TIME AND DATE

FUNCTIONS

FOR THE

HP-28S

by Kevin P. Jessup

DISCLAIMER

The material in this manual is provided AS IS and published without representation or warranty of any kind. Neither the publisher nor the author shall have any liability, consequential or otherwise arising from the use or misuse of any of the material contained herein.

COPYRIGHT NOTICE

INTRODUCTION

The HP-28S is undoubtedly the most powerful handheld calculator on the market. As a previous HP-41CX owner however, I do miss the input/output capability and the time and date functions. This software is an attempt to implement some of the time, date and alarm functions that were available for the HP-41CX as well as some other useful time and date functions and general purpose utilities.

Approximately 35 programs and utilities are described in this manual. The documentation describes their function, the stack input and resulting output. Like most programs written for the HP-28, these are highly structured and all programs in the package should be used together within a single directory unless you choose to locate certain general utilities in an upper level directory.

The first section of the manual is devoted to general purpose utilities. I suggest that the programs be placed in the HOME directory as they will then be available for use by other programs besides those contained herein. Section two is concerned with general purpose time and date utilities. You may wish to place some of these in the HOME directory as well if you believe they will be of use to you in other programs.

Sections three and four are devoted to the time/date programs and the alarm programs respectively. All of these routines should be located in the same subdirectory. You will probably want TIME to be the name of the subdirectory. If you plan on using alarms, you will have to load all the programs in this manual.

These programs are too lengthy to be of any value on the HP-28C. Collectively, the programs will require about 6500 bytes of memory. A basic time function for the HP-28C is described in the book *CUSTOMIZE YOUR HP-28* by W.A.C. Mier-Jedrzejowicz.

This manual assumes a solid understanding of the basic HP-28S functions and program entry. An understanding of time and date mathematics or extensive HP-28S programming experience is not required. Books on astronomy and computer science are available if you wish to further your understanding of Julian and Gregorian calendars or other time and date conversions. These programs should also serve as a solid base if you desire to further their capabilities. If you are not experienced in HP-28S program entry, read the owners manual before attempting to enter these programs.

GETTING STARTED

The programs will be described from the bottom up in order to provide a better understanding of just how the upper level programs function. Also, the upper level programs can not run without the low level subroutines, which is why they are presented in this order. If you prefer, just key them all in and run the CLOCK function. If you did everything right, the day of the week, date and time will be displayed on your HP-28S. That assumes of course that you set the date and time first using SETDT.

It is however, highly unlikely that you will load all programs without any errors. It is recommended that you key in each program and use the given example to verify its operation before proceeding with the entry of the next program. Unless otherwise specified, each program is required for both the time/date and alarm functions. Please note these programs require that the LAST option be enabled.

Do not locate programs that are associated with the manipulation of lists or arrays in an upper level directory. Keep all such programs and the global variables that they operate on in the same directory as the programs that call them as subroutines. For example, the DAT→S (date to string) program uses the MOY (month of year) list and both should be located in the same directory that contains the CLOCK program (CLOCK calls DAT→S which requires MOY). If this is confusing, just keep all programs in this manual in a subdirectory called TIME and no problems will occur.

Each program has a specific name. Do not change the name of the program. If the name is changed, other programs that call the program as a subroutine will not be able to find it.

A listing for each program or variable is provided. Key in the operations after the "Listing:" header in the documentation.

Beginning HP-28S programmers will undoubtedly be confused by the presence of a colon string (":") within some of the programs here presented. The colon character does not appear on any of the HP-28S keys. The following method can be used to load colons or any other non-keyable character into an HP-28S program.

1.) Determine the number of colons required in the program.
2.) Place 58 in level one. Execute the CHR command. A ":" string should now be on level one.
3.) Duplicate the colon string on the stack so there are as many colon strings as there are colons in the program.
4.) Use + to sum all the strings together.
5.) Edit the string of colons on level one.
6.) Go in and out of the editors insert mode to build your program around the colon string. Break up the colon string as needed to move the individual characters to wherever they are needed in the program.
7.) Hit ENTER to terminate the edit session.

Name: PAD
Type: Program
Function: Pad string with spaces. Left or right justified.

