Keeping Time with the Intersil CDP68HC68T1 Real Time Clock and the "Millennium Bug"

Application Note | November 1997 | AN9766

Introduction

The Intersil CDP68HC68T1 is a multifunctional CMOS real time clock with 32 bytes of general purpose RAM and various power sense and watchdog control circuitry. The main function of the CDP68HC68T1 (henceforth referred to as the "T1") is to keep accurate time in the form of seconds, minutes, hours, days, months and years. The interface to the T1 is through the SPI interface, available on many Intersil 68HC05 microcontrollers and other popular MCUs.

This application note deals with various techniques for keeping time with the T1, including keeping track of years into the 21st century and beyond.

This application note assumes that the reader has read and has access to the CDP68HC68T1 technical data sheet (Intersil file number 1547.3). A copy of this data sheet may be obtained from your local Intersil Corporation sales representative, downloaded from the Internet at http://www.intersil.com.

The "Millennium Bug" or "Y2K" Problem

The "Millennium Bug" (also referred to as the Year 2K and the Y2K problem) is one of the latest problems facing the software development industry. The problem is very simple - a lot of computer programs have been written to accept only two digits to represent the year for any given input. Thus, the program uses the two digits entered and *assumes* that the year is in the 20th century (i.e. the first two numbers are "19"). Thus, entry "74" for the year refers to the year "1974", and so on. Now that the turn of the century is approaching, the last two digits of the year are going to start over at "00". The result? Lots of computers are going to think that it's the year 1900. As you may image, a lot of chaos will ensue if the problem is not fixed. Paychecks will not be issued, air traffic control systems may shut down, your bank account may be frozen, and the list goes on. As this application note is being written, there are companies popping up all over the world for the express purpose of fixing this problem (it is estimated that it will cost over $600 billion US to fix!). However, keep in mind one important fact -- for the most part, the millennium bug is a SOFTWARE problem. Most hardware devices, the CDP68HC68T1 inclusive, will keep the correct time through the year 2000 if programmed correctly.

Now, you may be thinking that if the millennium bug is a software problem, there is nothing to worry about with using the T1. For the most part, this is true - if programmed correctly, the host MCU interfacing with the T1 will keep the correct time into the 21st century. Unfortunately, with all of the hype about the millennium bug problem in the software world, people have begun to worry about the problem arising in the hardware (i.e. real time clocks like the CDP6818 and the T1). Some companies have even designed devices to be "Year 2000 compliant". As stated before, the T1 and most other real time clock devices will function just fine in the year 2000 provided the application software is written correctly.

This application note will discuss more about the implications of the year 2000 and using the T1 later in the section **Tracking the year and the "Millennium Bug"**. First, we will discuss general time keeping techniques for use with the T1.

Keeping Time with the T1

Keeping time using the T1 is a very simple and easy task. The T1 keeps track of seconds, minutes, hours, days of the week, dates, months and years in BCD (binary coded decimal) format in a series of seven 8 bit internal registers. Once set, the T1 will keep track of all of these values, incrementing them as needed based on a 1Hz clock generated from an external oscillator. Values in the registers will "roll over" when appropriate so as to keep time according to the western Gregorian calendar. The T1 also accounts for leap years (i.e., Feb 28, 1996 will roll over to Feb 29, 1996. Feb 28, 1997 will roll over to Mar 1, 1997).

Timing and the T1

Before we start talking too much about how time is tracked in the T1, first we need to discuss the device's external and internal timing.

To keep track of time, the T1 needs an accurate clock source. There are three types of clocks that the T1 can use - an external square wave input on the XTALIN input pin, a standard crystal oscillator circuit using XTALIN and XTALOUT, or 50/60 Hz AC line voltage on the LINE input pin (standard 120V AC line current is regulated very accurately to 60Hz in North America and can act as a good clock source).

For the square wave and crystal oscillator setups, there are four frequencies that should be used with the T1. These are 32.768kHz, 1.048576MHz, 2.097152kHz and 4.194304MHz. Using frequencies other than these will directly affect the accuracy of the T1.

NOTE: Although the PC board layout of a crystal oscillator circuit is an important consideration in all designs, it is even more so when using a 32.768kHz crystal with a device like the T1. The reason is that these slow oscillating crystals (with respect to 2MHz and 4MHz crystals) have slower rise and fall times and as a result spent a lot more time around the midpoint of the CMOS inverter being used to oscillate the crystal. This makes the oscillator circuitry especially sensitive to external noise. When using a 32.768kHz crystal, be careful to place the crystal as close to the device as possible. The PC board traces for the oscillator circuit should never be more than an inch long and should not run parallel to each other. Also, it is a good idea to shield the oscillator traces by running a ground trace or ground plane around the circuitry. This will help to insure that no unnecessary clocks are generated internally to the T1 due to noise. Finally, as with most CMOS integrated circuits, it is a good idea to place a 0.1µF capacitor across the V_{DD} and V_{SS} supplies of the device to decouple and filter out any unwanted noise.

