CC-193

COMPUTATION CENTER
Massachusetts Institute of Technology
Cambridge 39, Massachusetts

March 30, 1962

To: All Programmers

From: Marjorie M. Daggett, F. J. Corbató, and J. R. Steinberg

Title: MITMR, an FMS subprogram for using the IBM Interval Timer clock

Purpose: MITMR is a general purpose subprogram that supplies the necessary co-ordination of the use of the IBM 7090 interval timer clock by the Fortran-FAP Monitor System and use by the individual monitor jobs and, as such, should be the only FMS programmer method of using the interval timer clock. It provides the user, during his execution time, with the ability to use the interval timer clock as either a stop watch or an alarm clock or both. Another entry to MITMR allows the user to interrogate the clock and determine the amount of time used since the start of his job. This subprogram has been co-ordinated with the system so that a run is terminated approximately n minutes after initiation, where n is the maximum running time as specified on the I.D. card. (See Procedures Handbook for I.D. card description.)

Description: The MIT-FMS system sets the clock to go off n minutes after the start of the job, where n is the maximum job time as specified on the I.D. card. An initializing entry from .SETUP to MITMR at the beginning of user execution time sets the time left for the job to run in a table with the entry location of the EXIT subprogram as its associated entry. When the clock goes off, control is transferred to EXIT and the job is terminated. Thus, in the normal case, even though a user's job time is up, a post-mortem can be taken.

The IBM clock increments in 60^{ths} of a second. This time unit is used in the calling sequences for both setting the clock and reading the clock.

Calling Sequences:

a. Two entries are provided to allow using the clock as a stop watch in order that sections of a program may be timed.

To reset the stopwatch to zero -

```
CALL    RSCLCK          in FORTRAN, or
EXECUTE RSCLCK.         in MAD.
```

To read the time in 60^{ths} of a second since the last call to RSLCK -

```
CALL    STØPCL(I)       in FORTRAN, or
EXECUTE STØPCL.(I)      in MAD,
```

where I is an integer variable that will contain upon exit from the routine the time used in 60^{ths} of a second.

b. At any time, a user may ask for the time since the beginning of the job in 60^{ths} of a second by using the call:

```
CALL    JØBTM(J)        in FORTRAN, or
EXECUTE JØBTM.(J)       in MAD,
```

where J is an integer variable that upon exit from the subprogram contains the time used since the start of the job in 60^{ths} of a second.

The amount of time left for a job to run can be found by issuing:

```
CALL    TIMLFT(J)       in FORTRAN, or
EXECUTE TIMLFT.(J)      in MAD,
```

where J is an integer variable that upon exit from the subprogram contains the time left for the job to run in 60^{ths} of a second.

c. The interval timer clock may also at any time be used as an alarm clock. The FORTRAN calling sequence for this is:

```
ASSIGN  S   TØ  N
CALL    TIMER(I , N)
```

where I is an integer variable that specifies the time for the clock to run in 60^{ths} of a second and S is the statement to which control is sent when the clock goes off. N. B. Due to the logic of the Fortran compiler, there must be a "seeming" path of flow to statement S. See Example 2 in section e.

The MAD calling sequence is:

```
EXECUTE   TIMER.(I , S)
```

where I is as above and S is a statement label to which control will transfer when the clock goes off.

There may be no more than nine (9) active calls to TIMER at any time.

After control has been sent to location S at the end of the timer interval and after examination of certain conditions, the user may desire to complete the set of instructions (or process) that was interrupted.

This may be done by issuing:

```
CALL    RSTRTN          in FORTRAN, or
EXECUTE RSTRTN.         in MAD.
```

The effect of this call is to restore the machine conditions to the state they were when the timer went off and to return to the instruction interrupted. Specifically, the only machine conditions disturbed and preserved are the AC, the MQ, the three index registers, and the AC overflow light.

d. If a TIMER call has been set up as a protective measure, it may be desired to turn off the alarm signal. This can be done by:

```
CALL    KILLTR          in FORTRAN, or
EXECUTE KILLTR.         in MAD.
```

It should be noted that this kills only the current alarm trap-time set up, unless the current alarm trap-time is the job shut-off time. The job shut-off setting cannot be killed even with repeated calls to KILLTR.

