The Real-Time Clock (RTC) on the IMSAI 8080

By Bruce E. Hall, W8BH

Did you know that the IMSAI 8080 replica, by TheHighNibble, comes with a built-in real time clock? You can query the time in CP/M 3.0. But what about CP/M 2.2? Are we out of luck?

Of course not. The RTC is available no matter what operating system is in use. This article describes how to access the RTC via assembly language programming (and BASIC!), and how to use the RTC to display time and date.

The RTC programming interface

In our emulated IMSAI 8080, the RTC is implemented by sending a command to the command I/O port (such as, "get hour") and then reading the result from the data port. This programming interface does not attempt to emulate any known clock hardware.

Here is a table of all the RTC commands. Sending a command is simple: output a command byte to port 41h. Then, to read the result, read a byte from port 42h. In Z80 assembler, it would look like this:

RTC Command	Description
00h	Get Second
01h	Get Minute
02h	Get Hour
03h	Get Days (LSB) since 1/1/1978
04h	Get Days (MSB) since 1/1/1978
05h	Get Day of Month
06h	Get Month (Jan=0, Feb=1, etc)
07h	Get Years since 1900
FFh	Toggle format of results from BCD (default) to Decimal

```
CLKCMD      .EQU    41h         ; RTC command port
CLKDAT      .EQU    42h         ; RTC data port

getRTC:     OUT  (CLKCMD),A     ; send RTC command
            IN   A,(CLKDAT)     ; get RTC value
            RET
```

Retrieving the time variables is just a matter of sending RTC commands and saving results:

```
getTime:    LD   A,0
            CALL getRTC         ; RTC 0 = seconds
            LD   (sec),A        ; save seconds
            LD   A,1
            CALL getRTC         ; RTC 1 = minutes
            LD   (min),A        ; save minutes
            LD   A,2
            CALL getRTC         ; RTC 2 = hours
            LD   (hour),A       ; save hours
            LD   A,3
            CALL getRTC         ; RTC 3 = days/LSB
            LD   (days),A       ; save days LSB
            LD   A,4
            CALL getRTC         ; RTC 4 = days/MSB
            LD   (days+1),A     ; save days MSB
            LD   A,5
            CALL getRTC         ; RTC 5 = day of month
            LD   (day),A        ; save day
            LD   A,6
            CALL getRTC         ; RTC 6 = month
            LD   (mon),A        ; save month
            LD   A,7
            CALL getRTC         ; RTC 7 = year
            LD   (year),A       ; save year
            RET
```

A note about the result format is in order. The default result format, except for commands 3 & 4, is binary coded decimal (BCD). BCD can be hard to manipulate arithmetically. To put the results in decimal format, send the FF command beforehand:

```
            LD   A,0FFh         ; Toggle format command –
            OUT  (CLKCMD),A     ; change BCD to decimal & vice versa
```

The toggle command Is super-sticky, which presents a unique problem. Run your application once, and the RTC format toggles to decimal. Run your application a second time, and the format toggles back to BCD! To use the toggle command effectively, you should check the current format and then toggle as needed. Here is example code to force decimal format:

```
forceDec:   IN   A,(CLKCMD)     ; read RTC format (0=BCD,1=decimal)
            OR   A              ; set flags
            RET  NZ             ; return if already decimal format
            LD   A,0FFh         ; must be in BCD format, so
            OUT  (CLKCMD),A     ; toggle format: BCD->decimal
            RET
```

BCD is easier to display. We will stick with BCD format and manipulate it when necessary.

Printing the Time

Here is a routine to print the time in 24-hour HH:MM:SS format:

```
pTime24:    LD   A,(hour)
            CALL pBCD           ; print hours
            LD   A,':'
            CALL putCh
            LD   A,(min)
            CALL pBCD           ; print minutes
            LD   A,':'
            CALL putCh
            LD   A,(sec)
            CALL pBCD           ; print seconds
            RET
```

The routine pBCD displays the contents of register A as a two-digit BCD value. See rtc1.asm for the full program listing.

Printing the Date

Printing time is straightforward; let's try the same approach with the date. Compare the code above with the following:

```
pDate:      LD   A,(mon)
            CALL pBCD           ; print month
            LD   A,'/'
            CALL putCh
            LD   A,(day)
            CALL pBCD           ; print day
            LD   A,'/'
            CALL putCh
            LD   A,(year)
            CALL pBCD           ; print year
            RET
```

Unfortunately, it doesn't work. Today is 8/10/25, but this routine returns the odd result of "07/10/<5". We need to make a few adjustments for it to work.

First, look back at the table on page 1, and you'll notice that the month result is a number, 0 through 11, where January=0, February=1, etc. We'll need to add 1 to convert it to the month number we expect. And we can't just increment the value, since it is stored in BCD format. Fortunately, the Z80 provides a special opcode, DAA, to handle this BCD math.

Second, the year is returned as the number of years since 1900. This worked great in 1980, when the returned value of "80" would be an appropriate date number. But in 2025, a date number of 125 just looks wrong. Moreover, there is a "Y2K" error in the emulation code: its BCD conversion routine does not expect a value greater than 99 (for year 1999). When it gets passed a value of 125 (for year 2025), it erroneously converts this into the *non-BCD* hex value C5 instead of the BCD value of 25. Oops!