Address Space and Register Addressing

Internally, the T1 can be viewed as a 64 byte address space, as shown in Figure 1. In this space are contained 13 timer control and status registers, 32 bytes of general purpose RAM and 19 unused bytes. The host MCU controls which byte is being written to or read from by sending an address byte at the beginning of every SPI data transfer. Since the highest addressable location in the T1 is location $32, only 6 address bits are needed to access the entire T1 address space. Thus, the address byte consists of the write/read bit, one unused bit, and six address bits. After the address byte is sent, the MCU either sends the data to be written or starts a transmission to read from the T1. For example, to write a $24 to the seconds register (location $20), the MCU would send $A0, $24. The $A0 byte is the address ($20) with the MSB set, indicating a write operation. To read the seconds register, the MCU would send $20, $xx, where $xx is a “don’t care byte” sent by the MCU to start the SPI transmission. The data sent to the T1 during a read operation is ignored. The data being read from the T1 is shifted out to the MCU through the MISO pins.

When writing to and reading multiple data bytes from the T1, it is advantageous to use the auto-increment feature of the address pointer. Each time a byte is written to or read form the T1, the internal address pointer increments to the next available address. By starting a subsequent transmission to the T1 without lowering the CE pin of the T1, the MCU can write or read multiple data bytes in one session. For example, the seconds, minutes and hours registers in the T1 are at locations $20, $21, and $22, respectively. To read all three in one transmission, the MCU would send the address byte $20 (note that the MSB of the address byte is low signifying a read operation), $xx, $xx, and $xx. After the address byte is sent, the MCU sends a dummy byte to start a SPI transmission to read the contents of the seconds register ($20). Once the T1 sends the seconds, its internal address pointer increments to $21. When the next transfer is started, the contents of this location are sent, and so on. The example code in Appendix A shows how this technique is used to set and read the complete date and time in one SPI transfer.

Note that the auto-increment feature confines the address pointer to either the RAM locations or the timer register locations, exclusively. When addressing locations in RAM space ($00-$1F), the address pointer will wrap to $00 after location $1F. Similarly, when accessing the timer registers, the address pointer will wrap from location $3F to $20. Thus, you cannot read RAM data and timer data in the same SPI transfer.

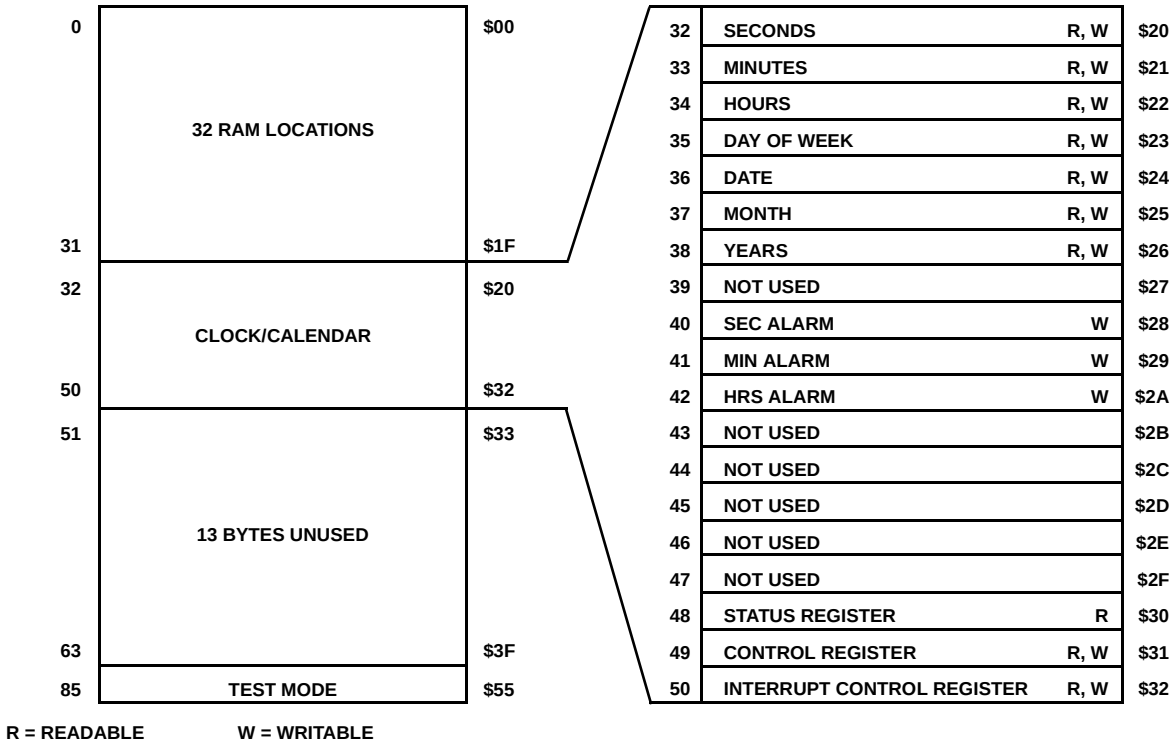

FIGURE 1. MEMORY MAP OF THE CDP68HC68T1

Data Format

One of the most important things to remember about the way the T1 keeps time is that the values in the clock registers are in a binary coded decimal (BCD) format. Each byte wide register is divided into two four bit nibbles; each nibble can represent decimal numbers 0 through 9 with the binary numbers %0000 through %1001. Thus, to represent 54 minutes the T1 would contain the number %01010100 ($54) in the minutes register. As a result, hexadecimal numbers $A-$F are not valid (except in the case of the hours register -- see explanation below). This is important to remember for those of you who are used to keeping track of numbers in hexadecimal format -- e.g., to represent December in the months register, you need to write a BCD “12” ($12) into the register, not $0C, the hexadecimal equivalent to a decimal 12.

