG places a value in the Location Counter. If encountered in absolute code, RORG also defines succeeding locations as program-relocatable. The operand usually specifies the value placed in the Location Counter. If the operand is not used, the Location Counter is replaced by:

- The *current maximum length of the program segment* of the program, if RORG appears in absolute or program-relocatable code.
- The *maximum length of the data segment* if RORG appears in data-relocatable code.
- The *maximum length of the common segment* if RORG appears in common-relocatable code.

The length of the program-, data-, or common-relocatable segment, at any time during assembly, is determined by either of the following:

1) The maximum value the Location Counter has ever attained as a result of the assembly of any preceding block of relocatable code.
2) Zero, if no relocatable code has been previously assembled.

Since the Location Counter begins at zero, the length of a segment and the next available address within that segment are identical.

If RORG appears in absolute code, a relocatable operand must be program-relocatable. In relocatable code, the operand's relocation type (data, common, or program) must match that of the current location counter.

In absolute code RORG places the operand value in the Location Counter and changes the Location Counter's relocation type to program-relocatable. In relocatable code RORG places the operand value in the Location Counter but does not change the Location Counter's relocation type.

Example 1

```
        RORG  $-10      Overlay ten bytes
```

The $ symbol contains the value of the current location. This example sets the Location Counter to the current location less ten bytes. The instructions and directives following the RORG directive replace the ten previously assembled words of relocatable code, permitting correction of the program without removing source records. If a label had been included, the label would have been assigned the value placed in the Location Counter.

Example 2

```
SEG2   RORG
```

The Location Counter contents depend upon preceding source statements. Assume that after defining data for a program that occupies >44 bytes, an AORG directive initiates an absolute block of code. The absolute block is followed by the RORG directive from the preceding example. This places >0044 in the Location Counter and defines the Location Counter as relocatable. Symbol SEG2 is a relocatable value, >0044. The RORG directive from the above example would have no effect except at the end of an absolute block or a dummy block.

SREF Secondary External Reference Directive SREF

Syntax [<label>] SREF <symbol>[,<symbol>] [<comment>]

Fields

Label Optional; if used, the label assumes the current value of the Location Counter.

Operand Contains one or more symbols, separated by commas, to be used in the operand field of a subsequent source statement.

Comment Optional

Description SREF provides access to one or more symbols defined in other programs. Unlike REF, SREF does not require a symbol to have a corresponding symbol listed in a DEF statement of another source module. The SREF'd symbol will be an unresolved reference but no error message will be produced.

Example

```
SREF    ARG1,ARG2
```

This example causes the link editor to include symbols ARG1 and ARG2 in the object code so that the corresponding addresses may be obtained from other programs.

Syntax [<label>] TEXT [-]'<string>' [<comment>]

Fields

Label Optional; if used, the label is assigned the location where the assembler places the first character.

Operand Contains a character string of up to 52 characters enclosed in single quotes; it may be preceded by a unary minus sign.

Comment Optional

Description TEXT places one or more characters in successive bytes of memory. The assembler negates the last character of the string when the string is preceded by a minus (-) sign (unary minus).

Example 1

```
MSG1   TEXT  'EXAMPLE'  Message heading
```

This example places the 8-bit ASCII representations of the characters in successive bytes. When the Location Counter is on an even address, the result is >4558, >414D, >504C, and >45xx. >xx, the contents of the rightmost byte of the fourth word, are determined by the next source statement. The label MSG1 is assigned the value of the first byte address, containing >45.

Example 2

```
MSG2   TEXT  -'NUMBER'
```

When the Location Counter is on an even address, the result is >4E55, >4D42, and >45AE. The label MSG2 is assigned the value of the byte address in which >4E is placed.

Syntax [<label>] TITL '<string>' [<comment>]

Fields

Label Optional; if used, the label assumes the current value of the Location Counter.

Operand Contains the title (<string>), a character string of up to 50 characters enclosed in single quotes. The assembler truncates a string longer than 50 characters and prints a truncation error message.

Comment Optional; the assembler does not print the comment but does increment the line counter.

Description TITL supplies a title to be printed in the heading of each page of the source listing. The title is printed on the next page after TITL is processed, and on subsequent pages until another TITL directive is processed. The TITL directive must be the first source statement submitted to the assembler if a title heading is desired on the listing's first page. This directive is not printed in the source listing.

Example

```
TITL  '**REPORT GENERATOR**'
```

This example prints the title **REPORT GENERATOR** in the page headings of the source listing.

Stop Source Listing Directive

Syntax [<label>] UNL [<comment>]

Fields

Label Optional; if used, the label assumes the value of the Location Counter.

Operand Not used

Comment Optional; if used, the assembler does not print the comment.

Description UNL halts the source listing output until a LIST directive is processed. It is not printed in the source listing, but the source line counter is incremented. This directive is frequently used in MACRO definitions to inhibit the listing of the macro expansion. It is useful for reducing assembly time and the size of the source listing.

5.6 Symbolic Addressing Techniques

The assembler processes symbolic memory addresses for addressing registers.

The following example illustrates this type of coding:

```
SUM         EQU     R33             Assign SUM for
                                    register 33
QUAN        EQU     R34             Assign QUAN for
                                    register 34

            ADD     QUAN,SUM        Add QUAN to SUM
*                                   Store in SUM
              .
              .
```

The two initial EQU directives assign meaningful labels to be used as register addresses in the subroutine.

5.7 Assembler Output

This section discusses assembler output, including source listings, error messages, a cross reference listing, and object code.

5.7.1 Source Listing

A source listing shows source statements and the object code they produce.

Each page of the source listing has a title line at the top. Any title supplied by a TITL directive is printed on this line. A page number is printed to the right of the title. A blank line follows the title line; subsequent lines contain the assembled source statements. Each assembled source statement contains a source statement number, a program counter value, the object code assembled, and the source statement as entered. If a source statement produces more than one byte of object code, the assembler prints the program counter value and object code on a separate line for each additional byte. Each added line is printed following the source statement line.

```
1       2    3 4          5

0018    F156  42          MOV   R10,R5
        F157  0A
        F158  05
```

The source statement number, 0018 in the example, is a 4-digit decimal number. Source records are numbered in the order in which they are entered including those source records that are not printed in the listing (TITL, LIST, UNL, and PAGE directives are not listed; source records between a UNL directive and a LIST directive are not listed). The difference between two consecutive source record numbers indicates if a source record was entered but not listed.

The next field on a line of the listing contains the program counter value (hexadecimal). In the example, F156 is the program counter value. Not all directives affect the program counter; the field is blank for those directives that do not affect it (the IDT, REF, DEF, EQU, SREF, and END directives leave the program counter field blank).

The third field normally contains a single blank. However, the assembler places a dash in this field when warning errors are detected.

The fourth field contains the hexadecimal representation of the object code, 420A05 in the preceding example. Note that the assembler produces a line containing the program counter value and the assembled object code for each byte of object code. All machine instructions and the BYTE, DATA, and TEXT directives use this field for object code. The EQU directive places the value corresponding to the label in the object code field.

The fifth field contains the characters of the source statement as they were scanned by the assembler. Spacing in this field is determined by the spacing in the source statement. The four source statement fields will be aligned in the listing only when they are aligned in the source statements or when tab characters are used.

5.7.2 Normal Completion Error Messages

The assembler issues two types of error messages: normal completion messages and abnormal completion messages (Section 5.7.3). When the assembler completes an assembly, it indicates any errors it encounters in the assembly listing. The assembler indicates errors following the source line in which they occur. At the end of a module (IDT-END pair), the corresponding messages are printed.

Table 5-3 lists error, warning, and information messages.

Table 5-3. Assembly Listing Errors

NONFATAL ERRORS	
MESSAGE	**EXPLANATION/RESPONSE**

WARNING - 'CEND' ASSUMED
WARNING - 'DEND' ASSUMED
WARNING - 'PEND' ASSUMED
WARNING - 'DSEG' ASSUMED
	This is a warning that the following two statements will produce:
	`CSEG 'DATA'`
	`DSEG`

WARNING - SYMBOL TRUNCATED	The maximum length for a symbol is six characters.
WARNING - STRING TRUNCATED	Check the syntax for the directive in question to determine the maximum length for the string.
WARNING - TRAILING OPERAND(S)	
WARNING - BYTE VALUE TRUNCATED	A value that is to be used as a byte value was larger than can be loaded into a byte.
**LAST WARNING	
FATAL ERRORS	
MESSAGE	**EXPLANATION/RESPONSE**
ABSOLUTE VALUE REQUIRED DISPLACEMENT TOO BIG	An instruction with an operand with a fixed upper limit was encountered that overflowed this limit.
INVALID EXPRESSION	This may indicate invalid use of a relocatable symbol in arithmetic.
EXPRESSION OUT OF BOUNDS	There is a range limit for the value being used that was exceeded.
DUPLICATE DEFINITION	The symbol appears as an operand of a REF statement, as well as in the label field of the source, OR, the symbol appears more than once in the label field of the source.
INVALID RELOCATION TYPE	The type of variable isn't relocatable.
INVALID OPCODE	The second field of the source record contained an entry that is not a defined instruction, directive, pseudo-op, DXOP, DFOP, or macro name.
INVALID OPTION	The option given in the OPTION directive are invalid.
INVALID REGISTER VALUE	The given register value is too large or too small.

Table 5-3. Assembly Listing Errors (Concluded)

FATAL ERRORS (CONTINUED)	
MESSAGE	**EXPLANATION/RESPONSE**
INVALID SYMBOL	The symbol being used has invalid characters in it.
VALUE TRUNCATED	The value used was too big for the field, so it has been truncated.
SYMBOL USED IN BOTH REF AND DEF	Symbol cannot be both referenced and defined in the same module.
COPY FILE OPEN ERROR	File does not exist or is already being used.
EXPRESSION SYNTAX ERROR	Unbalanced parentheses OR invalid operations on relocatable symbols.
INVALID ABSOLUTE CODE DIRECTIVE	The directive PEND, DEND and CEND have no meaning in absolute code.
LABEL REQUIRED BLANK MISSING	A blank is needed but one was not found. (Usually the blank is required in column 1.)
COMMA MISSING	Expected a comma but did not find one. Usually means that more operands were expected.
COPY FILENAME MISSING INDIRECT (*) MISSING	The indirect addressing (*) was needed.
SYMBOL REQUIRED OPERAND MISSING	There was no operand field.
REGISTER REQUIRED	A register should be used rather than a label or an absolute number.
CLOSE (') MISSING STRING REQUIRED	TEXT directive used with no text following.
PASS1/PASS2 OPERAND CONFLICT	The symbols in the symbol table did not have the same value in PASS1 and PASS2. Registers and peripheral files should be defined before they are used in an instruction. This error is also produced when the BSS directive is used to define a register name; use EQU instead.
SYNTAX ERROR	
UNDEFINED SYMBOL	The symbol being used has not been REF'ed or it has been DEF'ed but not used.
DIVIDE BY ZERO ILLEGAL SHIFT COUNT CANNOT INDEX BY REGISTER ZERO	The shift count being asked for is not valid.
INFORMATION MESSAGES	
MESSAGE	**EXPLANATION/RESPONSE**
OPCODES REDEFINED	As a result of an MLIB directive, one or more assembler opcodes has been redefined by a MACRO within a MACRO directory. You should take action if this is not intended.
MACROS REDEFINED	As a result of an MLIB directive, one or more currently defined macros has been redefined by a MACRO (of the same name) with a MACRO directory. You should take action if this is not intended.

5.7.3 Abnormal Completion Error Messages

Most abnormal completion error messages are issued by the operating system under which the assembly runs (messages in this category include those concerned with file I/O errors). Refer to the applicable operating system reference manual for detailed information. Table 5-4 lists the abnormal error messages.

Table 5-4. Abnormal Completion Error Messages

UNEXPECTED END OF PARSE
ERROR MAPPING PARSE - ASSEMBLER BUG
INVALID OPERATION ENCOUNTERED
NO OPCODE
INVALID LISTING ERROR ENCOUNTERED
SYMBOL TABLE ERROR
INVALID LIB COMMAND ID
UNKNOWN ERROR PASSED, CODE = XXXX

5.7.4 Cross-Reference Listing

The assembler prints an optional cross-reference listing following the source listing, as specified by the assembler OPTION directive. The format of the listing is shown in Figure 5-2.

```
LABEL      VALUE     DEFN      REFERENCES
-----      -----     ----      ----------

ADDT       01A8      0325      0314

ADSR  D    01A0      0316      0342  0343  0348  0349

GT         0006      0997

OBTCHN R             0088

SQUIB U                        0127  0233
```

Figure 5-2. Cross-Reference Listing Format

As Figure 5-2 shows,

- The assembler prints each symbol defined or referenced in the assembly in the label column. If a single character follows the symbol, it represents the symbol attribute. These symbol-attribute characters and their meanings are listed in Table 5-5.
- The second (value) column contains a four-digit hexadecimal number, the value assigned to the symbol. The number of the statement that defines the symbol appears in
- the third (definition) column. This column is left blank for undefined symbols.
- The fourth (reference) column lists the source statement numbers that reference the symbol. A blank in this column indicates that the symbol was never used.

Table 5-5. Symbol Attributes

CHARACTER	MEANING
R	External reference (REF)
D	External definition (DEF)
U	Undefined
M	Macro name
S	Secondary reference (SREF)
L	Force load (LOAD)

5.8 Object Code

The assembler produces object code that may be linked to other code modules or programs, and loaded directly into the computer. Object code consists of records containing up to 71 ASCII characters. You can correct record data manually for simple temporary changes for debugging. This prevents a lengthy re-assembly but it causes problems if you don't update the source. Figure 5-3 shows an example of object code.

SAMPLE 1 - ACTUAL CODE OUTPUT

```
K0000TESTPROG9F006B327BBB5ABB0002BCAFBB5246B0DA2B0000BA242B02A27F113F
B2003BA2FFB09A2BFF0BBA222B0AA2B4408B5208BD502BA2F0B0BCFBE32EB78047F0F6F
B0292B0A80B0AA2B000AB230FBE2EFBD202BBDE7BFB4DB0203BE206B4202B03727F127F
B1004B7D02B04E7B03DAB04D0BE205BD204B7401B050AB72FFB030A9F862*0B7F1D5F
9FFF4BF862BF862BF862BF862BF862BF862BF0067F7A4F
:       TESTPROG  11/28/84  15:59: 3     ASMMLP  2.1 83.074
```

SAMPLE 2 - EXPANDED CODE WITH KEYS (REFERENCE ONLY)

```
K0000TESTPROG9F006B327BBB5ABB0002BCAFBB5246B0DA2B0000BA242B02A27F113F
1 2     3      4 5  6 7                                              8 9  10
*88BF800B71EFBF788BF822B71EF*88BF848B71EF*88BF871B71EFBF788BF89781111F
11                                                                   12 13
BFFACB9C019FFFEBF8C47FAD1F
          14 15
:       TESTPROG  11/28/84  15:59: 3     ASMMLP  2.1 83.074
16
```

1) K – Begins each program
2) 0000 – Bytes of relocable code, always 0 for final linked code
3) TESTPROG – Name from the IDT statement of the program
4) 9 – Address follows
5) F006 – Beginning address
6) B – 16-bit word follows
7) 327B – 16-bit word, MSB first
8) 7 – Checksum follows
9) F113 – Checksum (2's complement of the sum of all ASCII characters prior to and including the 7 tag)
10) F – End of line
11) * – 8-bit byte to follow
12) 8 – Ignore checksum - useful when object code patching
13) 1111 – Any 4 numbers can follow an 8 tag
14) 9 – Address follows
15) FFFE – Address of vector area
16) : – Last line of object module

Note: Table 5-6 provides an explanation of the tag characters.

Figure 5-3. Sample Object Code

5.8.1 Object Code Format

Formatted object code contains records made up of fields sandwiched between tag characters. The specific tag character, defined by the assembler or linker, specifies the function of the fields with which it is associated. A tag character occupies the first position on each line of object code and identifies the fields it precedes to the loader. Table 5-6 details the various tag characters and their associated fields. Table 5-7 lists field and tag character information.

Table 5-6. Tag Characters

TAG CHARACTER	DESCRIPTION
K	Placed at the beginning of each program; followed by two fields. **Fields** - Field one contains the number of bytes of program relocatable code. - Field two contains the program identifier assigned to the program by an IDT directive. When no IDT directive is entered, field two is blank. The linker uses the program identifier to identify the program, and the number of bytes of program-relocatable code to determine the load bias for the next module or program.
M	Used when data or common segments are defined in the program; followed by three fields. **Fields** - Field one contains the length, in bytes, of data- or common-relocatable code. - Field two contains the data or common segment identifier, and field three contains a "common number." The identifier is a six-character field containing the name $DATA (padded on the right by one blank) for data segments and $BLANK for blank common segments. If a named common segment appears in the program, an M tag will appear in the object code with an identifier field corresponding to the operand in the defining CSEG directive(s). - Field three consists of a four-character hexadecimal number defining a unique common number to be used by other tags that reference or initialize data of that particular segment. For data segments, this common number is always zero. For common segments (including blank common), the common numbers are assigned in increasing order, beginning at one and ending with the number of different common segments. The maximum number of common segments that a program may contain is 127.
1,2	Used with entry addresses. **Fields** - The associated field is used by the linker to determine the entry point in which execution starts when linking is complete. Tag character 1 is used when the entry address is absolute; tag character 2 when the address is relocatable. The field lists the address in hexadecimal form.
3,4,X	Tag characters 3, 4, and X are used for external references. Tag character 3 is used when the last appearance of the externally referenced symbol is in program-relocatable code; tag character 4 when it is in absolute code; and the X tag when it is in data- or common-relocatable code. Tag characters 3 and 4 are associated with two fields. Tag character X may identify one additional field. **Fields** - Field one contains the location of the last appearance of the symbol. - Field two contains the symbol itself. - Field three is only used to supply the common number for the X tag.

Table 5-6. Tag Characters (Continued)

TAG CHARACTER	DESCRIPTION
E	Used for external references. An E tag is used when a nonzero quantity is to be added to a reference. **Fields** - Field 1 identifies the reference by occurrence in the object code (0, 1, 2, ...). In other words, the value in field one is an index into references identified by 3, 4, V, X, Y and Z tags in the object code. The list is maintained by order of occurrence (i.e., the first entry in the list is the symbol located in field two of the first 3, 4, V, X, Y, or Z tag). - Field 2 contains the value to be added to the reference after the reference is resolved.
@	Used for external references of an 8-bit value. It serves the same purpose for 8-bit values that the E-tag serves for 16-bit values.
5, 6, W	Used for external definitions. Tag character 5 is used when the location is program-relocatable. Tag character 6 is used when the location is absolute. Tag character W is used when the location is data- or common-relocatable. The fields are used by the linker to provide the desired linking to the external definition. **Fields** - Field one contains the location of the last appearance of the symbol. - Field two contains the symbol of the external definition. - Field three of tag character W contains the common number.
7	Precedes the checksum, and is placed at the end of the set of fields in the record. The checksum is an error detection word and is formed as the record is being written. It is the two's complement of the sum of the 8-bit ASCII values of the characters of the record from the first tag of the record through the checksum tag, 7.
9, A, S, P	Used with load addresses, required for data words that are to be placed at other than the next immediate memory addresses. Tag character 9 is used when the load address is absolute. Tag character A is used when the load address is program-relocatable. Tag character S is used when the load address is data-relocatable. Tag character P is used when the load address is common-relocatable. **Fields** - Field one contains the load address. - Field two is only present for tag character P and contains the common number.
*, B, C, T, N	Used with data words. Tag characters * and B are used when the data is absolute (i.e., an instruction word or a word that contains text characters or absolute constants). Tag * is used for absolute byte data (8 bits) and B is used for absolute word data (16 bits). Tag character C is used for a word that contains a program-relocatable address. Tag character T is used for a word that contains a data-relocatable address. Tag character N is used for a word that contains a common-relocatable address. **Fields** - Field one contains the data word. The linker places the data word in the memory location specified in the preceding load address field or in the memory location that follows the preceding data word. - Field two is only used with N and contains the common number.
G, H, J	Used when the symbol table option is specified. Tag character G is used when the location or value of the symbol is program-relocatable, tag character H is used when the location or value of the symbol is absolute, and tag character J is used when the location or value of the symbol is data- or common-relocatable. **Fields** - Field one contains the location or value of the symbol. - Field two contains the symbol to which the location is assigned. - Field three is used with tag character J only and contains the common number.

Table 5-6. Tag Characters (Concluded)

TAG CHARACTER	DESCRIPTION
U	Generated by the LOAD directive. The symbol specified is treated as if it were the value specified in an INCLUDE command to the linker. **Fields** - Field one contains zeros. - Field two contains the symbol for which the loader will search for a definition.
V, Y, Z	Used for secondary external references. Tag character V is used when the last appearance of the externally referenced symbol is in program-relocatable code; tag character Y when it is in absolute code; and the Z tag when it is in data- or common-relocatable code. Tag characters V and Y are associated with two fields. Tag character Z may identify one additional field. **Fields** - Field one contains the location of the last appearance of the symbol. - Field two contains the symbol itself. - Field three is only used to supply the common number for the Z tag.
8	Also associated with the checksum field, but used when the checksum field is to be ignored.
D	Specifies a load bias. Its lone associated field contains the absolute address that will be used by a loader to relocate object code. The Link Editor does not accept the D tag.
F	Placed at the end of the record. It may be followed by blanks.

The end of each record is identified by the tag character 7 followed by the checksum field and the tag character F (this data is described above). The assembler fills the rest of the record with blanks and a sequence number and begins a new record with the appropriate tag character.

The last record of an object module has a colon (:) in the first character position of the record, followed by blanks or time and date identifying data.

Table 5-7 defines the object record format and tags.

Table 5-7. Object Record Format and Tags

TAG	1ST FIELD	2ND FIELD	3RD FIELD
	MODULE DEFINITION		
K	PSEG Length	Program ID (8)	
M	DSEG Length	$DATA	0000
M	Blank Common Length	$BLANK	Common #
M	CSEG Length	Common Name (6)	Common #
	ENTRY POINT DEFINITION		
1	Absolute Address		
2	P-R Address		
	LOAD ADDRESS		
9	ABSOLUTE ADDRESS		
A	P-R Address		
S	D-R Address		
P	C-R Address	Common or CBSEG #	
	DATA		
*	Absolute 8-bit Value (2)		
B	Absolute 16-bit Value		
C	P-R Address		
T	D-R Address		
N	C-R Address	Common or CBSEG #	
	EXTERNAL DEFINITIONS		
6	Absolute Value	Symbol (6)	
5	P-R Address	Symbol (6)	
W	D-R/C-R Address	Symbol (6)	Common #
	EXTERNAL REFERENCES		
3	P-R Address of Chain	Symbol (6)	
4	Absolute Address of Chain	Symbol (6)	
X	D-R/C-R Address of Chain	Symbol (6)	Common *
E	Symbol Index Number	Absolute Offset	
@	Symbol Index Number	Offset (2)	Mask (2)
	SYMBOL DEFINITIONS		
G	P-R Address	Symbol (6)	
H	Absolute Value	Symbol (6)	
J	D-R/C-R Address	Symbol (6)	Common #
	FORCE EXTERNAL LINK		
U	0000	Symbol (6)	
	SECONDARY EXTERNAL REFERENCE		
V	P-R Address of Chain Entry	Symbol (6)	
Y	Absolute Address of Chain	Symbol (6)	
Z	D-R/C-R Address of Chain	Symbol (6)	Common #
	CHECK SUM		
7	Value		
	IGNORE CHECK SUM		
8	Any Value		
	LOAD BIAS		
D	Absolute Address		
	END OF RECORD		
F			
	END OF OBJECT MODULE		
:			

Notes: 1. All field widths are four characters unless otherwise specified by numbers in parenthesis.
2. If the first tag is 01 (hex), the file is in compressed object format.
3. P-R Program segment relative (address)
D-R Data segment relative (address)
C-R Common segment relative (address)

5.8.1.1 External References in Object Code

The Link Editor allows the use of external references in the object code. (See Section 7.)

5.8.1.2 Changing Object Code

In most cases, changing the object code is not the recommended way to correct errors in a program. All changes or corrections to a program should be made in the source code, then the program should be re-assembled. Failure to follow this procedure can make subsequent program correction or maintenance impossible. The information in the following paragraphs is intended for those rare instances when re-assembly is not possible. Any changes made directly to the object code should be thoroughly documented so that the programmers who come later can see what the program actually does, not what the source code says that it does.

To correct the object code without re-assembling a program, change the object code by changing or adding one or more records. One additional tag character is recognized by the loader to permit specifying an abolute address that will be used to relocate object code. The additional tag character, D, may be used in object records changed or added manually.

Tag character D is followed by a load bias (offset) value. The loader uses this value instead of the load bias computed by the loader itself. The loader adds the load bias to all relocatable entry addresses, external references, external definitions, load addresses, and data. The effect of the D tag character is to specify that area of memory into which the loader loads the program. The tag character D and the associated field must be placed ahead of the object code generated by the assembler.

Correcting the object code may require only changing a character or a word in an object code record. You may duplicate the record up to the character or word in error, replace the incorrect data with the correct data, and duplicate the remainder of the record up to the seven tag character. The changes will cause a checksum error when the checksum is verified as the record is loaded, so you must:

- Change the 7 tag character to an 8 tag character, in which case the checksum value is ignored,
  or
- Recalculate the checksum.

When more extensive changes are required, you may write an additional object code record or records. Begin each record with a tag character 9, A, S, or P, followed by an absolute load address or a relocatable load address. This may be an address into which an existing object code record places a different value. The new value on the new record will override the other value when the new record follows the other record in the loading sequence. Follow the load address with a tag character *, B, C, T, or N and an absolute data word or a relocatable data word. Additional data words preceded by appropriate tag characters may follow. When additional data is to be placed at a nonsequential address, write another load address tag character followed by the load address and data words preceded by tag characters. When the record is full, or all changes have been written, write tag character F to end the record.

When additional relocatable memory locations are loaded as a result of changes, you must change field one of tag character K, which contains the number of bytes of relocatable code. For example, if the object field written by the assembler contained 1000 hex bytes of relocatable code and you have added eight bytes in a new object record, additional memory locations will be loaded. You must find the K tag character in the object code file and change the value following the tag character from 1000 to 1008; you must also change the tag character 7 to 8 in that record, or recalculate the checksum.

When added records place corrected data in locations previously loaded, the added records must follow the incorrect records. The loader processes the records as they are read from the object file, and the last record that affects a given memory location determines the contents of that location at execution time.

The object code records that contain the external definition fields, the external reference fields, the entry address field, and the final program start field must follow all other object records. An additional field or record may be added to include reference to a program identifier. The tag character is 4, and the hexadecimal field contains zeros. The second field contains the first six characters of the IDT character string. External definitions may be added using tag character 5 or 6 followed by the relocatable or absolute address, respectively. The second field contains the defined symbol, filled to the right with blanks when the symbol contains less than six characters.

Note:

Both object code to be linked and object code to be downloaded can be changed without re-assembling the program. The link editor, though, will not accept tag character D in changed or added object records.

6. Assembly Language Instruction Set

The TMS7000 instruction set contains 61 instructions that control input, output, data manipulation, data comparison, and program flow. The instruction set can be divided into eight functional categories:

- Arithmetic Instructions
- Branch and Jump Instructions
- Compare Instructions
- Control Instructions
- Load and Move Instructions
- Logical Instructions
- Shift Instructions
- I/O Instructions

Note:

TMS70x2 and **TMS70Cx2** devices have 256 bytes of on-chip RAM; their register locations range from R0–R255. **TMS70x0** and **TMS70Cx0** devices have 128 bytes of on-chip RAM; their register locations range from R0–R127.

Topics in this section include:

6.1 Definitions

Table 6-1 lists and defines the symbols used in the instruction set.

Table 6-1. TMS7000 Symbol Definitions

SYMBOL	DEFINITION	SYMBOL	DEFINITION
A	Register A or R0 in Register File	B	Register B or R1 in Register File
Rn	Register n of Register File	Pn	Port n of Peripheral File ($0 \leq n \leq 255$)
s	Source operand	d	Destination operand
Rs	Source register in Register File	Ps	Source register in Peripheral File ($0 \leq s \leq 255$)
Rd	Destination register in Register File	Pd	Destination in Peripheral File ($0 \leq d \leq 255$)
Rp	Register pair	iop	Immediate operand
ST	Status Register	SP	Stack Pointer
PC	Program Counter	pcn	Location of the next instruction
$	Current value of Program Counter	b	Bit number, as in b7 ($0 \leq b \leq 7$)
offset	Relative Address (offset = ta - pcn)	ta	Target Address (ta = offset + pcn)
@	Indicates an address or label	%	Indicates immediate operand
*	Indicates Indirect Register File Addressing mode	<XADDR>	Indicates an extended address operand
?	Binary number	>	Hexadecimal number
MSB	Most significant byte	LSB	Least significant byte
MSb	Most significant bit	LSb	Least significant bit
cnd	Condition	( )	Contents of
→	Is assigned to	←	Becomes equal to
[ ]	Indicates an optional entry. The brackets themselves are not entered.	< >	Indicates something that must be typed in. For example, <offset> indicates that an offset must be entered. The brackets themselves are not entered.

6.2 Addressing Modes

TMS7000 Assembly Language supports eight addressing modes, listed in Table 6-2. Addressing modes that use 16-bit operands are sometimes referred to as extended addressing modes.