Listing: `<< → j << OVER SIZE - IF DUP 0 > THEN 1 SWAP START " " IF j THEN SWAP END + NEXT ELSE DROP END >> >>`

Input:			
	Level 3:	text	string
	Level 2:	desired length	real number
	Level 1:	n	real number
			Right justified if n=1, left justified if n=0.

Output:		
	padded text	string

Example: Pad the string "TEST" so it totals 15 characters and is right justified.

Level 3: "TEST"
Level 2: 15
Level 1: 1
PAD
Level 1: " TEST"

Notes: String is unchanged if string length is greater than or equal to the desired length.

Name: CENST
Type: Program
Function: Take a string from level 1 and append spaces to the front so when displayed on the LCD it will appear centered on the screen.

Listing: `<< DUP SIZE 2 / 11.5 + IP 1 PAD >>`

Input:			
	Level 1:	text	string

Output:			
	Level 1:	text	string

Example: Level 1: "ABCDEFG"
CENST
Level 2: " ABCDEFG"

Name: PRMT
Type: Program
Function: Prompt user for input using default value and text string on stack.

Listing: << DEPTH → str dep << CLLCD str 4 DISP HALT IF DEPTH dep == THEN SWAP DROP END >> >>

Input:

Level 2:	Default value	any object
Level 1:	Prompt	string

Output: User is prompted with the default value in level one. This value may be edited or replaced. A new object may simply be entered leaving the default object one level above it. The user MUST press CONT when ready to proceed. Do not DROP the default value without replacing it with an alternate value.

Notes: This program is used by the ALARM routines only.

Name: KEYW
Type: Program
Function: Wait for a key hit and return its text string value.

Listing: << DO KEY UNTIL END >>

Input: none

Output:

Level 1:	key	string

Example: KEYW
User hits the up arrow.
Level 1: "UP"

Notes: This program is used by the ALARM routines only.

Name: TONE
Type: Program
Function: Sound a beep sequence.

Listing: << 1 5 START 2000 .02 BEEP 4000 .01 BEEP NEXT >>

Input: none

Output: tone sequence

Notes: This program is used by the ALARM routines only.

Name: KYWT
Type: Program
Function: Wait for key with TONE.

Listing: << DO TONE 1 WAIT UNTIL KEY END >>

Notes: Performs like KEYW above but beeps tone till key hit. This program is used by the ALARM routines only.

Name: DLE
Type: Program
Function: Delete specified element from list.

Listing: << OVER RCL SIZE → n e s << IF e s ≤ THEN 1 e FOR i n i GE
NEXT DROP IF s e > THEN e 1 + s FOR i n i GET NEXT END s
- →LIST n STO END >> >>

Input:

Level 2:	'NAME'	list name
Level 1:	n	element number to delete

Output: Modified list stored in memory.

Example: List named TEST contains: { "HELLO" (2,4) 3.14 }

Level 2: 'TEST'
Level 1: 2
DLE

List named TEST contains: { "HELLO" 3.14 }

Notes: List must be stored in the directory from which you execute the program. This program is used by the alarm routines only.

Name: TPAR
Type: Binary integer
Function: Clock counter comparison value.

Notes: This variable is used to by the JDATE routine to calculate the current Julian date from the current clock counter value. It is set to its initial value by the SETDT function. Thereafter, it is never modified. This variable will be created when SETDT is run for the first time. SETDT must be run before the time/date or alarm routines can be used!

Name: MOY
Type: List
Function: Month of year text string list.

Listing:
```
{ "Jan"  "Feb"  "Mar"  "Apr"  "May"  "Jun"
  "Jul"  "Aug"  "Sep"  "Oct"  "Nov"  "Dec" }
```

Notes: This list is used to get a textual representation of the month based on a real number.

Name: DOW
Type: List
Function: Day of week text string list.

Listing:
```
{ "Sunday"  "Monday"  "Tuesday"  "Wednesday"
  "Thursday"  "Friday"  "Saturday" }
```

Notes: This list is used to get a textual representation of the day of week based on a real number.