If you do write a value that is out of range for a particular register (say, for example, you want to set the T1 day of the month register to the 15th, so you write a $0F instead of a $15), the T1 will not generate an error or let you know in any way that there is invalid data in the register. Instead, it will continue to increment the data as necessary (for our example, once every 24 hours) and will roll the data over when the last appropriate value is reached. In the day register, the T1 will increment from a $19 to a $20 (not a $1A) and from a $30 to a $31 (if in a 31 day month) or to a $01 (if in a 30 day month).

While keeping the values in the time registers in BCD format can be a burden when it comes to doing binary arithmetic, having information in BCD format facilitates displaying the time and date on an ASCII display device. In the example in Appendix A, the time read from the T1 is written to a standard Optrex interface LCD. Converting the minutes value to ASCII, for example, is trivial since all that is required is to simply add $30 to the BCD values read from the T1. The example code also shows how to easily display text for the day or the week (Sun, Mon, etc.) and the month using the values read from the T1. A table of the timer registers, their valid BCD ranges and examples is shown in Table 1.

Once a time and a date has been written into the T1, all of the values in the timer register will increment appropriately from the 1Hz clock divided down from the external oscillator. Thus, the minutes register will increment every time the seconds rolls from $59 to $00, the hours register will increment very time the minutes roll from $59 to $00, and so on. The T1 is designed to know that 1 day is 24 hours (rolling over at 11:59 p.m., or 23:59), there are 7 days in a week, there are 30 days in April, June, September, and November, there are 31 days in January, March, May, July, August, October and December, and there are 28 days in February except during leap years. It also knows to increment the year on December 31 at 11:59 p.m.

TABLE 1. CLOCK/CALENDAR AND ALARM DATA MODES

ADDRESS LOCATION (H)	FUNCTION	DECIMAL RANGE	BCD DATA RANGE	(NOTE 1) BCD DATE EXAMPLE
20	Seconds	0-59	00-59	18
21	Minutes	0-59	00-59	49
22	Hours 12 Hour Mode (Note 2)	1-12	81-92 (AM) A1-B2 (PM)	A3
	Hours 24 Hour Mode	0-23	00-23	15
23	Day of the Week (Sunday = 1)	1-7	01-07	03
24	Day of the Month (Date)	1-31	01-31	29
25	Month Jan = 1, Dec = 12	1-12	01-12	10
26	Years	0-99	00-99	85
28	Alarm Seconds	0-59	00-59	18
29	Alarm Minutes	0-59	00-59	49
2A	Alarm Hours (Note 3) 12 Hour Mode	1-12	01-12 (AM) 21-32 (PM)	23
	Alarm Hours 24 Hour Mode	0-23	00-23	15

NOTES:

1. Example: 3:49:18, Tuesday. Oct. 29,1985.
2. Most significant Bit, D7, is “0” for 24 hours, and “1” for 12 hour mode. Data Bit D5 is “1” for P.M. and ‘0” for A.M. in 12 hour mode.
3. Alarm hours. Data Bit D5 is “1” for P.M. and “0” for A.M. in 12 hour mode. Data Bits D7 and D6 are DON’T CARE.

The T1 also has the capability to track hours in a 12 hour or 24 hour format. A flag bit in hours register (bit 7) controls this option. If set, the T1 counts hours from 1 to 12. If B7 is clear, the T1 will count hours from 0 to 23. Note that in 12 hour mode, bit 5 of the hours register becomes a flag bit to indicate a.m. or p.m. The example code in Appendix A displays the hours in 12 hour format and uses this flag bit to display a.m. or p.m.

Tracking the Year and the "Millennium Bug"

Keeping track of all aspects of the current time with the exception of the year is very straight forward with the T1. For Example, to set the current time to 5:33:34 p.m., Wednesday, August 20, write the following to the T1:

- $34 to the seconds register ($20)
- $33 to the minutes register ($21)
- $A5 (bits 7 and 5 are set to indicate 12 hour mode and p.m.) to the hours register ($22)
- $04 (Wednesday is the 4th day of the week) to the day of the week register ($23)
- $20 to the day of the month register ($24)
- $08 (August is the 8th month of the year) to the month register ($25)

The year register, however, only has room for a two digit BCD number. So how do we represent a four digit year? Does $97 in the year register mean 1997 or 2097? One might think that because there is no century counter in the T1 that this part will obviously not work in the 21st century. The only solution is to get a more expensive real time clock with a four digit year counter. This is, however, not the case. The solution is very simple but is one that must be known to the application designer -- the year must be kept track of in software.

Unlike complex PC and mainframe based applications that must be able to trace dates in the past (birth dates into the 1800's) and well as into the future (driver's license expiration dates into the 21st century), the T1, like most real time clocks, is only concerned with the time and date right now. With that in mind it is quite easy to track the year in software.

For example, let's say we are designing a VCR and we are using the T1 as the clock. Our microcontroller must be able to read the T1 and know the exact time and date. This information will be displayed in the on-screen display and used to start or stop the VCR depending on how the user has programmed it. The time and date for the VCR is set in the T1 by inputs from the user and it can be assumed that the T1 will always be tracking the current time. Since it would be useless for most customers to set the VCR clock to a day in the past (8:37 p.m. on May 6, 1974 for example), the MCU program does not need to be concerned with dates in the past. With that in mind, we can pick a year, 1997 perhaps, that to VCR clock cannot be set before. By doing this the software can be written such that any year read from the T1 that is before $97 must refer to a date from 2000 to 2096. Otherwise, it must be a date from 1997 to 1999. This gives our VCR a 100 year usable date range from January 1, 1997 to December 31, 2096. This is acceptable since the odds that the VCR will still be around in 2095 are pretty slim.

