CTM-PER

Continuous-Period Counter

CTM-PER Continuous-Period Counter

Manual Part Number: 24826

Printed: March 1990

Rev. 1.0

WARNING
Keithley MetraByte assumes no liability for damages consequent to the use of this product. This product is not designed with components of a level of reliability suitable for use in life support or critical applications.

WARRANTY INFORMATION

All products manufactured by Keithley MetraByte are warranted against defective materials and worksmanship for a period of one year from the date of delivery to the original purchaser. Any product that is found to be defective within the warranty period will, at the option of Keithley MetraByte, be repaired or replaced. This warranty does not apply to products damaged by improper use.

CONTENTS

Chapter 1
INTRODUCTION

1.1 GENERAL

The CTM-PER is a PC-computer accessory board for monitoring timing changes in an on-going TTL signal. The board measures consecutive periods of a TTL signal and makes the results available for review, analysis, etc. For example, the results can be presented to the PC monitor in an array of consecutive measurements.

TTL signal frequencies may range from 0 (DC) to 80 KHz. The 80KHz upper frequency limit actually depends on frequency limits of the computer. In some computers, the upper frequency limit will be no more than 20 KHz.)

An important part of the CTM-PER package is the distributed software. The software enables the user to set up the board, to specify start/stop parameters for board operation, to control data flow, and to determine data formats for the array. The distributed software also contains CALL subroutines for use with BASIC, QUICKBASIC, PASCAL, C, and FORTRAN. In addition, there are commented examples and utility setup programs with sources.

CTM-PER software supports all common memory models for the following languages: Microsoft C (V4.0-5.1), Microsoft Quick-C (V1.0-2.0), Turbo C (V1.0-2.0), Microsoft Pascal (V3.0-4.0), Turbo Pascal (V3.0-5.0), Microsoft FORTRAN (V4.0-5.0), Lahey Personal Fortran (V1.0-2.0), QuickBASIC (V4.0 & higher), and GW, COMPAQ, and IBM BASIC (V2.0 & higher).

Typical CTM-PER applications include monitoring a Doppler signal and measuring output intervals of a rotating sensor. BNC connectors labeled SIGNAL and GATE are available at the board's rear plate. GATE can be programmed with selective polarity and used to enable measurements. SIGNAL can measure the timing of positive, negative, or both edges.

1.2 FUNCTIONAL DETAIL

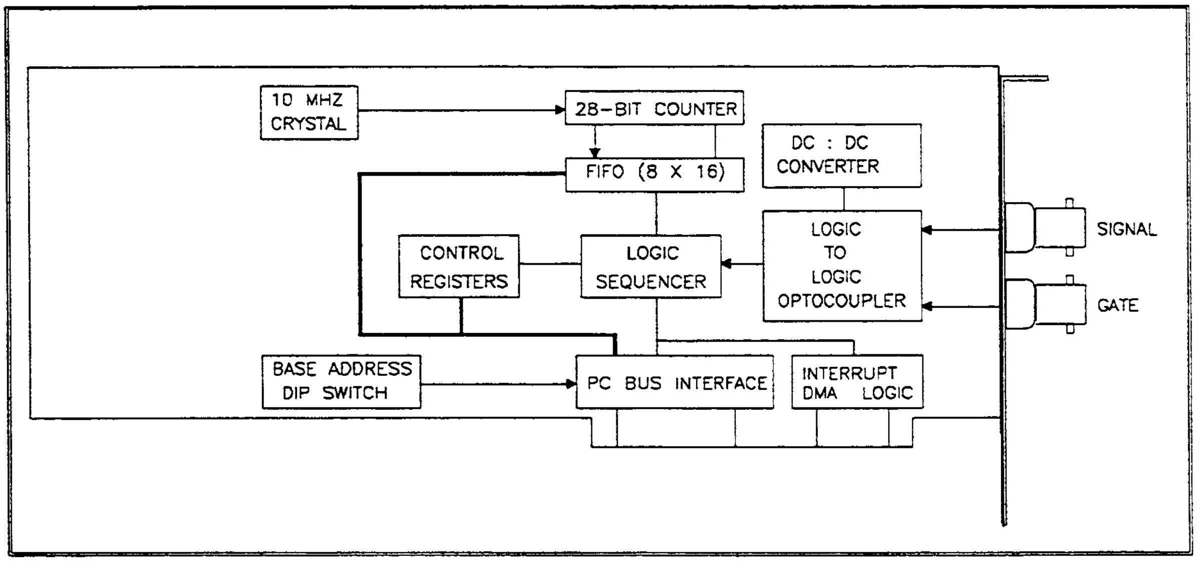


Figure 1-1. Block diagram of the CTM-PER Board.

The CTM-PER relies on an internal, crystal-controlled, 10MHz oscillator clock and a 28-bit up-counter register, as shown in Figure 1-1. It detects a signal edge by sensing a change in level. On detecting a signal edge, it loads the value of the 28-bit counter and the state of the GATE and SIGNAL into FIFO (First In, First Out) memory as four bytes (32 bits). Succeeding level changes cause successive counter values to load into FIFO memory.

While FIFO memory loads and unloads continuously, a slowing of the unload rate can cause a backup. FIFO memory can hold a backup of up to four counter values. Any attempt to load beyond the four-value limit causes an overrun error.

Load speed is a factor only while FIFO memory is holding less than four counter values (FIFO can hold no more than four counter values); it is a function of the rate at which the on-board state machine detects level changes and is limited to 1.0 MHz.

Unload speed depends on the technique for unloading FIFO and storing data in computer memory. DMA (Direct Memory Access) is the fastest technique and is the one used by the CTM-PER. This technique yields a speed range of 20KHz to 80KHz, depending on the computer. The lower limit of a period being timed can be considered as the roll-over time for the 10 MHz clock and the 28 bit counter: about 26.8 seconds.

Information about a measured period or interval is taken from the difference between two consecutive counter values. Because the clock is 10MHz, the resolution of the period is 0.1 microsecond.

Total data accumulation is limited only by total computer memory (or disk capacity if a product like MetraByte's STREAMER is used to stream data onto disk). The upper limit of frequency is controlled by two factors: the speeds at which FIFO memory loads and is unloaded.

1.3 SPECIFICATIONS

1.3.1 SIGNAL and GATE Inputs

Connectors:	Type: BNC (2)
SIGNAL/GATE Load:	1 LSTTL UNIT LOAD
High-Level Input:	2.0 VDC (minimum)
Low-Level Input:	0.8 VDC (maximum)
High-Level Input Current:	40 µA (maximum)
Low-Level Input Current:	-0.4 mA (maximum)
Absolute Maximum Input:	7 VDC
Input Protection:	100 ohms in series with input and 6.8 V zener
Transient Immunity: Common Mode)	5000 V/us (min)
Isolation Voltage:	500 VDC (Input to computer)

1.3.2 Crystal Clock and Counter

Frequency:	10.000 MHz
Frequency Stability:	+/-0.01% (+/-100 ppm)
Measurement Resolution:	0.1 microsecond
Measurement Rollover:	26.8 seconds
Measurement Bits:	28

1.3.3 Programmed Control

GATE:	Positive, negative, or none
Minimum GATE Trigger:	100 nsec.
Signal Edges:	Positive, negative, or both
Maximum Signal Frequency:	1.0 MHz (burst of 4), 20 to 80 KHz (continuous DMA) (Computer dependent)
Data Acquisition Modes:	Programmed, interrupt, or DMA
Interrupt Levels:	2, 3, 4, 5, 6, 7, or none
DMA Levels:	1, 3, or none

1.3.4 Environmental

Operating Temperature:	0 to 50 Degree Celcius
Bus:	IBM PC/XT Compatible
Power Requirements:	+5 VDC @ 600 mA (typical)

Chapter 2
INSTALLATION & SETUP

2.1 INTRODUCTION

CTM-PER distribution software is on a 5.25", 360K floppy diskette (DOS 2.10 format); it is also available on a 3.5" diskette. This software is licensed to permit multiple copies for non-commercial use, not for resale.

Installation of your CTM-PER Software will require the following procedures:

- Making a working copy of your CTM-PER Distribution diskette(s).
- Unpacking and inspecting the board.
- Selecting a Base Address for your CTM-PER driver board.
- Installation.

2.2 COPYING THE DISTRIBUTION DISKETTES

Make working copies of your CTM-PER Distribution Software diskette(s) and store your original copy in a safe place. To copy the Distribution diskette(s), use a procedure (from the two that follow) that suits your particular computer configuration. The first procedure is for a computer with dual floppy-disk drives, the second is for a computer with both a floppy- and a hard-disk drive.

2.2.1 Procedure for Dual Floppy-Disk Computers

1. With your your computer on and booted, place your DOS disk containing DISKCOPY.EXE in the A Drive.
2. Log to the A Drive by typing **A:** <Enter>
3. At the DOS *A>* prompt, type **A:DISKCOPY A: B:** <Enter>
4. Insert the *source* diskette (the CTM-PER Distribution diskette) into the A Drive. The system will prompt you through the disk copying process, asking you to insert the *target* diskette into the B Drive.
6. When you have completed copying, the computer will ask **COPY ANOTHER (Y/N)?**
7. If you are copying two diskettes, respond by typing **Y** <Enter> and follow the prompts to copy the second diskette. Otherwise, type **N** <Enter>.
8. Put the original CTM-PER diskette(s) in a safe place for storage. Then label your back-up disk(s) as your CTM-PER working copies.

2.2.2 Procedure for Hard-Disk Computers

1. With your computer on and booted, log to the drive to be used for your CTM-PER Distribution files. (In most cases, this will be the C Drive.)
2. The following instructions create a directory named *CTM* for the CTM-PER Distribution files. If you prefer a name other than *CTM*, substitute your preference in place of *CTM* in Step a. (immediately following). If you intend to use an existing directory, skip these instructions, log to that directory, and go to Step 3.
   a. Make a CTM subdirectory by typing **MD CTM** <Enter>
   b. Change to the CTM directory by typing **CD CTM** <Enter>

3. Place the CTM-PER Distribution Disk into a floppy Drive A.
3. Type the copy-all-files command, as **COPY A:*.*** <Enter>
5. If copying two diskettes, repeat Steps 3 and 4 for the second diskette.