Table 6-2. TMS7000 Addressing Modes

ADDRESSING MODE	EXAMPLE		
Single Register	LABEL	DEC	B
		INC	R45
		CLR	R23
Dual Register	LABEL	MOV	B,A
		ADD	A,R17
		CMP	R32,R73
Peripheral File	LABEL	XORP	A,P17
		MOVP	P42,B
Immediate	LABEL	AND	%>C5,R55
		ANDP	%VALUE,P32
		BTJO	%>D6,R80,LABEL
Program Counter Relative	LABEL1	JMP	LABEL
		DJNZ	A,LABEL
		BTJO	%>16,R12,LABEL
		BTJOP	B,P7,LABEL
Direct Memory	LABEL	LDA	@>F3D4
		CMPA	@LABEL
Register File Indirect	LABEL	STA	*R43
Indexed	LABEL2	BR	@LABEL(B)

6.2.1 Single Register Addressing Mode

Single Register Addressing mode instructions use a single register that contains an 8-bit operand. The register can be specified as Rn, where n is the Register File number in the range 0–127 or 0–255, depending upon the amount of on-chip RAM available.

A and B can denote R0 and R1, respectively. Single Register Addressing mode instructions that use registers A and B are also called *implied operand instructions*.

Single Register Addressing mode instructions that specify Rn are called *single operand instructions*. Figure 6-1 illustrates the object code generated by a single operand instruction for the the following cases:

```
Case 1:    <inst>  A
           <inst>  B
Case 2:    <inst>  Rn
```

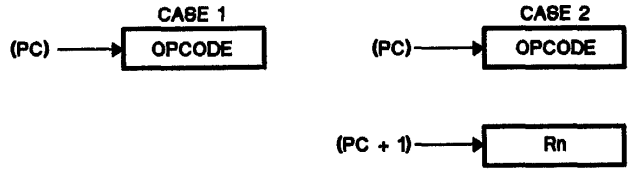


Figure 6-1. Single Register Addressing Mode Object Code

6.2.2 Dual Register Addressing Mode

Dual Register Addressing mode instructions use a source and a destination register that contain 8-bit operands. Assembly language syntax specifies the source register before the destination register. Figure 6-2 illustrates the byte requirements for all dual addressing instructions including the unique requirements of the Move instructions using this addressing mode.

SOURCE	DESTINATION A	DESTINATION B	DESTINATION Rd
A	2	1	2
B	1	2	2
iop	2	2	3
Rs	2	2	3

Bytes Needed for Move Instructions

SOURCE	DESTINATION A	DESTINATION B	DESTINATION Rd
A	2	2	3
B	1	2	3
iop	2	2	3
Rs	2	2	3

Bytes Needed for all Other Instructions

Figure 6-2. Dual Register Addressing Mode Byte Requirements

6.2.3 Peripheral-File Addressing Mode

Peripheral-File Addressing mode instructions perform I/O tasks. Each PF register is an 8-bit port that can be referred to as Pn.

Four instructions use Peripheral-File Addressing mode:

- MOVP,
- ANDP,
- ORP, and
- XORP.

These instructions may use Register A or B as the source register and Pn as the destination register. MOVP may also be executed using Pn as the source register and A or B as the destination register. (BTJOP and BTJZP are also Peripheral-File instructions but they have a different format.) Figure 6-3 illustrates the byte requirements of the instructions using the Peripheral-File Addressing mode.

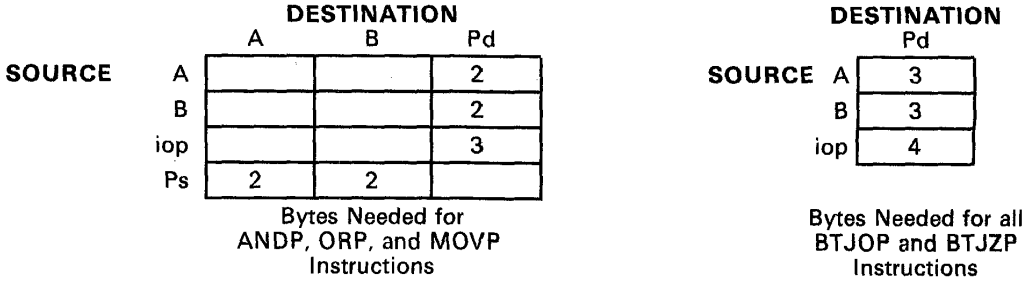


Figure 6-3. Peripheral-File Addressing Mode Byte Requirements

6.2.4 Immediate Addressing Mode

Immediate Addressing mode instructions use an immediate 8-bit operand. The immediate operand can be a constant value or a label preceded by a percent sign (%). The MOVD instruction uses 16-bit immediate operands in two special formats. Figure 6-4 illustrates the simplest case of an instruction using this mode.

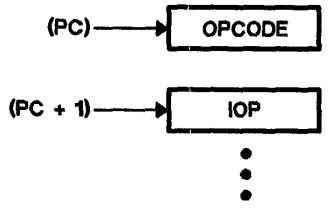


Figure 6-4. Immediate Addressing Mode Object Code

6.2.5 Program Counter Relative Addressing Mode

All Jump instructions use Program Counter Relative Addressing mode. The assembly language source statement for a jump instruction always includes a target address (ta). The microcomputer uses the target address to calculate an offset as follows: *offset = ta - pcn*, where **pcn** is the location of the next instruction and -128 ≤ ra ≤ 127. Figure 6-5 illustrates object code generated by a Jump instruction.

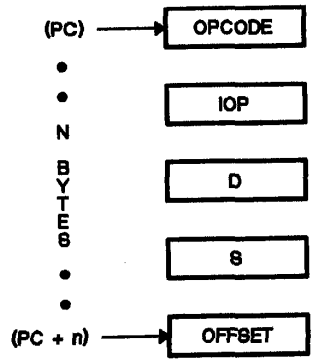


Figure 6-5. Program Counter Relative Addressing Mode Object Code

6.2.6 Direct Memory Addressing Mode

Direct Addressing mode instructions use a 16-bit address that contains the operand. The 16-bit address is preceded by an @ sign and can be written as a constant value or as a label. Figure 6-6 shows how the object code produced by an instruction using the Direct Memory Addressing mode generates a 16-bit effective address.

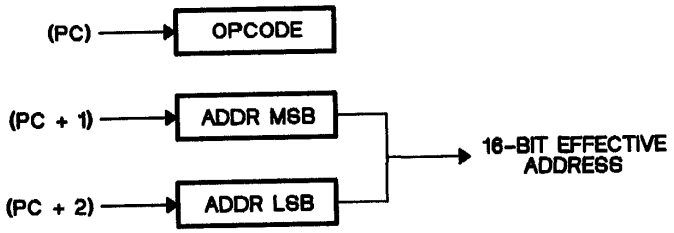


Figure 6-6. Direct Memory Addressing Mode Object Code

6.2.7 Register File Indirect Addressing Mode

Register File Indirect Addressing mode instructions use the contents of a register pair as a 16-bit effective address. The indirect Register File address is written as a register number (Rn) preceded by an asterisk (*), i.e.: *Rn. The LSB of the address is contained in Rn, and the MSB of the address is contained in the previous register (Rn-1). Figure 6-7 shows how the object code produced by an instruction using Register File Indirect Addressing mode generates a 16-bit effective address.

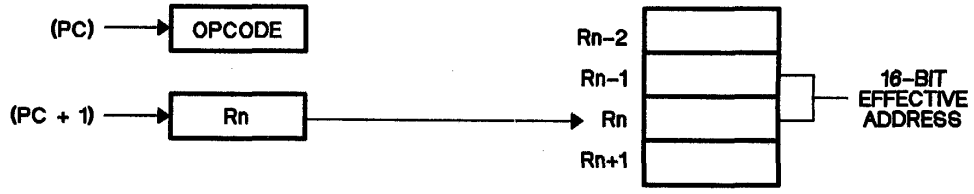


Figure 6-7. Register File Indirect Addressing Mode Object Code

6.2.8 Indexed Addressing Mode

Indexed Addressing mode instructions generate a 16-bit address by adding the contents of the B Register to a 16-bit direct memory address. The assembly language statement for the Indexed Addressing mode contains the direct memory address written as a 16-bit constant value or a label, preceded by an @ sign and followed by a B in parentheses: @LABEL(B). The addition automatically transfers any carries into the MSB. Figure 6-8 illustrates how the object code produced by an instruction using the Indexed Addressing mode generates a 16-bit effective address. Do not confuse this mode with the MOVD (Move Double) instruction's addressing mode.

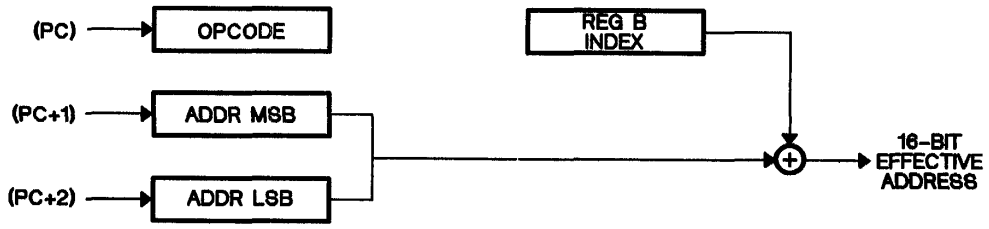


Figure 6-8. Indexed Addressing Mode Object Code

6.3 Instruction Set Overview

Table 6-3 lists all instruction formats, opcodes, byte lengths, cycles/instruction, operand types, status bits affected, and an operational description.

The TMS7000 Assembly Language instructions are presented in alphabetical order following the instruction overview table. All instructions may have optional labels preceding the mnemonic and comments following the operands. Labels, mnemonics, operands, and comments must be separated by at least one space:

```
START   MOVP  %>00,P0 Initialize to single chip
```

The byte count for each instruction may be determined from its instruction type and its operands.

Table 6-3. TMS7000 Family Instruction Overview

MNEMONIC		OPCODE	BYTES	CYCLES $T_C(C)$	STATUS C	N	Z	I	OPERATION DESCRIPTION
ADC	B,A	69	1	5	R	R	R	x	(s) + (d) + (C) → (d)
	Rs,A	19	2	8					Add the source, destination, and carry bit
	Rs,B	39	2	8					together. Store at the destination address.
	Rs,Rd	49	3	10					
	%iop,A	29	2	7					
	%iop,B	59	2	7					
	%iop,Rd	79	3	9					
ADD	B,A	68	1	5	R	R	R	x	(s) + (d) → (d)
	Rs,A	18	2	8					Add the source and destination operands at
	Rs,B	38	2	8					the destination address.
	Rs,Rd	48	3	10					
	%iop,A	28	2	7					
	%iop,B	58	2	7					
	%iop,Rd	78	3	9					
AND	B,A	63	1	5	0	R	R	x	(s) .AND. (d) → (d)
	Rs,A	13	2	8					AND the source and destination operands
	Rs,B	33	2	8					together and store at the destination
	Rs,Rd	43	3	10					address.
	%iop,A	23	2	7					
	%iop,B	53	2	7					
	%iop,Rd	73	3	9					
ANDP	A,Pd	83	2	10	0	R	R	x	(s) .AND. (Pn) → (Pn)
	B,Pd	93	2	9					AND the source and destination operands
	%iop,Pd	A3	3	11					together, and store at the destination
									address.
(1)					0	R	R	x	
BTJO	B,A,Ofst	66	2	7 (9)					If (s) .AND. (d) ≠ 0,
	Rn,A,Ofst	16	3	10 (12)					then (PC) + offset → (PC)
	Rn,B,Ofst	36	3	10 (12)					If the AND of the source and destination
	Rn,Rd,Ofst	46	4	12 (14)					operands ≠ 0, the PC will be modified to
	%iop,A,Ofst	26	3	9 (11)					include the offset.
	%iop,B,Ofst	56	3	9 (11)					
	%iop,Rn,Ofst	76	4	11 (13)					
(1)					0	R	R	x	
BTJOP	A,Pn,Ofst	86	3	11 (13)					If (s) .AND. (Pn) ≠ 0,
	B,Pn,Ofst	96	3	10 (12)					then (PC) + (offset) → (PC)
	%>iop,Pn,Ofst	A6	4	12 (14)					If the AND of the source and destination
									operands ≠ 0, the PC will be modified to
									include the offset.
(1)					0	R	R	x	
BTJZ	B,A,Ofst	67	2	7 (9)					If (s) .AND. NOT(d) ≠ 0,
	Rn,A,Ofst	17	3	10 (12)					then (PC) + (offset) → (PC)
	Rn,B,Ofst	37	3	10 (12)					If the AND of the source and NOT(destina-
	Rn,Rf,Ofst	47	4	12 (14)					tion operands ≠ 0, the PC will be modi-
	%>iop,A,Ofst	27	3	9 (11)					fied to include the offset.
	%>iop,B,Ofst	57	3	9 (11)					
	%>iop,Rn,Ofst	77	4	11 (13)					

Note: Add two to cycle count if branch is taken.

Legend:

0 Status Bit set always to 0.
1 Status Bit set always to 1.
R Status Bit set to a 1 or a 0 depending on results of operation.
x Status Bit not affected.
b Bit ( ) affected.
Ofst Offset

Table 6-3. TMS7000 Family Instruction Overview (Continued)

MNEMONIC		OPCODE	BYTES	CYCLES $T_{c(C)}$	STATUS C N Z I	OPERATION DESCRIPTION
(1)					0 R R x	
BTJZP	A,Pn,Ofst	87	3	11 (13)		If (s) .AND. NOT(Pn) ≠ 0,
	B,Pn,Ofst	97	3	10 (12)		then (PC) + offset → (PC)
	%>iop,Pn,Ofst	A7	4	12 (14)		If the AND of the source and NOT(destination) operands ≠ 0, the PC will be modified to include the offset.
BR	@Label	8C	3	10	x x x x	(d) → (PC)
	@Label(B)	AC	3	12		The PC will be replaced with the contents
	*Rn	9C	2	9		of the destination operand.
CALL	@Label	8E	3	14	x x x x	(SP) + 1 → (SP)
	@Label(B)	AE	3	16		(PC MSB) → (Stack)
	*Rn	9E	2	13		(SP) + 1 → (SP)
						(PC LSB) → (Stack)
						Operand Address → (PC)
CLR	A	B5	1	5	0 0 1 x	0 → (d)
	B	C5	1	5		Clear the destination operand.
	Rd	D5	2	7		
CLRC		B0	1	6	0 R R x	0 → (C)
						Clears the carry bit.
CMP	B,A	6D	1	5	R R R x	(d) - (s) computed
	Rn,A	1D	2	8		Set flags on the result of the source
	Rn,B	3D	2	8		operand subtracted from the destination
	Rn,Rn	4D	3	10		operand.
	%iop,A	2D	2	7		
	%iop,B	5D	2	7		
	%iop,Rn	7D	3	9		
CMPA	@Label	8D	3	12	R R R x	(A) - (s) computed
	@Label(B)	AD	3	14		Set flags on result of the source operand
	*Rn	9D	2	11		subtracted from A.
DAC	B,A	6E	1	7	R R R x	(s) + (d) + (C) → (d) (BCD)
	Rs,A	1E	2	10		The source, destination, and the carry bit
	Rs,B	3E	2	10		are added, and the BCD sum is stored at the
	Rs,Rd	4E	3	12		destination address.
	%>iop,A	2E	2	9		
	%>iop,B	5E	2	9		
	%>iop,Rd	7E	3	11		
DEC	A	B2	1	5	R R R x	(d) - 1 → (d)
	B	C2	1	5		Decrement destination operand by 1.
	Rd	D2	2	7		
DECD	A	BB	1	9	R R R x	(rp) - 1 → (rp)
	B	CB	1	9		Decrement register pair by 1.
	Rp	DB	2	11		C = 0 on 0 - FFFF transition.
DINT		06	1	5	0 0 0 0	0 → (global interrupt enable bit)
						Clear the I bit.

Note: Add two to cycle count if branch is taken.

Legend:
0 Status Bit set always to 0.
1 Status Bit set always to 1.
R Status Bit set to a 1 or a 0 depending on results of operation.
x Status Bit not affected.
b Bit ( ) affected.
Ofst Offset

Table 6-3. TMS7000 Family Instruction Overview (Continued)

MNEMONIC		OPCODE	BYTES	CYCLES $T_{c(C)}$	STATUS C N Z I	OPERATION DESCRIPTION
(1)					x x x x	
DJNZ	A,Ofst	BA	2	7 (9)		(d) - 1 → (d); If (d) ≠ 0, (PC) + (offset) → (PC)
	B,Ofst	CA	2	7 (9)		
	Rd,Ofst	DA	3	9 (11)		
DSB	B,A	6F	1	7	R R R x	(d) - (s) - 1 + (C) → (d) (BCD) The source operand is subtracted from the destination; this sum is then reduced by 1 and the carry bit is then added to it. The result is stored as a BCD number.
	Rs,A	1F	2	10		
	Rs,B	3F	2	10		
	Rs,Rd	4F	3	12		
	%>iop,A	2F	2	9		
	%>iop,B	5F	2	9		
	%>iop,Rd	7F	3	11		
EINT		05	1	5	1 1 1 1	1 → (global interrupt enable bit) Set the I bit.
IDLE		01	1	6	x x x x	(PC) → (PC) until interrupt (PC) + 1 → (PC) after return from interrupt Stops µC execution until an interrupt.
INC	A	B3	1	5	R R R x	(d) + 1 → (d) Increase the destination operand by 1.
	B	C3	1	5		
	Rd	D3	2	7		
INV	A	B4	1	5	0 R R x	NOT(d) → (d) 1's complement the destination operand.
	B	C4	1	5		
	Rd	D4	2	7		
JMP	Ofst	E0	2	7	x x x x	(PC) + (offset) → (PC) The PC is modified by an offset to create a new PC value.
(1)					x x x x	
JC	Ofst	E3	2	5 (7)		If conditions are met, then (PC) + offset → (PC) If the needed conditions are met, the PC is modified by the offset to form a new PC value.
JEQ	Ofst	E2	2	5 (7)		
JGE	Ofst	E5	2	5 (7)		
JGT	Ofst	E4	2	5 (7)		
JHS	Ofst	E3	2	5 (7)		
JL	Ofst	E7	2	5 (7)		
JNC	Ofst	E7	2	5 (7)		
JNE	Ofst	E6	2	5 (7)		
JNZ	Ofst	E6	2	5 (7)		
JP	Ofst	E4	2	5 (7)		
JPZ	Ofst	E5	2	5 (7)		
JZ	Ofst	E2	2	5 (7)		
LDA	@Label	8A	3	11	0 R R x	(s) → (A) Move the source operand to A.
	@Label(B)	AA	3	13		
	*Rn	9A	2	10		

Note: Add two to cycle count if branch is taken.

Legend:

0	Status Bit set always to 0.
1	Status Bit set always to 1.
R	Status Bit set to a 1 or a 0 depending on results of operation.
x	Status Bit not affected.
b	Bit ( ) affected.
Ofst	Offset

Table 6-3. TMS7000 Family Instruction Overview (Continued)

MNEMONIC		OPCODE	BYTES	CYCLES $T_C(C)$	STATUS C N Z I	OPERATION DESCRIPTION
LDSP		0D	1	5	x x x x	(B) → (SP) Load SP with Register B's contents.
MOV	A,B	C0	1	6	0 R R x	(s) → (d) Replace the destination operand with the source operand.
	A,Rd	D0	2	8		
	B,A	62	1	5		
	B,Rd	D1	2	7		
	Rs,A	12	2	8		
	Rs,B	32	2	8		
	Rs,Rd	42	3	10		
	%>iop,A	22	2	7		
	%>iop,B	52	2	7		
	%>iop,Rd	72	3	9		
MOVD	%>iop,Rp	88	4	15	0 R R x	(rp) → (rp) Copy the source register pair to the destination register pair.
	%>iop(B),Rp	A8	4	17		
	Rp,Rp	98	3	14		
MOVP	A,Pd	82	2	10	0 R R x	(s) → (d) Copy the source operand into the destination operand.
	B,Pd	92	2	9		
	%>iop,Pd	A2	3	11		
	Ps,A	80	2	9		
	Ps,B	91	2	8		
MPY	B,A	6C	1	44	0 R R x	(s) × (d) → (A,B) Multiply the source and destination operands, store the result in Registers A (MSB) and B (LSB).
	Rs,A	1C	2	47		
	Rs,B	3C	2	47		
	Rn,Rn	4C	3	49		
	%>iop,A	2C	2	46		
	%>iop,B	5C	2	46		
	%>iop,Rn	7C	3	48		
NOP		00	1	5	x x x x	(PC) + 1 → (PC) Add 1 to the PC.
OR	B,A	64	1	5	0 R R x	(s) .OR. (d) → (d) Logically OR the source and destination operands, and store the results at the destination address.
	Rs,A	14	2	8		
	Rs,B	34	2	8		
	Rs,Rd	44	3	10		
	%>iop,A	24	2	7		
	%>iop,B	54	2	7		
	%>iop,Rd	74	3	9		
ORP	A,Pd	84	2	10	0 R R x	(s) .OR. (d) → (d) Logically OR the source and destination operands, and store the results at the destination address.
	B,Pd	94	2	9		
	%>iop,Pd	A4	3	11		
POP	A	B9	1	6	0 R R x	(Stack Top) → (d) (SP) -1 → (SP) Copy the last byte on the stack into the destination address.
	B	C9	1	6		
	Rd	D9	2	8		

Note: Add two to cycle count if branch is taken.

Legend:
0 Status Bit set always to 0.
1 Status Bit set always to 1.
R Status Bit set to a 1 or a 0 depending on results of operation.
x Status Bit not affected.
b Bit ( ) affected.

Table 6-3. TMS7000 Family Instruction Overview (Continued)

MNEMONIC	OPCODE	BYTES	CYCLES $T_{C(C)}$	STATUS C N Z I	OPERATION DESCRIPTION
POP ST	08	1	6	0 R R x	(Stack Top) → (Status Register) (SP) - 1 → (SP) Replace the Status Register with the last byte of the stack.
PUSH A B Rs	B8 C8 D8	1 1 2	6 6 8	0 R R x	(s) → (Stack) (SP) + 1 → (SP) Copy the operand onto the stack.
PUSH ST	0E	1	6	0 R R x	(Status Register) → (Stack) (SP) + 1 → (SP) Copy the Status Register onto the stack.
RETI	0B	1	9	Loaded from the stack	Stack → (PC) LSByte (SP) - 1 → (SP) Stack → (PC) MSByte (SP) - 1 → (SP) Stack → Status Register (SP) - 1 → (SP)
RETS	0A	1	7	x x x x	(Stack) → (PC LSB) (SP) - 1 → (SP) (Stack) → (PC MSB) (SP) - 1 → (SP)
RL A B Rd	BE CE DE	1 1 2	5 5 7	b7 R R x	Bit(n) → Bit(n + 1) Bit(7) → Bit(0) and Carry
RLC A B Rd	BF CF DF	1 1 2	5 5 7	b7 R R x	Bit(n) → Bit(n + 1) Carry → Bit(0) Bit(7) → Carry
RR A B Rd	BC CC DC	1 1 2	5 5 7	b0 R R x	Bit(n + 1) → Bit(n) Bit(0) → Bit(7) and Carry
RRC A B Rd	BD CD DD	1 1 2	5 5 7	b0 R R x	Bit(n + 1) → Bit(n) Carry → Bit(7) Bit(0) → Carry
SBB B,A Rs,A Rs,B Rs,Rd %>iop,A %>iop,B %>iop,Rd	6B 1B 3B 4B 2B 5B 7B	1 2 2 3 2 2 3	5 8 8 10 7 7 9	R R R x	(d) - (s) - 1 + (C) → (d) Destination minus source minus 1 plus carry; stored at the destination address.
SETC	07	1	5	1 0 1 x	1 → (C) Set the carry bit.

Note: Add two to cycle count if branch is taken.

Legend:
0 Status Bit set always to 0.
1 Status Bit set always to 1.
R Status Bit set to a 1 or a 0 depending on results of operation.
x Status Bit not affected.
b Bit ( ) affected.

Table 6-3. TMS7000 Family Instruction Overview (Concluded)

MNEMONIC		OPCODE	BYTES	CYCLES $T_C(C)$	STATUS C N Z I	OPERATION DESCRIPTION
STA	@Label	8B	3	11	0 R R x	(A) → (d) Store A at the destination.
	@Label(B)	AB	3	13		
	*Rd	9B	2	10		
STSP		09	1	6	x x x x	(SP) → (B) Copy the SP into Register B.
SUB	B,A	6A	1	5	R R R x	(d) - (s) → (d) Store the destination operand minus the source operand into the destination.
	Rs,A	1A	2	8		
	Rs,B	3A	2	8		
	Rs,Rd	4A	3	10		
	%>iop,A	2A	2	7		
	%>iop,B	5A	2	7		
	%>iop,Rd	7A	3	9		
SWAP	A	B7	1	8	R R R x	d(Hn,Ln) → d(Ln,Hn) Swap the operand's hi and lo nibbles.
	B	C7	1	8		
	Rn	D7	2	10		
TRAP	0–23	E8–FF	1	14	x x x x	(SP) + 1 → (SP) (PC MSB) → (Stack) (SP) + 1 → (SP) (PC LSB) → (Stack) (Entry Vector)→ (PC)
TSTA		B0	1	6	0 R R x	0 → (C) Set carry bit; set sign and zero flags on the value of Register A.
TSTB		C1	1	6	0 R R x	0 → (C) Set carry bit; set sign and zero flags on the value in Register B.
XCHB	A	B6	1	6	0 R R x	(B) ←→ (d) Swap the contents of Register B with (d).
	Rn	D6	2	8		
XOR	B,A	65	1	5	0 R R x	(s) .XOR. (d) → (d) Logically exclusive OR the source and destination operands, store at the destination address.
	Rs,A	15	2	8		
	Rs,B	35	2	8		
	Rs,Rd	45	3	10		
	%>iop,A	25	2	7		
	%>iop,B	55	2	7		
	%>iop,Rd	75	3	9		
XORP	A,Pd	85	2	10	0 R R x	(s) .XOR. (Pn) → (Pn) Logically exclusive OR the source and destination operands, store at the destination.
	B,Pd	95	2	9		
	%>iop,Pd	A5	3	11		

Note: Add two to cycle count if branch is taken.

Legend:
0 Status Bit set always to 0.
1 Status Bit set always to 1.
R Status Bit set to a 1 or a 0 depending on results of operation.
x Status Bit not affected.
b Bit ( ) affected.

Syntax [<label>] ADC <s>,<Rd>

Execution (s) + (Rd) + (C) → (Rd)

Status Bits Affected

C Set to 1 on carry-out of (s) + (Rd) + (C)
Z Set on result
N Set on result

Description

ADC adds the contents of the source, the contents of the destination register, and the carry bit. It stores the result in the destination register.

Adding a 0 to the destination register is equivalent to a conditional increment (increment on carry).

ADC can implement multi-precision addition of signed or unsigned integers. For example, the 16-bit integer in register pair (R2,R3) may be added to the 16-bit integer in (A,B) as follows:

```
ADD     R3,B          Low order bytes added
ADC     R2,A          High order bytes added
```

Examples

```
LABEL1   ADC R66,R117   Adds the contents of
*                       register 66, register
*                       117, and the carry bit,
*                       and stores the sum in
*                       register 117

         ADC   B,A      Adds the contents of
*                       Register B, Register A,
*                       and the carry bit, and
*                       stores the sum in
*                       Register A

         ADC   %>3C,R29 Adds >3C, contents of
*                       register 29, and the
*                       carry bit, and stores
*                       the sum in register 29
```

Syntax [<label>] ADD <s>,<Rd>

Execution (s) + (Rd) → (Rd)

Status Bits Affected

C Set to 1 on carry-out of (s) + (Rd)
Z Set on result
N Set on result

Description ADD adds two bytes and stores the result in the destination register. It can be used for signed 2's complement or unsigned addition.

Examples

```
LABEL   ADD A,B          Adds the contents of
*                        Registers A and B, stores
*                        the results in B

        ADD  R7,A        Adds the contents of R7
*                        and A, and stores the
*                        results in A

        ADD  %TOTAL,R13  Adds the contents of
*                        TOTAL to R13 and stores
*                        the result in R13
```

Syntax [<label>] AND <s>,<Rd>

Execution (s) .AND. (Rd) → (Rd)

Status Bits Affected

C ← 0
N Set on result
Z Set on result

Description AND logically ANDs the two 8-bit operands. Each bit in the first operand is ANDed with the corresponding bit in the second operand. This is useful for clearing and resetting bits. If you need to clear a bit in the destination operand, then put a 0 in the corresponding source bit. A 1 in a source bit will not change the corresponding destination bit.