The code example in Appendix A demonstrates this technique. When the year is read from the T1, it is compared to the number $97. If the current year is lower ($00 - $96), the MCU adds the century on as "20". Otherwise the century "19" is added. For example, if the MCU reads the year as $05, it will display the year as 2005. If the year is read as $98, the MCU will display 1998. Thus, 11:59:59 p.m. on December 31, 1999 will be followed immediately by 12:00:00 on January 1, 2000 (this is, incidentally, not the beginning of the new millennium -- January 1, 2001 is). None of the values in the T1 timer registers need to be altered in this transition.

An alternative approach to the "century assumption" method detailed above is to have the host MCU keep track of the century in a RAM byte. In this situation, the MCU would simply increment this century counter in RAM every time it detects a transition from December 31 to January 1. The simplest way of handling this would be to store the month in a RAM byte every time the MCU reads the date from the T1. On the next date read, the MCU would check the current month (read from the T1 month register at location $25) against the month value read last time the T1 was accessed (stored in RAM). If the current month is less than the old month, MCU would increment the century counter. Keep in mind, this method assumes the T1 is accessed by the MCU at least once every 11 months. Ideally, the century byte and old month value could be stored in the RAM of the T1 itself, thus allowing the MCU to power down without losing track of the year. Utilizing this method allows the T1 to keep track of the correct time indefinitely.

Now, since it is easy to use the T1 right through the year 2000, there should be no other concerns, right? Well, not exactly. There is one more thing to be considered when dealing with the turn of the century. This concern is the fact that the year 2000 is a leap year and there will be a February 29, 2000.

Big deal, right? There is a leap year every four years, right? Well, not exactly. One of the features of the Gregorian calendar used by most of western civilization is that there is an extra day (February 29) in every year evenly divisible by 4, except for century years (1700, 1800, 1900, etc.) NOT divisible by 400. This was done to compensate for the fact that the earth revolves around the sun in 365.244 days, not 365.25. Thus, the year 1800, 1900, 2100 and 2200 are NOT leap years, even though they are evenly divisible by four.

So what does this mean in terms of the T1? Well, the T1 keeps track of leap year by incrementing the date to February 29 at midnight on February 28 when the value in the year counter is evenly divisible by 4. If the application software uses the year counter as described previously (where $00 in the year counter represents the year 2000), there is no problem. Since the year 2000 is a leap year, the T1 will keep perfect time and there will be no trace of the millennium bug.

If you use the year $00 to represent 1900, 2100, or any year not divisible by 400, the T1 will increment to February 29, erroneously considering the year to be a leap year. Since it is not likely that any applications using the T1 will need to track the year

1900 or 2100 (unless you are building a time machine), this is not a problem. Thus, there is no “Millennium Bug” in the T1. If an application using the T1 needs the value $00 in the year register to represent the year 1900, for example, it is simple enough to have the software check the date and, if it is equal to Feb 29, 00, increment the date to March 1.

Example Program

On the following pages is an example program written to demonstrate some of the time keeping techniques discussed in this application note. This program is written in 68HC05 assembly language and targets an 8-bit Intersil CDP68HC05C8B microcontroller. This code has been written so that even those with little to no experience with 68HC05 assembly language can hopefully follow along. For more information on CDP68HC05 microcontrollers, visit the Intersil 68HC05 web site on the Internet at http://www.intersil.com/68hc05.

The function of the example program is quite simple. The 68HC05 MCU reads the time data from the seven clock registers in the T1 (locations $20 to $26) and uses this data to display the current date and time on an Optrex LCD module. The display format for the data is:

Day HH:MM:SS a.m./p.m.
Month Date, 4 digit year

Thus, the current date and time as this application note is being written would be displayed as:

Thu 10:14:34 p.m.
Aug 21, 1997

NOTE: The LCD interface subroutines used by the 68HC05 in this program have not been listed in the listing file for brevity. For a listing and explanation of these subroutines, please refer to the Intersil application note AN9702, “Keypad scan and LCD interface for the CDP68HC05”.

When reading data from the T1, the 68HC05 does a “burst” read of all seven clock registers. As described before, this can be done because the T1 automatically increments its internal address pointer every time a register is read from or written to. In the example program, the 68HC05 sends the command to read from address $20 (the seconds register and the lowest addressed clock register). The 68HC05 then starts another SPI transmission to read the data from the seconds register. Once the data transfer is complete, the T1 increments its address pointer and is now pointing at location $21. When the 68HC05 starts the next SPI transmission (without lowering the CS pin of the T1), the T1 will shift out the contents of the minutes register ($21). The 68HC05 keeps initiating SPI transfers in this way until all seven T1 clock registers are read.

Now, the MCU could have lowered the CS of the T1 after it received the first byte of data. If the 68HC05 then wanted to read the minutes register, it would then raise the CS line, send the command to read from address $21, and then start another SPI transmission to read the data. Any of the registers in the T1 may be read in this fashion. However, since the clock registers are arranged sequentially in the T1 memory space, it is convenient and much faster to read them in the burst mode described above.

Once all of the data is read from the T1 and stored into the internal RAM of the 68HC05, the MCU uses this data to display the current time and date on the LCD. Since all of the data read from the T1 is in BCD format, displaying numerical data (seconds, minutes, day of the month, etc.) is as simple as taking each nibble of the BCD byte, adding $30 (ASCII ‘0’) to convert it to ASCII, and sending it to the LCD. Displaying the day, month, and year are a little different but still very simple.