Name: CLK12
Function: Sets 12 or 24 hour time display format.
Type: Real number

Notes: Create this variable and load it with 0 for 24 hour format or 1 for 12 hour AM/PM format.

Name: →DMY
Type: Program
Function: Convert compacted Day Month Year to expanded Day Month Year.

Listing: `<< IP LAST FP 100 * IP LAST FP 10000 * >>`

Input:	Level 1:	DD.MMYYYY	real number
Output:	Level 3:	DD	real number
	Level 2:	MM	real number
	Level 1:	YYYY	real number

Example:
Level 1: 27.111956
→DMY
Level 3: 27
Level 2: 11
Level 1: 1956

Name: DMY→
Type: Program
Function: Convert expanded Day Month Year to compacted Day Month Year.

Listing: `<< 10000 / + 100 / SWAP IP + >>`

Input:	Level 3:	DD	real number
	Level 2:	MM	real number
	Level 1:	YYYY	real number
Output:	Level 1:	DD.MMYYYY	real number

Example:
Level 3: 27
Level 2: 11
Level 1: 1956
DMY→
Level 1: 27.111956

Name: TM→S
Type: Program
Function: Convert time from real number format to text string.

Listing:
```
<< RCLF 0 → time f pm << 9 FIX IF CLK12 THEN IF time 12 ≥ THEN
1 'pm' STO IF time 13 ≥ THEN time 12 - 'time' STO END ELSE IF
time 1 < THEN time 12 + 'time' STO END END END time 100 + →STR
DUP 2 3 SUB ":" + OVER 5 6 SUB + ":" + SWAP 7 8 SUB + IF
CLK12 THEN IF pm THEN " PM" ELSE " AM" END + END f STOF >> >>
```

Input: Level 1: HH.MMSS real number

Output: Level 1: "HH:MM:SS" string

Example: Level 1: 17.3001
TM→S
Level 1: "05:30:01 PM" or "17:30:01" based on CLK12 value

Name: DAT→S
Type: Program
Function: Convert date from real number format to text string.

Listing:
```
<< RCLF 0 FIX SWAP →DMY 10000 + →STR 2 5 SUB 3 ROLLD "-"
'MOY' 3 ROLL GET + "-" + SWAP 100 + →STR 2 3 SUB SWAP +
SWAP + SWAP STOF >>
```

Input: Level 1: DD.MMYYYY real number

Output: Level 1: "DD-Mmm-YYYY" string

Example: Level 1: 27.111956
DAT→S
Level 1: "27-Nov-1956"

Name: G→JL
Type: Program
Function: Convert Gregorian date to Julian date.

Listing:
```
<< → d m y << IF m 3 < THEN m 12 + 'm' STO y 1 - 'y' STO END
IF IF 1582 y > THEN 0 ELSE IF 1582 y < THEN 1 ELSE IF 10 m <
THEN 1 ELSE IF 10 m > THEN 0 ELSE IF d 15 < THEN 0 ELSE 1 END
END END END END THEN y 100 / IP DUP 4 / IP 2 3 ROLL - + ELSE 0
END 365.25 y * IP + 30.6001 m 1 + * IP + d + 1720994.5 + >> >>
```

Input:			
	Level 3:	DD.ddddd	real number
	Level 2:	MM	real number
	Level 1:	YYYY	real number

Output:			
	Level 1:	DDDDDDDD.ddd	real number

Example: Find the Julian date for April 21, 1989 at 3:15:00 PM.

Place 3.1500 on the stack. Then convert the time to 24 hour format by adding 12 to the time. Then convert HMS to HRS by using the HMS→ function available in the TRIG menu. Divide by 24 to get the fractional portion of the date. Add 21, the current day of the month. You now should have 21.6354166667 on the stack. Continue as shown below...

Level 3:	21.6354166667
Level 2:	4
Level 1:	1989
G→JL	
Level 1:	2447638.13542 (the Julian date)

Name: JL→G
Type: Program
Function: Convert Julian date to Gregorian date.