With the CTM-PER Distribution files copied to your hard drive, put the original Utility Diskette(s) in a safe storage area.

2.3 UNPACKING AND INSPECTING

After you remove the wrapped board from its outer shipping carton, proceed as follows:

1. Place one hand firmly on a metal portion of the computer chassis (the computer must be turned Off and grounded). You place your hand on the chassis to drain off static electricity from the package and your body, thereby preventing damage to board components.
2. Allow a moment for static electricity discharge; carefully unwrap the board from its anti-static wrapping material.
3. Inspect the board for signs of damage. If any damage is apparent, return the board to the factory.
4. Check the contents of your CTM-PER package against its packing list to be sure the order is complete. Report any missing items to MetraByte immediately.

You may find it advisable to retain the packing material in case the board must be returned to the factory.

2.4 SELECTING AND SETTING THE BASE ADDRESS

The CTM-PER requires four consecutive address locations in I/O space. Since some I/O address locations are already occupied by internal I/O and other peripheral cards, you have the option of resetting the CTM-PER I/O base address by means of an on-board Base Address DIP switch. The Base Address switch is located as shown in Figure 2-1, and it appears as shown in Figure 2-2. Referring to Figure 2-2, you set the base address on a four-byte boundary to 3FC Hex (300 Hex is shown).

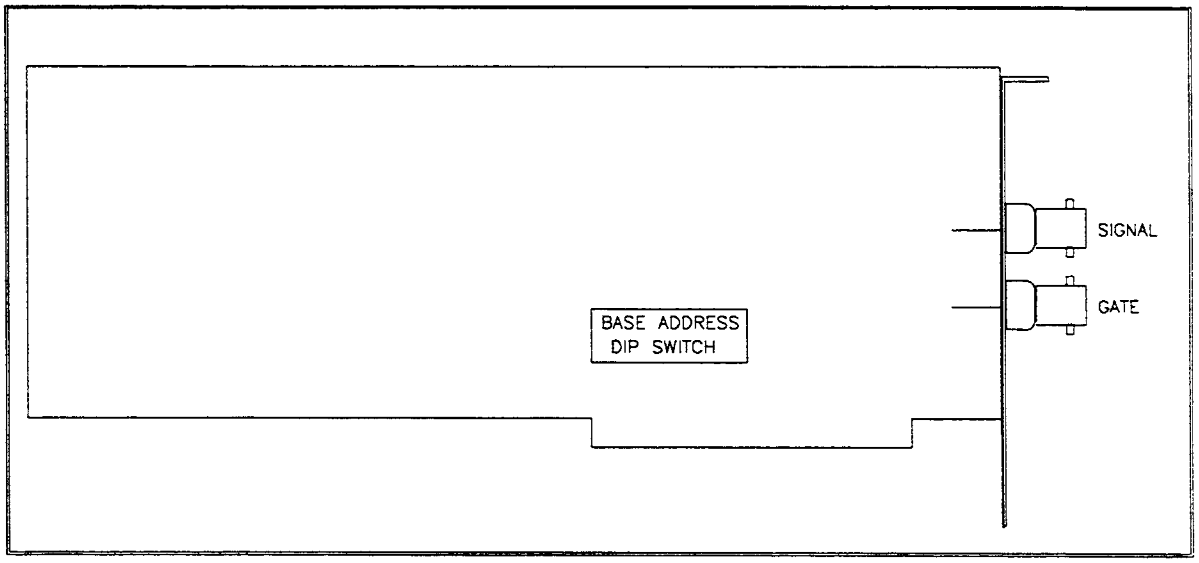


Figure 2-1. CTM-PER board outline, showing Base Address switch location.

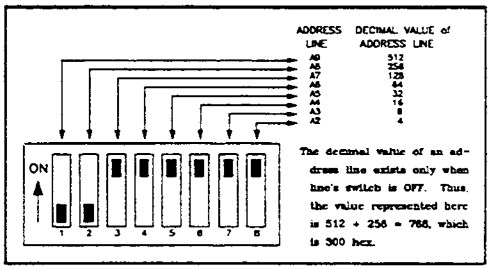

ADDRESS LINE
DECIMAL VALUE of ADDRESS LINE
A9 512
A8 256
A7 128
A6 64
A5 32
A4 16
A3 8
A2 4
ON
1
2
3
4
5
6
7
8
The decimal value of an address line exists only when line's switch is OFF. Thus, the value represented here is 512 + 256 = 768, which is 300 hex.

Figure 2-2. Base Address switch.

The board is preset for a base address of 300 HEX. If this address is not satisfactory, your distribution software contains a program called called *DIPSW.EXE* that asks for base address and shows a picture of the DIP switch setting. Use this program by logging to its location (to the floppy drive containing the distribution diskette or to the hard-drive directory containing your distribution files) and typing **DIPSW**

When the computer responds with *Desired base address?*, type your choice in decimal or IBM &H___ format and press <*Enter*>.

2.5 HARDWARE INSTALLATION

To install the CTM-PER in a PC, proceed as follows.

WARNING: ANY ATTEMPT TO INSERT OR REMOVE ANY ADAPTER BOARD WITH THE COMPUTER POWER ON COULD DAMAGE YOUR COMPUTER!

1. Turn Off power to the PC and all attached equipment.
2. Remove the cover of the PC as follows: First remove the cover-mounting screws from the rear panel of the computer. Then, slide the cover of the computer about 3/4 of the way forward. Finally, tilt the cover upwards and remove.
3. Choose an available option slot. Loosen and remove the screw at the top of the blank adapter plate. Then slide the plate up and out to remove.
4. Hold the CTM-PER board in one hand placing your other hand on any metallic part of the PC chassis (but not on any components). This will safely discharge any static electricity from your body.
5. Make sure the board switches have been properly set (refer to the preceding section).
5. Align the board connector with the desired accessory slot and with the corresponding rear-panel slot. Gently press the board downward into the socket. Secure the board in place by inserting the rear-panel adapter-plate screw.
7. Replace the computer's cover. Tilt the cover up and slide it onto the system's base, making sure the front of the cover is under the rail along the front of the frame. Replace the mounting screws.
8. Plug in all cords and cables. Turn the power to the computer back on.

MetraByte recommends that you retain the static-shield packaging for possible future removal and handling of the CTM-PER board.

2.6 CABLING

Connect the cable carrying the TTL signal to be monitored to BNC Connector J1 (Figure 2-1), which is labelled SIGNAL. BNC Connector J2 (GATE) is an optional TTL input. If GATE is not enabled in software, this connector can be left open.

2.7 REGISTER MAPS

Board Address + 0:

READ:	C3	C2	C1	C0	GT/	SG/	0	0
READ:	C11	C10	C9	C8	C7	C6	C5	C4
READ:	C19	C18	C17	C16	C15	C14	C13	C12
READ:	C27	C26	C25	C24	C23	C22	C21	C20

FIFO DATA (Data stored as 4 Bytes).
1st = LS Nibble Count + GATE + SIGNAL.
2nd = 3rd Byte Count (2^11 to 2^4).
3rd = 2nd Byte Count (2^19 to 2^12).
4th = MSB Count (2^27 to 2^20).

SG/ = 1/0 of signal after Trigger;
1 -> Falling Edge Trigger (negative signal trigger),
0 -> Rising Edge Trigger (positive signal trigger).

GT/ = 0/1 of GATE after Trigger.

WRITE:	-	-	-	-	-	-	-	-

Board Address + 1: (Not Used).

Board Address + 2:

READ:	CLR	OVR	GPL	GTE	GAT	SIG	S1	S0
WRITE:	CLR	-	GPL	GTE	TEN	TST	S1	S0

CLR = CLEAR; 0 -> Count to 0, FIFO cleared and OVR to 0.
1 -> Counters Count (Normal) + FIFO Normal.

OVR = OverRun Error; 1 -> FIFO overrun occurred since CLEAR.

GPL = GATE Polarity; 1/0 -> GATE 1/0 to count, 0/1 to hold.

GTE = GATE Enable; 1 -> GATE used, 0 -> GATE ignored.

GAT = GATE; 1/0 -> GATE Level at Trigger.

SIG = SIGNAL; 1/0 -> SIGNAL Level at Trigger.

TEN = 0 -> GATE used as Trigger, 1 -> GATE used as gate.

TST = Stop signal; 0 -> Stop data collection, 1 -> Collect.

S1 = Single or Dual Edge Signal Trigger; 0 -> Single, 1 -> Dual.

S0 = Positive or Negative Signal Trigger; 0 -> Positive, 1 -> Negative.

Board Address + 3:

READ:	DOR	IT/	DME	LEV	INE	IL2	IL1	IL0
WRITE:	-	-	DME	LEV	INE	IL2	IL1	IL0

DOR = DATA-OUT-READY; 1 -> Data In FIFO.

IT/ = INTERRUPT; 0 -> Interrupt pending.

DME = DMA ENABLE; 1 -> Enable DMA, 0 -> Disable.

LEV = DMA LEVEL; 0 -> Level 1, 1 -> Level 3.

INE = INTERRUPT ENABLE on data (if not DMA);
INTERRUPT ENABLE on TC (if DMA); 1 -> Enable, 0 -> Disable.

IL2, IL1, IL0 = INTERRUPT LEVEL;

000 -> None	100 -> IRQ4
001 -> None	101 -> IRQ5
010 -> IRQ2	110 -> IRQ6
011 -> IRQ3	111 -> IRQ7

Chapter 3 PROGRAMMING

3.1 INTRODUCTION

The CTM-PER is programmable at the lowest level using input and output instructions. In BASIC these are the INP(X) and OUT X,Y functions. Assembly language and most other high level languages have equivalent instructions.

To simplify program generation, the distribution software contains the I/O driver routine *CTMPER.BIN*. This routine is accessible from BASIC using a single-line CALL statement, and it covers the majority of common operating modes.