This is the truth table for the AND instruction:

Source Bit	Destination Bit	AND Result
0	0	0
0	1	0
1	0	0
1	1	1

amples

```
LABEL   AND   %>1,R12   Clear all bits in R12 except
                        Bit 0, which will remain
                        unchanged

*       AND   R7,A      AND the contents of R7 to A
                        and store the contents in A

*       AND   B,A       AND the contents of B to A
                        and store the contents in A
```

Syntax [<label>] ANDP <s>,<Pd>

Execution (s) .AND. (Pd) → (Pd)

Status Bits Affected

C	← 0
N	Set on result
Z	Set on result

Description ANDP clears one or more bits in a Peripheral-File register. It can reset an individual output line to zero when the source is an immediate operand and serving as a mask field. Since the peripheral register is read before it is ANDed, it may not work with some peripheral locations which have a different function when reading than when writing. **The only valid source operands are A, B, and %>iop.**

Examples

```
LABEL   ANDP   %>DF,P6    Clear bit 5 of Port B (P6)

        ANDP   %>FE,P9    Clear Bit 0 of Port C Data
                          Direction Register
                          (CDDR - P9)

        ANDP   A,P33      AND the contents of A and
                          P33 and store in P33
```

Syntax [<label>] BR <XADDR>

Execution (XADDR) → (PC)

Status Bits Affected None

Description BR branches to **any** location in the the 64K memory space, including the on-chip RAM. BR supports three extended addressing modes:

- Direct
- Indirect
- Indexed

The powerful concept of computed GOTOs is supported by the BR *Rn instruction. An indexed branch instruction of the form BR @TABLE(B) is an extremely efficient way to execute one of several actions on the basis of a control input. This is similar to the Pascal CASE statement. For example, suppose Register B contains a control value. The program can branch to label ACTION0 if B=0, ACTION1 if B=1, etc, for up to 128 different actions. This technique may also be used to transfer control on character inputs, error codes, etc.

Examples

```
LABEL1    BR    @THERE       Direct addressing
          BR    @TABLE(B)    Indexed addressing
          BR    *R14         Indirect addressing

LABEL2    EQU   $            Start execution here
          MOV   R3,B         Move control input to B
          RL    B            Multiply by 2 to get
*                            table offset
          BR    @TABLE(B)    Branch to correct J<cnd>
                             statement
DISPATCH  EQU   $            Dispatch table
          JMP   ACTION0
          JMP   ACTION1
          ...
          JMP   ACTIONn
ACTION0   EQU   $
*                            <Code for action 0>
ACTION1   EQU   $
*                            <Code for action 1>
          ...
ACTIONn   EQU   $
*                            <Code for action n>
```

Bit Test and Jump If One

Syntax

[<label>] BTJO <s>,<Rn>,<offset>

Execution

If (s [Bit x]) .AND. (Rn [Bit x]) ≠ 0, then (PC) + (offset) → (PC)

Status Bits Affected

C ← 0
N Set on (s) .AND. (Rn)
Z Set on (s) .AND. (Rn)

Description

BTJO tests for at least one bit position that contains a corresponding 1 in each operand. The source operand can be used as a bit mask to test for one or more 1 bits in the specified register. The operands are not changed by this instruction. If a corresponding 1 bit is found, the program branches to the offset.

Examples

```
LABEL   BTJO  %>14,R4,ISSET   Jump to ISSET if R4
*                             (bit 2) or R4 (bit
*                             4) is a 1

        BTJO  %>1,A,LOOP      Jump to LOOP if bit
*                             0 of Register A is
*                             a 1

        BTJO  R37,R113,START  Jump to START if any
*                             1 bit of R113 corre-
*                             sponds to a 1 bit
*                             in R37
```

Syntax [<label>] BTJOP <s>,<Pn>,<offset>

Execution If (s [Bit x]) .AND. (Pn [Bit x]) ≠ 0, then (PC) + (offset) → (PC)

Status Bits Affected

C ← 0
N Set on (s) .AND. (Pn)
Z Set on (s) .AND. (Pn)

Description BTJOP tests for at least one bit position that contains a corresponding 1 in each operand. The source operand can be used as a bit mask to test for at least one 1 bit in the Peripheral-File register.

Examples

```
LABEL   BTJOP   %>81,P4,THERE     Jump to THERE if
*                                 bit 0 or bit 7 of
*                                 Port A contain
*                                 a 1

        BTJOP   %>FF,P10,STORE    Test all bits of
*                                 Port D Data (P10);
*                                 jump to STORE if
*                                 any of the bits
*                                 are 1s

        BTJOP   B,P50,AGAIN       Jump to AGAIN if
*                                 any 1 bit of P50
*                                 corresponds to any
*                                 1 bit of the B
*                                 Register
```

Bit Test and Jump If Zero

Syntax [<label>] BTJZ <s>,<Rn>,<offset>

Execution If (s [Bit x]) .AND. NOT(Rn [Bit x]) ≠ 0, then (PC) + (offset) → (PC)

Status Bits Affected

C ← 0
N Set on (s) .AND. (NOT Rn)
Z Set on (s) .AND. (NOT Rn)

Description BTJZ tests for at least one bit position which has a 1 in the source and a 0 in the destination. The source operand can be used as a bit mask to test for zero bits in the specified register. The operands are unchanged by this instruction. The jump is calculated starting from the opcode of the instruction just after the BTJZ.

Examples

```
LABEL   BTJZ   A,R23,ZERO     If any 1 bits in A
*                             correspond to 0 bits
*                             in R23 then jump to
*                             ZERO to 0 bits in R23
*                             then jump to ZERO

        BTJZ   %>FF,A,NEXT    If A contains any 0
*                             bits, jump to NEXT

        BTJZ   R7,R15,OUT     If any 0 bits in R15
*                             correspond to 1 bits
*                             in R7, jump to OUT
```

Bit Test and Jump if Zero - Peripheral BTJZP

Syntax

[<label>] BTJZP <s>,<Pn>,<offset>

Execution

If (s [Bit x]) .AND. NOT(Pn [Bit x]) ≠ 0, then (PC) + (offset) → (PC)

Status Bits Affected

C	← 0
N	Set on (s) .AND. (NOT Pn)
Z	Set on (s) .AND. (NOT Pn)

Description

BTJZP tests for at least one bit position which has a 1 in the source and an 0 in the Peripheral-File register. The source operand can be used as a bit mask to test for zero bits in the Peripheral-File register. The operands are unchanged by this instruction. The jump is calculated starting from the opcode of the instruction just after the BTJZP.

Examples

```
LABEL   BTJZP   %>21,P4,THERE    Jump to THERE if P4
*                                (bit 0) or P4 (bit
*                                5) is 0

*       BTJZP   %>FF,P28,STORE   Jump to STORE if P28
                                 contains any 0s

*       BTJZP   B,P37,NEXT       Jump to NEXT if P37
*                                contains any 0 bits
*                                corresponding to 1
                                 bits in Register B
```

Call

Syntax

[<label>] CALL <XADDR>

Execution

(SP) + 1 → (SP)
(PC MSB) → (stack)
(SP) + 1 → (SP)
(PC LSB) → (stack)
(XADDR) → (PC)

Status Bits Affected

None

Description

CALL invokes a subroutine and pushes the PC contents on the stack. The operand indicates the starting address of the subroutine. Use the PUSH and POP instructions to save, pass, or restore Status or register values. The extended addressing modes of the CALL instruction allow powerful transfer of control functions.

Examples

```
LABEL   CALL   @LABEL4      Direct addressing
        CALL   @LABEL5(B)   Indexed addressing
        CALL   *R12         Indirect addressing
```

Clear

Syntax [<label>] CLR <Rd>

Execution 0 → (Rd)

Status Bits Affected

C ← 0
N ← 0
Z ← 1

Description CLR clears or initializes any file register including Registers A and B.

Examples

```
LABEL   CLR  B      Clear Register B
        CLR  A      Clear Register A
        CLR  R105   Clear register 105
```

Clear the Carry Bit

Syntax [<label>] CLRC

Execution Set status bits

Status Bits Affected

C	← 0
N	Set on value of Register A
Z	Set on value of Register A

Description CLRC clears the carry flag. This may be required before an arithmetic or rotate instruction. The logical and move instructions typically clear the carry bit. The CLRC opcode is equivalent to the TSTA opcode.

Example

```
LABEL   CLRC      Clear the carry bit
```

Syntax [<label>] CMP <s>,<Rn>

Execution (Rn) - (s) computed but not stored

Status Bits Affected

C 1 if (Rn) ≥ (s)
N Sign of result
Z 1 if (Rn) = (s)

Description CMP compares the destination operand to the source operand and sets the status bits. The CMP instruction is usually used in conjunction with a Jump instruction; Table 6-4 shows which Jump instructions can be used on status conditions set by CMP execution.

Table 6-4. Compare Instruction Examples - Status Bit Values

(S)	(Rn)	(Rn)-(S)	C	N	Z	INSTRUCTIONS THAT WILL JUMP					
FF	00	01	0	0	0	JL	JNC	JNE	JNZ	JP	JPZ
00	FF	FF	1	1	0	JHS	JC	JNE	JNZ	JN	
00	7F	7F	1	0	0	JHS	JC	JNE	JNZ	JN	JPZ
81	00	7F	0	0	0	JL	JNC	JNE	JNZ	JN	JPZ
00	81	81	1	1	0	JHS	JC	JNE	JNZ	JN	
80	00	80	0	1	0	JL	JNC	JNE	JNZ	JN	
00	80	80	1	1	0	JHS	JC	JNE	JNZ	JN	
7F	80	01	1	0	0	JHS	JC	JNE	JNZ	JN	JPZ
80	7F	FF	0	1	0	JL	JNC	JNE	JNZ	JN	
7F	7F	00	1	0	1	JHC	JC	JEQ	JZ	JPZ	
7F	00	81	0	1	0	JL	JNC	JNE	JNZ	JN	

Examples

```
LABEL   CMP   R13,R89   Set status bits on
*                       result of R89 minus R13

        CMP   B,R39     Set status bits on result
*                       of R39 minus (B)

        CMP   %>03,A    Set status bits on result
*                       of (A) minus >03
```

Syntax [<label>] CMPA <XADDR>

Execution (A) - (XADDR) computed but not stored

Status Bits Affected

C 1 if (A) logically ≥ (XADDR)
N 1 if (A) arithmetically < (XADDR)
Z 1 if (A) = (XADDR)

Description CMPA compares a long-addressed operand to the A register via direct, indirect, or indexed addressing modes. It is especially useful in table lookup programs that store the table either in extended memory or in program ROM. The status bits are set exactly as if Register A were the destination and the addressed byte the source.

Examples

```
LABEL   CMPA  @TABLE2     Direct addressing
        CMPA  @TABLE(B)   Indexed addressing
        CMPA  *R123       Indirect addressing
```

Syntax [<label>] DAC <s>,<Rd>

Execution (s) + (Rd) + (C) → (Rd), Produces a decimal result

Status Bits Affected

C	1 if value of (s) + (Rd) + C ≥ 100
N	Set on result
Z	Set on result

Description DAC adds bytes in binary-coded decimal (BCD) form. Each byte is assumed to contain two BCD digits. DAC is not defined for non-BCD operands. DAC with an immediate operand of zero value is equivalent to a conditional increment of the destination operand (increment destination on carry). The DAC instruction automatically performs a decimal adjust on the binary sum of (s) + (Rd) + C. The carry bit is added to facilitate adding multi-byte BCD strings, and so the carry bit must be cleared before execution of the first DAC instruction.

Examples

```
LABEL   DAC   %>24,A    Add the packed BCD value 24,
*                       and the carry bit to the
*                       Register A carry bit to
*                       Register A

        DAC   R55,R7    Add the BCD value of R55,
*                       and the carry bit to the
*                       BCD value of R7

        DAC   B,A       Add the carry bit to the
*                       BCD value in Register B
*                       to Register A
```

Decrement

Syntax [<label>] DEC <Rd>

Execution (d) - 1 → (Rd)

Status Bits Affected

C 0 if (Rd) decrements from >00 to >FF; 1 otherwise
N Set on result
Z Set on result

Description DEC subtracts 1 from any addressable operand. It is useful in counting and addressing byte arrays.

Examples

```
LABEL DEC   R102   Decrement R102 by 1
      DEC   A      Decrement Register A by 1
      DEC   B      Subtract 1 from the contents of
*                  Register B
```

Decrement Double

Syntax

[<label>] DECD <Rp>

Execution

(Rp) - 1 → (Rp)

Status Bits Affected

C 0 if most significant byte decrements from >00 to >FF; otherwise, C = 1
N Set on most significant byte of result
Z Set on most significant byte of result

Description

DECD decrements 16-bit indirect addresses stored in the *Register File*. Tables longer than 256 bytes may be scanned using this instruction.

The JZ (Jump on Zero) command is often used in conjunction with the DECD command. Note that JZ jumps when the **MSB** equals zero – not just when both bytes equal zero.

Example

```
LABEL   DECD  R51  Decrement (R50,R51) register
*                  pair, R51=LSB
```

Disable Interrupts

Syntax [<label>] DINT

Execution 0 → (Global interrupt enable status bit)

Status Bits Affected

I ← 0
C ← 0
N ← 0
Z ← 0

Description DINT simultaneously disables all interrupts. Since the interrupt enable flag is stored in the Status Register, the POP ST or RETI instructions may re-enable interrupts even though a DINT instruction has been executed. During the interrupt service, the interrupt enable bit is automatically cleared after the old Status Register value has been pushed onto the stack.

Example

```
LABEL   DINT    Disable global interrupt enable bit
```

DJNZ Decrement Register and Jump If Not Zero DJNZ

Syntax [<label>] DJNZ <Rd>,<offset>

Execution (Rd) - 1 → (d)
If (Rd) ≠ 0, then (PC) + (offset) → (PC)

Status Bits Affected None

Description DJNZ is used for looping control. It combines the DEC and the JNZ instructions, providing a faster and more compact instruction. DJNZ does not change the status bits.

Examples

```
LABEL   DJNZ   R15,THERE    Decrement R15. If R15 ≠
*                           0, jump to THERE

        DJNZ   A,AGAIN      Decrement A; if A ≠ 0,
*                           jump to AGAIN

        DJNZ   B,BACK       Decrement B; if B ≠ 0,
*                           jump to BACK
```

Syntax [<label>] DSB <s>,<Rd>

Execution (Rd) - (s) - 1 + (C) → (Rd) (decimal result)

Status Bits Affected

C 1 no borrow required, 0 if borrow required
N Set on result
Z Set on result

Description DSB performs multiprecision decimal BCD subtraction. A DSB instruction with an immediate operand of zero value is equivalent to a conditional decrement of the destination operand. The carry bit functions as a borrow bit, so if no borrow in is required, the carry bit should be set to 1. This can be accomplished by executing the SETC instruction.

Examples

```
LABEL   DSB   R15,R76     R76 minus R15 minus 1 plus
*                        the carry bit is stored
*                        in R76

        DSB   A,B        Register B minus Register
*                        A minus 1 plus the carry
*                        bit is stored in
*                        Register B

        DSB   B,R7       R7 minus Register B minus
*                        1 plus the carry bit
*                        stored in R7
```

Syntax [<label>] EINT

Execution 1 → (Global interrupt enable bit)

Status Bits Affected

I ← 1
C ← 1
N ← 1
Z ← 1

Description EINT simultaneously enables all interrupts. Since the interrupt enable flag is stored in the Status Register, the POP ST or RETI instructions may disable interrupts even though an EINT instruction has been executed. During the interrupt service, the interrupt enable bit is automatically cleared after the old Status Register value has been pushed onto the stack. Thus, the EINT instruction must be included inside the interrupt service routine to permit nested or multilevel interrupts.

Example

```
LABEL   EINT    All interrupts are enabled.
```

Syntax [<label>] IDLE

Execution (PC) → (PC) until interrupt
(PC) + 1 → (PC) after return from interrupt

Status Bits Affected None

Description For NMOS devices, IDLE suspends program operation until either an interrupt or reset occurs. It is the programmer's responsibility to assure that the interrupt enable status bit (and individual interrupt enable bits in the I/O control register) are set before executing the IDLE instruction. Upon return from an interrupt, control passes to the instruction following the IDLE instruction.

For CMOS devices, the IDLE instruction causes the device to enter one of two low-power modes, which use a fraction of the normal operating power. In Wake-Up mode, the on-chip oscillator remains active, and activating the timer interrupt or the external interrupts ($\overline{\text{RESET}}$, $\overline{\text{INT1}}$, or $\overline{\text{INT3}}$) releases the device from the low-power mode. In Halt mode, using the osc-off clock option, the oscillator and timers are disabled; the device can only be released from Halt mode by an external interrupt or $\overline{\text{RESET}}$. Using the osc-on clock option in Halt mode, the oscillator continues to operate and only the timers are disabled; the device can only be released from Halt mode by an external interrupt or $\overline{\text{RESET}}$.

For more information about low-power modes, see Section 3.5.

Example

```
LABEL   IDLE
```

Syntax [<label>] INC <Rd>

Execution (Rd) + 1 → (Rd)

Status Bits Affected

C 1 if (Rd) incremented from >FF to >00; 0 otherwise
N Set on result
Z Set on result

Description INC increments the value of any register. It is useful for incrementing counters into tables.

Examples

```
LABEL   INC   A      Increment Register A by 1
        INC   B      Register B is increased by 1
        INC   R43    Register 43 is increased by 1
```

Syntax [<label>] INV <Rd>

Execution NOT(Rd) → (Rd)

Status Bits Affected

C ← 0
N Set on result
Z Set on result

Description INV performs a logical or 1s complement of the operand. A 2's complement of the operand can be made by following the INV instruction with an increment (INC). A 1s complement reverses the value of every bit in the destination.

Examples

```
LABEL   INV   A       Invert Register A (0s become
*                     1s, 1s become 0s)

        INV   B       Invert Register B

        INV   R82     Invert register 82
```

Jump Unconditional

Syntax

[<label>] JMP <offset>

Execution

(PC) + (offset) → (PC)
(The PC contains the address of the instruction immediately following the jump.)

Status Bits Affected

None

Description

JMP jumps unconditionally to the address specified in the operand. The second byte of the JMP instruction contains the 8-bit *relative address* of the operand. The operand address must therefore be within -128 to +127 bytes of the location of the instruction following the JMP instruction. The assembler will indicate an error if the target address is beyond -128 to +127 bytes from the next instruction. For a longer jump the BR (branch) instruction can be used.

Example

```
LABEL   JMP THERE     Load the PC with the address
*                     of THERE
```

Syntax

[<label>] J<cnd> <offset>
(The PC contains the address of the instruction immediately following the jump.)

Execution

If tested condition is true, (PC) + (offset) → (PC)

Status Bits Affected

None

Description

Conditional Jump Instructions

INSTRUCTION	MNEMONIC	C	N	Z
Jump if Carry	JC	1	X	X
Jump if Equal	JEQ	X	X	1
Jump if Higher or Same	JHS	1	X	X
Jump if Lower	JL	0	X	X
Jump if Negative	JN	X	1	X
Jump if No Carry	JNC	0	X	X
Jump if Not Equal	JNE	X	X	0
Jump if Non-zero	JNZ	X	X	0
Jump if Positive	JP	X	0	0
Jump if Positive or Zero	JPZ	X	0	1
Jump if Zero	JZ	X	X	1

Use the J<cnd> instructions after a CMP instruction to branch according to the relative values of the operands tested. After MOV, MOVP, LDA, or STA operations, a JZ or JNZ may be used to test if the value moved was equal to zero. JN and JPZ may be used in this case to test the sign bit of the value moved.

Examples

```
LABEL   JNC   TABLE   If the carry bit is clear,
*                     jump to TABLE

        JP    HERE    If the negative and zero flags
*                     are clear, jump to HERE

        JZ    NEXT    If the zero flag is set, jump
*                     to NEXT
```

Syntax [<label>] LDA <XADDR>

Execution (XADDR) → (A)

Status Bits Affected

C ← 0
N Set on value loaded
Z Set on value loaded

Description LDA reads values stored anywhere in the full 64K-byte memory space. LDA uses three extended addressing modes:

- Direct Addressing mode provides an efficient means of directly accessing a variable in memory.
- Indexed addressing gives an efficient table look-up capability for most applications.
- Indirect addressing allows the use of very large look-up tables and the use of multiple memory pointers since any pair of registers can be used as the pointer.

Examples

```
LABEL   LDA   @LABEL4      Direct addresing
        LDA   @LABEL5(B)   Indexed addressing
        LDA   *R13         Indirect addressing
```

Syntax [<label>] LDSP

Execution (B) → (SP)

Status Bits Affected None

Description LDSP copies the contents of Register B to the Stack Pointer register. Use LDSP to initialize the Stack Pointer.

Example

```
LABEL   LDSP            Copy Register B to the
*                       Stack Pointer
```

Syntax [<label>] MOV <s>,<Rd>

Execution (s) → (Rd)

Status Bits Affected

C ← 0
N Set on value loaded
Z Set on value loaded

Description MOV transfers values within the register space. Immediate values may be loaded directly into the registers. A MOV that uses Register A or B as an operand produces shorter and quicker moves.

Examples

```
LABEL   MOV   A,B        Move the contents of Register
*                        A to Register B

        MOV   R32,R105   Move the contents of register
*                        32 to register 105

        MOV   %>10,R3    Move >10 to register 3
```

Syntax [<label>] MOVD <s>,<Rp>

Execution (s) → (Rp)

Status Bits Affected

C ← 0
N Set on MSb moved
Z Set on MSb moved

Description MOVD moves a two-byte value to the register pair indicated by the destination register number. (Note that Rp should be greater than 0 or the MSb may be lost.) The destination points to the LSB of the destination register pair. The source may be a 16-bit constant, another register pair, or an indexed address. For the latter case, the source must be of the form "%ADDR(B)" where ADDR is a 16-bit constant or address. This 16-bit value is added (via 16-bit addition) to the contents of the B register, and the result placed in the destination register pair. This stores an indexed address into a register pair, for use later in indirect addressing mode. This is not to be confused with the extended addressing instruction @LABEL(B).

Examples

```
LABEL   MOVD   %>1234,R3    Load register pair R2,R3
*                           with >1234

        MOVD   R5,R3        Copy R4,R5 to R2,R3;
*                           R5,R3 = LSB

        MOVD   %TAB(B),R3   Load register pair R2,R3
*                           with the effective
*                           address of TAB + B
```

Syntax

[<label>] MOVP <s>,<Pd>
or
[<label>] MOVP <Ps>,<d>

Execution

(s) → (Pd)
or
(Ps) → (d)

Status Bits Affected

C ← 0
N Set on value moved
Z Set on value moved

Description

MOVP transfers values to and from the Peripheral File. This may be used to input or output 8-bit quantities on the I/O ports. The Peripheral File also contains control registers for the interrupt lines, the I/O ports, and the timer controls. The operands supported by this instruction are A, B and %>iop.

During Peripheral-File instructions, a Peripheral-File port is always read before a write. The read can include output operations such as MOVP A,P6. If this read is undesirable because of hardware configuration, use a STA (Store A) instruction with the memory-mapped address of the peripheral register.

Examples

```
LABEL   MOVP   A,P6       Move the contents of
*                         Register A to Port B

RDPORT  MOVP   P4,B       Move Port A data into
*                         Register B

LOADD   MOVP   %>12,P27   Move the hex value 12 into
                          Register 27
```

Syntax [<label>] MPY <s>,<Rn>

Execution (s) × (Rn) → (A,B) Result always stored in A,B

Status Bits Affected

C ← 0
N Set on MSb of results (Register A)
Z Set on MSb of results (Register A)

Description MPY performs an 8-bit multiply for a general source and destination operand. The 16-bit result is placed in the A, B register pair with the most significant byte in A. Multiplying by a power of two is a convenient means of performing double-byte shifts. If a double byte shift is three places or less, then it may be faster to use RLC or RRC instead of multiply. If a single byte needs shifting then it is almost always faster to use RLC or RRC.

Examples

```
LABEL   MPY   R3,A      Multiply (R3) with (A), store
*                       result in A, B register pair

        MPY   %>32,B    Multiply >32 with (B), store
*                       in register pair A, B

        MPY   R12,R7    Multiply (R12) with (R7) and
*                       store in A, B register pair
```

Syntax [<label>] NOP

Execution (PC) + 1 → (PC)

Status Bits Affected None

Description NOP is useful as a pad instruction during program development, to "patch out" unwanted or erroneous instructions or to leave room for code changes during development. It is also useful in software timing loops.

Example

```
LABEL   NOP
```

Syntax [<label>] OR <s>,<Rd>

Execution (s) .OR. (Rd) → (Rd)

Status Bits Affected

C ← 0
N Set on result
Z Set on result

Description OR logically ORs the two operands. Each bit of the 8-bit result follows the truth table below. The OR operation is used to set bits in a register. If a register needs a 1 in the destination then a 1 is placed in the corresponding bit location in the source operand.

This is the truth table for the OR instruction:

Source Bit	Destination Bit	OR Result
0	0	0
0	1	1
1	0	1
1	1	1

Examples

```
LABEL   OR    A,R12     OR the A Register with R12,
*                       store in R12

        OR    %>0F,A    Set lower nibble of A to 1s,
*                       leave upper nibble unchanged

        OR    R8,B      OR (R8) with (B), store in B
```

Syntax [<label>] ORP <s>,<Pd>

Execution (s) .OR. (Pd) → (Pd)

Status Bits Affected

C ← 0
N Set on result
Z Set on result

Description ORP logically ORs the source operand with a Peripheral-File location, and write the result back to the Peripheral File. This may be used to set an individual I/O bit of a peripheral register. Since the peripheral register is read before it is ORed, it may not work with some peripheral locations which have a different function when reading than when writing.

Examples

```
LABEL   ORP   %>08,P0    Clear interrupt 2
*       ORP   A,P39      OR (A) with (P39), store
                         in P39
*       ORP   B,P90      OR (B) with (P90), store
                         in P90
```

Syntax [<label>] POP <Rd>

Execution
(Stack top) → (Rd)
(SP) - 1 → (SP)
(Move value then decrement SP)

Status Bits Affected

C	← 0
N	Set on value POPed
Z	Set on value POPed

Description POP pulls a value from the top of the stack. The data stack can be used to save or pass values, especially during subroutines and interrupt service routines.

The Status Register may be replaced with the contents on the stack by the statement `POP ST`. This one-byte instruction is usually executed in conjunction with a previously performed `PUSH ST` instruction.

Examples

```
LABEL   POP   R32   Load R32 with top of stack

        POP   ST    Load Status Register with
*                   top of stack
```

Syntax [<label>] PUSH <Rs>

Execution
(SP) + 1 → (SP)
(Rs) → (top of stack)
(Increment SP then move value)

Status Bits Affected
C ← 0
N Set on value PUSHed
Z Set on value PUSHed

Description PUSH places a value on the top of the stack. The data stack is used to save or pass values, especially during subroutines and interrupt service routines.

The Status Register may be pushed on the stack with the statement `LABEL PUSH ST`. This one-byte instruction is usually executed in conjunction with a subsequently performed `POP ST` instruction. The Status Register is unaffected.

Examples

```
LABEL   PUSH  A    Move (A) to top of stack
        PUSH  ST   Move status to top of stack
```

Syntax [<label>] RETI

Execution
(Stack) → (PC LSB)
(SP) - 1 → (SP)
(Stack) → (PC MSB)
(SP) - 1 → (SP)
(Stack) → (ST)
(SP) - 1 → (SP)

Status Bits Affected Status Register is loaded from the stack

Description RETI is typically the last instruction in an interrupt service routine. RETI restores the Status Register to its state immediately before the interrupt occurred and branches back to the program at the instruction boundary where the interrupt occurred. Registers A and B, if used, must be restored to original values before the RETI instruction.

Example
```
LABEL   RETI    Restore to program control
```

Syntax [<label>] RETS

Execution
(Stack) → (PC LSB)
(SP) - 1 → (SP)
(Stack) → (PC MSB)
(SP) - 1 → (SP)

Status Bits Affected None

Description RETS is typically the last instruction in a subroutine. RETS branches to the location immediately following the subroutine call instruction. In the called subroutine there must be an equal number of POPs and PUSHes so that the stack is pointing to the return address and not some other data.

Example

```
LABEL   RETS    Return to program control
```

Rotate Left

Syntax

[<label>] RL <Rd>

Execution

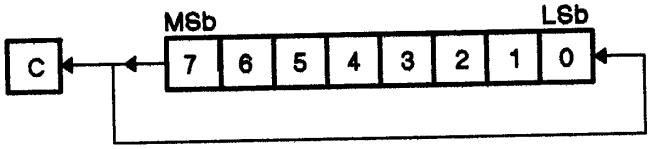
Bit(n) → Bit(n+1)
Bit(7) → Bit(0) and carry

Status Bits Affected

C Set to bit 7 of the original operand
N Set on result
Z Set on result

Description

RL circularly shifts the destination contents one bit to the left. The MSb is shifted into the LSb; the carry bit is also set to the original MSb value.



For example, if Register B contains the value >93, then RL changes the contents of B to >27 and sets the carry bit.

Examples

```
LABEL   RL   R102
        RL   A
        RL   B
```

Syntax

[<label>] RLC <Rd>

Execution

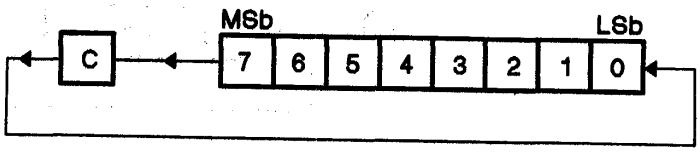
Bit(n) → Bit(n+1)
Carry → Bit(0)
Bit(7) → Carry

Status Bits Affected

C Set to bit 7 of the original operand
N Set on result
Z Set on result

Description

RLC circularly shifts the destination contents one bit to the left and through the carry. The original carry bit contents shift into the LSb, and the original MSb shifts into the carry bit.



For example, if Register B contains the value >93 and the carry bit is a zero, then the RLC instruction changes the operand value to >26 and the carry to one.

Rotating left effectively multiplies the value by 2. Using multiple rotates, any power of 2 (2, 4, 8, 16,...) can be achieved. This type of multiply is usually faster than the MPY (multiply) instruction. This instruction is also useful in rotates where a value is contained in more than one byte such as an address or in multiplying a large multibyte number by 2. Care must be taken to assure that the carry is at the proper value. The SETC or CLRC instructions may be use to setup the correct value.