For the day and the month, we would like to display their abbreviation, not just their number. To do this, we set up a table of “pointers” to the ASCII text messages set up in the 68HC05 memory. Using the day or month number, the 68HC05 can pick an address out of the pointer table that corresponds to the address of the text message. For example, let’s say the 68HC05 reads the day as “$05” from the T1. This represents Thursday, or “Thu”. The 68HC05 then loads this value into the index register and loads the accumulator with the value at dayTable, x. What this does is form an address by adding the contents of the index register ($05) and the address of dayTable ($27F). The accumulator is loaded with a byte from the resultant address ($284). This byte (decimal 24, hex $18) is the value of the lower byte of the address of the “Thu” message in memory (located at $318). The “txlcd1” subroutine, used to send text messages to the LCD, knows that the high byte of the address is $03. This is similar to loading a value from a C pointer array called `dayTable` by saying &message = dayTable[day]. In this way both the month and the day are displayed.

The full four digit year is displayed as apart of this program using the method talked about in the **Tracking the year and the “Millennium Bug”** section. The two digit year is read from the T1 and stored in the RAM of the 68HC05. When it comes to displaying the full date, this program has been written such that years “00” through “96” will have a “20” added as the century counter. Years “97” to “99” will have a “19” added as the century. Thus, this clock will roll over from 11:59:59 p.m, December 31, 1999 to 12:00:00 a.m., January 1, 2000.

Setting the time in this program is done by issuing a software interrupt (SWI). When this is done, the SWI interrupt service routine (ISR), located at address $262, will take the values in the 68HC05 RAM locations $50 to $56 and write them to the T1 clock register locations $20 to $26, respectively. This program was tested on a Intersil 68HC05ICEC1-EV in-circuit emulator system so that all that was necessary to change the time was to alter the 68HC05 RAM locations and reset the program. In a real application, the user program could possibly receive input from a keypad as a part of an IRQ interrupt, write this data into the proper RAM locations, call the SWI to set the time and date, and then return. Upon returning from the interrupt routine the new date and time would be set and the clock would continue on from the new time.

Conclusion

The CDP68HC68T1 is a powerful and lost cost method to keep track of real time in any MCU system. Despite the fact that the T1 does not have a four digit century counter, with the proper programming it can be impervious to the “Millennium Bug” and all of its consequences. Any MCU system will be able to keep time perfectly into the next century with the T1 if a few simple software precautions are taken.