The benefits of using CTMPER.BIN are largely in significant reduction of programming time. The driver also supports data collection on interrupt or DMA. Note, however, that BASIC has no interrupt or DMA processing functions, and so-called *background* data collection using these methods is available only by using the CALL routines.

3.2 LOADING THE CTMPER.BIN DRIVER ROUTINE (BASIC)

To use the CTMPER.BIN driver, load it into memory. Avoid loading it over any part of memory used by another program. An example of loading this routine using IBM BASIC is as follows:

```
100  CLEAR, 48*1024                        'CONTRACT WORKSPACE TO 48K
110  DEF SEG=0
120  SG=256*PEEK(&H511)+PEEK(&H510)        'BASIC WORKSPACE SEGMENT
130  SG=SG+48*1024/16
140  DEF SEG=SG
150  BLOAD "CTMPER.BIN",0                  'LOAD IN ASSEMBLY DRIVER
160  CTMPER=0
170  DIM D%(15)                            'DECLARE ARRAY
180  FLAG%=0                               'DECLARE VARIABLE
```

A second option applies when you have memory outside the BASIC workspace; it should be used for non-IBM BASIC (when the PEEKs of line 120 above will not work).

```
210  DEF SEG=&H7000
220  BLOAD "CTMPER.BIN",0                  'LOAD IN ASSEMBLY DRIVER
230  CTMPER=0
240  DIM D%(15)                            'DECLARE ARRAY
250  FLAG%=0                               'DECLARE VARIABLE
```

Before you try loading outside the workspace, be sure you really do have unused memory (large enough for the CTMPER.BIN file) at the location in line 210. You can change the line 210 DEF SEG=&H7000 and experiment with loading the CALL routine at other locations. Usually any clash with another program's use of the memory results in a failure to exit and return from the routine. The computer hangs up, and the only cure is to switch OFF, wait a few seconds, and turn on the power. Try a different memory location until the program works.

3.3 CALL STATEMENT FORMAT (BASIC)

Prior to entering the CALL, the DEF SEG=SG statement sets the segment address at which the CALL subroutine is located. The CALL statement for the CTMPER.BIN driver must use the form:

```
xxxxx CALL CTMPER(MD%, D%(0), FLAG%)
```

CTMPER is the address offset from the current segment of memory, as defined in the last DEF SEG statement. In all the examples, the current segment is defined to correspond with the starting address of the CALL routine. This offset is therefore zero and CTMPER=0 (see line 160).

The three variables within brackets are known as the CALL parameters; their meaning depends on the Mode, as described in the following sections. On executing the CALL, the addresses of the variables (pointers) are passed in the sequence written to BASIC's stack. The CALL routine unloads these pointers from the stack and uses them to locate the variables in BASIC's data space so data can be exchanged. Four important format requirements must be met:

1. The CALL parameters are positional. The subroutine knows nothing of the names of the variables, just their locations from the order of their pointers on the stack. The parameters must always be written in the correct order:

   (mode, data, errors)

2. The CALL routine expects its parameters to be integer-type variables and will write and read to the variables on this basis.
3. You cannot perform any arithmetic functions within the parameter list brackets of the CALL statement. For example, the following is an **illegal** statement:

   ```
   CALL CTMPER(MD%+2,D%(0)+8,FLAG%)
   ```

4. You cannot use constants for any of the parameters in the CALL statement. For example, the following is an illegal statement:

   ```
   CALL CTMPER(7,2,FLAG%)
   ```

Apart from these restrictions, you can name the integer variables what you want; the names in the examples are just convenient conventions. Strictly, you should declare the variables before executing the CALL.

3.4 USE OF THE CALL ROUTINE

The following subsections contain details and examples of using the CALL routine in all ten CTM-PER Modes. Note that *delta* is defined as the difference between the current and last data value; the *delta* is scaled by a power of two and stored as 16 bits (an unsigned integer). The Modes are selected by the MD% parameter in the CALL as follows:

MODE (MD%)	FUNCTION
0	Initialize, store CTM-PER base address, interrupt level, and DMA level.
1	Start the counter or to stop the counter and stop data collection. Shuts down interrupts and/or DMA.

(continued on next page)

MODE (MD%)	FUNCTION
2	Setup gate, edge, and scaling parameters.
3	Start data collection and return next data.
4	Start data collection and return array of data.
5	Start data collection and return array of deltas.
6	Start data collection of data into memory via interrupt.
7	Start data collection of deltas into memory via interrupt.
8	Start data collection of data into memory via DMA.
9	Transfer data/delta from memory into array.
10	Check status of data collection.

Note that the mode used for data collection depends on what else the computer is doing, the computer's speed, and the speed of the signal being measured. In all cases, it is recommended that the user vary the signal frequency, monitor the overrun error, and check the data for his/her particular computer and application program in order to establish full performance details.

MODE	MODE SPEED
3,4,5	slow (several hundred Hertz)
6,7	medium (several thousand Hertz)
8	high (up to 20 to 80 KHz)

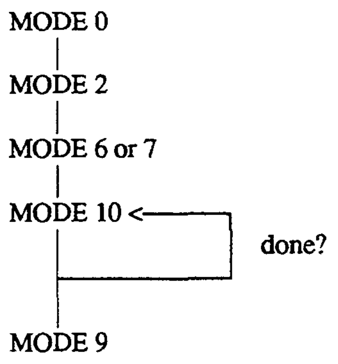
3.5 CALL SEQUENCE

Mode 0 must always be called initially. Mode 1 is optional to start the sequence and should be used to terminate interrupt or DMA operation when an operation is aborted before normal completion. Mode 2 must be called at least once before starting data collection. The normal minimum sequences of calls are as follows:

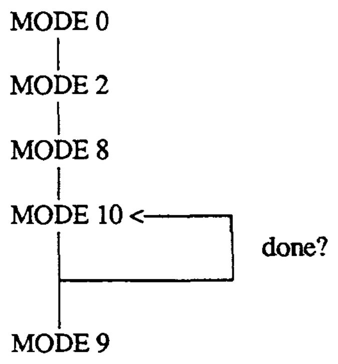
PROGRAMMED COLLECTION OF DATA/DELTA

```
MODE 0
  |
MODE 2
  |
MODE 3, 4, or 5
```

INTERRUPT COLLECTION OF DATA/DELTA



DMA COLLECTION OF DATA/DELTA



3.6 MODE CALL DESCRIPTIONS

3.6.1 MODE 0 - INITIALIZE

Mode 0 checks whether the base I/O address is in the legal range of 256 - 1020 (Hex 100 - 3FC) for the IBM PC. If not, an error exit occurs. The user-selected interrupt and DMA levels are then unloaded from the data array, checked and stored.

Data is passed in array:

ENTRY:	D%(0) = Base I/O address D%(1) = Interrupt level (0 for disabled) D%(2) = DMA level (0 for disabled)
EXIT:	D%(0) through D%(15) - unchanged
ERRORS:	Error #0, no error Error #3, base I/O address <255 or >1020 Error #4, interrupt level not 0 or <2 or >7 Error #5, DMA level not 0 or 1 or 3

EXAMPLE:

```
200   MD%=0
210   D%(0)=768 : D%(1)=3 : D%(2)=1
220   CALL CTMPER(MD%,D%(0),FLAG%)
230   IF FLAG%<> 0 THEN GOTO 1000
```

3.6.2 MODE 1 - START/STOP Clock

Mode 1 is used to start or stop the counter. Data collection is not started by this command (only the clock). The stop command clears any pending data out of the FIFO. An error is reported if, when the CTM-PER is stopped, an overrun error is pending. The clock may also be turned on through one of the data collection modes; MODE 1 is used to start the clock only if it is to be started prior to the start of data collection. The stop command will also shutdown ongoing interrupts or DMA operations.

Data is passed in array:

ENTRY:	D%(0) = 0 for stop, else for start
EXIT:	D%(0) through D%(15) - unchanged
ERRORS:	Error #0, no error Error #1, driver not initialized Error #6, overrun error pending

EXAMPLE:

```
580   MD%=1
590   D%(0)=0 '0=STOP  ELSE=START CLOCK
600   CALL CTMPER(MD%,D%(0),FLAG%)
610   IF FLAG%<> 0 THEN GOTO 1000
```

3.6.3 MODE 2 - Setup GATE, EDGES, SCALING

Mode 2 is used to setup (or change) the EDGE and GATE parameters for data collection and the SCALING factor for deltas. Data collection is not actually started with this mode. The scaling factor is used to reduce the delta count to 16 bits of resolution.

DELTA = $(T_{(I)} - T_{(I-1)})$ / 2^SCALE

where $T_{(I)}$ = current count
$T_{(I-1)}$ = last count
SCALE = D%(4) parameter (0 to 28)

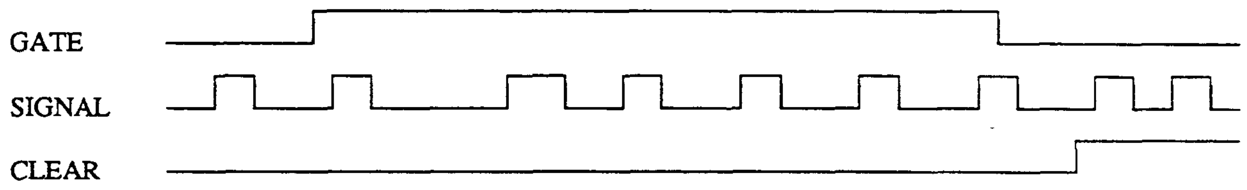
Data is passed in array:-

ENTRY:	D%(0) = Gate enable, 1 for gate used D%(1) = Gate polarity, 1/0 for 1/0 active D%(2) = Edges, 0 -> single, 1-> dual D%(3) = If single edge, 0 -> pos., 1 -> neg. D%(4) = Scaling factor power of 2 (0 to 28) D%(5) = 0 -> gate as trigger, 1 -> gate as gate
EXIT:	D%(0) through D%(14) - unchanged D%(15) - control word (for debug use only)

ERRORS:	Error #0, no error
	Error #1, driver not initialized
	Error #7, illegal gate parameters
	Error #8, illegal scale factor