Method: Both the FMS system and the user are making use of the interval timer for various purposes; hence, it is desirable to have one program in core at all times co-ordinating the use of the clock. Owing to space limitations in the existing system and the work involved in changing the system, an adequate but asymmetric way of dealing with the timer has been worked out that is compatible with the existing version of the Monitor.

The Monitor Sign-On record sets the clock to interrupt at the time specified on the I.D. card; it also sets the trapping location for the clock to transfer to a lower core program that only notes the fact the clock went off and returns to the instruction interrupted. This is called the "monitor setting" of the clock. Thus, if a user's job time is exceeded during any Monitor phase of operation, such as translation or loading, that phase of operation will be completed before the job is terminated. During an execution job, .SETUP, which must be in all main programs, calls an initializing entry of this subprogram (TIME) to change the trapping location for the clock to transfer internally to MITMR. Thus, the subprogram MITMR as well as .SETUP will be a part of every user's run. The initializing entry (TIME) enters the time left for the job to run in the table used by the entry TIMER and enters an associated transfer to EXIT. Thus, if the job time is exceeded during user execution, EXIT is called in the normal fashion, (i.e., before the job time shut off) it will reset the trapping location to the Monitor setting so that even if the clock then interrupts, the normal taking of a postmortem will be completed and the job terminated.

On each entry the user makes to TIMER, the alarm period is checked against the existing time periods in the table and is entered in the table in proper sorted sequence so that the table is always sorted according to ascending values of time periods. Thus, even though a user requests a time period which exceeds the job time left, the sort insures that the job time setting of the clock and its alarm will precede the excessive request. When the alarm clock goes off, the next entry in the table is used as the clock setting and control is sent to the location specified in the calling sequence to TIMER. The job time entry in the table is also marked so that repeated entries to KILLTR will never kill that entry in the table.

Restrictions: The table that TIMER uses is ten (10) registers long; hence, only nine (9) active entries at any time can be made to TIMER. If

the table is full, TIMER requests will be ignored.

The use of PDUMP and DUMP subprograms destroys any timing being done by the TIMER subroutine (except the job time). If PDUMP is entered, the trapping location for the timer is changed to the Monitor Setting and restored to the user setting upon completion. PDUMP uses KILLTR to remove all timer entries except the job setting and sets the clock for the remaining job time. The use of FTNPM and FTNBP also destroy any timing being done by the TIMER subroutine (except the job time). The procedure followed after a call to FTNPM or FTNBP is the same as that followed by PDUMP.

On chain jobs, at the end of a link, CHAIN will change the setting of the clock trapping location to the Monitor setting and set the clock for the job time that remains. This Monitor setting will be operative until the next link is brought in and the main program goes to (TIME) via .SETUP. At this time, the Monitor indicator is checked. If the timer went off while the link was being loaded, control is sent to EXIT; if the clock did not go off, the procedure is the same, i.e., the trapping location is changed for the user setting for this link and the user may time any sections of his program in this link. Thus, whenever CHAIN is entered, all TIMER settings are cancelled; there is no carry-over of information from one link to another.

e. Examples:

```
(1)             Timing a loop
                CALL RSCLCK
                DO  5  I = I,N
      5         A (I) = B(I) * C(I) + A(I)
                CALL STØPCL.(J)
                PRINT  3 , J
      3         FØRMAT (20 H TIME ØF DO LOOP IS I6, 19H 60THS OF A SECOND.)
(2)             Using the alarm clock
C               THIS LOOP IS WRONG IF IT TAKES MORE THAN TWO (2) SECONDS.
C               THE FOLLOWING THREE (3) STATEMENTS PROVIDE A PATH OF FLOW
C               TO STATEMENT 20.
                ASSIGN  1  TØ  N
                GØ  TØ  N, (1, 20)
      1         CØNTINUE
                ASSIGN  20  TØ  N
                CALL TIMER (2 * 60, N)
                DO  5  I = 1 , K
                DO  5  J = 1 , L
      5         A(I,J) = A(I,J) + Q(I,J) * T(I,J) / B(I,J)
                .
                .
                .
C               ERROR IN PARAMETERS IF MORE THAN 2 SECOND LOOP
C               PRINT VARIABLES THAT MAY BE OF INTEREST.
     20         PRINT 4, K, L
      4         FØRMAT (22 H DO 5 LØØP INTERRUPTED 5I10)
```