Examples

```
LABEL   RLC   R72
        RLC   A
        RLC   B
```

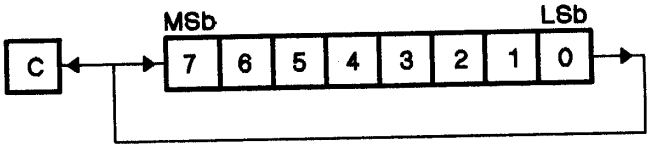
Syntax [<label>] RR <Rd>

Execution
Bit(n+1) → Bit(n)
Bit(0) → Bit (7) and carry

Status Bits Affected

C Set to bit 0 of the original value
N Set on result
Z Set on result

Description RR circularly shifts the destination contents one bit to the right. The LSb is shifted into the MSb, and the carry bit is also set to the original LSb value.



For example, if Register B contains the value >93, then the "RR B" instruction changes the contents of B to >C9 and sets the carry status bit.

Example

```
LABEL   RR   A
```

Syntax

[<label>] RRC <Rd>

Execution

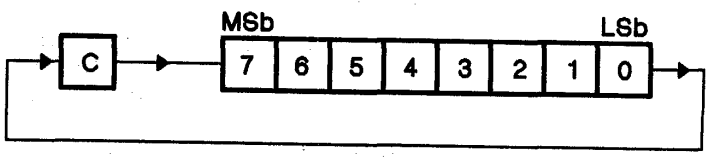
Bit(n+1) → Bit(n)
Carry → Bit(7)
Bit(0) → Carry

Status Bits Affected

C Set to bit 0 of the original value
N Set on result
Z Set on result

Description

RRC circularly shifts the destination contents one bit to the right through the carry. The carry bit contents shift into the MSb, and the LSb is shifted into the carry bit.



For example, if Register B contains the value >93 and the carry bit is zero, then RRC changes the operand value to >49 and sets the carry bit.

When the carry is 0 this instruction effectively divides the value by two. A value of >80 becomes >40. By using this instruction once more, the value can be divided by any power of two. Care must be taken to assure the correct value in the carry bit.

Example

```
LABEL   RRC R32
```

Subtract with Borrow

Syntax [<label>] SBB <s>,<Rd>

Execution (Rd) - (s) - 1 + (C) → (Rd)

Status Bits Affected

C Set to 1 if no borrow; 0 otherwise
N Set on result
Z Set on result

Description SBB performs multiprecision 2's complement subtraction. An SBB instruction with an immediate operand of zero value is equivalent to a conditional decrement of the destination operand. If (s)=0 and (C)=0 then (Rd) is decremented, otherwise it is unchanged. A borrow occurs if the result is negative. In this case, the carry bit is set to 0. The carry bit can be thought of as the "no-borrow" bit.

Examples

```
LABEL   SBB   %>23,B    Subtract (B) from >23, sub-
*                       tract 1, add the carry bit;
*                       store in Register B

        SBB   B,A       (B) minus (A) minus 1 plus
*                       the carry bit is stored
*                       in Register A

        SBB   %>33,R6   Subtract (R6) from >33, sub-
*                       tract the inverse of the
*                       carry bit
```

Syntax [<label>] SETC

Execution 1 → (C)

Status Bits Affected

C ← 1
N ← 0
Z ← 1

Description SETC sets the carry flag (if required) before an arithmetic or rotate instruction.

Example

```
LABEL   SETC
```

Store Register A

Syntax [<label>] STA <XADDR>

Execution (A) → (XADDR)

Status Bits Affected

C ← 0
N Set on value loaded
Z Set on value loaded

Description STA stores values anywhere in the 64K-byte memory address space. STA uses three extended addressing modes:

- Direct Addressing provides an efficient means of directly accessing a variable in memory.
- Indexed Addressing provides efficient table look-up.
- Indirect Addressing allows the use of very large look-up tables and the use of multiple memory pointers since any pair of registers can be used as the pointer.

Examples

```
LABEL   STA   @VALUE      Direct addressing
        STA   @TABLE(B)   Indexed addressing
        STA   *R13        Indirect addressing
```

Syntax [<label>] STSP

Execution (SP) → (B)

Status Bits Affected None

Description STSP copies the SP to Register B. This instruction can be used to test the stack size. The indexed addressing mode may be used to reference operands on the stack. For example, `STSP`; then `LDA @>0000(B)` will put the present value on top of the stack into Register A.

Example

```
LABEL   STSP    Copy the SP to Register B
```

Subtract

Syntax [<label>] SUB <s>,<Rd>

Execution (Rd) - (s) → (Rd)

Status Bits Affected

C Set to 1 if result ≥ 0, otherwise set to 0
N Set on result
Z Set on result

Description SUB performs 2's complement subtraction. The carry bit is set to 0 if a borrow is required. The carry bit could be renamed a "no-borrow" bit in this case.

Examples

```
LABEL   SUB   R19,B     (B) minus (R19) is
*                       stored in R19

        SUB   %>76,A    >76 minus (A) is stored
*                       in A

        SUB   R4,R9     (R4) minus (R9) stored
*                       in R9
```

Syntax [<label>] SWAP <Rn>

Execution Bits (7,6,5,4, / 3,2,1,0) → Bits (3,2,1,0, / 7,6,5,4)

Status Bits Affected

C Set to bit 0 of the result or bit 4 of the original
N Set on results
Z Set on results

Description SWAP exchanges the first four bits with the second four bits. This instruction is equivalent to four consecutive RL (rotate left) instructions. It manipulates four bit operands, especially useful for packed BCD operations.

Examples

```
LABEL   SWAP   R45    Switch Lo and Hi nibbles of R45
        SWAP   A      Switch Lo and Hi nibbles of A
        SWAP   B      Switch Lo and Hi nibbles of B
```

Syntax [<label>] TRAP <n> where n = 0–23

Execution

(SP) + 1 → (SP)
(PC MSB) → (stack)
(SP) + 1 → (SP)
(PC LSB) → (stack)
(Entry vector) → (PC)

Status Bits Affected None

Description Trap is a one-byte subroutine call. The operand <n> is a trap number which identifies a location in the trap vector table, addresses >FFD0 to >FFFF in memory. The contents of the two-byte vector location form a 16-bit trap vector to which a subroutine call is performed. TRAP is an efficient way to invoke a subroutine. The highest block of memory is the trap vector table, and can contain up to 23 subroutine addresses. The subroutine addresses are stored like all other addresses in memory, with the least significant byte in the higher-addressed location, as shown below.

TRAP VECTOR TABLE

>FFD0	Trap 23 address	MSB
>FFD1	Trap 23 address	LSB
:	:	:
>FFE0	Trap 15 address	MSB
>FFE1	Trap 15 address	LSB
:	:	:
>FFFA	Trap 2 address	MSB
>FFFB	Trap 2 address	LSB
>FFFC	Trap 1 address	MSB
>FFFD	Trap 1 address	LSB
>FFFE	Trap 0 address	MSB
>FFFF	Trap 0 address	LSB

Note that TRAPs 0, 1, 2, and 3 correspond to the hardware-invoked interrupts 0, 1, 2, and 3, respectively. The hardware-invoked interrupts, however, push the Program Counter and the Status Register before branching to the interrupt routine, while the TRAP instruction pushes only the Program Counter. TRAP 0 will branch to the same code executed for a system reset but will not set or clear all the registers like the hardware RESET.

Example

```
LABEL  TRAP 15
```

Test Register A

Syntax [<label>] TSTA

Execution C,N,Z bits set

Status Bits Affected

C ← 0
N Set on value in Register A
Z Set on value in Register A

Description TSTA sets the status bits according to the value in Register A. This instruction is equivalent to the CLRC (Clear Carry) instruction.

Example

```
LABEL   TSTA      Test Register A
```

Test Register B

Syntax [<label>] TSTB

Execution C,N,Z bits set

Status Bits Affected

C ← 0
N Set on value in Register B
Z Set on value in Register B

Description TSTB sets the status bits according to the value in Register B. It may be used to clear the carry bit. This instruction is equivalent to the XCHB B (exchange B with B) instruction.

Example

```
LABEL   TSTB   Test Register B
```

Exchange with Register B

Syntax [<label>] XCHB <Rn>

Execution (B) ←→ (Rn)

Status Bits Affected

C ← 0
N Set on original contents of B
Z Set on original contents of B

Description XCHB exchanges a register with Register B without going through an intermediate location. The XCHB instruction with the B Register as the operand is equivalent to the TSTB instruction.

Examples

```
LABEL   XCHB  A     Exchange Register B with
*                   Register A

        XCHB  R3    Exchange Register B with R3
```

Syntax [<label>] XOR <s>,<Rd>

Execution (s) .XOR. (Rd) → (Rd)

Status Bits Affected

C ← 0
N Set on result
Z Set on result

Description XOR performs a bit-wise exclusive OR operation on the operands. The XOR instruction can be used to complement bits in the destination operand. Each bit of the 8-bit result follows the truth table below. This operation can also toggle a bit in a register. If the bit value in the destination needs to be the opposite from what it currently is, then the source should contain a 1 in that bit location.

This is the truth table for the XOR instruction:

Source Bit	Destination Bit	XOR Result
0	0	0
0	1	1
1	0	1
1	1	0

Examples

```
LABEL   XOR   R98,R125    XOR (R98) with (R125),
*                         store in R125

        XOR   %>1,R20     Toggle bit 0 in R20

        XOR   B,A         XOR (B) with (A), store
*                         in A
```

Syntax [<label>] XORP <s>,<Pd>

Execution (s) .XOR. (Pd) → (Pd)

Status Bits Affected

C	← 0
N	Set on result
Z	Set on result

Description XORP performs a bit-wise exclusive OR operation on the operands. The XORP instruction can be used to complement bits in the destination PF register. Since the peripheral register is read before it is XORed, it may not work with some peripheral locations which have a different function when reading than when writing.

Examples

```
LABEL   XORP   %>01,P9     Invert bit 0 of P9 (Port C
*                          DDR); this inverts the
*                          direction of the pin

        XORP   %>AA,P29    Toggle odd bits of P29

        XORP   B,P99       XOR (B) with (P99), store
*                          in P99
```

7. Linking Program Modules

The TMS7000 Assembler creates both absolute and relocatable object code that can be linked to form executable programs from separately assembled modules. An entire program need not be assembled at one time. A long program can be divided into separately assembled modules, avoiding a long assembly and reducing the symbol table size. Caution must be observed when assembling a long program with excessive labels; this may cause an assembler error from symbol table overflow. Modules that are common to several programs can be assembled once and accessed when needed. These separately-generated modules can be linked together by the Link Editor, forming a single linked object module that is stored in a library and/or loaded as required.

The **Link Editor User's Guide** (literature number SPNU037) contains a complete description of the Link Editor, related files, linker commands, linking examples, and error messages. This section provides all the information that most TMS7000 users need to link program modules.

Section **Page**

7.1 Relocation Capability

Absolute code is appropriate for code that must be placed in dedicated areas of memory. It must always be loaded into the same memory area.

Relocatable code includes information that allows a loader to place the code in any available memory area, allowing the most efficient use of available memory.

Object code generated by an assembler contains machine language instructions, addresses, and data. The code may include **absolute** segments, **program-relocatable** segments, **data-relocatable** segments, and numerous **common-relocatable** segments. In assembly language source programs, symbolic references to locations within a relocatable segment are called *relocatable addresses*. These addresses are represented in the object code as displacements from the beginning of a specified segment. A *program-relocatable address*, for example, is a displacement into the program segment. At load time, all program-relocatable addresses are adjusted by a value equal to the load address. *Data-relocatable addresses* are represented by a displacement into the data segment. There may be several types of *common-relocatable addresses* in the same program, since distinct common segments may be relocated independently of each other.

Expressions may contain more than one symbol that is not previously defined. Expressions on either side of a multiplication or division symbol must be absolute; if they are relocatable, the expression is illegal. An expression in which the number of relocatable symobls or constants added to the expression exceeds the number of relocatable symbols or constants subtracted from the expression by more than one is illegal. That is, if:

NA = Number of relocatable values added, and
NS = Number of relocatable values subtracted

Then, if NA - NS =

0	The expression is absolute
1	The expression is relocatable
Neither	The expression is illegal

An expression containing relocatable symbols or constants of several different relocation types is absolute *if* it is absolute with respect to all relocation types. If it is relocatable with respect to one relocation type and absolute with respect to all other relocation types, it is relocatable.

Examples of valid expressions include:

BLUE+1	The value of symbol BLUE + 1
GREEN-4	The value of symbol GREEN - 4
2*16+RED	2 times 16 plus the value of symbol RED
440/2-RED	440 divided by two less the value of symbol RED. Red must be absolute.

Decimal, hexadecimal, and character constants are absolute. Assembly-time constants defined by absolute expressions are absolute, and assembly-time constants defined by relocatable expressions are relocatable.

Any symbol that appears in the label field of a source statement (other than an EQU directive) is **absolute** when the statement is in an *absolute block* of the program. Any symbol that appears in the label field of a source statement (other than an EQU directive) is **relocatable** when the statement is in a *relocatable block* of the program. The type of the label or an EQU directive is the type of an expression in an operand field.

7.2 Link Editor Operation

The Link Editor combines separate modules to produce a single linked output module. It resolves externally referenced symbols and definitions created by the REF and DEF directives. Without this function, all modules would have to be compiled or assembled at once. The Link Editor builds a list of symbols from the REF tags in the object modules that are to be included in the linking process. The Link Editor then resolves the references by matching DEF tag symbols with the REF tags and inserting the correct values for these symbols in the linked object code.

A **link control file**, which must be created before the assembly, controls the Link Editor operation. The link control file contains a set of link control commands (control stream) that direct the Link Editor in combining various object modules. Figure 7-1 shows a sample link control file. Table 7-1 summarizes the linker commands most often used to link TMS7000 program modules.

The link control commands define which modules are to be linked and how they are to be linked. The Link Editor automatically resolves the REF and DEF tag symbols between object modules specified in the INCLUDE commands. The Link Editor links the object modules in the order specified by the link control commands. Thus, the structure of the control stream determines the structure of the linked object module.

```
TASK     PROGNAME         Defines name (8 letters maximum)
INCLUDE  MYPRGRAM.MPO     Pathnames of object files,
                          compatible
INCLUDE  OTHERPGM.MPO     with user's computer system
END                       Last statement of link module
```

Figure 7-1. Sample Link Control File

Table 7-1. Linker Commands Used to Link TMS7000 Program Modules

COMMAND	SYNTAX AND DESCRIPTION
COMMON	Syntax: COMMON {<base>[,<name>][,<name>]..} Defines the starting address for the specified common segment (CSEG). Commons that are loaded at the specified address must be specifically identified within this command. COMMON is only valid when used with PROGRAM. <base> is the starting location of the common segment. It can be a decimal or a hexadecimal number. <name> is the name of the common segment.
DATA	Syntax DATA <base> Defines the absolute starting address for the data segment (DSEG) in the linked output. DATA is only valid when used with PROGRAM. <base> is the starting location of the data segment.
END	Syntax: END Indicates the end of the link control stream. This command is required in every link control file.
INCLUDE	Syntax: INCLUDE {<acnm>[,<acnm>]...,(<name>) [,(<name>)]...} Defines one or more modules to be included in the linking process. This is a required command. More than one INCLUDE statement may be used. <acnm> is the access name of a file containing the object module(s) to be inluded in the linking process, and (<name>) is a member in a library.
PROGRAM	Syntax: PROGRAM <base> Defines the absolute starting address for the program segment (PSEG) in the linked output. <base> is the starting location of the program segment.
TASK	Syntax: TASK [<name>] Defines the name of the task; this becomes the IDT name, placed on the last record of the object module. <name> is the task module identifier, and can have up to eight characters. If omitted, the IDT name of the first included module is used as the task name.

Avoid using AORG in object modules which will be linked. Linking a module that contains an AORG directive may produce an *Illegal immediate tag encountered* error at link time. Use the PSEG, CSEG, and DSEG directives instead to identify the locations in the source code. Use the PROGRAM, COMMON, and DATA commands in the link control file to define the locations.

The link control file will look similar to this example:

```
TASK    MYPROG
PROGRAM >F006   Program starting point (PSEG)
DATA    >FFD0   Trap and vector table stg pt (DSEG)
COMMON          Additional starting location (CSEG)
INCLUDE FILE1
INCLUDE FILE2
END
```

7.3 Directives Used for Linking

The assembler includes four directives used for linking program modules:

IDT Names the program module.
REF Names symbols used in the current module but defined in another module.
SREF Names symbols used in the current module that may not be defined in another module.
DEF Names symbols defined in the current module that can also be used by other modules.

For more information about directives, see Section 6, Assembler Directives.

7.3.1 IDT - Program Identifier Directive

The IDT directive assigns a name to the program module. Its syntax is:

```
[label]    IDT    <string>
```

where [label] is optional, and <string> contains the module name.

If a module will be linked, it must include an IDT directive. Each module name is limited to eight characters and must be unique.

7.3.2 DEF - External Definition Directive

Symbols defined in a program module and required by other program modules must be defined by the DEF directive. The following example shows a program named ROUTINES that DEFs a routine named SUBR1. The label SUBR1 must be defined in the program.

Example 7-1. File A

```
        IDT    'ROUTINES'
        DEF    SUBR1,SUBR2   Subroutines #1 and #2 entry
                              points
        .
SUBR1   EQU    $             Subroutine #1 starts here
        .
        .
        RETS
SUBR2   EQU    $             Subroutine #2 starts here
        .
        .
        RETS
        END
```

When the program in Example 7-1 is linked with the program in Example 7-2, the references are automatically resolved.

7.3.3 REF and SREF - External Reference Directives

If a module uses a symbol that is defined in a different module, it must be externally referenced by the REF or the SREF directive. The following example shows a program, MAIN, which REFs a subroutine named SUBR1. (SUBR1 is not defined in File B.)

Example 7-2. File B

```
IDT     'MAIN'
REF     SUBR1     Subroutine #1 entry point
.
.
.
CALL    @SUBR1    Execute subroutine #1 now
.
.
.
END
```

A. TMS7000 Bus Activity Tables

This section describes the internal and external bus activity during each instruction execution and hardware operation (for example, interrupts). The **external bus** activity is the information seen on the *expansion bus*. The **internal bus** refers to the *address and data buses* that are part of the TMS7000 internal architecture. The information on the address and data buses, as well as the control pins, can be monitored externally when the device operates in any mode but Single-Chip. The internal and external buses' activity is documented on a cycle-by-cycle basis. The information in this section is useful to:

- Understand the external expansion bus for the purpose of designing an interface
- Calculate instruction execution times
- Gain a better understanding of microcomputer operation

The information on the bus activity tables is the same for NMOS and CMOS devices except for the IDLE instruction. This difference is noted in Table A-8.

Topics covered in this appendix include:

Table A-1 contains an alphabetical listing of the TMS7000 instructions and indexes into the appropriate bus activity tables.

A.1 TMS7000 Operating Modes

The TMS7000 is a microcoded microcomputer with four operating modes:

- In the **Single-Chip mode**, there are four 8-bit I/O ports (Ports A, B, C, and D) that provide 32 general purpose I/O lines.
- In **Peripheral-Expansion mode**, one 8-bit port (Port C) becomes a multiplexed address and data bus and four output lines (the four most signigicant bits of Port B) become the bus control signals. This is called the *external expansion bus*. The 8-bit address/data bus allows the TMS7000 to access up to 256 bytes of externally memory-mapped peripherals (excluding the dedicated on-chip Peripheral-File locations).
- **Full-Expansion mode** is similar to Peripheral-Expansion mode, except that another Port D becomes the MSB of a 16-bit address (Port C supplies the LSB). This means that the TMS7000 can access up to 64K bytes externally minus the number of bytes of on-chip ROM.
- **Microprocessor mode** is the same as Full-Expansion mode, except that the on-chip ROM (if any) is ignored and the entire 64K bytes are mapped off chip.

A.2 TMS7000 Addressing Modes

Because the TMS7000 implements a microcoded architecture, the microcode that fetches the instructions and their operands can be shared by many instructions. The instruction can be grouped according to the types of operands the instructions require and how the instructions are fetched. Each instruction group is based on one of the addressing modes supported by the TMS7000:

- **Double Operand Functions (DOPFUN)**

  ADD, ADC, AND, BTJO, BTJZ, CMP, DAC, DSB, MOV, MPY, OR, SBB, SUB, XOR

  These instructions require two operands for execution.

- **Miscellaneous Functions (MISCFUN)**

  DINT, EINT, IDLE, LDSP, NOP, POP ST, PUSH ST, RETI, RETS, SETC, STSP

  These instructions need no operands because the instruction function is implied in the opcode.

- **Long Addressing Functions (LAFUN)**

  BR, CALL, CMPA, LDA, STA

  These instructions require a 16-bit address which is used to address the entire 64K-byte address range of the TMS7000.

- **Single Operand Functions - Special (SOPFUNS)**

  CLR, DEC, INC, INV, MOV A B, MOV A RN, MOV B RN, SWAP, TSTA/CLRC, TSTB, XCHB

  These instructions need one operand for execution.

- **Single Operand Functions - Normal (SOPFUNN)**

  DECD, DJNZ, POP, PUSH, RL, RLC, RR, RRC

  These instructions need one operand for execution. Two groups of single operand instructions are needed because of the way CPU control is implemented and the number of supported single operand instructions.

- **Double Operand Functions - Peripheral (DOPFUNP)**

  ANDP, BTJOP, BTJZP, MOVP, ORP, and XORP.

  These instructions require two operands and interact with the TMS7000 peripheral file registers.

- **Move Double (MOVD)**

  MOVD

  Moves a register pair to a register pair and is the only instruction in this group.

- **Relative Jumps (RJMP)**

  JMP, JN/JLT, JZ/JEQ, JC/JHS, JP/JGT, JPZ/JGE, JNZ/JNE, JNC, JL

  These conditional and unconditional jumps alter program flow by adding or subtracting an 8-bit value with the program counter.

- **Traps (TRAP)**

  Trap 0 through Trap 23.

  These instructions are used to perform subroutine calls.

A.3 Instruction Execution

There are three phases of instruction execution:

1) Opcode fetch (instruction acquisition mode)
2) Operand addressing (addressing mode)
3) Functional operation on the operands (functional mode)

Appendix A - TMS7000 Bus Activity Tables

The Bus Activity Tables, which list the number of cycles executed in each phase, are grouped according to these three phases:

- The **instruction acquisition sequence** is common to all instructions, so they are presented separately:

- To determine the number of **addressing mode** and **functional mode** cycles, locate the instruction's functional group (Table A-1) and reference the appropriate table. Table A-1 lists the TMS7000 instructions in alphabetical order with the corresponding addressing mode.

Add all these cycles together to obtain the bus activity present during that instruction's execution.

Each table indicates whether a read or a write is performed during that cycle. The R/$\overline{W}$ signal is high for reads and low (logic zero) for writes. The memory control signals, ALATCH and $\overline{ENABLE}$, are asserted during both reads and writes. Note that the $\overline{ENABLE}$ signal is asserted only during external reads and writes.

Accesses other than internal RAM are long memory cycle (two-cycle) accesses. The timing of these accesses for NMOS and CMOS devices is specified in the Memory Interface Timing specifications in Section 4. These long memory cycle accesses have been indicated by their grouping within the tables (two-cycle accesses are not separated by a horizontal line). For these cycle pairs, the first cycle uses the C and D ports for the address bus (C only for Peripheral-Expansion mode). In the second cycle, Port C becomes a data bus. Figure A-1 illustrates the read/write information. This timing diagram is the same for NMOS and CMOS devices, but the interface timing specifications are different.

Although short memory cycles (RAM cycles) influence the external bus activity, no valid information is seen and the timing cannot be specified.

The following terms are used throughout this appendix:

LSB least significant byte of a 16-bit value
MSB most significant byte of a 16-bit value
Rs (Rn source) the first operand listed
Rd (Rn destination) the second operand listed. The resulting value is stored at the Rd address.

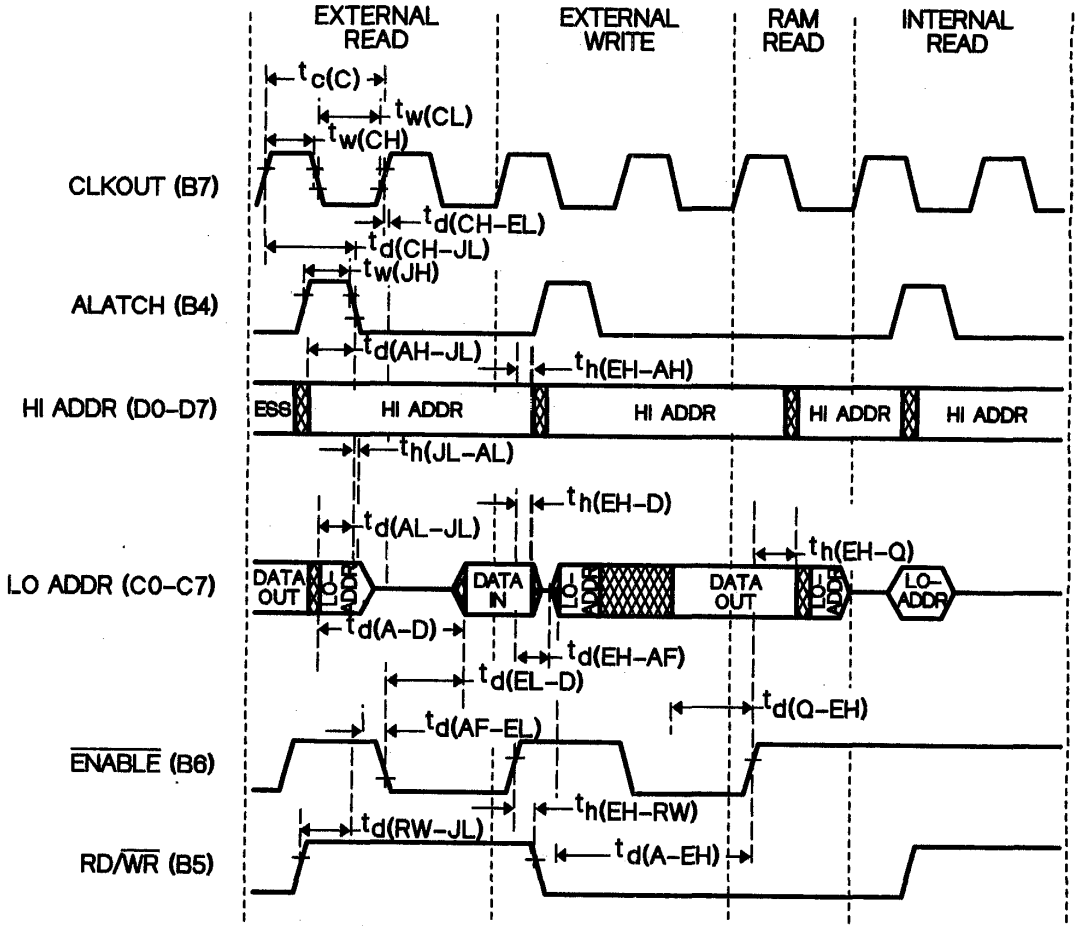


Figure A-1. Read and Write Timing Diagram

A.3.1 An Example Using the Bus Activity Tables

Example A-1 illustrates the execution steps produced by the instruction

```
ADD     R5, R6.
```

To construct the cycles required to execute the instruction, begin with the opcode fetch as shown in Example A-1. These three cycles:

1) Fetch the instruction opcode,
2) Increment the program counter, and
3) Prefetch register B.

Example A-1. Execution Steps for ADD (Instruction Acquisition)

ADDRESSING MODE	CYCLE	ADDRESS BUS	DATA BUS	R/$\overline{W}$
All Instructions	1 2	Opcode address Opcode address	Irrelevant data Instruction opcode	R R
	3 †	Register B address	Register B contents	R

† The first two cycles fetch the ADD instruction's opcode and increment the program counter. The third state prefetches register B to speed up instructions that reference register B.

Note: This information is from Table A-2.

Example A-2. Execution Steps for ADD (Addressing Modes)

ADDRESSING MODE	CYCLE	ADDRESS BUS	DATA BUS	R/$\overline{W}$
Rn, Rn	1 2	Opcode address + 1 Opcode address + 1	Irrelevant data Rs address	R R
	3	Rs address	Rs data	R
	4 5	Opcode address + 2 Opcode address + 2	Irrelevant data Rd address	R R
	6	Rd address	Operand data	R

Note: The addressing mode is entered next and is found in Table A-5.

The ADD instruction is a double operand function, requiring two operands. Double operand functions are described in Table A-5 and Table A-6. Cycles 1 and 2 of this mode read the R5 operand address. Cycle 3 reads the register contents.

> **Note:**
>
> The internal register read (or write) is a one cycle operation. All other reads/writes are two cycles long, requiring that the address bus be held stable for two complete machine cycles.

Each machine cycle corresponds to one clock period of the CLKOUT signal (pin 2), starting with the rising edge of this signal. Cycles 4 and 5 read the

Rd address, (R6) where the resultant value is placed. Cycle 6 reads the contents of register R6. Now, both operands are inside the CPU and the indicated function can be performed as shown Example A-3 for functional modes (excerpted from Table A-6).