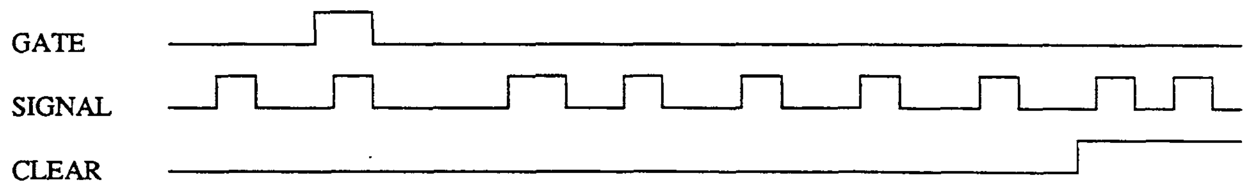
EXAMPLE:

```
250   MD%=2
260   D%(0)=0 '0=NOT GATED, 1=GATE USED
270   D%(1)=0 'GATE POLARITY 1/0 FOR 1/0 ACTIVE
280   D%(2)=0 'EDGES, 0=SINGLE  1=BOTH
290   D%(3)=0 'IF SINGLE EDGE, 0=POS  1=NEG
300   D%(4)=0 'SCALE FACTOR FOR DELTA MEASUREMENT
310   D%(5)=0 '0=TRIGGERED, 1=GATED
320           'NOTE:  DELTAS ARE 0.1 MICROSECONDS COUNTS TO 65535.
330           'IN OTHER WORDS, PERIODS TO 6.5535 MILLISECONDS CAN BE
340           'MEASURED WITHOUT SCALING.  IF LONGER PERIODS ARE TO BE
350           'MEASURED, D%(4) CAN BE SET TO ANY POWER OF 2, 0 TO 28.
360   CALL CTMPER(MD%,D%(0),FLAG%)
370   IF FLAG%<> 0 THEN GOTO 1000
```



```
                  |         |      |      |      |      |
COUNT  ---100   --200--   ---300--      -400--        500-----0--
                  |         |      |      |      |      |
DATA             180       270    330    390    450    510
                  |         |      |      |      |      |
DELTA            180        90     60     60     60     60
```

Figure 2-3. GATE and SIGNAL waveforms with GATE Enabled, positive polarity, GATE used as GATE, single-positive-edge triggering.



```
                  |         |      |      |      |      |
COUNT  ---100   --200--   ---300--      -400--        500-----0--
                  |         |      |      |      |      |
DATA             180       270    330    390    450    510
                  |         |      |      |      |      |
DELTA            180        90     60     60     60     60
```

Figure 2-4. GATE and SIGNAL waveforms with GATE Enabled, positive polarity, GATE used as Trigger, single-positive-edge triggering.

3.6.4 MODE 3 - Data Collection, Single

Mode 3 is used to collect data, one point at a time, under program control. The program must be able to keep up with the data or else an overrun error will occur. Pressing any key, once Mode 3 is started, will produce an immediate return to calling program with error code 11.

Data is passed in array:

ENTRY:	D%(0) = Last LSB (2^3 to 2^0, GAT/, SIG/, 00)
	D%(1) = Last 3rd Byte (2^11 to 2^4)
	D%(2) = Last 2rd Byte (2^19 to 2^12)
	D%(3) = Last MSB (2^27 to 2^20)

Note: Use D%(0)=D%(1)=D%(2)=D%(3)=0 initially

EXIT:	D%(0) = Current LSB (2^3 to 2^0, GAT/, SIG/, 00)
	D%(1) = Current 3rd Byte (2^11 to 2^4)
	D%(2) = Current 2rd Byte (2^19 to 2^12)
	D%(3) = Current MSB (2^27 to 2^20)
	D%(4) = Current scaled DELTA (unsigned integer)
	D%(5) through D%(15) - unchanged
ERRORS:	Error #0, no error
	Error #1, driver not initialized
	Error #6, overrun error pending
	Error #9, control not setup (Mode 2 call)
	Error #10, DELTA overflow, >65535
	NOTE: D%(0) to D%(3) set correctly, DELTA, D%(4), set to 0
	Error #11, Keyboard termination

3.6.5 MODE 4 - Data Collection, Multiple

Mode 4 is used to collect an array of data under program control. Data is stored in the array using two words of the integer array. The first word being the first two bytes out of the FIFO (the 2^11 bit to 2^0 bit plus GATE/, SIGNAL/, and two zeros). The second word being the next two bytes out of the FIFO (the 2^27 bit to 2^12 bit). The program must be able to keep up with the data or else an overrun error will occur. The data is passed as two successive words in the array. Pressing any key, once Mode 4 is started, will produce an immediate return to calling program with error code 11.

Data is passed in array:

ENTRY:	D%(0) = Number of conversions required
	NOTE: Array size (integer words) must be twice the number of conversions
	D%(1) = Segment of array (-1 if caller's segment)
	D%(2) = Offset of array
EXIT:	D%(0) through D%(15) - unchanged
ERRORS:	Error #0, no error
	Error #1, driver not initialized
	Error #6, overrun error pending
	Error #9, control not setup (Mode 2 call)
	Error #11, Keyboard termination
	Error #12, Sample count 0 or negative

EXAMPLE:

```
100   DIM X%(19),X(19)
 .
 .
180   MD%=4
190   D%(0)=10 : D%(1)=-1 : D%(2)=VARPTR(X%(0))
200   CALL CTMPER(MD%,D%(0),FLAG%)
210   IF FLAG% <> 0 THEN GOTO 1000
220   FOR I=0 TO 19
230   IF X%(I)<0 THEN X(I)=65536!+X%(I) ELSE X(I)=X%(I)
240   NEXT
250   FOR I=0 TO 18 STEP 2
260   T=INT(X(I)/16)+X(I+1)*4096
270   PRINT T
280   NEXT
```

3.6.6 MODE 5 - Data Collection, Multiple Delta

Mode 5 is used to collect an array of deltas under program control. The program must be able to keep up with the data or else an overrun error will occur. An initial value of zero for the data is assumed. Note that the DELTA overflow error is not indicated in this mode but the DELTA value comes out zero. The delta has been scaled by the parameter setup in Mode 2. Pressing any key, once Mode 5 is started, produces an immediate return to calling program with Error Code 11.

Data is passed in array:

ENTRY:	D%(0) = Number of conversion DELTAs required D%(1) = Segment of array (-1 if caller's segment) D%(2) = Offset of array
EXIT:	D%(0) through D%(15) - unchanged
ERRORS:	Error #0, no error Error #1, driver not initialized Error #6, overrun error pending Error #9, control not setup (Mode 2 call) Error #11, Keyboard termination Error #12, Sample count 0 or negative

EXAMPLE:

```
200   MD%=5
210   D%(0)=20 : D%(1)=-1 : D%(2)=VARPTR(X%(0))
220   CALL CTMPER(MD%,D%(0),FLAG%)
230   IF FLAG%<>0 THEN GOTO 1000
240   FOR I=0 TO 19
250   IF X%(I)<0 THEN X=65536!+X%(I) ELSE X=X%(I)
260   PRINT X*2^SCALING
270   NEXT
```

3.6.7 MODE 6 - Data Collection Via Interrupt

Mode 6 is used to collect data into memory via interrupt control. The program must be able to keep up with the data or else an overrun error will occur. MODE 9 can be used to move the data (two successive words) or deltas (one word), into an array. Data collection is stopped and the FIFO cleared when the sample count is done or when an overrun is encountered.

Data is passed in array:-

ENTRY:	D%(0) = Number of conversion required NOTE: MEMORY size in words must be twice D%(0) (four times in bytes) D%(1) = Segment of MEMORY (-1 if caller's segment) D%(2) = Offset of MEMORY to receive data D%(3) = Recycle flag: 0 -> Non-recycle, done flag set after number of samples 1 -> Recycle, samples continuously written and re-written
EXIT:	D%(0) through D%(1)5 - unchanged
ERRORS:	Error #0, no error Error #1, Driver not initialized Error #6, Overrun error pending Error #9, Control not setup (Mode 2 call) Error #12, Sample count 0 or negative Error #13, DMA/Interrupt already active Error #16, Interrupt must be installed for this mode.

EXAMPLE:

```
210      MD%=6
220      D%(0)=20 : D%(1)=&H7000 : D%(2)=0 : D%(3)=0
230      CALL CTMPER(MD%,D%(0),FLAG%)
240      IF FLAG%<>0 THEN GOTO 1000
```

3.6.8 MODE 7 - Delta Collection Via Interrupt

Mode 7 is used to collect Deltas (one unsigned integer word) into memory via interrupt. The program must be able to keep up with the data or an overrun error occurs. MODE 9 can be used to move the Deltas, passed as one word, into an array. Data collection is stopped and the FIFO cleared when the sample count is done or when an overrun is encountered. For collection of Deltas, an initial value of zero is used. Deltas greater then 65535 are indicated by a zero, no other error flag is used for the Delta overflow.

Data is passed in array:

ENTRY:	D%(0) = Number of conversions required D%(1) = Segment of MEMORY (-1 if caller's segment) D%(2) = Offset of MEMORY to receive data D%(3) = Recycle flag 0 -> Non-recycle, done flag set after number of samples 1 -> Recycle, samples continuously written and re-written
EXIT:	D%(0) through D%(15) - unchanged

ERRORS:	Error #0, no error Error #1, Driver not initialized Error #6, Overrun error pending Error #9, Control not setup (Mode 2 call) Error #12, Sample count 0 or negative Error #13, DMA/Interrupt already active Error #16, Interrupt must be installed for this mode.

EXAMPLE:

```
210  MD%=7
220  D%(0)=20 : D%(1)=&H7000 : D%(2)=0 : D%(3)=0
230  CALL CTMPER(MD%,D%(0),FLAG%)
240  IF FLAG%<>0 THEN GOTO 1000
```

3.6.9 MODE 8 - Data Collection Via DMA

Mode 8 is used to collect data via DMA. In general, this is the mode normally used to collect data. MODE 9 can be used to move the data or Deltas, passed as two successive words or one word, into an array.