Listing:
```
<< .5 + DUP IP SWAP FP 0 0 0 0 → i f c d e g << IF i 2299160 >
THEN i 1867216.25 - 36524.25 / IP DUP 4 / IP - 1 + i + ELSE i END
1524 + 'c' STO c 122.1 - 365.25 / IP 'd' STO d 365.25 * IP 'e' STO
c e - 30.6001 / IP 'g' STO c e - f + g 30.6001 * IP - IF g DUP
13.5 < THEN 1 ELSE IF g 13.5 > THEN 13 ELSE 0 END END - IF DUP
2.5 > THEN 4716 ELSE IF DUP 2.5 < THEN 4715 ELSE 0 END END d
SWAP - >> >>
```

Input:			
	Level 1:	DDDDDDDD.ddd	real number

Output:			
	Level 3:	DD.ddddd	real number
	Level 2:	MM	real number
	Level 1:	YYYY	real number

Example: Find the Gregorian equivalent of Julian day number 2435804.5

Level 1: 2435804.5
JL→G
Level 3: 27
Level 2: 11
Level 1: 1956

Name: RSCLK
Type: Program
Function: Read system timer.

Listing: << #11CAh SYSEVAL >>

Input: none

Output: Level 1: count 48 bit binary integer

Notes: For version 2BB of the HP-28S. If your machine has a different SYSEVAL call for the system clock, modify this routine with the corresponding address. See the table below.

ROM Version	SYSEVAL ADDRESS
1BB	#123E hex
1CC	#1266 hex
2BB	#11CA hex
New Version	#????? hex

The binary word size of the HP-28S must be at least 48 bits in length for this function to return an accurate value. Note that the upper level time and date routines that make use of this function take care of that for you. You can find the version of your HP-28 by placing #Ah on the stack and executing SYSEVAL. The system timer provides the time standard for all the time date and alarm functions. The programs in this manual will not run if this routine does not return the system timer value. The author is not responsible for changes to the HP-28S internal program code that result in a timer SYSEVAL address other then those listed above.

Name: SSCLK
Type: Program
Function: Continuously display system timer.

Listing: << RCLF 64 STWS CLLCD DO RSCLK 1 DISP UNTIL KEY END DROP STOF CLMF >>

Input: none

Output: Timer count on line 1 of LCD display.

Notes: Hit any key except ON to exit. This routine is not required by any of the time, date or alarm functions.

Name: LPYR
Type: Program
Function: Determines if year is a leap year.

Listing: `<< → y << y 4 MOD NOT y 100 MOD AND y 400 MOD NOT OR >> >>`

Input:	Level 1:	yyyy	real number
Output:	Level 1:	n	real number

Note: Output is 1 if a leap year, else 0.

Notes: None of the programs in this manual use this routine. It is provided for your conveniance only.

Name: RDOW
Type: Program
Function: Determine day of week given Gregorian date.

Listing: `<< RCLF 0 FIX SWAP →DMY G→JL 1.5 + 7 MOD SWAP STOF >>`

Input:	Level 1:	DD.MMYYYY	real number
Output:	Level 1:	n	real number
Example:	Level 1:	27.111956	November 27, 1956
	RDOW		
	Level 1:	2	Tuesday

Notes: Output ranges from 0 to 6 for Sunday to Saturday.

Name: SDOW
Type: Program
Function: Same as DOW above but output is a text string.

Listing: `<< 'DOW' SWAP RDOW 1 + GET >>`

Example: Above example would return "Tuesday" to the stack.

Name: DDATE
Type: Program
Function: Calculate new date given count and starting date on stack.

Listing: `<< →DMY G→JL + JL→G DMY→ >>`

Input:	Level 2:	count	real number
	Level 1:	DD.MMYYYY	real number
Output:	Level 1:	DD.MMYYYY	real number

Example: What was the date 1000 days previous to April 1, 1987?

Level 2:	-1000	
Level 1:	1.041987	
DDATE		
Level 1:	5.071984	July 5, 1984

Name: DDAYS
Type: Program
Function: Calculate number of days between two dates.

Listing: `<< →DMY G→JL SWAP →DMY G→JL - >>`

Input:	Level 2:	DD.MMYYYY	real number (start)
	Level 1:	DD.MMYYYY	real number (end)
Output:	Level 1:	n	real number (delta)

Example: Find the number of days between November 27, 1956 and January 31, 1989.