Appendix A - CDP68HC05 to CDP68HC68T1 Interface Example Code Listing

```
INTERSIL Corporation  (c)1990 - 1997
68HC05 Assembler Version 3.0.2
Filename:      T1YEAR2K.LST
Source Created:08/21/97, 10:30 am
Assembled:     08/21/97, 10:30 am
00001                   ;*************************************************************************
00002                   ;       aaaaaaaaaaaaa
00003                   ;     aH"""""HHHH"""""Ha
00004                   ;    aHH    HHHH    HHa       COPYRIGHT 1995,1996,1997
00005                   ;       ,-.  ,-.  ,-.         INTERSIL CORPORATION
00006                   ;    __/  |_/  |_/  |__
00007                   ;    HHH    HHHH    HHH
00008                   ;     HH    HHHH    HH
00009                   ;      "HHHHHHHHHHHH"
00010                   ;
00011                   ;      File:   t1year2k.s
00012                   ;
00013                   ;      Version: 1.0
00014                   ;
00015                   ;      Description:
00016                   ;         This is a simple program to demonstrate using the Intersil
00017                   ;         CDP68HC68T1 RTC in a simple time/date display circuit.
00018                   ;         This program interfaces a Intersil CDP68HC05C8B microcontroller
00019                   ;         to both the T1 and an Optrex LCD module. The clock data
00020                   ;         read from the T1 is displayed in the proper format on the LCD
00021                   ;         display.
00022                   ;
00023                   ;         Note: The LCD interface subroutines have not been included
00024                   ;         in this listing file to conserve space. A full listing and
00025                   ;         explanation of these routines can be found in the Intersil
00026                   ;         application note AN9702, "Keypad scan and LCD interface for
00027                   ;         the CDP68HC05".
00028                   ;
00029                   *************************************************************************
00030
00031                   ;*************************************************************************
00032                   ;   -- Constant definitions --
00033                   ;*************************************************************************
00034
00035                   #include <c8b.s>
---------- START OF INCLUDE F:\6805\C8B.S ----------
00001                   #nolist
00123                   #list
---------- END OF INCLUDE F:\6805\C8B.S ----------
00036
00037 $0002 =     2     lcd           equ     portc
00038 $0001 =     1     lcdctl         equ     portb
00039 $0006 =     6     rs            equ     6
00040 $0005 =     5     rw             equ     5
00041 $0004 =     4     e             equ     4
00042 $0001 =     1     t1csp           equ     portb
00043 $0007 =     7     t1cs           equ     7
00044
00045                   ;*************************************************************************
00046                   ;   -- Variable definitions --
00047                   ;*************************************************************************
00048
00049     0050                  section ramVars, $50
00050
00051     0050          seconds       ds      1
00052     0051          minutes       ds      1
00053     0052          hours         ds      1
00054     0053          day           ds      1
00055     0054          date          ds      1
00056     0055          month         ds      1
```

```
00057     0056          year          ds     1
00058     0057          lcdTemp1      ds     1
00059     0058          lcdTemp2      ds     1
00060     0059          lcdTemp3      ds     1
00061     005A          lcdTemp4      ds     1
00062     005B          lcdTemp5      ds     1
00063     005C          lcdTemp6      ds     1
00064     005D          mytemp        ds     1
00065
00066                   ;*************************************************************************
00067                   ;   -- Macro definitions --
00068                   ;*************************************************************************
00069
00070                   #macro sendspi
00071                         sta     spdr
00072                         brclr   spif,spsr,*
00073                         lda     spdr
00074                   #endmacro
00075
00076                   ;*************************************************************************
00077                   ;   -- Program code --
00078                   ;*************************************************************************
00079
00080     0100                section code, $100
00081
00082  [2] 0100  9C             rsp                     ;reset stack pointer
00083  [2] 0101  9B             sei                     ;mask interrupts
00084  [6] 0102  CD0241           jsr     lcdIni              ;initialize LCD
00085  [5] 0105  1E05           bset    t1cs,t1csp+4        ;make t1 CS output
00086  [5] 0107  1F01           bclr    t1cs,t1csp          ;disable T1
00087  [2] 0109  A654           lda     #(2!spe+2!mstr+2!cpha) ;SPI on, master, phase=pol=0, 1MHz
00088  [4] 010B  B70A           sta     spcr
00089
00090                   ;*************************************************************************
00091                   ;   -- Configure the CDP68HC68T1 --
00092                   ;
00093                   ;   The T1 is set up with the control register ($31) = $81. This
00094                   ;   sets the START bit to enable the counter, sets the T1 up to use
00095                   ;   a 4.194304MHz oscillator, and to generate a 2MHz output on the
00096                   ;   clock out pin. All interrupts for the T1 are disabled by writing
00097                   ;   a $00 to the interrupt control register ($32).
00098                   ;*************************************************************************
00099
00100  [2] 010D  A681           lda     #$81            ;Write an $81 to the control
00101  [2] 010F  AE31           ldx     #$31            ;register in the T1 at $31
00102* [6] 0111  AD7D           jsr     writet1
00103
00104  [2] 0113  A600           lda     #$00            ;Write a $00 to the interrupt
00105  [2] 0115  AE32           ldx     #$32            ;register in the T1 at $32
00106* [6] 0117  AD77           jsr     writet1
00107
00108                   ;*************************************************************************
00109                   ;   -- Set the time --
00110                   ;
00111                   ;   In this code, the time is read from the T1 from the clock registers
00112                   ;   at $20 to $26 and placed in 6805 ram locations $50 to $56. From the
00113                   ;   data in these RAM locations the time display on the LCD is created.
00114                   ;   If an SWI is issued, the values in these RAM locations will be written
00115                   ;   to the T1 such as to set the current time.
00116                   ;*************************************************************************
00117
00118 [10] 0119  83             swi                     ;this causes the data in RAM to be
00119                                                   ;be written to the T1
00120
00121                   ;*************************************************************************