Data is passed in array:

ENTRY:	D%(0) = Number of samples required (1 to 16384) D%(1) = Segment of MEMORY (-1 if caller's segment) D%(2) = Offset of MEMORY to receive data NOTE: two words (four bytes) per sample D%(3) = Recycle flag 0 -> Non-recycle, done flag set after number of samples 1 -> Recycle, samples continuously written and re-written
EXIT:	D%(0) through D%(15) - unchanged
ERRORS:	Error #0, no error Error #1, Driver not initialized Error #6, Overrun error pending Error #9, Control not setup (Mode 2 call) Error #12, Sample count 0 or negative Error #13, DMA/Interrupt already active Error #15, DMA wrap around of page

EXAMPLE:

```
210  MD%=8
220  D%(0)=1024 : D%(1)=&H7000 : D%(2)=0 : D%(3)=0
230  CALL CTMPER(MD%,D%(0),FLAG%)
240  IF FLAG%<>0 THEN GOTO 1000
250  MD%=10
260  CALL CTMPER(MD%,D%(0),FLAG%)
270  IF D%(0)<> 1 THEN GOTO 2000           'MUST BE DMA OPERATION
280  PRINT "SAMPLE COUNT = ";D%(2)
290  IF D%(1)=1 THEN GOTO 260              'WAIT ON DONE
```

3.6.10 MODE 9 - Transfer From Memory To Array

Mode 9 transfers data from any segment/offset of memory to an integer array in BASIC's workspace. Data can be converted in the process from data to Deltas. NOTE that the initial value of the count used is zero. NOTE ALSO that for languages other than BASIC, this mode is not usually needed; it can be used to convert data to Delta for any language.

Data is passed in array:

```
ENTRY:        D%(0) = Number of samples to transfer (1 to 32767)
              D%(1) = Source segment in memory
              D%(2) = Source offset in memory
              D%(3) = Starting sample offset number
                      0 to 32767 for deltas
                      0 to 16383 for data
                           offset (bytes) = samples * 2 for deltas
                           offset (bytes) = samples * 4 for data
              D%(4) = Segment of destination array (-1 if caller's segment)
              D%(5) = Offset of destination array
                      NOTE: Two array (2 words) positions per data
                           One array (1 word) position per delta
              D%(6) = data/DELTA flag
                      0 -> data
                      1 -> DELTA
                      2 -> data available, convert to DELTA

EXIT:         D%(0) through D%(1)5 - unchanged

ERRORS:       Error #0, no error
              Error #1, Driver not initialized
              Error #14, Illegal start offset or number of samples.
              Error #10, DELTA overflow, >65536
                      NOTE: Non-fatal, DELTA set to 0
```

EXAMPLE:

```
310   MD%=9
320   D%(0)=20 : D%(1)=&H7000 : D%(2)=0 : D%(3)=0
330   D%(4)=-1 : D%(5)=VARPTR(X%(0)) : D%(6)=2
340   CALL CTMPER(MD%,D%(0),FLAG%)
360   FOR I=0 TO 19
370   IF X%(I)<0 THEN X=65536!+X%(I) ELSE X=X%(I)
380   PRINT X
390   NEXT
```

3.6.11 MODE 10 - Monitor Status

Mode 10 is used to monitor the status of the data collection during interrupt or DMA operations.

Data is passed in array:

```
ENTRY:        D%(0) through D%(15) - not used
```

EXIT:

D%(0) = Type of operation (last or current)
- 0 = none or programmed
- 1 = DMA
- 2 = interrupt

D%(1) = Status
- 0 = done or inactive
- 1 = active

D%(2) = Current sample count

D%(3) = Error flag (set only after status D%(1) inactive, cleared by this mode)
- 0 = no error
- 1 = overrun error

D%(4) = Read of base+2 (control register)
see register map

D%(5) = Read of base+3 (interrupt/DMA register)
see register map

ERRORS:

Error #0, no error
Error #1, Driver not initialized

EXAMPLE:

```
250 MD%=10
260 CALL CTMPER(MD%,D%(0),FLAG%)
270 IF D%(0)<> 1 THEN GOTO 2000     'MUST BE DMA OPERATION
280 PRINT "SAMPLE COUNT = ";D%(2)
290 IF D%(1)=1 THEN GOTO 260        'WAIT ON DONE
```

..or..

```
250 MD%=10
260 CALL CTMPER(MD%,D%(0),FLAG%)
270 IF D%(0)<> 2 THEN GOTO 2000              'MUST BE INTERRUPT
280 PRINT "SAMPLE COUNT = ";D%(2)
290 IF D%(1)=1 THEN GOTO 260                 'WAIT ON DONE
```

Chapter 4
CALIBRATION & TEST

The CTM-PER board requires no adjustments. Accuracy is soley a function the 10 MHz Crystal and can be checked using an external counter on the 10 MHz Crystal or by inputting a known frequency to the SIGNAL input of the board and allowing the CTM-PER to measure the periods. It is important that you use a very stable and accurate source for this purpose.

Chapter 5
FACTORY RETURN INFORMATION

Before returning any equipment for repair, please call 508/880-3000 to notify MetraByte's technical service personnel. If possible, a technical representative will diagnose and resolve your problem by telephone. If a telephone resolution is not possible, the technical representative will issue you a Return Material Authorization (RMA) number and ask you to return the equipment. Please reference the RMA number in any documentation regarding the equipment and on the outside of the shipping container.

Note that if you are submitting your equipment for repair under warranty, you must furnish the invoice number and date of purchase.

When returning equipment for repair, please include the following information:

1. Your name, address, and telephone number.
2. The invoice number and date of equipment purchase.
3. A description of the problem or its symptoms.

Repackage the equipment. Handle it with ground protection; use its original anti-static wrapping, if possible.

Ship the equipment to

Repair Department
Keithley MetraByte
440 Myles Standish Boulevard
Taunton, Massachusetts 02780

Telephone 508/880-3000
Telex 503989
FAX 508/880-0179

Be sure to reference the RMA number on the outside of the package!

Appendix A
SUMMARY OF ERROR CODES

Error codes are returned in the FLAG% variable as follows:

CODE	ERROR DESCRIPTION
0	No Errors
1	Driver not initialized.
2	Mode number <0 or >N.
3	Invalid base address, <256 or >1020.
4	Interrupt level out of range, not 0 or <2 or >7.
5	DMA level not 0 or 1 or 3.
6	Overrun error encountered.
7	Illegal gate parameters
8	Illegal scale factor
9	Control not setup (Mode 2 call)
10	DELTA overflow, >65535
11	Keyboard termination
12	Sample count 0 or negative
13	DMA/Interrupt already active
14	Illegal start offset or number of samples
15	DMA wrap around of page
16	This mode needs an Interrupt (can't be 0)

Appendix B

CTM-PER: PASCAL, C, FORTRAN DRIVERS FOR CTM-PER

TABLE OF CONTENTS

B.1 INTRODUCTION

B.1.1 CTM-PER General Description

Keithley MetraByte's CTM-PER is a programming tool for writing data acquisition and control routines in Pascal, C, and Fortran for for the CTM-PER. CTM-PER supports all memory models for the following languages; Microsoft C (V4.0-5.1), Microsoft Quick-C (V1.0-2.0), Turbo C (V1.0-2.0), Microsoft PASCAL (V3.0-4.0), Turbo PASCAL (V3.0-5.0), Microsoft FORTRAN (V4.0-4.1), Lahey Personal Fortran (V1.0-2.0), QuickBASIC (V4.0 & higher), and GW, COMPAQ, and IBM BASIC (V2.0 & higher). CTM-PER consists of several assembly-language drivers and example programs for each supported language. This Section is structured to illustrate useage for each memory model of each supported language, and it includes an example program at the end of each language section. Full source listings are included in the distribution software. You should be familiar with the board's operating MODES, PARAMETERS, and ERROR codes before attempting CTM-PER implementation.

B.1.2 Implementation

Software drivers of the CTM-PER are limited to the actual language interface for the supported languages. To simplify programming and illustrate actual language interface, the following sections contain a very brief introductory explanation followed by an actual example for each language. Each interface driver (implemented via a CALL statement) consists of three position-dependent parameters, namely MODE, ARGUMENT (or PARAM), and FLAG. MODE is the type of function to be executed by the board. PARAM is the function-dependent arguments required for execution. FLAG is the error number, if any, corresponding to selected MODE.

B.2 MICROSOFT 'C' (V4.0-5.0) & QUICK'C' (V1.0-2.0)

B.2.1 Small Model

Model:	Small ("/AS") switch on command line
Passes:	Word size pointers (offset, no DS register)
Sequence:	Arguments Passed Right to Left
Default Calling Convention:	Arguments Passed by Value(Passing pointers to a subroutine is considered pass-by-value convention)

Example

'C' Call:	mscs_ctmper (&Mode, Params, &Flag);
'C' Declaration:	extern void mscs_ctmper(int*,int*);
.ASM Subroutine:	

The following assembly code shows how the driver handles user arguments:

```
_mscs_ctmper proc near
                push bp          ; save base pointer
                mov bp,sp        ; save stack pointer
                .                ; [bp+4] holds offset of Mode
                .                ; [bp+6] holds offset of Params
                .                ; [bp+8] holds offset of Flag
                .                ; Program execution here
                .                ;
                .                ;
                pop bp           ;restore bp & sp prior to exit
                ret              ;return
_mscs_ctmper endp
```

Other

This information is provided for those wishing to create their own drivers:

- _mscs_ctmper is declared "PUBLIC" in the .ASM file
- mscs_ctmper is declared "extern" in the "C" file
- The .ASM file contains the ".model small" directive (MASM & TASM only)
- Add leading underscore "_" to all mscs_ctmper occurrences in .ASM file
- mscs_ctmper is a near call
- mscs_ctmper must be in a segment fname_TEXT (where fname is the name of the file where mscs_ctmper resides) if .ASM file contains mixed model procedures.