Example A-3. Execution Steps for ADD (Functional Modes)

INSTRUCTION	CYCLE	ADDRESS BUS	DATA BUS	R/$\overline{W}$
ADD	1	Register address	Register data	$\overline{W}$

Once both operands are inside the CPU, only one cycle is needed to perform the add operation. The result is written back to register R6 during this cycle. A total of 10 cycles is required to perform an `ADD R5, R6`.

Table A-1. Alphabetical Index of Instruction Groups

INSTRUCTION	ADDRESS MODE	TABLE NUMBER	FUNCTION
ADC	DOPFUN	Table A-5	Add with carry
ADD	DOPFUN	Table A-5	Add
AND	DOPFUN	Table A-5	And
ANDP	DOPFUNP	Table A-15	And value with peripheral port
BTJO	DOPFUN	Table A-5	Test bit and jump if one
BTJOP	DOPFUNP	Table A-15	Test peripheral bit and jump if one
BTJZ	DOPFUN	Table A-5	Test bit and jump if zero
BTJZP	DOPFUNP	Table A-15	Test peripheral bit and jump if zero
BR	LAFUN	Table A-9	Long branch
CALL	LAFUN	Table A-9	Subroutine call
CLR	SOPFUNS	Table A-11	Clear
CLRC	SOPFUNS	Table A-11	Clear status carry bit
CMP	DOPFUN	Table A-5	Compare value
CMPA	LAFUN	Table A-9	Compare value with Register A
DAC	DOPFUN	Table A-5	Decimal add with carry
DEC	SOPFUNS	Table A-11	Decrement value
DECD	SOPFUNN	Table A-13	Decrement double register pair
DINT	MISCFUN	Table A-7	Disable interrupts
DJNZ	SOPFUNN	Table A-13	Decrement and jump if not zero
DSB	DOPFUN	Table A-5	Decimal subtract
EINT	MISCFUN	Table A-7	Enable interrupts
IDLE	MISCFUN	Table A-7	Idle (PC is held unchanged)
INC	SOPFUNS	Table A-11	Increment
INV	SOPFUNS	Table A-11	Invert
JMP	REL JUMPS	Table A-19	Unconditional relative jump
J<cnd>	REL JUMPS	Table A-19	Conditional relative jumps (JN/JLT, JZ/JEQ, JL, JC/JHS, JP/JGT, JPZ/JGE, JNZ/JNE, JNC)
LDA	LAFUN	Table A-9	Load Register A from long address
LDSP	MICSFUN	Table A-7	Load Stack Pointer
MOV	DOFUN	Table A-5	Move a data value
MOV	SOPFUNS	Table A-11	Move with implied operand
MOVD	MOVD	Table A-17	Move a 16-bit value to register pair
MOVP	DOPFUNP	Table A-15	Move a data value to/from port
MPY	DOPFUN	Table A-5	Multiply two 8-bit values
NOP	MISCFUN	Table A-7	No operation

Table A-1. Alphabetical Index of Instruction Groups (Concluded)

INSTRUCTION	ADDRESS MODE	TABLE NUMBER	FUNCTION
OR	DOPFUN	Table A-5	OR two values together
ORP	DOPFUNP	Table A-15	OR port value with another value
POP	SOPFUNN	Table A-13	POP a value off the stack
POPST	MISCFUN	Table A-7	POP stack value into Status Register
PUSH	SOPFUNN	Table A-13	PUSH a value onto the stack
PUSHST	MISCFUN	Table A-7	PUSH Status Register onto stack
RETI	MISCFUN	Table A-7	Return from interrupt
RETS	MISCFUN	Table A-7	Return from subroutine
RL	SOPFUNN	Table A-13	Rotate left
RLC	SOPFUNN	Table A-13	Rotate left through carry bit
RR	SOPFUNN	Table A-13	Rotate right
RRC	SOPFUNN	Table A-13	Rotate right through carry bit
SBB	DOPFUN	Table A-5	Subtract with borrow
SETC	MISCFUN	Table A-7	Set carry bit
STA	LAFUN	Table A-9	Store Register A to long address
STSP	MISCFUN	Table A-7	Store Stack Pointer to Register B
SUB	DOPFUN	Table A-5	Subtract
SWAP	SOPFUNS	Table A-11	Swap nibbles of an 8-bit value
TSTA	SOPFUNS	Table A-11	Test Register A and set status
TSTB	SOPFUNS	Table A-11	Test Register B and set status
TRAP n	TRAP	Table A-20	Trap to subroutine
XCHB	SOPFUNS	Table A-11	Exchange value with Register B
XOR	DOPFUN	Table A-5	Exclusive OR
XORP	DOPFUNP	Table A-15	Exclusive OR with peripheral port

Table A-2. Instruction Acquisition Mode - Opcode Fetch

ADDRESSING MODE	CYCLE	ADDRESS BUS	DATA BUS	R/$\overline{W}$
All Instructions	1†	Opcode address	Irrelevant data	R
	2	Opcode address	Instruction opcode	R
	3‡	Register B address	Register B contents	R

† Go to interrupt code listed for cycle 3 if an interrupt is pending.
‡ Go to addressing modes (Table A-5 through Table A-20).

Notes: 1. This mode is executed for all instructions to fetch the instruction's opcode.
2. Register B is prefetched to speed up the execution of instructions that reference register B.
3. The Program Counter is incremented during cycles 1 and 2 of this mode.
4. An interrupt check is performed during cycle 2. If an interrupt is detected, cycle 3 is not executed. Control is passed immediately to the interrupt handling code shown next.

Table A-3. Instruction Acquisition Mode - Interrupt Handling

FUNCTION	CYCLE	ADDRESS BUS	DATA BUS	R/$\overline{W}$
Interrupts	1†	Irrelevant data	Irrelevant data	–
	2	Irrelevant data	Irrelevant data	–
	3	Irrelevant data	Irrelevant data	–
	4	Irrelevant data	Irrelevant data	–
	5	SP register	Status register	$\overline{W}$
	6	Irrelevant data	Irrelevant data	–
(Reset entry)	7	Irrelevant data	Irrelevant data	–
	8	Irrelevant data	Irrelevant data	–
	9	Address >FF00 + vector	Irrelevant data	R
	10	Address >FF00 + vector	LSB INT vector	R
	11	Address >FF00 + vector	Irrevelent data	R
	12	Address >FF00 + vector	MSB INT vector	R
	13	SP contents	PCH contents	$\overline{W}$
	14	Irrelevant data	Irrelevant data	–
	15	SP + 1 contents	PCL contents	$\overline{W}$
	16	Irrelevant data	Irrelevant data	–
	17	Irrelevant data	Irrelevant data	–

† Jump to cycle number 5 if opcode was IDLE (>01). If it was an IDLE instruction, do not decrement PC because desired return is past the IDLE instruction.

Notes:
1. The Program Counter is decremented during cycles number 3 and 4. This is done because the instruction that the PC had pointed at has not been executed.
2. The Status Register is saved on the stack during Cycle 5. The Program Counter is saved during cycles 13 and 15.
3. The vector is selected by hardware depending upon which interrupt was asserted.

Table A-4. Instruction Acquisition Mode - Reset

FUNCTION	CYCLE	ADDRESS BUS	DATA BUS	R/$\overline{W}$
Reset	1	Irrelevant data	Irrelevant data	R
	2	Irrelevant data	Zeroes	–
	3†	Address >0100	Zeroes	$\overline{W}$
	4	Address >0100	Zeroes	$\overline{W}$

† Jump to interrupt cycle 7 (see Reset Entry).

Notes:
1. A read operation is done the first cycle even though the address and data buses contain irrelevant data. This read is done to protect memory in case a long write was in progress when the Reset action occurred.
2. The write to address >0100 is done to disable all interrupts.
3. The Stack Pointer is initialized to >01.
4. The Program Counter is stored in the register pairs A and B.
5. The RESET function is initiated when the $\overline{\text{RESET}}$ line of the TMS7000 device is held at a logic zero level for at least five clock cycles. When an active signal is detected on $\overline{\text{RESET}}$, the sequence shown above is entered immediately after the current machine cycle is done.

Table A-5. Double Operand Functions - Addressing Modes (ADD,ADC,AND,BTJO,BTJZ,CMP,DAC,DSB,MOV,MPY,OR,SBB,SUB,XOR)

FUNCTION†	CYCLE	ADDRESS BUS	DATA BUS	R/$\overline{W}$
Rn, A	1	Opcode address + 1	Irrelevant data	R
	2	Opcode address + 1	Rn address	R
	3	Rn address	Rn data	R
	4	Register A address	Register A data	R
%n, A	1	Opcode address + 1	Irrelevant data	R
	2	Opcode address + 1	Immediate value (%n)	R
	3	Register A address	Register A data	R
Rn, B	1	Opcode address + 1	Irrelevant data	R
	2	Opcode address + 1	Rn address	R
	3	Rn address	Rn data	R
	4	Register B address	Operand data	R
Rn, Rn	1	Opcode address + 1	Irrelevant data	R
	2	Opcode address + 1	Rs address	R
	3	Rs address	Rs data	R
	4	Opcode address + 2	Irrelevant data	R
	5	Opcode address + 2	Rd address	R
	6	Rd address	Rd data	R
%n, B	1	Opcode address + 1	Irrelevant data	R
	2	Opcode address + 1	Immediate data	R
	3	Register B address	Register B data	R
B, A	1	Register A address	Register A data	R
%n, Rn	1	Opcode address + 1	Irrelevant data	R
	2	Opcode address + 1	Immediate data	R
	3	Opcode address + 2	Irrelevant data	R
	4	Opcode address + 2	Rn address	R
	5	Rn address	Rn data	R

† See functional modes in Table A-6.

Table A-6. Double Operand Functions - Functional Modes (ADD,ADC,AND,BTJO,BTJZ,CMP,DAC,DSB,MOV,MPY,OR,SBB,SUB,XOR)

INSTRUCTIONS†	CYCLE	ADDRESS BUS	DATA BUS	R/$\overline{W}$
MOV	1	Register address	Register data	$\overline{W}$
AND	1	Register address	Register data	$\overline{W}$
OR	1	Register address	Register data	$\overline{W}$
XOR	1	Register address	Register data	$\overline{W}$
ADD	1	Register address	Register data	$\overline{W}$
ADC	1	Register address	Register data	$\overline{W}$
SUB	1	Register address	Register data	$\overline{W}$
SBB	1	Register address	Register data	$\overline{W}$
CMP	1	Irrelevant data	Irrelevant data	–
DAC	1 2 3 †	Register address Register address Register address	Register data Register data Register data	$\overline{W}$ R $\overline{W}$
DSB	1	Register address	Register data	$\overline{W}$
	2	Register address	Register data	R
	3	Register address	Register data	$\overline{W}$
MPY (Note 1)	1	Register B address	Register B data	$\overline{W}$
	2	Irrelevant data	Irrelevant data	–
	3	Irrelevant data	Irrelevant data	–
	4	Register B address	Register B data	R
	5	Register B address	Register B data	$\overline{W}$
9 iterations	6	Irrelevant data	Irrelevant data	–
	7	Irrelevant data	Irrelevant data	–
	8	Register A address	MSB mult. product	$\overline{W}$
	9	Irrelevant data	Irrelevant data	–
BTJO,BTJZ (Note 2)	1	Irrelevant data	Irrelevant data	–
	2	Opcode address + 1	Irrelevant data	R
	3	Opcode address + 1	Jump PC offset	R
	4	Opcode address + 1	Jump PC offset	R
	5	Irrelevant data	Irrelevant data	–
	6	Irrelevant data	Irrelevant data	–
	7	Irrelevant data	Irrelevant data	–

† Jump to instruction acquisition sequence.

Notes: 1. MPY – This microcode iterates to perform the multiply. The functional portion of the MPY instruction requires 40 states for execution.

2. BTJO, BTJOP – Not all states are executed. Either state 2 or state 3 is executed, but not both. The same applies to states 6 and 7.

Table A-7. Miscellaneous Functions - Addressing Modes (DINT,EINT,IDLE,LDSP,NOP,POP ST,PUSH ST,RETI,RETS,SETC,STSP)

ADDRESSING MODE	CYCLE†	ADDRESS BUS	DATA BUS	R/$\overline{W}$
	1	SP contents	Stack value	R

† See functional modes in Table A-8.

Table A-8. Miscellaneous Functions - Functional Modes (DINT,EINT,IDLE,LDSP,NOP,POP ST,PUSH ST,RETI,RETS,SETC,STSP)

ADDRESSING MODE	CYCLE	ADDRESS BUS	DATA BUS	R/$\overline{W}$
EINT	1	Irrelevant data	Irrelevant data	–
DINT	1	Irrelevant data	Irrelevant data	–
SETC	1 †	Irrelevant data	Irrelevant data	–
POP ST	1	SP contents	Stack data	R
	2 †	Irrelevant data	Irrelevant data	–
STSP	1	Irrelevant data	Irrelevant data	–
	2 †	Register B address	SP contents	$\overline{W}$
RETS	1	Irrelevant data	Irrelevant data	–
	2	Register address	Register data	R
	3 †	Irrelevant data	Irrelevant data	–
RETI	1	Irrelevant data	Irrelevant data	–
	2	Register address	Register data	R
	3	Irrelevant data	Irrelevant data	–
	4	SP contents	Register data	R
	5 †	Irrelevant data	Irrelevant data	–
LDSP	1 †	Irrelevant data	Irrelevant data	–
PUSH ST	1	Irrelevant data	Irrelevant data	–
	2 †	SP contents	Status register	$\overline{W}$
IDLE	1	Irrelevant data	Irrelevant data	–
	2 †	Irrelevant data	Irrelevant data	–

† Jump to instruction acquisition sequence.

Notes: 1. NOP does not have an execution state. From the addressing mode control is passed back to the instruction acquisition microcode.

2. The bus activity shown for the IDLE instruction corresponds to the NMOS parts only. For these parts, the microcode loops by jumping back to its own instruction acquisition. For the CMOS parts, an IDLE corresponds to a microcode halt. Because of this, it may take up to 6 cycles longer to interrupt out of an NMOS idle.

Table A-9. Long Addressing Functions - Addressing Modes (BR,CALL,CMPA,LDA,STA)

ADDRESSING MODE	CYCLE	ADDRESS BUS	DATA BUS	R/$\overline{W}$
@n	1	Opcode address + 1	Irrelevant data	R
	2	Opcode address + 1	MSB of long address	R
	3	Opcode address + 2	Irrelevant data	R
	4	Opcode address + 2	LSB of long address	R
	5 †	Irrelevant data	Irrelevant data	–
*Rn	1	Opcode address + 1	Irrelevant data	R
	2	Opcode address + 1	Rn address	R
	3	Rn address	LSB of long address	R
	4 †	Rn - 1 address	MSB of long address	R
@n(B)	1	Irrelevant data	Irrelevant data	–
	2	Opcode address + 1	Irrelevant data	R
	3	Opcode address + 1	MSB of long address	R
	4	Opcode address + 2	Irrelevant data	R
	5	Opcode address + 2	LSB of long address	R
	6	Irrelevant data	Irrelevant data	–
	7 †	Irrelevant data	Irrelevant data	–

† See functional modes in Table A-10.

Table A-10. Long Addressing Functions - Functional Modes (BR,CALL,CMPA,LDA,STA)

ADDRESSING MODE	CYCLE	ADDRESS BUS	DATA BUS	R/$\overline{W}$
LDA	1	Operand address	Irrelevant data	R
	2	Operand address	Operand data	R
	3 †	Register A address	Operand data	$\overline{W}$
STA	1	Register A address	Register A contents	R
	2	Operand address	Register A contents	$\overline{W}$
	3 †	Operand address	Register A contents	$\overline{W}$
BR	1	Irrelevant data	Irrelevant data	–
	2 †	Irrelevant data	Irrelevant data	–
CMPA	1	Operand address	Irrelevant data	R
	2	Operand address	Operand data	R
	3	Register A address	Register A contents	R
	4 †	Irrelevant data	Irrelevant data	–
CALL	1	Irrelevant data	Irrelevant data	–
	2	SP contents	PCH contents	$\overline{W}$
	3	Irrelevant data	Irrelevant data	–
	4	SP + 1	PCL	$\overline{W}$
	5	Irrelevant data	Irrelevant data	–
	6 †	Irrelevant data	Irrelevant data	–

† Jump to instruction acquisition sequence.

Table A-11. Single Operand Functions, Special - Addressing Modes (CLR,DEC,INC,INV,MOV A B,MOV A RN,MOV B RN,SWAP,TSTA/CLRC,TSTB,XCHB)

ADDRESSING MODE	CYCLE	ADDRESS BUS	DATA BUS	R/$\overline{W}$
A	1 †	Register A address	Register A contents	R
B	1 †	Register B address	Register B contents	R
Rn	1	Opcode address + 1	Irrelevant data	R
	2	Opcode address + 1	Rn address	R
	3 †	Rn address	Rn data	R

† See functional modes in Table A-12.

Table A-12. Single Operand Functions, Special - Functional Modes (CLR,DEC,INC,INV,MOV A B,MOV A RN,MOV B RN,SWAP,TSTA/CLRC,TSTB,XCHB)

ADDRESSING MODE	CYCLE	ADDRESS BUS	DATA BUS	R/$\overline{W}$
DEC	1	Register address	Register data	$\overline{W}$
INC	1	Register address	Register data	$\overline{W}$
INV	1	Register address	Register data	$\overline{W}$
CLR	1 †	Register address	Register data	$\overline{W}$
XCHB	1	Register B address	Register data	$\overline{W}$
	2 †	Register address	Register data	$\overline{W}$
SWAP	1	Irrelevant data	Irrelevant data	–
	2	Irrelevant data	Irrelevant data	–
	3	Irrelevant data	Irrelevant data	–
	4 †	Register address	Register data	$\overline{W}$
MOV A,B	1	Register A address	Register A data	R
	2 †	Register B address	Register A data	$\overline{W}$
MOV A,Rn	1	Register A address	Register A data	R
	2 †	Register address	Register A data	$\overline{W}$
MOV B,Rn	1 †	Register address	Register B data	$\overline{W}$
TSTA/CLRC	1	Register A address	Register A data	R
	2 †	Register address	Register data	$\overline{W}$
TSTB	1 †	Register B address	Register data	$\overline{W}$

† Jump to instruction acquisition sequence.

Table A-13. Single Operand Functions, Normal - Addressing Modes (DECD,DJNZ,POP,PUSH,RL,RLC,RR,RRC)

ADDRESSING MODE	CYCLE	ADDRESS BUS	DATA BUS	R/$\overline{W}$
A	1 †	Register A address	Register A data	R
B	1 †	Register B address	Register B data	R
Rn	1 2	Opcode address + 1 Opcode address + 1	Irrelevant data Rn address	R R
	3 †	Rn address	Rn data	R

† See functional modes in Table A-14.

Table A-14. Single Operand Functions, Normal - Functional Modes (DECD,DJNZ,POP,PUSH,RL,RLC,RR,RRC)

INSTRUCTION	CYCLE	ADDRESS BUS	DATA BUS	R/$\overline{W}$
PUSH	1	Irrelevant data	Irrelevant data	–
	2 †	SP contents	Register data	$\overline{W}$
POP	1	SP contents	Register data	R
	2 †	Register data	Register data	$\overline{W}$
RR	1	Register data	Register data	$\overline{W}$
RRC	1	Register data	Register data	$\overline{W}$
RL	1	Register data	Register data	$\overline{W}$
RLC	1 †	Register data	Register data	$\overline{W}$
DECD	1	Register data	Register data	$\overline{W}$
	2	Irrelevant data	Irrelevant data	–
	3	Irrelevant data	Irrelevant data	–
	4	Register address	Register data	R
	5 †	Register address	Register data	$\overline{W}$
DJNZ	1	Register address	Register data-1	$\overline{W}$
	2 ‡	Opcode address + 1	Irrelevant data	R
	3 †	Opcode address + 1	Jump PC offset	R
	4	Opcode address + 1	Jump PC offset	R
	5 §	Irrelevant data	Irrelevant data	–
	6 †	Irrelevant data	Irrelevant data	–
	7 †	Irrelevant data	Irrelevant data	–

† Jump to instruction acquisition sequence.
‡ If result is not = 0, jump to state 4.
§ If jump PC offset is positive, jump to state 7.

Table A-15. Double Operand Functions, Peripheral - Addressing Modes (ANDP,BTJOP,BTJZP,MOVP,ORP,XORP)

ADDRESSING MODE	CYCLE	ADDRESS BUS	DATA BUS	R/$\overline{W}$
A, Pn	1	Register A address	Register A data	R
	2	Opcode address + 1	Irrelevant data	R
	3	Opcode address + 1	Pn address	R
	4	Pn address	Irrelevant data	R
	5	Pn address	Pn data	R
	†			
B, Pn	1	Opcode address + 1	Irrelevant data	R
	2	Opcode address + 1	Pn address	R
	3	Pn address	Irrelevant data	R
	4	Pn address	Pn data	R
	†			
%n, Pn	1	Opcode address + 1	Irrelevant data	R
	2	Opcode address + 1	%n -immediate data	R
	3	Opcode address + 2	Irrelevant data	R
	4	Opcode address + 2	Pn address	R
	5	Pn address	Irrelevant data	R
	6	Pn address	Pn data	R
	†			
Pn, A	1	Register A address	Register A data	R
	2	Opcode address + 1	Irrelevant data	R
	3	Opcode address + 1	Pn address	R
	4	Pn address	Irrelevant data	R
	5	Pn address	Pn data	R
	†			
Pn, B	1	Opcode address + 1	Irrelevant data	R
	2	Opcode address + 1	Pn address	R
	3	Pn address	Irrelevant data	R
	4	Pn address	Pn data	R
	†			

† See functional modes in Table A-16.

Notes: 1. Addressing modes "A, Pn" and "Pn, A" fetch their operands the same way.
2. Addressing modes "B, Pn" and "Pn, B" fetch their operands the same way.

Table A-16. Double Operand Functions, Peripheral - Functional Modes (ANDP,BTJOP,BTJZP,MOVP,ORP,XORP)

INSTRUCTION	CYCLE	ADDRESS BUS	DATA BUS	R/$\overline{W}$
MOVP X, Pn	1	Pn address	Peripheral register data	$\overline{W}$
	2 †	Pn address	Peripheral register data	$\overline{W}$
MOVP Pn, A	1	Register A address	Register data	$\overline{W}$
MOVP Pn, B	1 †	Register B address	Register data	$\overline{W}$
ANDP	1	Pn address	Peripheral register data	$\overline{W}$
	2 †	Pn address	Peripheral register data	$\overline{W}$
ORP	1	Pn address	Peripheral register data	$\overline{W}$
	2 †	Pn address	Peripheral register data	$\overline{W}$
XORP	1	Pn address	Peripheral register data	$\overline{W}$
	2 †	Pn address	Peripheral register data	$\overline{W}$
BTJOP	1	Irrelevant data	Irrelevant data	–
	2 ‡	Opcode address + 1	Irrelevant data	R
	3 †	Opcode address + 1	Jump PC offset	R
	4	Opcode address + 1	Jump PC offset	R
	5 §	Irrelevant data	Irrelevant data	–
	6 †	Irrelevant data	Irrelevant data	–
	7 †	Irrelevant data	Irrelevant data	–
BTJZP	1	Irrelevant data	Irrelevant data	–
	2 ¶	Opcode address + 1	Irrelevant data	R
	3 †	Opcode address + 1	Jump PC offset	R
	4	Opcode address + 1	Jump PC offset	R
	5 §	Irrelevant data	Irrelevant data	–
	6 †	Irrelevant data	Irrelevant data	–
	7 †	Irrelevant data	Irrelevant data	–

† Jump to instruction acquisition sequence.
‡ If bit tested is equal to a 1, jump to state 4.
§ If jump PC offset is positive, jump to state 7.
¶ If bit tested is equal to a 0, jump to state 4.
Notes: 1. MOVP X, Pn - X is either register A or B, or an 8-bit immediate value %n.

Table A-17. Move Double - Addressing Mode (MOVD)

INSTRUCTION	CYCLE	ADDRESS BUS	DATA BUS	R/$\overline{W}$
%n, Rn	1 2	Opcode address + 1 Opcode address + 1	Irrelevant data MSB of immediate data	R R
	3 4	Opcode address + 2 Opcode address + 2	Irrelevant data LSB of immediate data	R R
	5 †	Irrelevant data	Irrelevant data	–
Rn, Rn	1 2	Opcode address + 1 Opcode address + 1	Irrelevant data Rn source address	R R
	3	Rn source address	Rn data - LSB	R
	4 †	Rn - 1 source addr.	Rn - 1 data - MSB	R
%n(B), Rn	1	Irrelevant data	Irrelevant data	–
	2 3	Opcode address + 1 Opcode address + 1	Irrelevant data MSB of immediate data	R R
	4 5	Opcode address + 2 Opcode address + 2	Irrelevant data LSB of immediate data	R R
	6	Irrelevant data	Irrelevant data	–
	7 †	Irrelevant data	Irrelevant data	–

† See functional mode in Table A-18.

Table A-18. Move Double - Functional Mode (MOVD)

INSTRUCTION	CYCLE	ADDRESS BUS	DATA BUS	R/$\overline{W}$
MOVD	1	Irrelevant data	Irrelevant data	–
	2 3	Opcode address + 2/3 Opcode address + 2/3	Irrelevant data Destination Rn address	R R
	4 5	Irrelevant data Dest. Rn address	Irrelevant data LSB register data	– $\overline{W}$
	6	Irrelevant data	Irrelevant data	–
	7 †	Dest. Rn-1 address	MSB register data	$\overline{W}$

† Jump to instruction acquisition sequence.

Notes: 1. MOVD – States 2 and 3 will be Opcode address + 2 for the "%n, Rn" and the "Rn, Rn" addressing modes. States 2 and 3 will be Opcode address + 3 for the "%n(B), Rn" addressing mode.

Table A-19. Relative Jumps - Addressing and Functional Modes (JMP,JN/JLT,JZ/JEQ,JC/JHS,JP/JGT,JPZ/JGE,JNZ/JNE,JNC,JL)

RELATIVE JUMPS	CYCLE	ADDRESS BUS	DATA BUS	R/$\overline{W}$
	1 ‡	Opcode address + 1	Irrelevant data	R
	2 †	Opcode address + 1	Jump PC offset	R
	3	Opcode address + 1	Jump PC offset	R
	4 §	Irrelevant data	Irrelevant data	–
	5 †	Irrelevant data	Irrelevant data	–
	6 †	Irrelevant data	Irrelevant data	–

† Jump to instruction acquisition sequence.
‡ If jump condition is true, jump to state 3.
§ If jump offset is positive go to state 6.

Notes: 1. Cycle 1 tests the jump condition. If the jump is true, go to state 3, else execute state 2 and return to the instruction acquisition sequence.
2. Cycle 4 tests whether the jump offset is positive or negative. If the jump offset is positive, go to state 6.

Table A-20. Traps - Addressing and Functional Modes (Trap 0 through Trap 23)

TRAPS	CYCLE	ADDRESS BUS	DATA BUS	R/$\overline{W}$
Trap 0–7 (Group A) Trap 8–15 (Group B) Trap 16–23 (Group C)	1 1 1	Irrelevant data Irrelevant data Irrelevant data	Irrelevant data Irrelevant data Irrelevant data	– – –
	2	Irrelevant data	Irrelevant data	–
	3 4	Address >FF00+Opcode Address >FF00+Opcode	Irrelevant data LSB trap vector	R R
	5 6	Address >FF00+Opcode-1 Address >FF00+Opcode-1	Irrelevant data MSB trap vector	R R
	7	SP contents	PCH contents	$\overline{W}$
	8	Irrelevant data	Irrelevant data	–
	9	SP + 1 contents	PCL contents	$\overline{W}$
	10	Irrelevant data	Irrelevant data	–
	11 †	Irrelevant data	Irrelevant data	–

† Jump to instruction acquisition sequence.

B. TMS7500/TMS75C00 Data Encryption Device

The TMS7500 and TMS75C00 Data Encryption Devices (DED)[10] are peripheral devices designed to perform the National Bureau of Standards (NBS) Data Encryption Standard (DES) algorithm as specified in the Federal Information Processing Standard (FIPS) Publication 46. The TMS7500 and the TMS75C00 can be designed into computer systems requiring the use of the Data Encryption Standard. The TMS7500 and TMS75C00 are firmware products derived from two Texas Instruments 8-bit single-chip microcomputers, the TMS7020 and TMS70C20. Because of the similarities between the TMS7020 and TMS70C20, the TMS7500 and TMS75C00 are pin-to-pin and functionally identical in operation. The only difference is that the TMS7500 is built using NMOS technology, while the TMS75C00 is built using CMOS technology. Because the TMS7500 and TMS75C00 are each based on 8-bit single-chip microcomputers that are in high volume production, they can be a very cost-effective solution for low-cost data encryption requirements.

The TMS7500 and TMS75C00 devices are available from Texas Instruments in a standard 600-mil, 40-pin plastic package with 100-mil pin-to-pin spacings. The TMS7500 requires a single 5-volt power supply and all I/O pins are TTL compatible. The TMS75C00 requires a single 3-volt to 5.5-volt power supply and features a low current requirement of 5.5 mA typical.

For the sake of simplicity, this appendix will use the term *TMS7500* to refer to both the TMS7500 and TMS75C00 devices unless otherwise stated.

Topics covered in this appendix include:

[10] The products covered by this document (TMS7500 and TMS75C00) are within the group of electronic products that are wholly or partly of U.S. origin or technology, the export of which is subject to export license control by the U.S. Government. Therefore, prior to exportation, you are obligated to obtain the required export license from the U.S. Department of State (refer to Title 22, Code of Federal Regulations).