Level 2:	27.111956	
Level 1:	31.011989	
DDAYS		
Level 1:	11753	positive difference

Name: TDS
Type: Program
Function: Get text strings for time, date and day of week.

Listing:
```
<< DUP DAT→S SWAP SDOW 3 ROLL TM→S 3 ROLLD >>
```

Input:	Level 2:	hh.mmss	real number
	Level 1:	dd.mmyyyy	real number
Output:	Level 3:	time	string
	Level 2:	date	string
	Level 1:	day of week	string

Example:
```
Level 2:  16.1530
Level 1:  8.061989
TDS
Level 3:  "04:15:30 PM"
Level 2:  "08-Jun-1989"
Level 1:  "Thursday"
```

Name: SETDT
Type: Program
Function: Set current date and time.

Listing:
```
<< RCLF 64 STWS RSCLK 4 ROLL →DMY 6 ROLLD 6 ROLLD 4 ROLL
HMS→ 24 / + 5 ROLL 5 ROLL G→JL 707788800 * R->B -
#9CCEC6C7C6B4AFB5h XOR 'TPAR' STO STOF >>
```

Input:	Level 2:	DD.MMYYYY	real number (todays date)
	Level 1:	HH.MMSS	real number (current time)

SETDT

Output: TPAR updated so routines involving current date and time return correct values. Routine leaves no parameters on stack.

Example: Set date and time to April 24, 1989 at 2:00:01 PM.

```
Level 2:  24.041989
Level 1:  14.0001
SETDT
```

Notes: SETDT must be run if the remaining routines in this manual are to produce a valid output.

Name: JDATE
Type: Program
Function: Get todays Julian date.

Listing:
```
<< RCLF 64 STWS RSCLK TPAR #9CCEC6C7C6B4AFB5h XOR - B→R
707788800 / SWAP STOF >>
```

Input: none

Output: n real number

Notes: Run SETDT first to set the current date and time.

Name: DATE
Type: Program
Function: Return two real numbers representing the current date and time to the stack.

Listing:
```
<< JDATE JL→G 3 PICK FP 24 * →HMS 4 ROLLD DMY→ >>
```

Input: none

Output:

Level 2:	HH.MMSS	real number
Level 1:	DD.MMYYYY	real number

Example: DATE

Level 2:	21:5902	(9:59:02 PM)
Level 1:	1.021986	(February 1, 1986)

Notes: Run SETDT first to set the current date and time.

Name: CLOCK
Type: Program
Function: Make a digital clock on the LCD display.

Listing:
```
<< CLLCD DO DATE TDS " " + SWAP + CENST 1 DISP CENST 3 DISP
UNTIL KEY END DROP CLMF >>
```

Input: none

Output: Clock on LCD display.

Notes: Hit any key except ON to gracefully exit. Run SETDT first to set the current date and time.

Name: PTD
Type: Program
Function: Print time and date. (HP 82240 printer required)

Listing: << DATE TDS " " + SWAP + CENST PR1 CR DROP CENST PR1 DROP CR >>

Input: none

Output: Time and date is printed.

Notes: Run SETDT first to set the current date and time.

Name: ASET
Type: Program
Function: Set an alarm.

Listing:
```
<< CLLCD DATE SWAP "Time (HH.MMSS) ?" PRMT HMS→ 24 / SWAP
"Date (DD.MMYYYY) ?" PRMT →DMY 4 ROLLD 4 ROLLD + 3 ROLL 3
ROLL G→JL "" "Message ?" PRMT 0 "Reschedule (HHH.MMSS) ?"
PRMT HMS→ 24 / 3 →LIST 1 →LIST 'ADAT' IFERR RCL THEN STO
ELSE + 'ADAT' STO END >>
```

Input: User is prompted.

Output: ADAT variable created or modified.

Notes: ASET will prompt for...
1.) Alarm time (with current time default in level 1)
2.) Alarm date (with current date default in level 1)
3.) Alarm message (with null text string default in level 1)
4.) Reschedule interval (with 0 default in level 1)

Example: Set an alarm for 5:30 AM on January 1st, 1990 so you can get up and enjoy your hangover. Just to make sure you do get up, set a 10 minute reschedule interval.