B.2.2 Medium Model

Model:	Medium ("/AM") switch on command line
Passes:	Word size pointers (offset, no DS register)
Sequence:	Arguments Passed Right to Left
Default Calling Convention:	Arguments Passed by Value

Example

'C' Call:	mscm_ctmper (&Mode, Params, &Flag);
'C' Declaration:	extern void mscm_ctmper(int*,int*,int*);
.ASM Subroutine:	

The following assembly code shows how the driver handles user arguments:

```
_mscm_ctmper proc far                 ; far CALL (dword return address)
                  push bp             ; save base pointer
                  mov bp,sp           ; save stack pointer
                  .                   ; [bp+6] holds offset of Mode
                  .                   ; [bp+8] holds offset of Params
                  .                   ; [bp+10] holds offset of Flag
                  .                   ; Program execution here
                  .                   ;
                  .                   ;
                  pop bp              ;restore bp & sp prior to exit
                  ret                 ;return
_mscm_ctmper endp
```

Other

This information is provided for those wishing to create their own drivers:

- _mscm_ctmper is declared "PUBLIC" in the .ASM file
- mscm_ctmper is declared "extern" in the "C" file
- The .ASM file contains the ".model medium" directive (MASM & TASM only)
- Add leading underscore "_" to all mscm_ctmper occurrences in .ASM file
- mscm_ctmper is a far call
- mscm_ctmper must be in a segment fname_TEXT (where fname is the name of the file where mscm_ctmper resides), else Linker returns an error.

B.2.3 Large Model

Model:	Large ("/AL") switch on command line
Passes:	dword size pointers (offset and DS register)
Sequence:	Arguments Passed Right to Left
Default Calling Convention:	Arguments Passed by Value

Example

'C' Call:	mscm_ctmper (&Mode, Params, &Flag);
'C' Declaration:	extern void mscl_ctmper(int*,int*,int*);
.ASM Subroutine:	

The following assembly code shows how the driver handles user arguments:

```
_mscl_ctmper proc far                 ; far CALL (dword return address)
                push bp               ; save base pointer
                mov bp,sp             ; save stack pointer
                .                     ; [bp+6] holds offset of Mode
                .                     ; [bp+10] holds offset of Params
                .                     ; [bp+14] holds offset of Flag
                .                     ; Program execution here
                .                     ;
                .                     ;
                pop bp                ;restore bp & sp prior to exit
                ret                   ;return
_mscl_ctmper endp
```

Other

This information is provided for those wishing to create their own drivers:

- _mscl_ctmper is declared "PUBLIC" in the .ASM file
- mscl_ctmper is declared "extern" in the "C" file
- The .ASM file contains the ".model large" directive (MASM & TASM only)
- Add leading underscore "_" to all mscl_ctmper occurrences in .ASM file
- mscl_ctmper is a far call
- mscm_ctmper must be in a segment fname_TEXT (where fname is the name of the file where mscl_ctmper resides), else Linker returns an error.

B.2.4 Microsoft 'C' Example

```
/*************************************************************************/
/*  MSCEXAMPLE.C
/*  CTM-PER EXAMPLE OF MODE 0
/*  USING MICROSOFT C MEDIUM MODEL
/*************************************************************************/

#include  "stdio.h"
#include  "conio.h"

   extern mscm_ctmper(int*,int*,int*);              /* declare driver call


main()
{
   int Mode, Flag, Params[15];

   /* Initialize CTM-PER using Mode 0

   Mode = 0;
   Params[0] = 768;                                 /* Base Address of Board
   Params[1] = 2;                                        /* Interrupt level
   Params[2] = 3;                                        /* DMA level
   Params[3] = 0;                                        /* No Auto-Calibration

      mscm_ctmper(&Mode, Params, &Flag);
      if(Flag !=0)
         {
            printf("\n\nMode 0 Error FLag = %d\n",Flag);
         }
   REMAINDER OF CODE
   .
   .
   .
}
```

B.3 BORLAND TURBO 'C' (V1.0-2.0)

B.3.1 Small Model

Model:	Small ("-ms") switch on command line
Passes:	word size pointers (offset, no DS register)
Sequence:	Arguments Passed Right to Left
Default Calling Convention:	Arguments Passed by Value

Example

'C' Call:	tcs_ctmper (&Mode, Params, &Flag);
'C' Declaration:	extern void tcs_ctmper(int*,int*,int*);
.ASM Subroutine:	

The following assembly code shows how the driver handles user arguments:

```
_tcs_ctmper proc near
                    push bp             ; save base pointer
                    mov bp,sp           ; save stack pointer
                    .                   ; [bp+4] holds offset of Mode
                    .                   ; [bp+6] holds offset of Params
                    .                   ; [bp+8] holds offset of Flag
                    .                   ; Program execution here
                    .                   ;
                    .                   ;
                    pop bp              ;restore bp & sp prior to exit
                    ret                 ;return
_tcs_ctmper endp
```

Other

This information is provided for those wishing to create their own drivers:

- _tcs_ctmper is declared "PUBLIC" in the .ASM file
- tcs_ctmper is declared "extern" in the "C" file
- The .ASM file contains the ".model small" directive (MASM & TASM only)
- Add leading underscore "_" to all tcs_ctmper occurrences in .ASM file
- tcs_ctmper is a near call
- tcs_ctmper must be in a segment fname_TEXT (where fname is the name of the file where tcs_ctmper resides), else Linker returns an error.

B.3.2 Medium Model

Model:	Medium ("-mm") switch on command line
Passes:	word size pointers (offset, no DS register)
Sequence:	Arguments Passed Right to Left
Default Calling Convention:	Arguments Passed by Value

Example

'C' Call:	tcm_ctmper (&Mode, Params, &Flag);
'C' Declaration:	extern void tcm_ctmper(int*,int*,int*);
.ASM Subroutine:	

The following assembly code shows how the driver handles user arguments:

```
_tcm_ctmper proc far                    ; dword pointer return address
                  push bp               ; save base pointer
                  mov bp,sp             ; save stack pointer
                  .                     ; [bp+6] holds offset of Mode
                  .                     ; [bp+8] holds offset of Params
                  .                     ; [bp+10] holds offset of Flag
                  .                     ; Program execution here
                  .                     ;
                  .                     ;
                  pop bp                ;restore bp & sp prior to exit
                  ret                   ;return
_tcm_ctmper endp
```

Other

This information is provided for those wishing to create their own drivers:

- _tcm_ctmper is declared "PUBLIC" in the .ASM file
- tcm_ctmper is declared "extern" in the "C" file
- The .ASM file contains the ".model medium" directive (MASM & TASM only)
- Add leading underscore "_" to all tcm_ctmper occurrences in .ASM file
- tcm_ctmper is a near call
- tcm_ctmper must be in a segment fname_TEXT (where fname is the name of the file where tcm_ctmper resides), else Linker returns an error.

B.3.3 Large Model

Model:	Large ("-ml") switch on command line
Passes:	dword size pointers (offset, no DS register)
Sequence:	Arguments Passed Right to Left
Default Calling Convention:	Arguments Passed by Value

Example

'C' Call:	tcl_ctmper (&Mode, Params, &Flag);
'C' Declaration:	extern void tcl_ctmper(int*,int*,int*);
.ASM Subroutine:	

The following assembly code shows how the driver handles user arguments:

```
_tcl_ctmper proc far ; dword pointer return address
                push bp          ; save base pointer
                mov bp,sp        ; save stack pointer
                .                ; [bp+6] holds offset of Mode
                .                ; [bp+10] holds offset of Params
                .                ; [bp+14] holds offset of Flag
                .                ; Program execution here
                .                ;
                .                ;
                pop bp           ;restore bp & sp prior to exit
                ret              ;return
_tcl_ctmper endp
```

Other

This information is provided for those wishing to create their own drivers:

- _tcl_ctmper is declared "PUBLIC" in the .ASM file
- tcl_ctmper is declared "extern" in the "C" file
- The .ASM file contains the ".model large" directive (MASM & TASM only)
- Add leading underscore "_" to all tcl_ctmper occurrences in .ASM file
- Both code & data use dword (segment/offset) pointers
- tcl_ctmper must be in a segment fname_TEXT (where fname is the name of the file where tcl_ctmper resides), else Linker returns an error.

B.3.4 Turbo 'C' Example

```
/***************************************************************************/
/*  TCEXAMPLE.C
/*  CTM-PER EXAMPLE OF MODE 0
/*  USING TURBO C MEDIUM MODEL
/**************************************************************************

#include  "stdio.h"
#include  "conio.h"

    extern tcm_ctmper(int*,int*,int*);                /* declare driver call

main()
{
    int Mode, Flag, Params[15];

    /* initialize CTM-PER using Mode 0

    Mode = 0;
    Params[0] = 768;                                  /* Base Address of Board
    Params[1] = 2;                                        /* Interrupt level
    Params[2] = 3;                                        /* DMA level
    Params[3] = 0;                                        /* No Auto-Calibration

        tcm_ctmper(&Mode, Params, &Flag);

        if(Flag !=0)
        {
            printf("\n\nMode 0 Error FLag = %d\n",Flag);
        }
    REMAINDER OF CODE
    .
    .
    .
}
```

B.4 MICROSOFT PASCAL (V3.0-4.0)

B.4.1 Medium Model

Model:	Medium
Passes:	word size pointers (offset address only)
Sequence:	Arguments Passed Left to Right
Default Calling Convention:	Arguments Passed by Value

Example

PASCAL Call:	Result = msp_ctmper (Var1, Var2, Var3);
'C' Declaration:	FUNCTION msp_ctmper(VAR Var1:integer;VAR Var2;VAR Var3: integer):integer;external;
.ASM Subroutine:	

The following assembly code shows how the driver handles user arguments:

```
msp_ctmper proc far ; far call (dword return address)
                        push bp             ; save base pointer
                        mov bp,sp; save stack pointer
                        .                   ; [bp+4] holds offset of Mode
                        .                   ; [bp+6] holds offset of Params
                        .                   ; [bp+8] holds offset of Flag
                        .                   ; Program execution here
                        .                   ;
                        .                   ;
                        mov ax,n            ; Return Value for Function In ax register
                        pop bp              ;
                        ret 6               ; return and pop bp & sp values prior to exit
msp_ctmper endp
```