B.1 Key Features

A number of key features, most of which are user programmable, enable the TMS7500 to enhance the flexibility of any system using data encryption. The device can store two keys at a time and operate in two of the standard data encryption modes. Some of the key features are highlighted below:

- Validated by the National Bureau of Standards (NBS)
- Can store both a Master and an Active 64-bit key
- Active key can be encrypted or decrypted by master key internally
- Electronic Codebook (ECB) or Cipher Feedback (CFB) internal modes of operation
- Dual 8-bit data bus operation possible, one for plain data and one for ciphered data
- Command register programmable from data bus or from external pins on chip
- Status is displayed on external pins and can be read from the data bus
- On-chip oscillator uses crystal or ceramic resonator
- Maximum data rate of 3200 bits per second at 5 MHz for ECB and 400 bits per second for 8-bit CFB with the TMS7500
- Maximum data rate of 2304 bits per second at 3.6 MHz for ECB and 288 bits per second for 8-bit CFB with the TMS75C00
- Single power source requirement (5 V nominal)
- TMS75C00 offers a low power supply current requirement of 5.5 mA typical

B.2 Typical Applications

The TMS7500 is particularly well suited for any system requiring a low-cost, medium-speed data encryption device. It is easily interfaced into the system and is capable of maintaining the data rates required by most modems and terminals without sacrificing system performance. Typical applications include:

- Computer to terminal communication links
- Home banking communication links
- Teller machines for banks
- Portable terminals
- Point-of-sale terminals
- Personnel data handling
- Small business systems
- Trade market software protection

B.3 Functional Block Diagram

The functional block diagram of the TMS7500 Data Encryption Device (DED) in Figure B-1 illustrates the firmware architecture organized around the registers, buffers, and I/O buses, which are all linked together through data selectors. All of the necessary data path sequences through these selectors are determined by a 5-bit command register and eight external control/handshake pins. The device status is stored in the Status Register and is also available on the status output pins. The 64-bit key values and encryption data are passed along the 8-bit main data bus and cipher data bus.

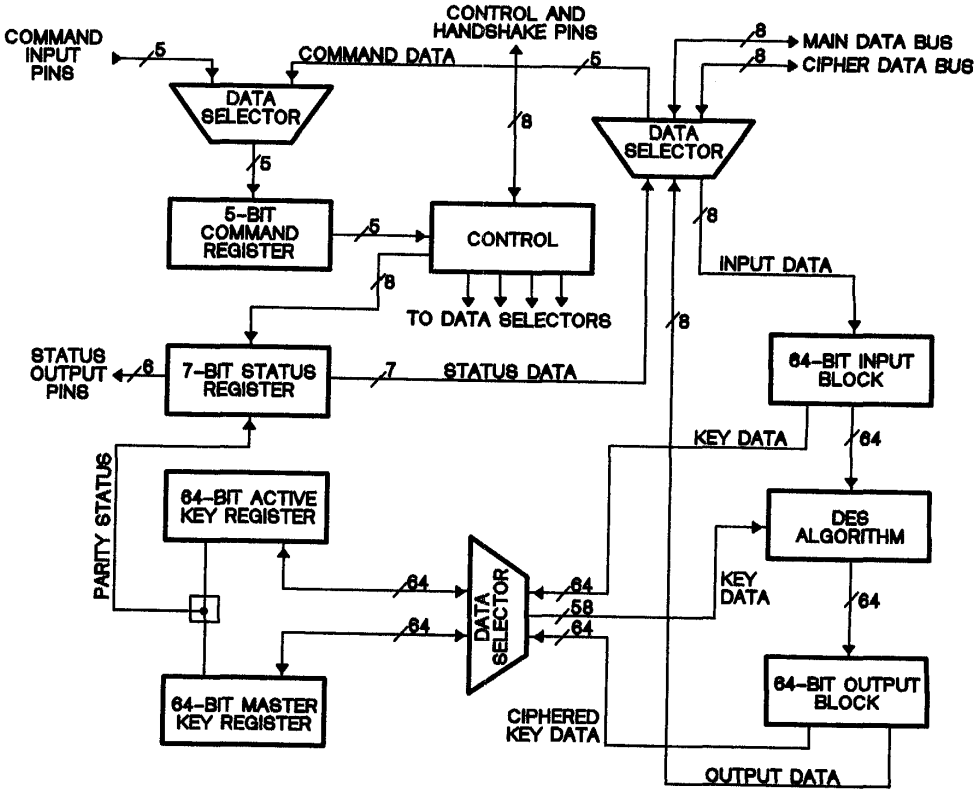


Figure B-1. TMS7500 Functional Block Diagram

B.4 Reference Documents

The following document is available from your Texas Instruments distributor or a Texas Instruments Regional Technology Center. It contains a complete functional description, interface timing specifications, and hardware/software interface examples for the TMS7500 and TMS75C00 data encryption devices.

TMS7500/TMS75C00 User's Guide (literature number SPNU004)

The following list contains related documents on the Data Encryption Standard issued by the U.S. Government. These are available from the National Technical Information Service, U.S. Department of Commerce, 5285 Port Royal Road, Springfield, VA 22161.

FIPS PUB 46, Specifications for the Data Encryption Standard

FIPS PUB 74, Guidelines for Implementing and Using the NBS Data Encryption Standard

FIPS PUB 81, DES Modes of Operation

FED STD-1026, Telecommunications, Interoperability Requirements for Use of the Data Encryption Standard in the Physical Layer of Data Communications

FED STD-1027, General Security Requirements for Equipment Using the Data Encryption Standard

IEEE TRANSACTIONS ON COMMUNICATIONS, VOL. COM-29, NO. 6, June 1981, Integrating the Data Encryption Standard into Computer Networks, by Miles E. Smid

The following documents are available from the Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20234.

NBS Special Publication 500-2, Validating the Correctness of Hardware Implementations of the NBS Data Encryption Standard

NBS Special Publication 500-27, Computer Security and the Data Encryption Standard

NBS Special Publication 500-54, A Key Notarization System for the Data Encryption Standard

NBS Special Publication 500-61, Maintenance Testing for the Data Encryption Standard

C. TMS70x1 Devices

The TMS70x1 devices include the TMS7001, TMS7041, and the SE70P161. These devices contain the same features as the TMS70x0 devices, and enhance communication ability with the addition of a serial I/O port. The TMS7041 has 4K bytes of on-chip ROM; the TMS7001 has no on-chip ROM.

Each TMS70x1 member has 128 bytes of on-chip RAM, and has the capability (through memory-expansion modes) to access up to 64K bytes of address space.

The SE70P161 is a prototyping component for the TMS7001. It is pin-compatible with the TMS7041, and uses the same instruction set. The SE70P161 is commonly referred to as a *piggyback* device because it's packaging allows a standard TMS2764 or TMS27128 EPROM device to be plugged into the top. This two-chip unit acts as a form-fit and function emulator for the TMS7041 microcontroller.

The TMS70x1 devices are not recommended for new designs. For designs that require an on-chip UART, we recommend using the enhanced features and performance of the TMS70x2, TMS70Cx2, or the TMS7742-EPROM devices.

Topics covered in this appendix include:

C.1 Key Features

- Family member with 4K bytes of on-chip ROM as well as a ROMless version
- 128-byte on-chip RAM Register File
- Flexible on-chip serial port:
  - Asynchronous, Isosynchronous, and Serial I/O modes
  - Two multiprocessor communication formats
  - Fully software programmable
  - Internal or external baud-rate generator
  - Separate baud-rate timer, useable as a third timer
- 32 TTL-compatible I/O pins:
  - 22 bidirectional pins
  - 8 output pins
  - 2 high-inpedance input pins
- Full-feature data/program stack
- Memory-mapped ports for easy addressing
- 256-byte Peripheral File
- Memory expansion capability
  - 64K-byte address space
- 8-bit instruction word
- Eight powerful addressing formats, including:
  - Register-to-register arithmetic
  - Indirect addressing on any register pair
  - Indexed and indirect branches and calls
- 2's complement arithmetic
- Single-instruction binary-coded decimal (BCD) add and subtract
- Two external, maskable interrupts
- Flexible interrupt handling
  - Priority servicing of simultaneous interrupts
  - Software execution of hardware interrupts
  - Precise timing of interrupts with the capture latch
  - Software monitoring of interrupt status
- NMOS, 5V ± 10% power supply
- 40-pin, 600-mil, dual-inline package, 100-mil, pin-to-pin spacing packages

C.2 TMS70x1 Pinouts and Pin Descriptions

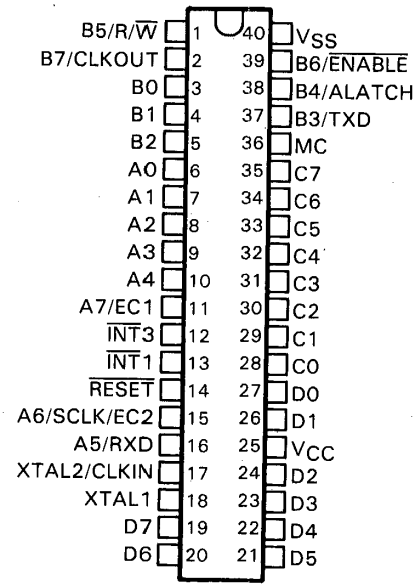


Figure C-1. TMS70x1 Pinout

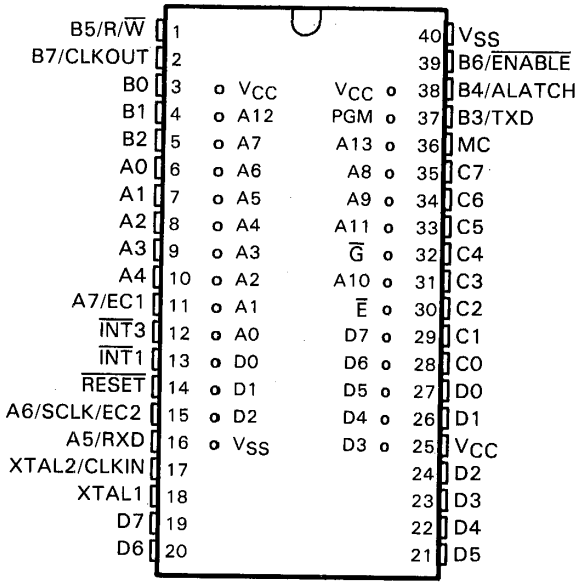


Figure C-2. SE70P161 Pinout

Table C-1. TMS70x1 and SE70P161 Pin Descriptions

SIGNAL	PIN	I/O	DESCRIPTION
A0 LSb A1 A2 A3 A4 A5/RXD A6/SCLK/EC2 A7/EC1	6 7 8 9 10 16 15 11	I/O I/O I/O I/O I/O I I/O I/O	Port A. Pins A0–A4 and A7 are general-purpose bidirectional pins. Pin A7/EC1 may also be used to clock the on-chip Timer 1 event counter. Pin A5/RXD is used as the UART receiver. Pin A6/SCLK/EC2 is the serial clock I/O pin and Timer 2 event counter
B0 B1 B2 B3/TXD B4/ALATCH B5/R/$\overline{W}$ B6/$\overline{ENABLE}$ B7/CLKOUT	3 4 5 37 38 1 39 2	O O O O O O O O	Port B. B0–B7 are general-purpose output-only pins. B4–B7 become memory expansion control signals in Peripheral-Expansion, Full-Expansion, and Microprocessor modes. Pin B3 is used as the UART transmitter.
C0 C1 C2 C3 C4 C5 C6 C7	28 29 30 31 32 33 34 35	I/O I/O I/O I/O I/O I/O I/O I/O	Port C. C0–C7 can be individually selected in software as general-purpose input or output pins in Single-Chip mode. C0–C7 become the LSB address/data bus in Peripheral-Expansion, Full-Expansion, and Microprocessor modes.
D0 D1 D2 D3 D4 D5 D6 D7	27 26 24 23 22 21 20 19	I/O I/O I/O I/O I/O I/O I/O I/O	Port D. D0–D7 can be individually selected in software as general-purpose input or output pins in Single-Chip or Peripheral-Expansion modes. D0–D7 become the MSB address/data bus in Full-Expansion and Microprocessor modes.
$\overline{INT1}$	13	I	Highest-priority maskable interrupt
$\overline{INT3}$	12	I	Lowest-priority maskable interrupt
$\overline{RESET}$	14	I	Reset
MC	36	I	Mode control pin, V_{CC} for Microprocessor mode
XTAL2/CLKIN	17	I	Crystal input for control of internal oscillator
XTAL1	18	0	Crystal output for control of internal oscillator
V_{CC}	25		Supply voltage (positive)
V_{SS}	40		Ground reference

C.3 TMS70x1 Architecture

The following sections describe the featuers and functions of the TMS70x1 microcomputers. The TMS70x1 devices are not recommended for new designs. For designs that require an on-chip UART, we recommend using the enhanced features and performance of the TMS70x2, TMS70Cx2, or the TMS7742-EPROM devices.

C.3.1 On-Chip RAM and Registers

The TMS70x1 devices contain the same on-chip registers as the TMS70x2 devices, with the exception of on-chip RAM. The TMS70x1 devices have 128 bytes of on-chip RAM, a 256-byte Peripheral File, a Stack Pointer (SP), a Status Register (ST), and a 16-bit Program Counter (PC).

C.3.2 On-Chip General-Purpose I/O Ports

The TMS70x1 devices have 32 I/O pins organizes as four 8-bit parallel ports, A, B, C, and D. These ports are memory mapped identically and accessed via the same control registers as on the TMS70x2 devices (see Section 3.2).

C.3.3 Memory Modes

The TMS70x1 devices can address up to 64K bytes of ROM and RAM. Four memory modes can be selected by a combination of software and hardware: Single-Chip, Peripheral Expansion, Full Expansion, and Microprocessor modes. These modes are identical to the other TMS7000 family memory modes (see Section 3.3).

C.3.4 I/O Control Registers

The TMS70x1 devices contain identical I/O control registers in the same memory-mapped locations that are on the TMS70x2 devices. The only difference is that bit 7 of serial control register 1 (SCTL1) is a don't care for the TMS70x1 devices, whereas on the TMS70x2 devices, this bit is the Timer 3 start/stop bit. (See Section 3 for more information.)

C.3.5 Interrupts

The TMS70x1 devices contain the same interrupt sources that are on the TMS70x2 devices. However, the external interrupts on the TMS70x1 devices are edge and level active rather than edge-only as on the TMS70x2 devices.

C.3.6 Clock Options

Clock options for the TMS70x1 are ÷2 and ÷4 of the oscillator frequency. (See Section 3.4 for more information.)

C.3.7 Programmable Timer/Event Counters

The TMS70x1 devices contain the same three timer/event counters found in the TMS70x2 devices. These timers function the same on each device with the exception of the start/stop function of Timer 3. The TMS70x1 devices do not have a start/stop function for Timer 3. (See Section 3.6 for more information.)

C.3.8 Serial Port

The TMS70x1 devices' serial port uses the same control registers and operates identically to the serial port of the TMS70x2 devices, with the exception of the asynchronous mode baud rate. The TMS70x1 operates half as fast in the asynchronous mode as do the TMS70x2 devices. This is because the TMS70x2 devices require 8 SCLK pulses to send a bit of data, while the TMS70x1 devices require 16 SCLK pulses. (See Section 3.8 for more information.)

These are the baud-rate equations for TMS70x1 devices using Asynchronous or Isosynchronous communcations.

Asynchronous baud rate

$$\frac{1}{64 \times (PR + 1) \times (TR + 1) \times t_{c(C)}}$$

Isosynchronous baud rate

$$\frac{1}{4 \times (PR + 1) \times (TR + 1) \times t_{c(C)}}$$

C.4 Standard Instruction Set/Development Support

The TMS70x1 devices use the same instruction set as all other TMS7000 family devices. Also, the TMS70x1 uses identical development tools such as the XDS, EVM, assemblers, and linkers, as do the other TMS7000 devices.

C.5 Electrical Specifications

The electrical specifications and memory interface timings of the TMS70x1 devices are identical to those of the TMS70x0 devices (see Section 4 for electrical specifications and memory interface timings).

D. Character Sets

The TMS7000 Assembler recognizes the ASCII character set listed in Table D-1. Table D-2 lists characters that the assembler does not recognize, but may be recognized and acted upon by other programs. The device service routine for the card reader accepts and stores into the calling program's buffer all the characters listed.

Table D-1. ASCII Character Set

HEX (Low nibble)	0-	1-	2-	3-	4-	5-	6-	7-
-0	NUL 0	DLE 16	SP 32	0 48	@ 64	P 80	] 96	p 112
-1	SOH 1	DC1 17	! 33	1 49	A 65	Q 81	a 97	q 113
-2	STX 2	DC2 18	" 34	2 50	B 66	R 82	b 98	r 114
-3	ETX 3	DC3 19	- 35	3 51	C 67	S 83	c 99	s 115
-4	EOT 4	DC4 20	$ 36	4 52	D 68	T 84	d 100	t 116
-5	ENQ 5	NAK 21	% 37	5 53	E 69	U 85	e 101	u 117
-6	ACK 6	SYN 22	& 38	6 54	F 70	V 86	f 102	v 118
-7	BEL 7	ETB 23	] 39	7 55	G 71	W 87	g 103	w 119
-8	BS 8	CAN 24	(40	8 56	H 72	X 88	h 104	x 120
-9	HT 9	EM 25	) 41	9 57	I 73	Y 89	i 105	y 121
-A	LF A	SUB 26	* 42	: 58	J 74	Z 90	j 106	z 122
-B	VT B	ESC 27	+ 43	; 59	K 75	[91	k 107	{ 123
-C	FF C	FS 28	' 44	< 60	L 76	\ 92	l 108	: 124
-D	CR D	GS 29	- 45	= 61	M 77	] 93	m 109	} 125
-E	SO E	RS 30	. 46	> 62	N 78	^ 94	n 110	~ 126
-F	SI F	US 31	/ 47	? 63	O 79	- 95	o 111	DEL 127

(High nibble)

Table D-2. Control Characters

HEX VALUE	DECIMAL VALUE	CHARACTER
00	0	NUL
01	1	SOH
02	2	STX
03	3	ETX
04	4	EOT
05	5	ENQ
06	6	ACK
07	7	BEL
08	8	BS
09	9	HT
0A	10	LF
0B	11	VT
0C	12	FF
0D	13	CR
0E	14	SO
0F	15	SI
10	16	DLE
11	17	CD1
12	18	CD2
13	19	CD3
14	20	CD4
15	21	NAK
16	22	SYN
17	23	ETB
18	24	CAN
19	25	EM
1A	26	SUB
1B	27	ESC
1C	28	FS
1D	29	GS
1E	30	RS
1F	31	US
7F	127	DEL

E. Hexadecimal Instruction Table/Opcode Map

HIGH / LOW	0000 0	0001 1	0010 2	0011 3	0100 4	1010 5	0110 6	0111 7	1000 8	1001 9	1010 A	1011 B	1100 C	1101 D	1110 E	1111 F
0000 0	NOP								MOVP Pn,A			TSTA/ CLRC	MOV A,B	MOV A,Rn	JMP	TRAP 15
0001 1	IDLE									MOVP Pn,B			TSTB	MOV B,Rn	JN/ JLT	TRAP 14
0010 2		MOV Rn,A	MOV %n,A	MOV Rn,B	MOV Rn,Rn	MOV %n,B	MOV B,A	MOV %n,Pn	MOVP A,Pn	MOVP B,Pn	MOVP %n,Pn	DEC A	DEC B	DEC Rn	JZ/ JEQ	TRAP 13
0011 3		AND Rn,A	AND %n,A	AND Rn,B	AND Rn,Rn	AND %n,B	AND B,A	AND %n,Pn	ANDP A,Pn	ANDP B,Pn	ANDP %n,Pn	INC A	INC B	INC Rn	JC/ JHS	TRAP 12
0100 4		OR Rn,A	OR %n,A	OR Rn,B	OR Rn,Rn	OR %n,B	OR B,A	OR %n,R	ORP A,Pn	ORP B,Pn	ORP %n,Pn	INV A	INV B	INV Rn	JP/ JGT	TRAP 11
0101 5	EINT	XOR Rn,A	XOR %n,A	XOR Rn,B	XOR Rn,Rn	XOR %n,B	XOR B,A	XOR %n,R	XORP A,Pn	XORP B,Pn	XORP %n,Pn	CLR A	CLR B	CLR Rn	JPZ/ JGE	TRAP 10
0110 6	DINT	BTJO Rn,A	BTJO %n,A	BTJO Rn,B	BTJO Rn,Rn	BTJO %n,B	BTJO B,A	BTJO %n,R	BTJOP A,Pn	BTJOP B,Pn	BTJOP %n,Pn	XCHB A	XCHB B	XCHB Rn	JNZ/ JNE	TRAP 9
0111 7	SETC	BTJZ Rn,A	BTJZ %n,A	BTJZ Rn,B	BTJZ Rn,Rn	BTJZ %n,B	BTJZ B,A	BTJZ %n,R	BTJZP A,Pn	BTJZP B,Pn	BTJZP %n,Pn	SWAP A	SWAP B	SWAP Rn	JNC/ JL	TRAP 8
1000 8	POP ST	ADD Rn,A	ADD %n,A	ADD Rn,B	ADD Rn,Rn	ADD %n,B	ADD B,A	ADD %n,R	MOVD %n,Rn	MOVD Rn,Rn	MOVD %n(B), Rn	PUSH A	PUSH B	PUSH Rn	TRAP 23	TRAP 7
1001 9	STSP	ADC Rn,A	ADC %n,A	ADC Rn,B	ADC Rn,Rn,	ADC %n,B	ADC B,A	ADC %n,R				POP A	POP B	POP Rn	TRAP 22	TRAP 6
1010 A	RETS	SUB Rn,A	SUB %n,A	SUB Rn,B	SUB Rn,Rn	SUB %n,B	SUB B,A	SUB %n,R	LDA @n	LDA *Rn	LDA @n(B)	DJNZ A	DJNZ B	DJNZ Rn	TRAP 21	TRAP 5
1011 B	RETI	SBB Rn,A	SBB %n,A	SBB Rn,B	SBB Rn,Rn	SBB %n,B	SBB B,A	SBB %n,R	STA @n	STA *Rn	STA @n(B)	DECD A	DECD B	DECD Rn	TRAP 20	TRAP 4
1100 C		MPY Rn,A	MPY %n,A	MPY Rn,B	MPY Rn,Rn	MPY %n,B	MPY B,A	MPY %n,R	BR @n	BR *Rn	BR @n(B)	RR A	RR B	RR Rn	TRAP 19	TRAP 3
1101 D	LDSP	CMP Rn,A	CMP %n,A	CMP Rn,B	CMP Rn,Rn	CMP %n,B	CMP B,A	CMP %n,R	CMPA @n	CMPA *Rn	CMPA @n(B)	RRC A	RRC B	RRC Rn	TRAP 18	TRAP 2
1110 E	PUSH ST	DAC Rn,A	DAC %n,A	DAC Rn,B	DAC Rn,Rn	DAC %n,B	DAC B,A	DAC %n,R	CALL @n	CALL *Rn	CALL @n(B)	RL A	RL B	RL Rn	TRAP 17	TRAP 1
1111 F		DSB Rn,A	DSB %n,A	DSB Rn,B	DSB Rn,Rn	DSB %n,B	DSB B,A	DSB %n,R				RLC A	RLC B	RLC Rn	TRAP 16	TRAP 0

A – Register A
B – Register A
Rn – Register File register
Pn – Peripheral File register
%n – Immediate Addressing
@n – Direct Addressing
*Rn – Indirect Addressing

F. Instruction Opcode Set

	SINGLE OPERAND			DUAL OPERAND									PERIPHERAL					EXTENDED			Other	STATUS WORD	
	A	B	Rn	A,B	B,A	Rn, A	%n, A	Rn, B	%n, B	Rn, Rn	%n, Rn	A, Rn	B, Rn	A, Pn	Pn, A	B, Pn	Pn, B	%n, Pn	†	‡	§		¶	»
ADC					69	19	29	39	59	49	79												X	
ADD					68	18	28	38	58	48	78												X	
AND					63	13	23	33	53	43	73												X	
ANDP														83		93		A3					X	
BTJO					66	16	26	36	56	46	76												X	
BTJOP														86		96		A6					X	
BTJZ					67	17	27	37	57	47	77												X	
BTJZP														87		97		A7					X	
BR																			8C	9C	AC			
CALL																			8E	9E	AE			
CLR	B5	C5	D5																				X	
CLRC																						B0	X	
CMP					6D	1D	2D	3D	5D	4D	7D												X	
CMPA																			8D	9D	AD			
DAC					6E	1E	2E	3E	5E	4E	7E												X	
DEC	B2	C2	D2																				X	
DECD	BB	CB	DB																				X	
DINT																						06	X	X
DJNZ	BA	CA	DA																				X	
DSB					6F	1F	2F	3F	5F	4F	7F												X	
EINT																						05	X	X
IDLE																						01	X	
INC	B3	C3	D3																				X	
INV	B4	C4	D4																				X	
JMP																						E0		
JC/JHS																						E3		
JN/JLT																						E1		
JNC/JL																						E7		
JNZ/JNE																						E6		
JP/JGT																						E4		
JPZ/JGE																						E5		
JZ/JEQ																						E2		
LDA																			8A	9A	AA		X	
LDSP																						0D		

† Direct
‡ Indirect
§ Indexed
¶ Condition Bits
» Interrupt Enable

Appendix F - Instruction Opcode Set

	SINGLE OPERAND			DUAL OPERAND										PERIPHERAL					EXTENDED			Other	STATUS WORD	
	A	B	Rn	A,B	B,A	Rn, A	%n, A	Rn, B	%n, B	Rn, Rn	%n, Rn	A, Rn	B, Rn	A, Pn	Pn, A	B, Pn	Pn, B	%n, Pn	†	‡	§		¶	»
MOV				C0	62	12	22	32	52	42	72	D0	D1									X		
MOVD																			88	98	A8		X	
MOVP														82	80	92	91	A2				X		
MPY					6C	1C	2C	3C	5C	4C	7C												X	
NOP																						00		
OR					64	14	24	34	54	44	74												X	
ORP														34		94		A4				X		
POP	89	C9	D9																			08	X	
PUSH	88	C8	D8																			0E	X	
RETI																						0B		
RETS																						0A		
RL	BE	CE	DE																				X	
RLC	BF	CF	DF																				X	
RR	BC	CC	DC																				X	
RRC	BD	CD	DD																				X	
SBB					6B	1B	2B	3B	5B	4B	7B												X	
SETC																						07	X	
STA																			8B	9B	AB		X	
STSP																						09	X	
SUB					6A	1A	2A	3A	5A	4A	7A												X	
SWAP	B7	C7	D7																				X	
TSTA																						80	X	
TSTB																						C1	X	
TRAP																						E3-EF	X	
XCHB	B6		D6																				X	
XOR					65	15	25	35	55	45	75												X	
XORP														35		95		A5				X		

† Direct
‡ Indirect
§ Indexed
¶ Condition Bits
» Interrupt Enable

G. CrossWare Installation

This section contains step-by-step instructions for installing, verifying, and relinking the TMS7000 Family Macro Assembler and Link Editor. This CrossWare can be installed on five operating systems:

Digital Equipment Corporation VAX-11[11]

- VMS operating system - page G-2

TI/IBM PC[12]

- MS-DOS[13] (TI PC) and PC-DOS (IBM PC) operating systems - page G-8

IBM Mainframes[12]

- MVS operating system - page G-14
- CMS operating system - page G-26

TI 990[14]

- DX10 operating system - page G-31

These style and symbol conventions are used throughout this section:

- The symbol **<CR>** indicates that a carriage return should be entered; **<enter>** indicates that the enter key should be pressed.
- Angle brackets (< and >) indicates a word which must be typed out; for example, <directory> indicates that you should type a directory name. The brackets themselves are not entered.
- Screen displays are shown in a `special font`.
- Portions of a display that are user responses are underscored.

Texas Instruments suggests that you conform to these procedures as closely as possible during the initial installation, allowing you to verify the installation with a minimum of trouble.

---

[11] VAX-11 and VMS are trademarks of Digital Equipment Corporation.

[12] MVS, CMS, and PC-DOS are trademarks of International Business Machines.

[13] MS is a trademark of Microsoft Corporation.

[14] TI 990 and DX10 are trademarks of Texas Instruments, Inc.

G.1 VAX/VMS CrossWare Installation

The TMS7000 CrossWare tape was created with the VMS BACKUP utility. The package is contained in two directories, shipped in two save sets.

G.1.1 Restore Procedures

In the following examples, **MFA0** represents the tape drive name and **DUA2** represents the hard disk drive name. Actual tape and disk drive names may differ.

- **Mount the Tape**

  Place the tape on a tape drive. Mount it by entering:

  ```
  ALLOC MFA0: <CR>
  MOUNT MFA0:/OVER=ID/FOR/DEN=1600 <CR>
  ```

  If the mount is successful, the screen displays:

  ```
  ASM7 MOUNTED ON MFA0
  ```

- **Restore the Macro Assembler**

  Use the BACKUP utility to read the ASM7 save set from the tape:

  ```
  BACKUP/LOG MFA0:ASM7 DUA2:[<directory>]*.* <CR>
  ```

  The CrossWare package can reside in either your directory or a system directory. The following examples copy the package into your directory, copying the ASM7 directory structure on the tape into [<directory>] on disk DUA2.