The following code handles both issues, and assumes dates in the 2000-2099 range:

```
pDate:      LD   A,(mon)        ; months 0-11
            INC  A              ; conv to months 1-12
            DAA                 ; adjust for BCD math
            CALL pBCD           ; print month
            LD   A,'/'
            CALL putCh
            LD   A,(day)
            CALL pBCD           ; print day
            LD   A,'/'
            CALL putCh
            LD   A,(year)
            SUB  0A0h           ; change baseyear 1900 to 2000
            CALL pBCD           ; print year
            RET
```

The full program listing is given in rtc2.asm.

Calculating the Day of the Week

There is no RTC variable for day of the week, so we'll need to derive it from the values we have. The calculation is relatively simple: day of the week = (days-1) % 7, where % is the modulo (remainder) operator. This formula works because day #1, January 1st 1978, fell on a Sunday. Seven days later it was also a Sunday, etc. Imagine a very long calendar beginning in January 1978, where each day is represented by days since 1/1/78:

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
0	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	32	33	34...

As you can see, for any day on that calendar, the remainder following divide-by-7 gives us the number of days since the preceding Sunday... which is the day of the week, 0-6.

There is just one small problem. Our Z80 instruction set does not have a modulo function. We don't even have multiplication or division. So, we can't do divide-by-seven. We could create (or copy) a general-purpose divide routine, but this is overkill for our immediate needs. Keeping in mind that division can be thought of as serial subtraction, let's try subtracting 7 from the day until we can't go any further. The subtractions don't need to be counted, because we aren't trying to determine the division result (quotient). We just want the remainder. Consider the following code:

```
calcDOW:    LD   HL,(days)      ; HL = days since 1/1/78
            DEC  HL             ; dont count current day
            LD   DE,7           ; divisor is 7
```

```
            XOR  A              ; clear carry for next op
jDW:        SBC  HL,DE          ; subtract the divisor
            JR   NC,jDW         ; keep subtracting until neg
            ADD  HL,DE          ; undo last subtraction
            LD   A,L            ; remainder now in HL
            LD   (dow),A        ; store remainder
            RET
```

Amazingly, the modulo function is implemented by the three highlighted instructions. First, a two-instruction loop repeatedly subtracts 7 until the carry bit is set. The carry indicates that we've gone one subtraction too far, so we add back 7 to get the remainder. Neat!

Printing the Day of the Week

I created a 28-character string containing 4 characters for each day of the week:

S	u	n	0	M	o	n	0	T	u	e	0	W	e	d	0	T	h	u	0	F	r	i	0	S	a	t	0

The day of the week value, multiplied by 4, is a convenient byte offset into the string. For example, if the day of the week is 3 (Wednesday), start with the character at byte offset 3*4 = 12. "WED" starts 12 bytes from the beginning of the string.

```
printDOW:   LD   A,(dow)        ; A = day of week
            RLCA                ; A x 2
            RLCA                ; A x 4
            LD   B,0
            LD   C,A            ; offset BC = dow x 4
            LD   HL,stDow       ; point to days of week
            ADD  HL,BC          ; add in offset
            CALL putStr         ; print day of week
            RET
```

See rtc3.asm for a complete program that prints the date, including the day of the week.

Overcoming limitations of the CP/M 3 interface

CP/M 3 directly provides time information, shielding the user from the RTC implementation details. However, it does not provide all the variables listed above. Through BDOS function 105, CP/M 3 provides the hour, minute, second, and days (since 1/1/1978). It does not provide the day, month, and year! How can you get those?

One way is to bypass BDOS and use the RTC command set above. To stay within CP/M 3 though, things get a little more complicated. It is possible to determine the day, month, and year from the day counter alone. To do this, I took inspiration from the breaktime() routine in Paul Stroffregen's Arduino Time library. The idea is simple:

1. From 1978 onwards, start subtracting a year's worth of days (accounting for leap years) until you cannot subtract any more days without going negative. This is the current year. The remaining count is the day-of-the-year.
2. Calculate the month by subtracting a month's worth of days at a time (accounting for the actual number of days in each month) until you cannot subtract any more days without going negative. This is the current month. The remainder is the day of the month.

To see this code, refer to the program listing in rtc4.asm.

Putting it all together

Use the routines above to create whatever time and/or date string you want. For example, I chose to print the time in 12-hour format, followed by the day of the week, followed by the date.

```
start:      LD   SP,1000h       ; establish local stack
            CALL getTime        ; hour/min/sec/days, etc
            LD   HL,stTime
            CALL putStr         ; "The time is "
            CALL pTime12        ; print time in 24-hr format
            CALL printDOW       ; print day of week
            LD   A,' '
            CALL putCh          ; followed by space
            CALL pDate          ; then print the date
            CALL crlf
fin:        JP   0000           ; return to CP/M
```

See rtc5.asm for the complete program listing. Or, if you prefer using XYBASIC, check out rtc.bas, which displays time and date using the same principles.

Happy Hacking!

Bruce.

Last updated: 12 Aug 2025

Other articles in this series: IMSAI 8080 articles