Other

This information is provided for those wishing to create their own drivers:

- msp_ctmper is declared "PUBLIC" in the .ASM file
- msp_ctmper is declared external in the calling program
- msp_ctmper resides in segment_TEXT (default of the .model command)

B.4.2 MicroSoft PASCAL Example

```
/******************************************************/
/*  MCPEXAMPL.PAS
/*  CTM-PER EXAMPLE OF MODE 0/*USING MICROSOFT PASCAL
/******************************************************/

Type

Parray = array[1..16] of word;

Var

Params      : Parray;
Mode,Flag   : integer;
Result      : integer;

(* Define Driver Function Call *)

FUNCTION msp_ctmper(VAR Mode:integer;VAR Params:Parray;VAR
Flag:integer):INTEGER;EXTERN;

(* MAIN *)

BEGIN
   Mode:= 0;
   Params[1]:= 768;        (* Base Address of Board *)
   Params[2]:= 2;                                (* Interrupt level *)
   Params[3]:= 3;                                (* DMA level *)
   Params[4]:= 0;                                (* No Auto-Calibration *)
      Result:= msp_ctmper(Mode,Params,Flag);

      if(Result <> 0) then
         WriteLn('Mode 0 Error # = ',Result);

REMAINDER OF CODE
   .
   .
   .
END.
```

B.5 BORLAND TURBO PASCAL (VER 3.0 - 4.0)

Borland's Turbo PASCAL supports a compact- and a large-memory model. The compact model supports one code segment and multiple data segments. In this model, the code segment is limited to 64K with assembly routine calls being near calls. The data segment is unlimited. The large model permits unlimited code and data segments with assembly calls and data access being far calls.

The program (TINST.EXE) shipped with TURBO PASCAL can change the calling convention so that you may not know which convention you are using. The default state is "OFF" or compact mode. In order to ascertain which mode you are using, run the "TINST.EXE" program.

B.5.1 Compact Model

Model:	Compact (Forces far call "OFF" in TINST.EXE)
Passes:	dword size pointers (offset and segment)
Sequence:	Arguments Passed Left to Right
Default Calling Convention:	Arguments Passed by Value

Example

PASCAL Call:	Result = tp_ctmper (Var1, Var2, Var3);
PASCAL Declaration:	FUNCTION tp_ctmper(VAR Var1:integer;VAR Var2;VAR Var3: integer):integer;external;
.ASM Subroutine:	(Either Model)

The following assembly code shows how the driver handles user arguments:

```
tp_ctmper       proc near       ; near call (single word return address)
                push bp         ; save base pointer
                mov bp,sp       ; save stack pointer
                .               ; [bp+4] holds offset of VAR3
                .               ; [bp+6] holds offset of VAR2
                .               ; [bp+8] holds offset of VAR1
                .               ; Program execution here
                .               ;
                .               ;
                mov ax,n        ; return Value for Function In ax register
                pop bp
                ret 12          ; return & pop values prior to exit
tp_ctmper       endp
```

Other

This information is provided for those wishing to create their own drivers:

- Use the $L 'Metacommand' to link the object file containing external function tp_ctmper, i.e. {$l tpctmper} (Link to file tpctmper.obj).
- The VAR declarative forces pass by reference (address of variable) in the function declaration. Default is pass by value (pushing the actual integer value onto the stack).tp_ctmper is declared external in the calling program along with the type of return value (integer).
- Remember that in PASCAL, functions return a value whereas procedures never do.
- The .ASM file contains an explicit declaration of the code segment containing tp_ctmper. Turbo PASCAL handles segments in a primitive manner which is not compatible with the '.model' statements available in MASM or TASM. The function tp_ctmper must reside in a segment called 'CODE'! Turbo PASCAL will not accept any other segment name. If tp_ctmper is not in

segment "the linker returns an "unresolved external" error. The Segment Declaration for "CODE" in the .ASM file must appear as:

```
CODE SEGMENT WORD PUBLIC
ASSUME CS:CODE
.
.                       ; CODE GOES HERE
.
CODE ENDS
```

B.5.2 Large Model

Model:	Large (Forces far call "ON" in TINST.EXE)
Passes:	dword size pointers (offset and segment)
Sequence:	Arguments Passed Left to Right
Default Calling Convention:	Arguments Passed by Value

Example

PASCAL Call:	Result = tp_ctmper (Var1, Var2, Var3);
PASCAL Declaration:	FUNCTION tp_ctmper(VAR Var1:integer;VAR Var2;VAR Var3: integer):integer;external;
.ASM Subroutine:	(Either Model)

The following assembly code shows how the driver handles user arguments:

```
tp_ctmper       proc near       ; far call (dword return address)
                push bp         ; save base pointer
                mov bp,sp       ; save stack pointer
                .               ; [bp+4] holds offset of VAR3
                .               ; [bp+8] holds offset of VAR2
                .               ; [bp+12] holds offset of VAR1
                .               ; Program execution here
                .               ;
                .               ;
                mov ax,n        ; return Value for Function In ax register
                pop bp
                ret 12          ; return & pop values prior to exit
tp_ctmper       endp
```

Other

This information is provided for those wishing to create their own drivers:

- Use the $L 'Metacommand' to link the object file containing external function tp_ctmper, i.e. {$1 tpctmper} (Link to file tpctmper.obj).
- The VAR declarative forces pass by reference (address of variable) in the function declaration. Default is pass by value (pushing the actual integer value onto the stack).tp_ctmper is declared external in the calling program along with the type of return value (integer).
- Remember that in PASCAL, functions return a value whereas procedures never do.
- The .ASM file contains an explicit declaration of the code segment containing tp_ctmper.

B.5.3 Turbo PASCAL Example

```
{S$R-}
{$I-}
{$B+}
{$S+}
{$N-}
{$L TURBOCTM}  (* Link TURBOCTM for CTM-PER *)
{$M 65500, 16384, 655360}
(***************************************************************)
(*  TPEXAMPLE.PAS     *)
(*  CTM-PER EXAMPLE OF MODE 0*)
(*USING TURBO PASCAL*)
(***************************************************************)

Type
Parray = array[1..16] of word;
Var

Params       : Parray;
Mode,Flag   : integer;
Result     : integer;

(* Define Driver Function Call *)

FUNCTION tp_ctmper(VAR Mode:integer;VAR Params:Parray;VAR Flag:integer):INTEGER;EXTERN;

(* MAIN *)

BEGIN
Mode:= 0;                    (* Use Mode 0 *)
Params[1]:= 768;                                    (* Base Address of Board *)
Params[2]:= 2;               (* Interrupt level *)
Params[3]:= 3;               (* DMA level *)
Params[4]:= 0;               (* No Auto-Calibration *)

   Result:= tp_ctmper(Mode,Params,Flag);

   if(Result <> 0) then
      WriteLn('Mode 0 Error # = ',Result);

REMAINDER OF CODE
   .
   .
END.
```

B.6.2 FORTRAN INOUT.FOR Function Example

```
C       INOUT.FOR
C       Example for using INBYT & OUTBYT Functions

        program inout
        integer*2 port,outdat
        integer*1 indat

        port=0
        outdat=0

        do 35 i=1,10,1
        write (*,10)
10 format('Enter Port Address(Decimal): ')

        read(*,15) port
15 format(i3)

        write(*,20)
20 format(' Enter data to write(-1 = exit) ')

        read(*,25) outdat
25 format(i3)

        if(outdat .EQ. -1) go to 45

        write (*,30) outdat
30 format(' Data Written = ',z)

        call outbyt(port,outdat)
        indat=inbyt(port)

35 write(*,40) indat
40 format(' Data Read = ',z)
45 end
```

B.6.3 Integer (Default) Function or Subroutine

The following assembly code shows how the driver handles user arguments:

```
fctmper         proc far        ; dword pointer return address
                push bp         ; save base pointer
                mov bp,sp       ; save stack pointer
                .               ; [bp+6] holds offset of VAR3
                .               ; [bp+10] holds offset of VAR2
                .               ; [bp+14] holds offset of VAR1
                .               ; Program execution here
                .               ;
                .               ;
                mov ax,n        ; return Value for Function In ax register
                pop bp
                ret             ;
fctmper         endp
```

B.7 LAHEY PERSONAL FORTRAN (V1.0 & HIGHER)

B.7.1 Large Model (Only Model Available)

Model:	Large
Passes:	dword size pointers (offset and DS register)
Sequence:	Arguments Passed Left to Right (Opposite MS)
Default Calling Convention:	Arguments Passed by Reference

Example

FORTRAN Call:	call lhy_ctmper(Var1, Var2, Var3);
FORTRAN Declaration:	**None necessary in FORTRAN source file** (Fortran assumes that undeclared subroutines or functions are external. It is left to the linking process to provide the required .LIB or .OBJ files. However, the function name should conform to ANSI FORTRAN rules for integer functions.
.ASM Subroutines:	

NOTE: FORTRAN integer functions (those beginning with the letters i, j, or k) return results in the ax register, whereas non-integer functions reserve 4 bytes on the calling stack for a far pointer to the returned result. Non-integer functions pass their arguments starting at location bp+10 after the "push bp" and "mov bp,sp" instructions have been executed. MetraByte's FORTRAN <--> Assembly routines predominantly use type integer so that this is not a problem. Using non-integer functions may become a problem when returning pointers, floating point results, long integers, etc. To avoid undue problems, use the IMPLICIT INTEGER (A-Z) which causes all Functions and Variables to be implicitly type integer unless expressly declared otherwise.

Also note that FORTRAN Calls By Reference. This method places the address of the passed parameters onto the stack at the time of the call to any function or subroutine rather than the parameters themselves.