  A README file explaining the Macro Assembler validation procedure is contained in this directory:

  ```
  [<directory>.ASM7]README.DAT
  ```

  If you do not want to install the Link Editor, skip the next step and unload the tape.

- **Restore the Link Editor**

  Use the BACKUP utility to copy the LINKER save set from the tape:

  ```
  BACKUP/LOG MFA0:LINKER.BCK DUA2:[<directory>]*.* <CR>
  ```

  The string '...' within the brackets is for a directory name, required for the system to construct subdirectories.

  The LINKER.BCK directory structure on the tape is copied into [<directory>] on disk DUA2.

  A README file explaining the Link Editor validation procedure is contained in this directory:

  ```
  [<directory>.LINKER]README.DAT
  ```

- **Dismount the Tape**

  Dismount the tape by entering:

  DISMOUNT MFA0: <CR>

  Remove the tape from the drive. Deallocate the tape drive by entering:

  DEALLOCATE MFA0: <CR>

G.1.2 Installing Command Files

Two command procedures have been provided to ensure correct system-dependent parse features. If your VAX/VMS system runs under Version 2.5, use the PARSE.C25 command procedure by renaming it PARSE.COM. If your system runs under Version 3.0, use the default PARSE.COM.

Set the default directory to the directory the Assembler and Linker have been restored to. Edit the Assembler and Linker command files, replacing existing pathnames with the pathnames that the Assembler and Linker have been restored to.

Edit the file: [<directory>.ASM7]XASM.COM

Substitute the appropriate file pathnames in three places:

- Two calls to the PARSE command, which appear within the first 20 lines as:

  $ @[MOORE.ASM7]PARSE 'P1'....

  Change them to:

  $ @DUA2:[<directory>.ASM7]PARSE 'P1'....

- One RUN statement, which appears near the bottom of the file as:

  $ RUN[MOORE.ASM7]ASM7000

  Change it to:

  $ RUN DUA2:[<directory>.ASM7]ASM7000

Edit the file: [<directory>.LINKER]LINKER.COM

Substitute the appropriate file pathnames in three places:

- Two calls to PARSE, marked in the file by a preceding line '******... . The actual command appears similar to the PARSE commands in the assembler command file. Change them to:

  $ @DUA2:[<directory>.LINKER]PARSE 'P1'...

- One RUN statement near the end of the file. Change it to:

  $ RUN DUA2:[<directory>.LINKER]LINKER

G.1.3 Providing Transparent Access

It is not feasible to set the default directory (SET DEF) each time the Assembler or Link Editor is executed. Use the following procedure to provide transparent access for all users. Once the directories are on disk, make the following assignments into the LOGIN.COM file:

```
$ X7 :== @DUA2:[<directory>.ASM7]XASM.COM
$ XLINK :== @DUA2:[<directory>.LINKER]LINKER.COM
```

This defines the X7 and XLINK commands, which execute the Macro Assembler and Link Editor. Execute the Macro Assembler by entering **X7** at the terminal in System Mode. Similarly, execute the Link Editor by entering **XLINK**.

G.1.4 Verifying Installation

This verification procedure is not designed to perform an exhaustive test, it simply verifies that the installation procedures were executed correctly. It also provides familiarity with the basic operation and data flow of this package.

1) Create a test directory. Copy the TEST.ASM, TEST1.ASM, TEST2.ASM, and TEST1.CON files from [.ASM7] and [.LINKER] into the directory by entering these commands:

```
$ CREATE/DIR [<userid>.TEST] <CR>
$ SET DEF [<userid>.TEST] <CR>
$ COPY [<directory>.ASM7]TEST.ASM * <CR>
$ COPY [<directory>.LINKER]TEST1.ASM * <CR>
$ COPY [<directory>.LINKER]TEST2.ASM * <CR>
$ COPY [<directory>.LINKER]TEST1.CON * <CR>
```

2) In System Mode, enter: X7 <CR>

For the first input parameter, enter TEST.ASM, TEST1.ASM, and TEST2.ASM, respectively, for the three assembler runs (ASM is the default extension). The command procedure parses the pathname and generates defaults for the output listing and object files. Take the defaults by pressing the carriage return, or specify alternate file pathnames following the prompts:

```
$ X7 TEST <CR>
Object file (TEST.MPO): <CR>
Listing file (TEST.LIS): <CR>
Messages (_TTA3:): <CR>

$ X7 TEST1
Object file (TEST1.MPO): <CR>
Listing file (TEST1.LIS): <CR>
Messages (_TTA3:): <CR>

$ X7 TEST2
Object file (TEST2.MPO): <CR>
Listing file (TEST2.LIS): <CR>
Messages (_TTA3:): <CR>
```

This creates the TEST.MPO, TEST.LIS, TEST1.MPO, TEST1.LIS, TEST2.MPO and TEST2.LIS files in the directory [<userid>.TEST].

3) In System Mode, enter: XLINK <CR>

As the first input parameter, enter: TEST1.CON

For the second and third parameters, the command procedure parses the pathname and generates defaults for the output, load, and map files. This procedure links the object files for TEST1 and TEST2 into a single executable object file in TEST1.LOD (CON is the default for the first parameter):

```
$ XLINK TEST1 <CR>
Linked object file (TEST1.LOD): <CR>
Map file (TEST1.MAP): <CR>
```

This creates the files TEST1.LOD and TEST1.MAP. These files should agree with the precompiled versions in the product directories for the Macro Assembler and Link Editor.

G.1.5 Relinking the Macro Assembler and Link Editor

There should be no reason to relink the Macro Assembler or Link Editor, but command files have been provided to allow for this contingency.

To relink the Macro Assembler, edit the LINKASM.COM procedure file to put the correct pathname for the runtime library in the logical assignment statement. In System Mode, execute LINKASM.COM to relink the ASM7.EXE file:

```
$ SET DEF [<directory>.ASM7] <CR>
$ @LINKASM <CR>
```

Similarly, to relink the Link Editor, edit the LINKLINK.COM procedure file to put the correct pathname for the runtime library in the logical assignment statement. In System Mode, execute LINKLINK.COM to relink the LINKER.EXE file:

```
$ SET DEF [<directory>.LINKER] <CR>
$ @LINKLINK <CR>
```

G.1.6 Product Directories

The following listing contains the product directories found in the CrossWare package. These two directories contain a total of 28 files.

```
SET DEF [<directory>] <CR>
DIR <CR>

Directory [<directory>]
ASM7.DIR;1      LINKER.DIR;1
Total: 2 files

DIR [<default directory>.ASM7] <CR>

Directory [<directory>.ASM7]
ASM.OBJ;1     ASM7000.EXE;1 LINKASM.COM;1  PARSE.C25;1
PARSE.COM;1   README.LIS;1  ASMRTS.OLB;1   TEST.ASM;1
TEST.LIS;1    TEST.MPO;1    XASM.COM;1
Total: 11 files

DIR [<default directory>.LINKER] <CR>

Directory [<directory>.LINKER.]
LINKER.COM;1 LINKER.EXE;1  LINKER.OBJ;1   LINK-
LINK.COM;1
PARSE.C25;1   PARSE.COM;1   README.LIS;1   LINKRTS.OLB;1
TEST1.ASM;1   TEST1.CON;1   TEST1.LIS;1    TEST1.LOD;1
TEST1.MAP;1   TEST1.MPO;1   TEST2.ASM;1    TEST2.LIS;1
TEST2.MPO;1
Total: 17 files
```

G.1.7 Using the MLIB Directive

The directory pathname under VAX/VMS can be less than or equal to nine characters. However, the MLIB directive issues an Invalid Macro Library Pathname error message when the directory pathname is more than eight characters. The following code segment shows the correct response when using eight characters for the macro directory pathname.

```
NO$IDT  TMS7000 ASSEMBLER VAX/VMS 2.1 83.088 14:30:25     8/1/84
                                                          PAGE 0001
0001           *
0002           * 7000 Format 1 test procedure
0003           * This is a test file with pathname eight
0004           * characters long
0005 0000            MLIB  'DUA1:[MD0273.ABCDEFGH]'
0006           *
0007 0000            PSEG
0008           *
0009                 X1    B,A
0001           *
0002 0000  69        ADC   B,A
0010                 X1    R2,A
0001           *                    "a" = 9
0002 0001  19        ADC   R2,A
     0002  02
0011                 X1    R2,B
0001           *                    "a" = 9
0002 0003  39        ADC   R2,B
     0004  02
0012                 X1    %01,A
0001           *                    "a" = 9
0002 0005  29        ADC   %01,A
     0006  01
NO ERRORS, NO WARNINGS
```

G.2 TI and IBM PC MS/PC-DOS CrossWare Installation

The TMS7000 CrossWare package is shipped on a double-sided, dual-density diskette. The Macro Assembler and Link Editor execute in batch mode on MS-DOS (TI PC) and PC-DOS (IBM PC) systems. At least 256K bytes of memory space must be available.

Instructions are included for both hard disk systems and dual floppy drive systems. The examples use these symbols for drive names:

A: Floppy disk drive for hard disk systems *or* source drive for dual floppy drive systems.
B: Destination or system disk drive for dual floppy drive systems.
E: Winchester (hard disk) for hard disk systems.

G.2.1 Diskette Files

The diskette contains the following files:

Executable Modules:

LINKER.EXE	Executes the Link Editor
XASM7.EXE	Executes the Macro Assembler

Macro Assembler Test Files:

TEST1.ASM	Source file for Assembler test program #1
TEST1.LST	Correct output listing file for Assembler test program #1
TEST1.MPO	Correct output object file for Assembler test program #1
TEST2.ASM	Source file for Assembler test program #2
TEST2.LST	Correct output listing file for Assembler test program #2
TEST2.MPO	Correct output object file for Assembler test program #2

Link Editor Test Files:

TEST.CTL	Linker test program (link control file)
TEST.MAP	Correct output listing file for the Linker test program
TEST.LOD	Correct output object file for the Linker test program

G.2.2 Restoring the Macro Assembler and Link Editor

These instructions are for both hard disk systems and dual floppy drive systems. On a dual floppy drive system, the MS/PC-DOS system diskette should be in drive B.

1) Make a backup diskette of the product diskette.

   – On *PC-DOS systems*, place a blank diskette in drive A. Enter:

   FORMAT A: <CR>

   DISKCOPY A: A: <CR>

   Follow the prompts, removing and inserting the source and destination diskettes as directed.

- On *MS-DOS systems*, insert the source (product) diskette in drive A. Enter:

  DISKCOPY A: A:/F/V <CR>

  The /F switch tells MS-DOS to format the new (destination) diskette before copying begins. The /V switch tells MS-DOS to verify that the source and destination diskettes are identical after the diskcopy is complete. When MS-DOS *first* prompts for the destination diskette, remove the source diskette and insert a blank diskette. Follow the prompts, removing and inserting the source and destination diskettes as directed.

  When MS/PC-DOS prompts:

  COPY ANOTHER (Y/N)?

  respond with N.

2) Copy the Macro Assembler onto the hard disk or the system disk:

   On *hard disk* systems, enter:

   COPY A:XASM7.EXE E:*.*/V <CR>

   On *dual floppy drive* systems, enter:

   COPY A:XASM7.EXE B:*.*/V <CR>

3) Copy the Link Editor onto the hard disk or the system disk:

   On *hard disk* systems, enter:

   COPY A:LINKER.EXE E:*.*/V <CR>

   On *dual floppy drive* systems, enter:

   COPY A:LINKER.EXE B:*.*/V <CR>

G.2.3 Executing the Macro Assembler

To execute the Macro Assembler enter: XASM7

The command line parser prompts for the source, listing, and object file names:

Source File	Enter the source file name (if the source file does not have an extension, then type the file name with an explicit '.').
Listing File	Enter the output listing file name.
Object File	Enter the output object file name.

MS/PC-DOS creates defaults for the listing and object files and/or their extensions. The default extensions are:

- Source file - .ASM
- Listing file - .LST
- Object file - .MPO

A source file name can be followed by a semicolon, either on the command line or in response to a prompt; this causes the Macro Assembler to generate the default files without displaying further prompts.

Examples:

XASM7 <filename>.SRC;
- Uses <filename> with extension SRC.
- Generates defaults for the listing file <filename.LST> and object file <filename>.MPO.

XASM7 <filename>;
- Uses <filename> with default extension ASM.
- Generates defaults for the listing and object files as indicated above.

XASM7 <filename>,<newname>;
- Uses <filename> with default extension ASM.
- Generates listing file <newname>.LST and object file <newname>.MPO.

XASM7 <filename>,<newname>
- Uses <filename> with default extension ASM.
- Generates listing file <newname>.LST and prompts for object file name.

G.2.4 Executing the Link Editor

To execute the Linker enter: LINKER

The command line parser will prompt for the control, linkmap, and load file names.

Control File	Enter the control file name with extension (if the control file does not have an extension, type the file name with an explicit '.').
Map File	Enter the linkmap file name with extension.
Load File	Enter the load module file name with extension.

MS/PC-DOS generates defaults for the linkmap and load files and/or their extensions. The default extensions are:

- Control file - .CTL
- Linkmap file - .MAP
- Load file - .LOD

A source file name can be followed by a semicolon, either on the command line or in response to a prompt; this causes the Macro Assembler to generate the default files without displaying further prompts.

Examples:

LINKER <filename>.SRC;
- Uses <filename> with extension SRC.
- Generates defaults for the linkmap and load files as indicated above.

LINKER <filename>;
- Uses <filename> with default extension CTL.
- Generates defaults for the linkmap and load files as indicated above.

LINKER <filename>,<newname>;
- Uses <filename> with default extension CTL.
- Generates linkmap file <newname>.MAP and load file <newname>.LOD.

LINKER <filename>,<newname>
- Uses <filename> with default extension CTL.
- Generates linkmap file <newname>.MAP and prompts for the load file name.

G.2.5 Testing the Macro Assembler

Hard Disk Systems:

1) Copy the TEST1.ASM and TEST2.ASM files from the backup diskette onto the hard disk using the MS/PC-DOS COPY utility:

```
COPY A:*.ASM E:*.*/V <CR>
```

2) Execute the Macro Assembler using TEST1.ASM and TEST2.ASM as source files. In response to the system prompt, enter:

```
XASM7 TEST1;
```

The Assembler generates the default object file TEST1.MPO and default listing file TEST1.LST.

3) Compare the listing and object files just created to those on backup diskettes. Only lines which contains the date and time the files were created should be different.

- On *MS-DOS systems*, use the FILCOM utility:

```
FILCOM TEST1.MPO A:TEST1.MPO <CR>
FILCOM TEST1.LST A:TEST1.LST <CR>
FILCOM TEST2.MPO A:TEST2.MPO <CR>
FILCOM TEST2.LST A:TEST2.LST <CR>
```

MS/DOS will display the lines that are different.

- On *PC-DOS systems*, use the TYPE utility to print the contents of each file on the screen and visually check for differences:

```
TYPE TEST1.MPO <CR>
TYPE A:TEST1.MPO <CR>

TYPE TEST1.LST <CR>
TYPE A:TEST1.LST <CR>

TYPE TEST2.MPO <CR>
TYPE A:TEST2.MPO <CR>

TYPE TEST2.LST <CR>
TYPE A:TEST2.LST<CR>
```

Floppy Drive Systems:

1) Insert the backup diskette into the default floppy drive.

2) Execute the Macro Assembler using TEST1.ASM and TEST2.ASM as source files. It is important to use a different name for the object and listing files, otherwise the Assembler will write over these files on the backup diskette, and there will be no correct files to compare the created files to. In response to the system prompt, enter:

```
XASM7 TEST1,MYTEST1;
```

The Assembler generates object file MYTEST1.MPO and listing file MYTEST1.LST.

3) Compare the listing and object files just created to those on backup diskettes. Only lines which contains the date and time the files were created should be different.

- On *MS-DOS systems*, use the FILCOM utility:

```
FILCOM TEST1.MPO MYTEST1.MPO <CR>
FILCOM TEST1.LST MYTEST1.LST <CR>
FILCOM TEST2.MPO MYTEST2.MPO <CR>
FILCOM TEST2.LST MYTEST2.LST <CR>
```

MS/DOS will display the lines that are different.

- On *PC-DOS systems*, use the TYPE utility to print the contents of each file on the screen and visually check for differences:

```
TYPE TEST1.MPO <CR>
TYPE MYTEST1.MPO <CR>

TYPE TEST1.LST <CR>
TYPE MYTEST1.LST <CR>

TYPE TEST2.MPO <CR>
TYPE MYTEST2.MPO <CR>

TYPE TEST2.LST <CR>
TYPE MYTEST2.LST<CR>
```

G.2.6 Testing the Link Editor

Hard Disk Systems:

1) Copy the TEST.CTL, TEST1.MPO, and TEST2.MPO files from the backup diskette onto the hard disk using the MS/PC-DOS COPY utility:

```
COPY A:TEST.CTL E:*.*/V <CR>
COPY A:TEST*.MPO E:*.*/V <CR>
```

2) Execute the Link Editor using TEST.CTL as the control file. In response to the system prompt, enter:

```
LINKER TEST;
```

The Linker generates the default linkmap file TEST.MAP and default load file TEST.LOD.

3) Compare the listing and object files just created to those on backup diskettes. Only lines which contains the date and time the files were created should be different.

   - On *MS-DOS* systems, use the FILCOM utility:

     ```
     FILCOM TEST.MAP A:TEST.MAP <CR>
     FILCOM TEST.LOD A:TEST.LOD <CR>
     ```

     MS/DOS will display the lines that are different.

   - On *PC-DOS systems*, use the TYPE utility to print the contents of each file on the screen and visually check for differences:

     ```
     TYPE TEST.MAP <CR>
     TYPE A:TEST.MAP <CR>

     TYPE TEST.LOD <CR>
     TYPE A:TEST.LOD <CR>
     ```

Floppy Drive Systems:

1) Insert the backup diskette into the default floppy drive.

2) Execute the Link Editor using TEST.CTL as the control file. It is important to use a different name for the map and load files, otherwise the Linker will write over these files on the backup diskette, and there will be no correct files to compare the created files to. In response to the system prompt, enter:

   ```
   LINKER TEST,MYTEST;
   ```

   The Linker generates linkmap file MYTEST.MAP and load file MYTEST.LOD.

   - On *MS-DOS systems*, use the FILCOM utility:

     ```
     FILCOM TEST.MAP MYTEST.MAP <CR>
     FILCOM TEST.LOD MYTEST.LOD <CR>
     ```

     MS/DOS will display the lines that are different.

   - On *PC-DOS systems*, use the TYPE utility to print the contents of each file on the screen and visually check for differences:

     ```
     TYPE TEST.MAP <CR>
     TYPE MYTEST.MAP <CR>

     TYPE TEST.LOD <CR>
     TYPE MYTEST.LOD <CR>
     ```

G.3 IBM/MVS CrossWare Installation

This section explains how to install the TMS7000 CrossWare package on an IBM/MVS system.

G.3.1 Tape Transfer to Datasets

Section G.3.1.1 describes the files that are shipped on the product tape. They are grouped according to file type, i.e., all JCL files are in a dataset, all load modules are in a dataset, and all object modules are in a dataset. Section G.3.1.2 provides instructions for creating the partitioned datasets that will contain these files. Section G.3.1.3 contains the JCL needed to restore these files into the partitioned datasets on the virtual machine.

To submit a file, enter edit mode using the desired file, and type `SUBMIT` on the command line. This submits the file as a batch job.

G.3.1.1 Module Descriptions

The following lists describe the files provided on the tape, grouped according to modules:

- CNTL - Control Files (JCL)

  ASSEMBLE Invokes the assembler test program
  LINKASM Relinks the TMS7000 family Assembler
  LINKER Invokes the Link Editor test program
  LINKLINK Relinks the TMS7000 family Link Editor
  RANDINIT Invokes a utility that initializes random files (for the Assembler)

- LOAD - Load Modules

  ASM7000 The Assembler load module
  LINKER The Link Editor load module
  RANDINIT Random file initialization utility load module

- TEXT - Object Modules

  ASM7000 Assembler object file
  LINKER Link Editor object file
  TEST1 Benchmark test Assembler object code
  TEST2 Assembler test object code

- RUNTIME - Runtime Support Modules

  Contains the object modules for the TI Pascal runtime support needed to relink the Assembler and the Linker. They are not listed here, since there are about 240 members in this set.

- TEST - Source Modules

  TEST1 Test program used for Assembler and Link Editor
  TEST2 Test program used for Assembler and Link Editor

G.3.1.2 Creating the Datasets

Use the MVS dataset utility to create partitioned datasets with the following names and characteristics. (A different library name may be used to replace LIBNAME.)

- Create dataset LIBNAME.ASM7000.CNTL (library of JCL files)

Device Type	3350
Organization	PO
Record Format	FB
Record Length	80
Block Size	3200
1st Extent Tracks	10
Secondary Tracks	0
Directory Blocks	10

- Create dataset LIBNAME.ASM7000.LOAD (library of load modules)

Device Type	3350
Organization	PO
Record Format	U
Record Length	80
Block Size	13030
1st Extent Tracks	3
Secondary Tracks	0
Directory Blocks	30

- Create dataset LIBNAME.ASM7000.TEST (library of source code test programs)

Device Type	3350
Organization	PO
Record Format	FB
Record Length	80
Block Size	2960
1st Extent Tracks	1
Secondary Tracks	0
Directory Blocks	20

- Create dataset LIBNAME.ASM7000.TEXT (library of object modules)

Device Type	3350
Organization	PO
Record Format	FB
Record Length	80
Block Size	2960
1st Extent Tracks	1
Secondary Tracks	0
Directory Blocks	10

- Create dataset LIBNAME.ASM7000.RUNTIME (library of runtime support object modules)

Device Type	3350
Organization	PO
Record Format	U
Record Length	80
Block Size	10030
1st Extent Tracks	1
Secondary Tracks	0
Directory Blocks	50

G.3.1.3 Restoring the Tape

Use an editor to create a sequential file called **TRESTORE** which contains the JCL shown below. This JCL restores the tape. Insert the name of the tape (written on the tape label) in **<TAPE NAME>**. If a different library name was used for LIBNAME, insert it as the partitioned dataset name wherever the JCL uses LIBNAME. The member names provided should remain the same for the sake of clarity.

```
//RESTOR JOB      <job card>
//TAPEDMP         PROC   DSNX='DUMMY',LNO=1,FB=U,BSZ=3200
//STEP1 EXEC PGM=IEBCOPY
//SYSPRINT  DD SYSOUT=%
//INPDS     DD DSNAME=&DSNX,DISP=OLD
//BACKUP    DD DSNAME=<TAPE NAME>,UNIT=TAPE,DISP=OLD,
//          LABEL=(&LNO,NL),
//          DCB=(RECFM=&FB,LRECL=80,BLKSIZE=&BSZ,DEN=3),
//          VOL=(,RETAIN)
//SYSUT1    DD UNIT=SPACE,DISP=(NEW,DELETE),SPACE=(80,(60,45))
//SYSUT2    DD UNIT=SPACE,DISP=(NEW,DELETE),SPACE=(80,(60,45))
//       PEND
//DOIT1 EXEC TAPEDMP, DSNX='LIBNAME.ASM7000.CNTL',LNO=1,FB=FB,
//       BSZ=3200
//SYSIN     DD *
//          COPY OUTDD=INPDS,INDD=BACKUP
//DOIT2 EXEC TAPEDMP, DSNX='LIBNAME.ASM7000.LOAD',LNO=2,FB=FB,
//       BSZ=3200
//SYSIN     DD *
//          COPY OUTDD=INPDS,INDD=BACKUP
//DOIT3 EXEC TAPEDMP, DSNX='LIBNAME.ASM7000.TEXT',LNO=3,FB=FB,
//       BSZ=3200
//SYSIN     DD *
//          COPY OUTDD=INPDS,INDD=BACKUP
//DOIT4                              EXEC                    TAPEDMP,
          DSNX='LIBNAME.ASM7000.RUNTIME',LNO=4,FB=FB,
//       BSZ=3200
//SYSIN     DD *
//          COPY OUTDD=INPDS,INDD=BACKUP
//DOIT5                              EXEC                    TAPEDMP,
          DSNX='LIBNAME.ASM7000.RUNTIME',LNO=4,FB=FB,
//       BSZ=3200
//SYSIN     DD *
//          COPY OUTDD=INPDS,INDD=BACKUP
//
//
```

G.3.2 Installing the Assembler and Link Editor

The JCL in Section G.3.1.3 installs the following software components:

- *Assembler Load Modules*

  LIBNAME.ASM7000.LOAD(ASM7000) Contains the complete load module for the Macro Assembler. It may be executed as is, or used to relink the Assembler (see Section G.3.3).

  LIBNAME.ASM7000.LOAD(RANDINIT) The JCL uses this load module to initialize random files used by the Macro Assembler.

- *Assembler Object Modules*

  LIBNAME.ASM7000.TEXT(ASM7000)
  LIBNAME.ASM7000.RUNTIME This load module and this dataset relink the Assembler.

- *Assembler Control Files*

  LIBNAME.ASM7000.CNTL(ASSEMBLE) Executes the Macro Assembler. (See verification procedures, Section G.3.4.)

  LIBNAME.ASM7000.CNTL(LINKASM) Relinks the Macro Assembler.

- *Link Editor Load Modules*

  LIBNAME.ASM7000.LOAD(LINKER) Load module for the Link Editor. No other load modules are necessary for Link Editor execution.

- *Link Editor Object Modules*

  LIBNAME.ASM7000.TEXT(LINKER)
  LIBNAME.ASM7000.RUNTIME This file and this dataset relink the Link Editor.

- *Link Editor Control Files*

  LIBNAME.ASM7000.CNTL(LINKER) Executes the Link Editor. (See verification procedure, Section G.3.4.)

  LIBNAME.ASM7000.CNTL(LINKLINK) Relinks the Link Editor.

G.3.3 Relinking the Assembler and Link Editor

Assembler

Execute the following steps to relink the Assembler.

1) Edit the control file LIBNAME.ASM7000.CNTL(LINKASM).

2) Change LIBNAME to the correct partitioned dataset name where appropriate. In the data definition card below, insert the name of the dataset for the output load module. (It may be easier to use the load module library created above for verification.) Replace the load module name in the NAME card with the desired member name. The (R) specifies to replace an earlier version of the load module.

```
//SYSLMOD DD DISP=OLD,DSN=LIBNAME.ASM7000.LOAD
     .
     .
     .
NAME ASM7000(R)
```

3) Save the edited file and submit the JCL to the system. A condition code of zero indicates a successful link. Be sure to use the correct load module in the verification procedures in Section G.3.4.

Link Editor

The Link Editor load module may be executed as is. If the Link Editor is to be relinked on the new system, execute the following procedure:

1) Edit the control file LIBNAME.ASM7000.CNTL(LINKLINK).

2) Change LIBNAME to the correct partitioned dataset name where appropriate. In the data definition card below, insert the name of the dataset for the output load module. Replace the load module name in the NAME card with the member name desired. The (R) specifies to replace an earlier version of the load module.

```
//SYSLMOD    DD   DISP=OLD,DSN=LIBRARY.ASM7000.LOAD
  .
  .
  .
NAME LINKER(R)
```

3) Save the edited file and submit the JCL file to the system. A condition code of zero indicates a successful link. Use this load module in the Link Editor for verification procedures in Section G.3.4.

G.3.4 Verifying Installation

These verification procedures are not designed to perform an exhaustive test. They simply verify that the installation procedures were executed correctly. They also provide familiarity with the package's basic operation and data flow.

Software Components Used for Assembler Verification

- *Control Files*

  LIBNAME.ASM7000.CNTL(ASSEMBLE) Contains the JCL to execute the Assembler installation verification.

- *Load Modules*

  LIBNAME.ASM7000.LOAD(RANDINIT) Initializes the random files (direct access files) used by the Macro Assembler. If random file initialization is performed automatically on an open to a random file, this step is not necessary, and may be deleted from the JCL. If, however, the random file initialization is not performed automatically, the random files must be explicitly initialized as direct access files.

  LIBNAME.ASM7000.LOAD(ASM7000) Contains the load module for the Assembler. If the Assembler has been relinked, use the new load module name for verification.

- *Test Programs*

  LIBNAME.ASM7000.TEST(TEST1)
  LIBNAME.ASM7000.TEST(TEST2) Contain the test program module. These tests consist of assembly language programs containing directives, macro definitions, macro calls, and assembly instructions for each opcode.

Software Components Used for Link Editor Verification

- *Control File*

  LIBNAME.ASM7000.CNTL(LINKER) Contains the JCL to execute the Link Editor installation verification.

- *Load Module*

  LIBNAME.ASM7000.LOAD(LINKER) Contains the TMS7000 Link Editor. If the Link Editor was relinked on this system, use the new load module name.

- *Test Programs*

  LIBNAME.ASM7000.TEST This dataset contains two object modules, TEST1 and TEST2. This test links these modules together.

Assembler Verification Procedure

This procedure assembles a test program that contains all instruction opcodes, basic directives, macro definitions, and macro calls.

1) If the Assembler has been relinked, edit the file: LIBNAME.ASM7000.CNTL(ASSEMBLE). Substitute the correct load module and dataset names in the following JCL card:

```
//ASSEM  PROC ASM=ASM7000,STACK=10K,HEAP=100K
  .
  .
  .
//STEPLIB  DD DISP=SHR,LIBNAME.ASM7000.LOAD
```

2) Allocate an object output dataset called LIBNAME.ASM7000.OBJECT and specify it in the following DD card:

```
//ASMGO.OBJECT DD DSN=LIBNAME.ASM7000.OBJECT(TEST1),DISP=OLD
```

3) Save the file and submit the JCL to the system. A condition code of 0 indicates a successful assembly. There should be no error messages from the results of this assembly and the file LIBNAME.ASM7000.TEXT(TEST1).