00122                   ;   -- Displaying the Time --
00123                   ;
```

```
00124                ;   In this program, all the 6805 MCU does is read the time and date
00125                ;   values from the T1 and display them on a standard Optrex LCD. To do
00126                ;   this, the 6805 does a burst read of the T1 clock locations $20 to $26
00127                ;   and stores them in RAM at locations $50 to $56. The display section of
00128                ;   this loop then takes this data, converts it to the appropriate display
00129                ;   data and writes it to the LCD.
00130                ;*************************************************************************
00131
00132                displayLoop
00133
00134  [5] 011A  1E01          bset    t1cs,t1csp          ;enable t1
00135  [2] 011C  A620          lda     #$20                ;read all clock registers
00136  [M] 011E              sendspi                       ;starting at address $20
00137  [2] 0125  AE50          ldx     #seconds            ;point X at 6805 RAM locations
00138
00139                getLoop
00140  [M] 0127              sendspi                       ;get all 7 clock registers in
00141  [4] 012E  F7            sta     x                   ;burst xfer mode
00142  [3] 012F  5C            incx                        ;and store them to 6805 RAM
00143  [2] 0130  A356          cpx     #$56
00144  [3] 0132  26F3          bne     getLoop
00145  [5] 0134  1F01          bclr    t1cs,t1csp          ;when done, disable T1 CS
00146
00147                ;*************************************************************************
00148                ;   -- Display Time and Date --
00149                ;
00150                ;   This section of code takes the data read from the T1 and stored in
00151                ;   the RAM of the 6805 and displays it as the time and date in the
00152                ;   following format:
00153                ;                   Day HH:MM:SS a.m.
00154                ;                   Month Day, Year
00155                ;   Thus, the current time as this code is being written is:
00156                ;                   Thu 10:42:03 a.m.
00157                ;                   Aug 21, 1997
00158                ;*************************************************************************
00159
00160                display
00161  [3] 0136  BE53          ldx     day                 ;display day
00162  [5] 0138  DE027F          ldx     dayTable,x          ;use pointer table to send address
00163  [6] 013B  CD01D7          jsr     txlcd1              ;of day message to txlcd1 routine
00164  [3] 013E  B652          lda     hours               ;display hours (set up as am/pm)
00165  [2] 0140  A41F          and     #$1F                ;mask upper 3 bits (am/pm and mode)
00166  [6] 0142  CD01C4          jsr     txbcd               ;send to LCD
00167  [2] 0145  A63A          lda     #':'                ;send colon
00168  [6] 0147  CD01EF          jsr     wdlcd
00169  [3] 014A  B651          lda     minutes             ;send the minutes to the LCD
00170* [6] 014C  AD76           jsr     txbcd
00171  [2] 014E  A63A          lda     #':'                ;and then another colon
00172  [6] 0150  CD01EF          jsr     wdlcd
00173  [3] 0153  B650          lda     seconds             ;and then the seconds
00174* [6] 0155  AD6D           jsr     txbcd
00175  [2] 0157  A620          lda     #$20                ;and then a space
00176  [6] 0159  CD01EF          jsr     wdlcd
00177  [2] 015C  A670          lda     #'p'                ;test B5 of the hours data. If set,
00178  [5] 015E  0A5202          brset   5,hours,pm          ;send "p.m." to the LCD. If clear,
00179  [2] 0161  A661          lda     #'a'                ;send "a.m."
00180                pm
00181  [6] 0163  CD01EF          jsr     wdlcd
00182  [2] 0166  A66D          lda     #'m'
00183  [6] 0168  CD01EF          jsr     wdlcd
00184  [3] 016B  BE55          ldx     month               ;use month data as pointer into
00185  [5] 016D  DE0287          ldx     monthTable,x        ;string table. Send address of
00186* [6] 0170  AD65           jsr     txlcd1              ;month text to txlcd1 routine
00187  [3] 0172  B654          lda     date                ;send the day of the month
00188* [6] 0174  AD4E           jsr     txbcd
00189  [2] 0176  A62C          lda     #$2c                ;send a comma
00190* [6] 0178  AD75           jsr     wdlcd
```

```
00191  [2] 017A  A620          lda    #$20              ;send a space
00192* [6] 017C  AD71          jsr    wdlcd
00193  [2] 017E  A619          lda    #$19              ;send the 1st two digits of the year
00194  [3] 0180  BE56          ldx    year              ;if the year data is >$96, send
00195  [2] 0182  A396          cpx    #$96              ;a $19. If less, send $20. This will
00196  [3] 0184  2202          bhi    20th              ;give this program a date range from
00197  [2] 0186  A620          lda    #$20              ;1-1-1997 to 12-31-2096
00198                  20th
00199* [6] 0188  AD3A          jsr    txbcd             ;send upper digits of the year
00200  [3] 018A  B656          lda    year              ;then the lower digits
00201* [6] 018C  AD36          jsr    txbcd
00202
00203* [3] 018E  208A          jmp    displayLoop       ;and loop
00204
00205                  ;*************************************************************************
00206                  ;   -- Subroutines --
00207                  ;*************************************************************************
00208
00209                  ;*************************************************************************
00210                  ;   -- writet1 --
00211                  ;
00212                  ;   This subroutine writes a byte of data, contained in A, to the
00213                  ;   address contained in X.
00214                  ;*************************************************************************
00215
00216                  writet1
00217  [5] 0190  1E01          bset   t1cs,t1csp        ;enable T1
00218  [4] 0192  B757          sta    lcdtemp1          ;preserve A in temp location
00219  [3] 0194  B60B          lda    spsr              ;read SPSR and SPDR to clear
00220  [3] 0196  B60C          lda    spdr              ;the SPIF flag
00221  [2] 0198  9F            txa                      ;X contains the T1 reg. address
00222  [2] 0199  AA80          ora    #$80              ;Set write bit (B7)
00223  [4] 019B  B70C          sta    spdr              ;and send to T1
00224  [5] 019D  0F0BFD        brclr  spif,spsr,*       ;wait for tx
00225  [3] 01A0  B657          lda    lcdtemp1          ;get data byte to send
00226  [4] 01A2  B70C          sta    spdr              ;and send it
00227  [5] 01A4  0F0BFD        brclr  spif,spsr,*       ;wait for tx
00228  [3] 01A7  B60C          lda    spdr              ;read SPDR to clear SPIF
00229  [5] 01A9  1F01          bclr   t1cs,t1csp        ;disable T1
00230  [6] 01AB  81            rts                      ;and return
00231
00232                  ;*************************************************************************
00233                  ;   -- readt1 --
00234                  ;
00235                  ;   This subroutine reads a byte of data, from the address contained