B.7.2 Integer (Default) Function or Subroutine

The following assembly code shows how the driver handles user arguments:

```
lhy_ctmper      proc far        ; dword pointer return address
                push bp         ; save base pointer
                mov bp,sp       ; save stack pointer
                .               ; [bp+6] holds offset of VAR3
                .               ; [bp+10] holds offset of VAR2
                .               ; [bp+14] holds offset of VAR1
                .               ; Program execution here
                .               ;
                .               ;
                pop bp          ; Restore bp & sp prior to exit
                ret             ;
lhy_ctmper      endp
```

NOTES: 1. VAR3 = Return Value of Function.
2. Function lhy_ctmper must be declared as an integer * 2.

B.7.3 Lahey Personal FORTRAN Example

```
C*****************************************************************************
C* LHYEXAMPLE.FOR
C* CTM-PER EXAMPLE OF MODE 0
C* USING Lahey Personal FORTRAN
C*****************************************************************************

integer*2 Params(16), Mode, Flag, lhy_ctmper

    Mode = 0;                       (* Use Mode 0 *)
    Params(1):= 768;                (* Base Address of Board *)
    Params(2):= 2;                  (* Interrupt level *)
    Params(3):= 3;                  (* DMA level *)
    Params(4):= 0;                  (* No Auto-Calibration *)

        call lhy_ctmper(Mode, Params(1), Flag);

          if (Flag .NE. 0) then
          print *,'Mode 0 Error # =',Flag

REMAINDER OF CODE
  .
  .
  .
```

B 8 INTERPRETED BASIC (GW, COMPAQ, IBM, ETC.)

B.8.1 Medium Model (Only Model Available)

Model:	Medium (Far Single Data)
Passes:	word size pointers (offset and no DS Register)
Sequence:	Arguments Passed Left to Right
Default Calling Convention:	Arguments Passed by Reference

Example

BASIC Call:	12500 CALL CTMPER(MODE%, PARAMS%(0), FLAG%)
BASIC Declaration:	NONE NECESSARY IN BASIC SOURCE CODE. However, a "BLOAD" (Binary load of .BIN file) of the binary file containing the external subroutine must be done prior to calling that subroutine.
.ASM Subroutine:	

The following assembly code shows how the driver handles user arguments:

```
Location 0 (Beginning of Code Segment)
                    jmp ctmper
                    .
                    .
 ctmperproc         far                ; far call (dword return address)
                    push bp            ; save base pointer
                    mov bp,sp          ; save stack pointer
                    .                  ; [bp+6] holds offset of Mode
                    .                  ; [bp+8] holds offset of Params
                    .                  ; [bp+10] holds offset of Flag
                    .
                    .                  ; Program execution here
                    .
                    .
                    pop bp             ; restore bp & sp prior to exit
                    ret
 ctmper             endp
```

NOTE: BASIC requires that the .BIN file containing the callable subroutine "ctmper(Mode%, Params%(0), Flag%)" reside at location 0 in the .ASM segment or to "jmp" (unconditional jump) to the .BIN file. A BASIC "jmp " will always jump to location 0 in the .ASM code segment. Creation of a .BIN file is accomplished as follows:

1. Create the .ASM Source Code File 'EXAMPLE.ASM'
2. Assemble 'EXAMPLE.ASM' thus creating 'EXAMPLE.OBJ'
3. Link 'EXAMPLE.OBJ' to create 'EXAMPLE.EXE'
4. Run EXE2BIN on 'EXAMPLE.EXE' (DOS Utility) to create 'EXAMPLE.COM'
5. Run MAKEBIN.EXE (MetraByte Utility) on 'EXAMPLE.COM' to create'EXAMPLE.BIN'

```
MASM EXAMPLE ;
LINK EXAMPLE ;
EXE2BIN EXAMPLE.EXE EXAMPLE.COM MAKEBIN      EXAMPLE.COM
```

The .BIN file is loaded at a certain location within a specified segment defined by the "DEF SEG" command. This location is then supplied to BASIC via a pointer residing at locations &h510 and &H511. This allows the user to perform a BLOAD at a known address in relation to BASIC's starting address. GW-BASIC does not supply this information so that the user must specify the address when BLOADing the .BIN file. Notice that the example program arbitrarily uses &H8000 for the BLOAD segment. Caution should be exerhowever, to avoid overwriting any existing programs loaded in high memory.

B.8.2 Interpreted BASIC Example Program

```
100  '*********************************************************************
110  '*BASEXAMP.BAS
120  '*CTM-PER EXAMPLE OF MODE 0
130  '*USING BASIC
140  '*********************************************************************
150  SG = &H8000
160  DEF SEG = SG
170  BLOAD "CTMPER.BIN", 0
180  DIM PARAMS%(15)
190  MODE% = 0'USE MODE 0
200  PARAMS%(0) = 768'BASE ADDRESS
210  PARAMS%(1) = 2'SET INTERRUPT LEVEL
220  PARAMS%(2) = 3'SPECIFY DMA LEVEL
225  PARAMS%(3) = 0'NO AUTOCALIBRATION
230  CALL CTMPER(MODE%,PARAMS%,FLAG%)'CALL TO DRIVER
240  '
250  IF FLAG% <> 0 THEN PRINT "MODE 0 ERROR #",FLAG%
260  '
270  .
280  .
   etc.
```

B.9 QUICK BASIC

B.9.1 Medium Model (Only Model Available)

Model:	Medium (Far Single Data)
Passes:	word size pointers (offset and no DS Register)
Sequence:	Arguments Passed Left to Right
Default Calling Convention:	Arguments Passed by Reference

Example

BASIC Call: CALL QBCTMPER(MODE%, VARPTR(PARAMS%(0)), FLAG%)

BASIC Declaration: The Declaration tells QuickBASIC that the subroutine expects three arguments and that the *middle* argument is to be passed by value. Remember that BASIC normally passes all arguments by reference (address). This is the only method for passing an array to a subroutine in BASIC: passing the value of the address of the array in effect passes the array by reference. To make use of the callable assembly routine, a ".QLB" (Quick Library) file is created out of the original .ASM source file. Although the format of the subroutine is identical to those used by interpreted BASIC packages, both the Quick BASIC integrated development environment (QB.EXE) and the command line complier (BC.EXE) expect the subroutine to be in a specially formatted .QLB library file. Unlike interpreted BASIC packages, Quick BASIC actually links to the assembly .QLB library file so it is not necessary to include the "jmp QBCTMPER" instruction at location 0 (of the source file) as in interpreted BASIC.

.ASM Subroutine:

The following assembly code shows how the driver handles user arguments:

```
QBCTMPER    proc far        ; far call (dword return address)
            push bp         ; save base pointer
            mov bp,sp       ; save stack pointer
            .               ; [bp+6] holds offset of Mode
            .               ; [bp+8] holds offset of Params
            .               ; [bp+10] holds offset of Flag
            .
            .               ; Program execution here
            .
            .
            pop bp          ; restore bp & sp prior to exit
            ret
QBCTMPER    endp
```

NOTE When creating a .QLB file, it is good practice to make a .LIB of the same version as a backup file. Creation of a .QLB file is accomplished as follows:

1. Create the .ASM Source Code File 'EXAMPLE.ASM'

2. Assemble 'EXAMPLE.ASM' thus creating 'EXAMPLE.OBJ'

3. Link 'EXAMPLE.OBJ' with the "/q" option to create 'EXAMPLE.QLB'
```
MASM EXAMPLE ;
LINK /q EXAMPLE ;
```

scs_ctmper(mode,param,flag)	:	Call from Microsoft C Small Model
mscm_ctmper(mode,param,flag)	:	Call from Microsoft C Medium Model
mscl_ctmper(mode,param,flag)	:	Call from Microsoft C Large Model
tcs_ctmper(mode,param,flag)	:	Call from Turbo C Small Model
tcm_ctmper(mode,param,flag)	:	Call from Turbo C Medium Model
tcl_ctmper(mode,param,flag)	:	Call from Turbo C Large Model
msp_ctmper(mode,param,flag)	:	Call from Microsoft PASCAL
QBctmper(mode,param,flag)	:	Call from Microsoft QuickBASIC
fctmper(mode,param,flag)	:	Call from Microsoft FORTRAN
lhy_ctmper(mode,param,flag)	:	Call from Lahey FORTRAN

Linking the Library "ctmper.lib" to the user program is accomplished after program compilation by including it in the link line as follows:

```
link userprog.obj,userprog,,user.lib_ctmper.lib;
```

- userprog.obj is an object module produced by compilation of the user program
- userprog should be used for the resultant executable .EXE file
- user.lib is any other user library, if applicable

For TurboPASCAL, the entry point is:

tp_ctmper(mode,param,flag) : Call from TurboPASCAL program

The user program should have the directive

```
{$L turboctm} Link in turboctm.obj 30
```

B.10.2 Files Listing for CTM-PER

FILE NAME	DESCRIPTION
FILES.DOC	File Listing
PCFCTMPER.DOC	Information concerning multi-language call structures
README.DOC	REV. 1 Information on CTM-PER
MSCDEMO.EXE	Microsoft "C" example of Mode 3 usage
TPDEMO.EXE	Turbo PASCAL example of Mode 3 usage
MSPDEMO.EXE	Microsoft PASCAL example of Mode 3 usage
CTMPER.LIB	Driver for "C", PASCAL, and FORTRAN
TURBOCTM.OBJ	CTM-PER Driver for Turbo PAS

B.10.3 Source C, PASCAL, and FORTRAN Programs

FILE NAME	DESCRIPTION
CTMPER.ASM	Source code CTM-PER Driver (Common Modes)
CTMPERPCF.ASM	CTM-PER Source code Language Interface
MSCDEMO.C	Microsoft "C" example of Mode 3 usage
TCDEMO.C	Turbo "C" example of Mode 3 usage
TPDEMO.PAS	Turbo PASCAL example of Mode 3 usage
MSPDEMO.PAS	Microsoft PASCAL example of Mode 3 usage
MSFDEMO.FOR	Microsoft FORTRAN example of Mode 3 usage
LHYDEMO.FOR	Lahey Fortran Example of Mode 3 usage
DEMOG.C	Microsoft "C" Example of Mode 6
DEMO6L.C	Microsoft "C", Large Model Example of Mode 6