SECTION V

TIMING SOFTWARE

Timing software enables the user to specify a preset length of time and to signal the user when that time has elapsed. In theory, up to 255 software timers are available to the user.

The Z80-CPU's non-maskable interrupt (NMI) input, which comes from the VDP interrupt output, forms the time base for all the timers. In the U.S., it is about every 1/60 second. In the European version, it is about every 1/50 second. TIME_MGR is the routine responsible for generating the time base at the desired intervals.

All timer routines use a CRAM area designated as the TIMER_TABLE. The size of this table depends on the number of timers in use and their types. There are two types of timers, non-repeating and repeating.

The user will be notified of the status of the timer only when he checks it.

5.1 Non-Repeating Timers

These timers will not repeat themselves after time out. The user will be notified and their timers are set free.

5.2 Repeating Timers

These timers only need to be set once. After each time out they will notify the user and repeat themselves.

For both types of timers, the timer length can be either short or long:

(a) Short - 1 to 255 units of time base.
(b) Long - 256 to 65535 units of time base.

5.3 TIMER_TABLE:

As a timer is requested, it is placed into TIMER_TABLE. Each timer consists of a Mode_Byte and a two-byte Value_Word.

The appropriate amount of CRAM should be reserved using the following formula:

```
TIMER_TABLE          DEFS      Num_of_Timers * 3
TIMER_DATA_BLOCK     DEFS      Num_of_Long_Repeating * 4
```

NOTE: Num_of_Timers is the total number of timers.

5.3.1 Mode_Byte

Mode_Byte is one byte of data for each timer containing the information of done bit, repeat bit, free bit, long bit and last-timer-in-table bit (Refer to Appendix G for details).

5.3.2 Value_Word

A two-byte value which can have several meanings depending on the type of timer.

(a) Short Timers:

The Value_Word is the actual timer in this case.

The first byte is the value to be decremented and the second byte is the initial timer value. In the case of a repeating timer, the second byte is used to restart the timer.

(b) Long Non-Repeating Timers:
The Value_Word is the value of the timer and is decremented as a tyo-byte quantity.

(c) Long Repeating Timers:
The Value_Word is the address of the location in the TIMER_DATA_BLOCK where the first word is the value to be decremented and the second word is the initial timer value.

5.3.3 TIMER_DATA_BLOCK

This is the data area in CRAM where four bytes are designated for each long repeating timer. The table's size is expandable under user control, so care should be taken not to write over other pertinent data.

5.4 INIT_TIMER

Calling Sequence:

```
LD    HL, TIMER_TABLE
LD    DE, TIMER_DATA_BLOCK
CALL  INIT_TIMER
```

Description:

INIT_TIMER initializes timer data areas to the locations defined as inputs.

Parameters:

TIMER_TABLE	This is the CRAM address where the timer information will be placed.
TIMER_DATA_BLOCK	This is the address where long repeating timer data will be placed.

Side Effects:

- Destroys DE and HL.

5.5 TIME_MGR

Calling Sequence:

```
CALL  TIME_MGR
```

Description:

TIME_MGR is responsible for maintaining all OS software timers. The task of maintenance is defined as updating all timers, setting the proper signal code when a timer times out, and restarting repeating timers.

Each call to TIME_MGR will cause all active timers to be decremented by one. There is no limit as to when the routine could be called, but typically it is every NMI from VDP which forms the system time base.

An active timer is defined as a timer with its repeat bit set or its done bit not set, or both.

If an entire timer value decrements to zero, the done bit will be set in Mode_Byte. In addition, the timer will be restarted if it is a repeating type.

Parameters: None.

Side Effects:

- Destroys AF, DE and HL.

5.6 REQUEST_SIGNAL

Calling Sequence:

```
LD    HL, TIMER_LENGTH
LD    A, REPEAT
CALL  REQUEST_SIGNAL
```

Description:

REQUEST_SIGNAL accepts a time interval and a repeat code (Boolean) as input. The REPEAT parameter, when set, instructs TIME_MGR to re-initialize the timer when it times out instead of relinquishing the timer memory locations.

REQUEST_SIGNAL sets up a timer and assigns that timer a number in the accumulator. The routine also determines the type of timer and allocates space in the TIMER_TABLE accordingly.

Short Timer:

A short timer has a counter value of 255 or less and uses one Mode_Byte and Value_Word.

Long Timer:

A long timer has counter values greater than 255.

(a) Non-Repeating:

A non-repeating timer uses a Mode_Byte and Value_Word.

(b) Repeating:

A repeating timers uses a Mode_Byte and Value_Word in addition to four bytes starting at the first available location in the TIMER_DATA_BLOCK.

The user should save the timer number. This value, referred to as SIGNAL_NUM, should subsequently be used when calling TEST_SIGNAL to find the status of the signal or when calling FREE_SIGNAL to release a timer.

Parameters:

TIMER_LENGTH	The number of the time base units of a timer. Values range from 1 (shortest) to 0FFFFH (longest).
REPEAT	1 = repeating timer; 0 = non-repeating timer.
Output:	Value of timer number returned in accumulator. User should save it in CRAM location SIGNAL_NUM.

Side Effects:

- Destroys AF, BC, DE, and HL.

5.7 TEST_SIGNAL

Calling Sequence:

```
LD    A, SIGNAL_NUM
CALL  TEST_SIGNAL
```

Description:

TEST_SIGNAL takes a signal number and tests to see whether the indicated timer has timed out since the last time it was tested. If so, it returns with "true" in the accumulator; otherwise, it returns "false". The zero flag reflects the contents of the accumulator.

If the timer of SIGNAL_NUM tested has its Done bit set and the timer is non-repetitive, then the Free bit will be set to release the timer for further use.

If no timer of a particular signal number exists then the routine will return a false.

Although it has been defined that testing a non-existing signal number will return a false value, a common error in use of timing routines is the testing of an undefined signal.

The error occurs when one module, with a given SIGNAL_NUM, calls TEST_SIGNAL with that SIGNAL_NUM as input. If this module receives a "true" from TEST_SIGNAL, then another module which is rightfully using a timer with that SIGNAL_NUM will not receive a "done" signal.

Parameters:

SIGNAL_NUM	Timer number.

Side Effects:

- Destroys AF (output), BC, DE, and HL.

5.8 FREE_SIGNAL

Calling Sequence:

```
LD    A, SIGNAL_NUM
CALL  FREE_SIGNAL
```

Description:

FREE_SIGNAL takes a SIGNAL_NUM value as input and upon finding a timer assigned to that number, releases it to the free list. If no timer of that SIGNAL_NUM is found, no action will be taken. This routine will free a timer regardless of its current value or its REPEAT parameter.

Special case of long repeating timer:

This routine will release a portion of the TIMER_DATA_BLOCK that a particular timer uses and moves the remaining contents up. The Value_Words of other

repeating timers will also be modified to reflect this move.

NOTE: In this special case, TIME_MGR, or any other routine that accesses or modifies the TIMER_TABLE, should NOT be called during the execution of FREE_SIGNAL. (This may occur if TIME_MGR was called on interrupt). ColecoVision Bulletin No. 0010 (Appendix D) suggests the solution of using DEF_INT to defer interrupts.

Parameters:

SIGNAL_NUM	Previously defined output from REQUEST_SIGNAL.

Side Effects:

- Destroys AF, BC, DE and HL.