ASET	
ASET prompts:	Time (HH.MMSS) ?
You enter:	5.3 then press CONT
ASET prompts:	Date (DD.MMYYYY)
You enter:	1.011990 then press CONT
ASET prompts:	Message ?
You enter:	"Get up." then pres CONT
ASET prompts:	Reschedule ?
You enter:	.1 then press CONT

The variable ADAT should have been created. Press ADAT in the user menu to recall it to the stack. It should look like this:

```
{ { 2447892.72917 "Get up." 6.94444444446E-3 } }
```

The first element is the Julian date at which the timer will alarm. The second element is the alarm message and the third is the reschedule interval in days. If the reschedule interval is set to 0, the alarm will not be rescheduled after it times out. In other words, if you want the alarm to sound only once, enter a reschedule interval of zero. As more alarms are added the ADAT list will grow in size. You MUST run ASET from the directory in which the time, date and alarm programs are located.

Name: ADISP
Type: Program
Function: Display an ADAT alarm element.

Listing:
```
<< 1 GETI JL→G 3 PICK FP 24 * →HMS 4 ROLLD DMY→ TDS DROP
"DATE:  " SWAP + 1 DISP "Time:  " SWAP + 2 DISP GETI 4 DISP
GET IF DUP THEN 24 * →HMS CLK12 0 'CLK12' STO SWAP TM→S
SWAP 'CLK12' STO "Repeat: " SWAP + 3 DISP ELSE DROP END >>
```

Input: Level 1: { n "string" n } ADAT list element

Output: Alarm data displayed.

Example: Level 1: { 2447892.72917 "Get up." 6.94444444446E-3 }
ADISP
Will display...

Date:	01-Jan-1990	on line 1
Time:	05:30:00 AM	on line 2
Repeat:	00:10:00	on line 3
Get up.		on line 4

Notes: This is a subroutine called by ALIST and ACHEK described below. If the reschedule interval of the alarm is $\geq$ 100 hours, the hours portion will be truncated to 2 digits. If the reschedule interval is zero, it will not be displayed.

Name: ALIST
Type: Program
Function: Lists all alarms.

Listing:
```
<< IFERR 'ADAT' RCL THEN DROP ELSE SIZE 1 'ADAT' → s i n <<
CLLCD WHILE s i ≥ i AND REPEAT n i GET ADISP IF KEYW DUP "C"
== THEN DROP "DELETED" CENST CLLCD 2 DISP n i DLE i 1 - 'i'
STO s 1 - 's' STO ELSE IF "Q" == THEN s 'i' STO END END i 1 +
'i' STO END CLMF >> END >>
```

Input: none

Output: Alarms are displayed as described in ADISP above. Display halts with each alarm. Hit any key except ON, C or Q to display the next alarm. Hitting C will clear the displayed alarm. Hitting Q will exit the program. Program will automaticly exit after displaying the last alarm in the list.

Name: ACHEK
Type: Program
Function: Sound an alarm if it has timed out.

Listing:
```
<< IFERR 'ADAT' RCL THEN DROP ELSE IF SIZE DUP THEN 1
SWAP → i e << DO 'ADAT' i GET IF DUP 1 GET JDATE ≤
THEN CLLCD DUP ADISP IF KYWT CLLCD "ENTER" == THEN " ALARM
ACKNOWLEDGED..." 1 DISP IF DUP 3 GET THEN "RESCHEDULED"
CENST 3 DISP JDATE OVER 1 GET - OVER 3 GET / LAST SWAP
DROP SWAP IP 1 + * OVER 1 GET + 1 SWAP PUT 'ADAT' SWAP i
SWAP PUT ELSE "DELETED" CENST 3 DISP 'ADAT' i DLE i 1 - 'i'
STO e 1 - 'e' STO DROP END ELSE DROP END CLMF ELSE DROP
END UNTIL i 1 + 'i' STO i e > END >> ELSE DROP END END >>
```

Input: none

Output: If an alarm has timed out, it will be displayed as in ADISP above with a repeating beep tone till a key is hit. If the ENTER key is hit, the alarm will be rescheduled based on the reschedule interval. If the reschedule interval is 0, the alarm will be deleted. If any other key (except ON) is hit, the program will not acknowledge the alarm. This process will repeat till all alarms have been checked.