The same procedure can be followed for source file TEST2 by simply replacing member name TEST1 with TEST2 in the ASMGO.OBJECT and ASMGO.SYSIN DD cards.

Link Editor Verification Procedure

This test may be performed with the test object modules provided on the tape, or it may be used in tandem with the Assembler test by using the object modules produced from testing the Assembler. Substitute the appropriate dataset and member names for the test modules desired.

1) If the Link Editor has been relinked, edit the JCL file, changing these JCL cards to the new load module dataset name:

```
//LINKER  PROC  LKED=LINKER,STACK=20K,HEAP=400K,TMPSIZE=1,
  .
  .
  .
 //STEPLIB  DD DSN=LIBNAME.ASM7000.LOAD,DISP=SHR
```

2) Create an output load module dataset called LIBNAME.ASM7000.LOAD3 and place the name in the following DD card:

```
//TESTIT EXEC LINK-
          ER.OBJLIB='LIBNAME.ASM7000.LOAD3',OBJMEM='LOAD3'
```

The next DD card in the JCL for executing the Link Editor (see Section G.3.6) is:

```
MYOBJXXX DD DSN=LIBNAME.ASM7000.TEXT,DISP=OLD
```

The MYOBJXXX DD card specifies the object input modules. If you want to test other object modules, substitute LIBNAME.ASM7000.OBJECT for the

dataset name and TEST1 and TEST2 for the member names in the INCLUDE statements in the JCL.

3) Save the edited file and submit the JCL to the system. A condition code of 0 indicates a successful link. The load object code will be in the file LIBNAME.ASM7000.LOAD3(LOAD3).

G.3.5 JCL for Executing the Assembler

This JCL is contained in the file LIBNAME.ASM7000.CNTL(ASSEMBLE).

```
//ASSEM   JOB 'NAME    000 000  0000000-00      000102 0512P C'
//*MAIN ORG=00000
//ASSEM   PROC ASM=ASM7000,STACK=10K,HEAP=100K
//            OBJLIB='&&OBJLIB',OBJMEM=ASM7000
//*****************************************************************
//*
//*   TMS7000 MACRO ASSEMBLER VERSION 2.1
//*
//*****************************************************************
//ASMGO EXEC PGM=&ASM,PAR='&STACK,&HEAP'
//*                         PROGRAM FILE
//STEPLIB         DD DISP=SHR,DSN=LIBNAME.ASM7000.LOAD
//*                         SOURCE FILE
//INPUT           DD DDNAME=SYSIN
//*                         INPUT FILE
//OBJECT          DD DSN=&OBJLIB(&OBJMEM),
//                DISP=(NEW,KEEP)
//                UNIT=SPACE,SPACE=(CYL,(3,1,10)),
//                DCB=(RECFM=FB,LRECL=80,BLKSIZE=2960)
//*                         OUTPUT FILE
//OUTPUT          DD SYSOUT=A
//*                         TEMPORARY FILE
//TEMPFILE        DD DISP=(NEW,DELETE),
//                UNIT=SPACE,SPACE=(CYL,1),
//                DCB=(RECFM=FB,LRECL=80,BLKSIZE=2960)
//NEWLIB          DD UNIT=SPACE,SPACE=(TRK,1),DISP=(NEW,PASS),
//                DCB=(DSORG=DA)
//                PEND
//*FORMAT PR,DDNAME=OBJECT,CONTROL=SINGLE
// EXEC ASSEM
//ASMGO.OBJECT    DD DSN=LIBNAME.ASM7000.TEXT(TEST1),
//                DISP=OLD
//ASMGO.OUTPUT    DD SYSOUT=A,DCB=RECFM=FBA
//ASMGO.SYSIN     DD DSN=LIBNAME.ASM7000.TEST(TEST1),DISP=SHR
//
```

G.3.6 JCL for Executing the Link Editor

This JCL is contained in the file LIBNAME.ASM7000.CNTL(LINKER).

```
//LINKER  JOB 'NAME    000 000  0000000-00      000102 0512P C',
//*MAIN ORG=00000
//LINKER PROC LKED=LINKER,STACK=20K,HEAP=100K,TMPSIZE=1
//       OBJLIB='TEMPLIB',OBJMEM=TEMPNAME
//LINK   EXEC PGM=&LKED,PARM=(&STACK,&HEAP)
//STEPLIB  DD DSN=LIBNAME.ASM7000.LOAD,DISP=SHR
//OUTPUT   DD SYSOUT=A
//INPUT    DD DDNAME=SYSIN
//TEMPFILE DD DISP=NEW,UNIT=SPACE,SPACE=(CYL,&TMPSIZE),
//         DCB=DSORG=DA
//OBJECT   DD DISP=SHR,DSN=&OBJLIB(&OBJMEM)
//         PEND
//TESTIT EXECLINKER,OBJLIB='LIBNAME.ASM7000.LOAD',OBJMEM='LOAD3'
//MYOBJXXX DD DSN=LIBNAME.ASM7000.TEXT,DISP=OLD
//SYSIN    DD *
 TASK JUNK
 DATA    0
 COMMON  128
 PROGRAM 256
 INCLUDE MYOBJXXX(TEST1)
 INCLUDE MYOBJXXX(TEST2)
 END
/*
```

G.3.7 JCL for Relinking the Assembler

This JCL is contained in the file LIBNAME.ASM7000.CNTL(LINKASM).

```
//LINKA   JOB 'NAME    000 000  0000000-00          000102 0512P
C',
//*MAIN    ORG=00000
//  EXEC   PGM=IEWL,PARM='MAP,LIST,LET,CALL,SIXE=(118K,24K)'
//SYSLIB   DD DISP=SHR,DSN=LIBNAME.ASM7000.RUNTIME
//SYSLIN   DD DISP=SHR,DSN=LIBNAME.ASM7000.TEXT(ASM7000)
//         DD DDNAME=SYSIN
//SYSPRINTDD SYSOUT=A
//SYSUT1   DD UNIT=SPACE,SPACE=(CYL,(1,1))
//SYSLMOD  DD DISP=OLD,DSN=LIBNAME.ASM7000.LOAD
//SYSIN    DD *
 ENTRY P$MAIN
 INCLUDE SYSLIB(STACLIKE)
 INCLUDE SYSLIB(ASMTEXT)
 INCLUDE SYSLIB(PUTREC)
 INCLUDE SYSLIB(MAIN)
 NAME ASM7000(R)
//
```

G.3.8 JCL for Relinking the Link Editor

This JCL is contained in the file LIBNAME.ASM7000.CNTL(LINKLINK).

```
//LINKA    JOB 'NAME     000 000  0000000-00       000102 0512P C',
//*MAIN     ORG=00000
//   EXEC   PGM=IEWL,PARM='MAP,LIST,LET,CALL,SIXE=(118K,24K)'
//SYSLIB    DD DISP=SHR,DSN=LIBNAME.ASM7000.RUNTIME
//SYSLIN    DD DISP=SHR,DSN=LIBNAME.ASM7000.TEXT(LINKER)
//          DD DDNAME=SYSIN
//SYSPRINT DD SYSOUT=A
//SYSUT1    DD UNIT=SPACE,SPACE=(CYL,(1,1))
//SYSLMOD   DD DISP=OLD,DSN=LIBNAME.ASM7000.LOAD
//SYSIN     DD *
 ENTRY P$MAIN
 INCLUDE SYSLIB(STACLIKE)
 INCLUDE SYSLIB(PUTREC)
 INCLUDE SYSLIB(ASCII$)
 INCLUDE SYSLIB(MAIN)
 NAME LINKER(R)
//
```

G.3.9 JCL for Random File Initialization

This JCL is contained in the file LIBNAME.ASM7000.CNTL(RANDINIT).

```
//LINKA    JOB 'NAME     000 000  0000000-00       000102 0512P C',
//*MAIN     ORG=00000
//   EXEC   PGM=RANDINIT
//STEPLIB    DD DISP=SHR,DSN=LIBNAME.ASM7000.LOAD
//OUTPUT     DD SYSOUT=A
//FILE1      DD DSN=LIBNAME.ASM7000.FILE,DISP=OLD,DCB=DSORG=DA
//INPUT      DD *
DDNAME=FILE1,LENTH=80,NUMBER=100
//
```

G.3.10 Using the COPY Directive

In Assembler Text:

The COPY statement syntax is:

```
[<label>]    COPY    <filename>    [<comment>]
```

where:

[<label>] is an optional label beginning in column 1.

<filename> has been defined on a DD card in the JCL. Filenames may be members of partitioned datasets or sequential files. Names may be delimited by parentheses, blanks, or periods.

[<comment>] is an optional comment.

```
        IDT 'TEST'
 * COPY STATEMENT TEST PROGRAM
        COPY DATASET(MEMBER)
        COPY SEQUEN
        END
```

In the JCL:

```
     .
     .
     .
//DATASET   DD DSN=LIBNAME.DATA.LIBRARY
//SEQUEN    DD DSN=LIBNAME.DATA.LIBRARY(FILE1)
     .
     .
     .
```

This example copies the file named MEMBER from the dataset LIBNAME.DATA.LIBRARY and the sequential file FILE1 from the same dataset.

G.3.11 Using the MLIB Directive

In Assembler Text:

The MLIB statement syntax is:

```
[<label>]    MLIB    '<pathname>'    [<comment>]
```

where:

[<label>] is an optional label beginning in column 1.

<pathname> is a quote enclosed filename, previously defined on a DD card in the JCL. The filename must be a partitioned dataset. Only one name may be specified for each MLIB directive.

[<comment>] is an optional comment.

```
        IDT    'TEST'
* MLIB STATEMENT TEST PROGRAM
A       BSS    2
B       DATA   >1000
        MLIB   'DATASET'
        MAC1   A,B
        END
```

In the JCL:

```
        .
        .
        .
//DATASET     DD DSN=LIBNAME.DATA.LIBRARY
        .
        .
        .
```

In this example, the MLIB statement causes the Assembler to search for the member MAC1 in the dataset LIBNAME.DATA.LIBRARY (since it is not a valid opcode or an internally defined macro). The Assembler first searches for a special member of the dataset named MLIST to determine if it should replace any opcodes. MLIST contains a list of all macros defined as members of the dataset.

G.4 IBM/CMS CrossWare Installation

This section contains directions for installing the TMS7000 Macro Assembler and Link Editor on an IBM/CMS system. The CrossWare tape was created with the CMS TAPE DUMP command.

G.4.1 Tape Files

The product tape contains the following files:

ASM7000	MODULE	Assembler executable module
ASM7000	OBJECT	Assembler object file
ASM7000	EXEC	Exec to invoke the assembler
ASMDEFX	EXEC	Exec to set up assembler filedefs
LINKER	MODULE	Linker executable module
LINKER	OBJECT	Linker object file
LINK7000	EXEC	Exec to invoke the linker
LINKDEFX	EXEC	Exec to set up linker filedefs
RELOAD	EXEC	Exec to re-generate executable modules
TEST	ASM7000	Sample assembler source
TEST	LIST7000	Sample assembler output listing
TEST	OBJ7000	Sample assembler output object
TEST1	ASM7000	Sample assembler source
TEST1	LIST7000	Sample assembler output listing
TEST1	OBJ7000	Sample assembler output object
TEST	LINKCTL	Sample link control file
TEST	OUTPUT	Sample linker map listing
TEST	OBJECT	Sample linker output object
TIPL	EXEC	Exec needed to generate load modules
ASCII$	TEXT	IBM object file needed to re-link
RUNTIME	TXTLIB	IBM object library for re-link
STACLIKE	TXTLIB	IBM object library for re-link

G.4.2 Restoring the Macro Assembler and Link Editor

1) Mount the tape.

   Set up a virtual tape drive with a density of 6250 BPI. The tape drive must be attached to the userid that is restoring the tape. CMS usually reserves virtual addresses 181 through 184 for tape devices. If, for example, a userid is attached to a tape drive at virtual address 181, CMS will display the following message on that userid's terminal:

   ```
   TAPE 181 ATTACHED
   ```

2) Use the TAPE SCAN command to display a list of the files on the tape:

```
TAPE SCAN  <enter>
```

This list should be the same as the list in Section G.4.1, followed by the message:

```
END-OF-FILE OR END-OF-TAPE
```

Rewind the tape before reading the files from it:

```
TAPE REW  <enter>
```

3) Use the TAPE LOAD command to read in the files on the tape.

> **Caution:**
>
> **Files loaded from tape replace files with the same filename, filetype, and filemode.**

The command syntax is:

```
TAPE LOAD <filename> <filetype> <filemode>
```

Two methods are recommended for using this command:

- Read one file at a time by specifying the individual filename, filetype, and filemode. This example loads file ASM7000 MODULE onto minidisk F.

  ```
  TAPE LOAD ASM7000 MODULE F <enter>
  ```

- Read all the files at once (placing them on the same minidisk). This example loads all the files on the tape to minidisk A.

  ```
  TAPE LOAD * * A  <enter>
  ```

Rewind the tape after loading the files.

G.4.3 Executing the Macro Assembler

To execute the TMS7000 Macro Assembler, enter:

```
ASM7000 <filename> <filemode>
```

<filename> is the name of the source file; it must have a filetype of *ASM7000*. The <filemode> is optional. If no filemode is specified, CMS will search all accessible disks and uses the first occurrence of `<filename> ASM7000`. If a filemode is specified, the Macro Assembler uses the file `<filename> ASM7000 <filemode>` as input.

The Macro Assembler creates three output files and places them on the A disk; it is the user's responsibility to assure there is enough available disk space. The Macro Assembler output files are:

```
<filename>  LIST7000  A  Listing file
<filename>  OBJ7000   A  Object file
<filename>  MESSAGE   A  Run-time support message file
```

G.4.4 Executing the Link Editor

To execute the TMS7000 Link Editor, enter:

```
LINK7000 <filename> <filemode>
```

<filename> is the name of the source file; it must have a filetype of *LINKCTL*. The <filemode> is optional. If no filemode is specified, CMS will search all accessible disks and uses the first occurrence of `<filename> LINKCTL`. If a filemode is specified, the Link Editor uses the file `<filename> LINKCTL <filemode>` as input.

The Link Editor creates three output files and places them on the A disk; it is the user's responsibility to assure there is enough available disk space. The Link Editor output files are:

`<filename>`	`OBJECT`	`A`	Output object module
`<filename>`	`OUTPUT`	`A`	Linker map listing
`<filename>`	`MESSAGE`	`A`	Run-time support message file

G.4.5 Testing the Macro Assembler

This test procedure verifies that the Macro Assembler has been installed correctly. These examples use files TEST ASM7000 and TEST1 ASM7000 as source files, and create the LIST7000, OBJ7000, and MESSAGE output files described in Section G.4.3. The examples assume that the files were loaded from the tape onto the A disk.

1) Copy the correct versions of the output files onto another disk (the Macro Assembler will write over these files on the A disk; copying them to another disk saves them for comparison). This example copies the files onto the B disk; if the B disk is not available, use the next available read/write disk.

```
COPYFILE TEST LIST7000 A = = B <enter>
COPYFILE TEST OBJ7000 A = = B <enter>
COPYFILE TEST1 LIST7000 A = = B <enter>
COPYFILE TEST1 OBJ7000 A = = B <enter>
```

2) Execute the Macro Assembler:

```
ASM7000 TEST <enter>
===> TMS 7000 Macro Assembler Started
===> Assembly for ' TEST ' complete , RC = ( 0 ).
R;

ASM7000 TEST1  <enter>
===> TMS 7000 Macro Assembler Started
===> Assembly for ' TEST1 ' complete , RC = ( 0 ).
R;
```

A revision code of 0 indicates a successful assembly.

3) Compare the output files created by the Macro Assembler to the output files that were shipped on the tape:

```
COMPARE TEST LIST7000 A TEST LIST7000 B <enter>
COMPARE TEST OBJ7000 A TEST OBJ7000 B <enter>
COMPARE TEST1 LIST7000 A TEST1 LIST7000 B <enter>
COMPARE TEST1 OBJ7000 A TEST1 OBJ7000 B <enter>
```

In each comparison, only lines containing times or dates should differ. For example,

```
COMPARE TEST LIST7000 A TEST LIST7000 B <enter> .
COMPARING TEST LIST7000 A WITH TEST LIST7000 B
TEST LIST7000 A  <line with time and/or date>
TEST LIST7000 B <same line with different time and/or date>
R;
```

G.4.6 Testing the Link Editor

This test procedure verifies that the Link Editor has been installed *correctly*. These examples use the file TEST LINKCTL as a source file, and create the OUTPUT, OBJECT, and MESSAGE output files described in Section G.4.4. The examples assume that the files were loaded from the tape onto the A disk.

1) Copy the correct versions of the output files onto another disk (the Link Editor will write over these files; copying them to another disk saves them for comparison). This example copies the files onto the B disk; if the B disk is not available, use the next available read/write disk.

```
COPYFILE TEST OUTPUT A = = B  <enter>
COPYFILE TEST OBJECT A = = B  <enter>
```

2) Execute the Link Editor:

```
LINK7000 TEST <enter>
....370 X 7000 CROSS LINK EDITOR V3.2 STARTED.....
===> RC = ( 0 ).
R;
```

A revision code of 0 indicates a successful link.

3) Compare the output files created by the Link Editor to the output files that were shipped on the tape:

```
COMPARE TEST OUTPUT A TEST OUTPUT B  <enter>
COMPARE TEST OBJECT A TEST OBJECT B  <enter>
```

In each comparison, only lines containing times or dates should differ. For example,

```
COMPARE TEST OUTPUT A TEST OUTPUT B  <enter> .
COMPARING TEST OUTPUT A WITH TEST OUTPUT B
TEST OUTPUT A  <line with time and/or date>
TEST OUTPUT B <same line with different time and/or date>
R;
```

G.4.7 Macro Assembler and Link Editor Regeneration

If the ASM7000 or LINK7000 execs are accidentally destroyed, they can be regenerated from the object files (ASM7000 OBJECT and LINKER OBJECT) by executing the RELOAD exec. RELOAD calls the TIPL exec to include the proper run-time files.

G.4.8 Using the MLIB Directive

The CMS implementation of the MLIB directive requires that macro libraries are logically grouped by filetype. For example, the macro definition files might be:

```
MAC1    MACRO A
MAC2    MACRO A
MAC3    MACRO A
```

In the assembler source file, the MLIB directive would look like this:

```
MLIB    'MACRO'
or
MLIB    'MACRO A'
or
MLIB    'MACRO *'
```

In the first MLIB example, the filemode is not given. CMS will search all minidisks in Search Order. In the second example, the filemode specifies the A disk, so only the A disk will be searched for macros. If this method is used, all macros called by the source file **must** be located on the A disk (or, if another disk is specified, on that disk). The third example is the same as the first example.

G.4.9 Using the COPY Directive

The CMS implementation of the COPY directive requires that the file(s) to be copied into the source file **must** have the same filetype as the source file. Otherwise, the copied file will not be copied into during assembly time, and no assembler error or warning will be issued. However, the copied file does not have to be in the same minidisk as the source file. In the assembler source file, the COPY directive syntax is:

```
COPY    SUB1
```

G.5 TI 990/DX10 CrossWare Installation

TMS7000 CrossWare for TI 990/DX10 is available on several types of media, including magnetic tape and hard discs. The magnetic tapes were created with the backup directory command (BD). The hard discs were created with the copy directory command (CD).

The CrossWare contains the TMS7000 Macro Assembler, Link Editor a utility to convert absolute TMS7000 object modules to a form acceptable to the standard PROM utility, and the PROM utility. (Absolute TMS7000 object modules can be generated by either the Assembler, using the AORG directive, or by the Linker, using the PROGRAM <absolute value> directive.)

The DVS7000 directory, contained on mag tape or hard disk, contains the following files:

```
PROCS                    LINK
PROGRAM                  M$LC
README                   PROM
ASM                      QUIT
CONVRT
```

G.5.1 Macro Assembler and Link Editor Installation

1) If your CrossWare package is contained on magnetic tape, you must transfer it to a hard disc before you can use it. Mount the tape and enter the following:

```
RD  <CR>

RESTORE DIRECTORY
  SEQUENTIAL ACCESS NAME:  MT01
      DIRECTORY PATHNAME:  <directory>.DVS7000
    LISTING ACCESS NAME:   <directory>.LST7000
                OPTIONS:   ADD
```

This places the files on the tape into the directory <directory>.DVS7000. To create a hard disc copy, execute a Copy Directory command:

```
CD  <CR>
```

The resulting directory is named DVS7000.

2) The directory DVS7000 may be used by:

   a) Copying it to the system disc,
   b) Changing the directory name with the Modify File command (MFN), or
   c) Leaving it on the hard disc.

3) At this point, you should read the instructions in <directory>.DVS7000.-README.

G.5.2 Executing the Macro Assembler

To execute the TMS7000 Macro Assembler, enter: ASM. The following prompts will appear:

```
ASSEMBLE 7000 SOURCE MODULE
              SOURCE FILE:  <access name>
              OBJECT FILE:  <access name>
             LISTING FILE:  <access name>
   FOREGROUND/BACKGROUND:  F
```

The Macro Assembler creates defaults for the listing and object files and/or their extensions. The default extensions are:

- Source file - .ASM
- Listing file - .LST
- Object file - .MPO

G.5.3 Executing the Link Editor

To execute the TMS7000 Link Editor, enter: LINK. The following prompts will appear:

```
LINK EDIT OBJECT MODULES
            CONTROL FILE:  <access name>
      LINKED OBJECT FILE:  <access name>
       LINK LISTING FILE:  <access name>
   FOREGROUND/BACKGROUND:  F
```

The Link Editor creates defaults for the listing and object files and/or their extensions. The default extensions are:

- Control file - .CTL
- Linkmap file - .MAP
- Load file - .LOD

G.5.4 Using the DX Conversion Utility

To invoke the DX conversion utility, type: CONVRT. The following prompts will appear:

```
7000 TO 9900 FORMAT CONVERSION UTILITY REV 1.0
         INPUT FILE:  <access name>
        OUTPUT FILE:  <access name>
```

G.5.5 Using the DX PROM Utility

To invoke the DX PROM utility, type: PROM. The following prompts will appear:

```
PROM PROGRAMMING UTILITY
          CRU ADDRESS:  <valid CRU address>
    INITIAL PROM TYPE:  <valid PROM/EPROM type>
          990/12 CRU?:  NO
```

H. Glossary

ADDR: Port A Data-Direction Register

ALU: Arithmetic Logic Unit

APORT: Port A Data Register

assembler: Any program that converts mnemonic and symbolic machine code into machine language

ASYNC: Communications Mode, bit 1 in the serial mode register (SMODE)

Asynchronous Communication mode: A mode used by the serial port to communicate with peripheral devices. Requires framing bits but does not require a synchronizing clock.

BPORT: Port B Data Register

BRKDT: Break Detect, bit 6 in the serial port Status Register (SSTAT)

C bit: Carry bit in the Status Register

CDDR: Port C Data-Direction Register

CHAR1, CHAR2: Number of Bits per Character, bits 2 and 3 in the serial mode register (SMODE)

CLK: Serial Clock Source, bit 6 in serial control register 1 (SCTL1)

CPORT: Port C Data Register

CRC: Customer Response Center

CrossWare: Texas Instruments macro assemblers and linkers

DDDR: Port D Data-Direction Register

DDR: Data Direction Register

Direct Memory Addressing mode: Uses a 16-bit address that contains an operand

DIP: Dual-inline package

directive: A mnemonic instruction to the assembler, executed during assembly

DPORT: Port D Data Register

Dual Register Addressing mode: Uses a source and a destination register as 8-bit operands

EC1: Timer 1 event counter

EC2: Timer 2 event counter

ER: Error Reset, bit 4 of serial control register 0 (SCTL0).

EVM: Evaluation module

expression: A sequence of symbols, contants, and operators, to which a numerical value can be assigned during assembly

Extended Addressing mode: An addressing mode which uses a 16-bit address

FE: Framing Error, bit 6 of the serial port status register (SSTAT)

FFE: form factor emulator; an EPROM or piggyback device which to emulates or replaces a masked-ROM device

Fosc: External oscillator frequency

Full-Expansion mode: A TMS7000 operating mode which extends addressing capability to the full 64K-byte limit

Halt mode: A low-power mode entered by the CMOS devices in which the on-chip timer logic is disabled

I bit: Global interrupt enable bit (in the Status Register)

Immediate Addressing mode: Uses an immediate 8-bit address

Indexed Addressing mode: Generates a 16-bit address by adding the contents of register B to a 16-bit direct memory address

IOCNT0: I/O control register 0

IOCNT1: I/O control register 1

IOCNT2: I/O control register 2

Isosynchronous Communication mode: A hybrid communications protocol which combines features of Asynchronous and Serial I/O communications; uses framing bits and a serial clock

link control file: Contains commands which control the link process

linker: Collects and interconnects relocatable elements to produce an absolute element

mask option: A device option, such as a clock option, which is placed on a manufacturing template, or mask, copying the actual circuit onto the silicon device; cannot be changed by software.

MC pin: Mode Control pin. When this pin is set to 1 (5 V), the Microprocessor mode of device operation is entered

Microprocessor mode: A mode of operation intended for applications which do not justify the use of on-chip ROM. All memory accesses except for internal RAM and on-chip Peripheral File locations are addressed externally.

MULTI: Multiprocessor mode, bit 0 of the serial mode register (SMODE)

N bit: Sign bit in the status register

NCRF: New Code Release Form

OE: Overrun Error, bit 4 in the serial port status register (SSTAT)

PC: Program Counter

PE: Parity Error, bit 3 in the serial port status register (SSTAT)

PEN: Parity Enable, bit 4 in the serial mode register (SMODE)

Peripheral-Expansion mode: An operating mode which allows use of on-chip ROM and also allows addressing off-chip locations (peripheral devices)

Peripheral File Addressing mode: Refers to instructions which perform I/O tasks; either the source or the destination is a peripheral file register

Peripheral File instructions: MOVP, ANDP, ORP, XORP, BTJOP, and BTJZP

PEVEN: Parity Even, bit 5 of the serial mode register (SMODE)

PF: Peripheral File

piggyback: A device used as a form-factor emulator for masked-ROM devices

PLA: Programmed Logic Array

PLCC: Plastic-leaded chip carrier

Program Counter Relative Addressing mode: Used by all jump instructions; adds an offset to the PC value to form the address

RF: Register File

RTC: Regional Technology Center

RXBUF: Receiver Buffer

RXD: Receive Data, line A5

RXEN: Receiver Enable, bit 2 in serial control register 0 (SCTL0).

RXRDY: Receiver Ready, bit 1 in the serial status register (SSTAT).

RXSHF: RX Shift register

SCAT: Strip Chip Architecture Technology

SCLK: serial clock source, pin A6

SCTL0: Serial port control register 0

SCTL1: Serial port control register 1

Serial I/O Mode: A serial-port communication mode which uses an external clock to synchronize the receiver and the transmitter; Stop bits are also used

Single Register Addressing mode: Uses a single register that contains an 8-bit operand

Single-Chip mode: An operation mode in which the device functions as a standalone microcomputer with no off-chip memory expansion bus

SIO: Serial I/O or Communications mode, bit 6, serial mode register (SMODE)

SLEEP: Sleep, bit 5, serial control register 1 (SCTL1)

SMODE: Serial port mode register

SP: Stack Pointer

SSTAT: Serial port status register

ST: Status Register

START: Timer 3 start, bit 7, serial control register 1 (SCTL1)

STOP: Stop, bit 7, serial mode register (SMODE)

TMP: Prefix for devices that conform to the final electrical specifications but have not completed quality and reliability verification

TMS: Device prefix for fully qualified production devices

TMX: Device prefix for experimental devices that are not representative of the device's final electrical specifications

TXBUF: Transmitter Buffer, write-only PF register P23

TXD: Transmission data, uses line B3

TXEN: Transmit Enable, bit 0, serial control register 0 (SCTL0)

TXRDY: Transmitter Ready, bit 0, serial port Status Register (SSTAT)

TXSHF: transmitter shift register

T1CTL: Timer 1 control register

T1CTL0: Timer 1 control register 0/LSB capture reload register value

T1CTL1: Timer 1 control register 1/MSB readout reload register

T1DATA: Timer 1 data register

T1LSDATA: Timer 1 LSB decrementer latch/LSB decrementer value

T1MSDATA: Timer 1 MSB decrementer latch/MSB readout latch

T1OUT: Timer 1 output

T2CTL: Timer 2 control register

T2CTL0: Timer 2 control register 0/LSB capture latch value

T2CTL1: Timer 2 control register 1/MSB readout reload register

T2DATA: Timer 2 data register

T2OUT: Timer 2 output

T2LSDATA: Timer 2 LSB decrementer latch/LSB decrementer value

T2MSDATA: Timer 2 MSB decrementer latch/MSB readout latch

T3DATA: Timer 3 data register

T3ENB: Timer 3 Enable, bit 2, serial control register 1 (SCTL1)

T3FLG: Timer 3 Flag, bit 3, serial control register 1 (SCTL1)

UR: Software UART reset, bit 6, serial control register 0 (SCTL0)

Wake-Up mode: A low-power mode entered by the CMOS devices in which the oscillator and timer logic remain active

WU bit: Wake-Up, bit 4, serial control register 1 (SCTL1)

WUT: Wake-Up temporary flag

XDS: Extended Development Support

Z bit: zero bit, Status Register

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Z