00236                  ;   in X and returns it in A.
00237                  ;*************************************************************************
00238
00239                  readt1
00240  [5] 01AC  1E01          bset   t1cs,t1csp        ;enable T1
00241  [3] 01AE  B60B          lda    spsr              ;clear the SPIF flag
00242  [3] 01B0  B60C          lda    spdr
00243  [2] 01B2  9F            txa                      ;move the address to A
00244  [2] 01B3  A47F          and    #$7F              ;clear W/R bit (B7)
00245  [4] 01B5  B70C          sta    spdr              ;and send
00246  [5] 01B7  0F0BFD        brclr  spif,spsr,*       ;wait for tx
00247  [4] 01BA  B70C          sta    spdr              ;write anything to SPDR to start tx
00248  [5] 01BC  0F0BFD        brclr  spif,spsr,*       ;wait for tx
00249  [3] 01BF  B60C          lda    spdr              ;get received data
00250  [5] 01C1  1F01          bclr   t1cs,t1csp        ;disable T1
00251  [6] 01C3  81            rts                      ;and return
00252
00253                  ;*************************************************************************
00254                  ;   -- txbcd --
00255                  ;
00256                  ;   This subroutine takes a two BCD digits, contained in A, and writes
00257                  ;   them to the LCD. The data in A MUST BE BCD DIGITS!
```

```
00258                ;*************************************************************************
00259
00260                txbcd
00261  [4] 01C4  B75D          sta     mytemp              ;preserve A
00262  [3] 01C6  44            lsra                        ;shift high nibble to low nibble
00263  [3] 01C7  44            lsra                        ;to send the upper BCD digit to
00264  [3] 01C8  44            lsra                        ;the LCD
00265  [3] 01C9  44            lsra
00266  [2] 01CA  AB30          add     #$30                ;add $30 to convert to ASCII
00267* [6] 01CC  AD21          jsr     wdlcd               ;and write the data to the LCD
00268  [3] 01CE  B65D          lda     mytemp              ;get the original value
00269  [2] 01D0  A40F          and     #$0F                ;and mask to the lower nibble
00270  [2] 01D2  AB30          add     #$30                ;add $30 to convert to ASCII
00271* [6] 01D4  AD19          jsr     wdlcd               ;and send to the LCD
00272  [6] 01D6  81            rts
00460
00461                ;*************************************************************************
00462                ;   -- SWI Interrupt Routine --
00463                ;
00464                ;   When called, this ISR will take the values in the 6805 RAM locations
00465                ;   $50 to $56 and write them into the T1 clock data registers so as
00466                ;   to set the time.
00467                ;*************************************************************************
00468                swiVec
00469  [5] 0262  1E01          bset    t1cs,t1csp          ;enable t1
00470  [2] 0264  A6A0          lda     #$A0                ;write to all clock registers
00471  [M] 0266              sendspi                       ;starting at address $20
00472  [2] 026D  AE50          ldx     #seconds            ;point X at 6805 RAM locations
00474                setLoop
00475  [3] 026F  F6            lda     x                   ;get data to send
00476  [M] 0270              sendspi                       ;and write to all 7 clock registers
00477  [3] 0277  5C            incx                        ;($20-$26) in the T1
00478  [2] 0278  A356          cpx     #$56
00479  [3] 027A  26F3          bne     setLoop
00480  [5] 027C  1F01          bclr    t1cs,t1csp          ;when done, disable T1 CS
00481  [9] 027E  80            rti
00482
00483                ;*************************************************************************
00484                ;   -- Pointer Tables --
00485                ;
00486                ;   These two tables are used by to get the address of the weekday and
00487                ;   month display strings. This address is used by txlcd1 to display
00488                ;   the text on the LCD.
00489                ;*************************************************************************
00490
00491                dayTable
00492     027F  0000060C        db      0,0,6,12,18,24,30,36
         0283  12181E24
00494                monthTable
00495     0287  2A2A3036        db      42,42,48,54,60,66,72,78,84,90,42,42,42,42,42,42,96,102,108
         028B  3C42484E
         028F  545A2A2A
         0293  2A2A2A2A
         0297  60666C
00497                ;*************************************************************************
00498                ;   -- Display Text --
00499                ;
00500                ;   This text is used to show the weekday and the month on the LCD.
00501                ;   The BCD data from the T1 is used as a pointer into the dayTable and
00502                ;   monthTable tables (above). The data loaded from the table is the
00503                ;   offset of the text message from mesPage label. This offset is used
00504                ;   by txlcd1 to find and display the text. Note that the first byte of
00505                ;   each message is an LCD control character used to position the display
00506                ;*************************************************************************
00507
00508     0300               section mesPage,(*&$FF00+$100)      ;start at even page boundary
00509
```

```
00510     0300  8053756E        db      $80,"Sun ",$00
          0304  2000
00511     0306  804D6F6E        db      $80,"Mon ",$00
          030A  2000
00512     030C  80547565        db      $80,"Tue ",$00
          0310  2000
00513     0312  80576564        db      $80,"Wed ",$00
          0316  2000
00514     0318  80546875        db      $80,"Thu ",$00
          031C  2000
00515     031E  80467269        db      $80,"Fri ",$00
          0322  2000
00516     0324  80536174        db      $80,"Sat ",$00
          0328  2000
00518     032A  C04A616E        db      $C0,"Jan ",$00
          032E  2000
00519     0330  C0466562        db      $C0,"Feb ",$00
          0334  2000
00520     0336  C04D6172        db      $C0,"Mar ",$00
          033A  2000
00521     033C  C0417072        db      $C0,"Apr ",$00
          0340  2000
00522     0342  C04D6179        db      $C0,"May ",$00
          0346  2000
00523     0348  C04A756E        db      $C0,"Jun ",$00
          034C  2000
00524     034E  C04A756C        db      $C0,"Jul ",$00
          0352  2000
00525     0354  C0417567        db      $C0,"Aug ",$00
          0358  2000
00526     035A  C0536570        db      $C0,"Sep ",$00
          035E  2000
00527     0360  C04F6374        db      $C0,"Oct ",$00
          0364  2000
00528     0366  C04E6F76        db      $C0,"Nov ",$00
          036A  2000
00529     036C  C0446563        db      $C0,"Dec ",$00
          0370  2000
00531     1FF4                  section vectors, $1ff4
00532     1FF4  0100              dw    code
00533     1FF6  0100              dw    code
00534     1FF8  0100              dw    code
00535     1FFA  0100              dw    code
00536     1FFC  0262              dw    swiVec
00537     1FFE  0100              dw    code
```