ALARM USAGE

Detecting an alarm requires that you manually execute the ACHEK program. There is no hardware control that will automaticly sound an alarm as on the HP41. One thing you may want to do is place an ACHEK call within the CLOCK program. That way, alarms will sound whenever one times out while the CLOCK program is running. If you want to do this, insert the ACHEK call just before the UNTIL command in the CLOCK program described above.

When setting an alarm, the alarm time may vary by as much as one second from the time you intended. For example, an alarm set for 8:00 AM may be displayed as 07:59:59. This is normal and due to rounding errors.

When a continuous alarm is rescheduled, it is set to the next alarm interval after the current time. For example, if a continuous alarm was due at 10:00 AM with a 1 hour reschedule interval and it is 1:15 PM when detected, the alarm will be rescheduled for 2:00 PM.

It is impractical to set alarm reschedule intervals of less than 1 hour. Unless you have no other goals in life, you probably don't want to spend your evenings watching alarms time out! The alarm programs were written to provide a simple method of reminding you of daily meetings or other activities. They should not be used to provide audible indications of time intervals that require the accuracy of a stopwatch.

When an alarm is detected, the 28S will beep till acknowledged with a key hit. You should not be away from the 28S (at such a distance that you can not hear the alarm) for an extended period of time with alarm monitoring active or battery drain will be excessive. You can run the clock program continuously if you like but remember that running any program continuously can cause accelerated battery drain.

TIMER ROLL-OVER

The HP-28S has a 48-bit timer that these programs use as a time base. With 8192 increments of this timer occurring every second, this gives a range of (2 ^ 48) / 8192 or 34359738368 seconds or about 1088 years. Unfortunately, this is only true if the timer value is 0 when the time is set. If the timer rolls over or becomes corrupted for any other reason, the time and date will have to be reset. To check when timer roll-over will occur on your system, follow the procedure below. Make sure the binary word size is set to 49 bits or greater before proceeding.

1.) Place #1000000000000h on the stack.
2.) Run the RSCLK program described in this manual.
3.) Push the - key (subtract the two numbers).
4.) Execute B→R (binary to real).
5.) Divide by 707788800.

This gives the number of days till timer roll over. Divide by 365.25 to get the number of years till roll-over. In all probability, one of the following will occur before this happens.

1.) You will lose battery power and the timer will change to a random value.
2.) You will accidentally write over the timer with a complex assembly language program.
3.) The 28S will fail.
4.) HP will have made something much better than the 28S so you won't be using it anyway!
5.) You won't be around for the event!

As you can see, timer roll-over should not be a source of great anxiety when other factors have a much greater affect on its long term stability or importance.

LONG TERM ACCURACY

The accuracy of the time and date over an extended period is entirely dependent on the accuracy of the HP-28S oscillator hardware. The routines were tried on five different HP-28S systems and accuracy was within 3 seconds per month on one machine and within 2 minutes per month on the worst of the five. If accuracy is a problem, please note that it is not that difficult to reset the time and date once or twice a month.

SYSTEM SPEED MODIFICATIONS

There are a variety of ways to increase the clock speed of an HP-28S. Some of these affect the 48-bit system timer, others do not. An unmodified HP-28S increments the system timer 8192 times a second. If the speed up you have implemented changes this increment rate, the SETDT and JDATE algorithms will have to be modified to account for a greater system clock speed. If, for example, you have done a 2 times speed up, double the constant of 707788800 (this is the number of ticks per day) found in the SETDT and JDATE routines. The author is not responsible for changes to the HP-28S circuit design or program code that change the timer increment rate, thus rendering these programs inaccurate.

Congratulations. By now you have finished the laborious, but hopefully worthwhile entry of the time and date functions. Words of praise, death threats, job offers and other commentary may be sent directly to me at the address below.

Kevin P. Jessup
9118 North 85th Street
Milwaukee, WI 53224