JGT has seen

Micro-Concurrent Pascal: 1802 System Evaluation

Nigel Maund

August 5, 1983

DIVISION OF SEISMOLOGY AND GEOMAGNETISM

Earth Physics Branch
Energy, Mines and Resources Canada
Ottawa, Canada

Internal Report

1984-2 (S)

Table of Contents

Introduction

Micro-Concurrent Pascal, a Pascal dialect, is a high-level language that supports real-time, inter-tasking programming. The mCP language offers process and monitor constructs that allow any number of processes to run independently but at the same time share data and communicate with each other. An mCP program requires a certain amount of organization in defining the structure of these software constructs. This underlying structure, different from a standard Pascal program, promotes a systematic approach to creating error-free real time systems.

MCP programs are compiled on a host computer which performs extensive compile-time checking. The compiler produces pseudo code which may then be downloaded to the target system. On the target system a 4.6 kilobyte Interpreter/Kernel executes the program. The P-code (pseudo code) that is produced is position-independent, reentrant, and ROMable. The mCP program can also access assembly language routines.

The mCP program which must be interpreted will not execute as fast as an equivalent assembly language program. The 1802 Interpreter/Kernel will execute approximately 1950 P-codes per second (using a 2.4576 Mhz crystal). This P-code execution speed plays a major factor in the interrupt handling efficiency of mCP. Since most real time applications invlove interrupts, mCP must be able to handle interrupts efficiently. This report will examine the specifics of the interrupt handling scheme of mCP. The Enertec real time clock program, ERTC18, will be used in order to outline the steps taken by the kernel in servicing an interrupt.

It is assumed that the reader is familiar with the mCP inter-processing organization. An outline of the ERTC18 program will be given although a more comprehensive description is given in the Enertec manual.

Steps in Handling Interrupts

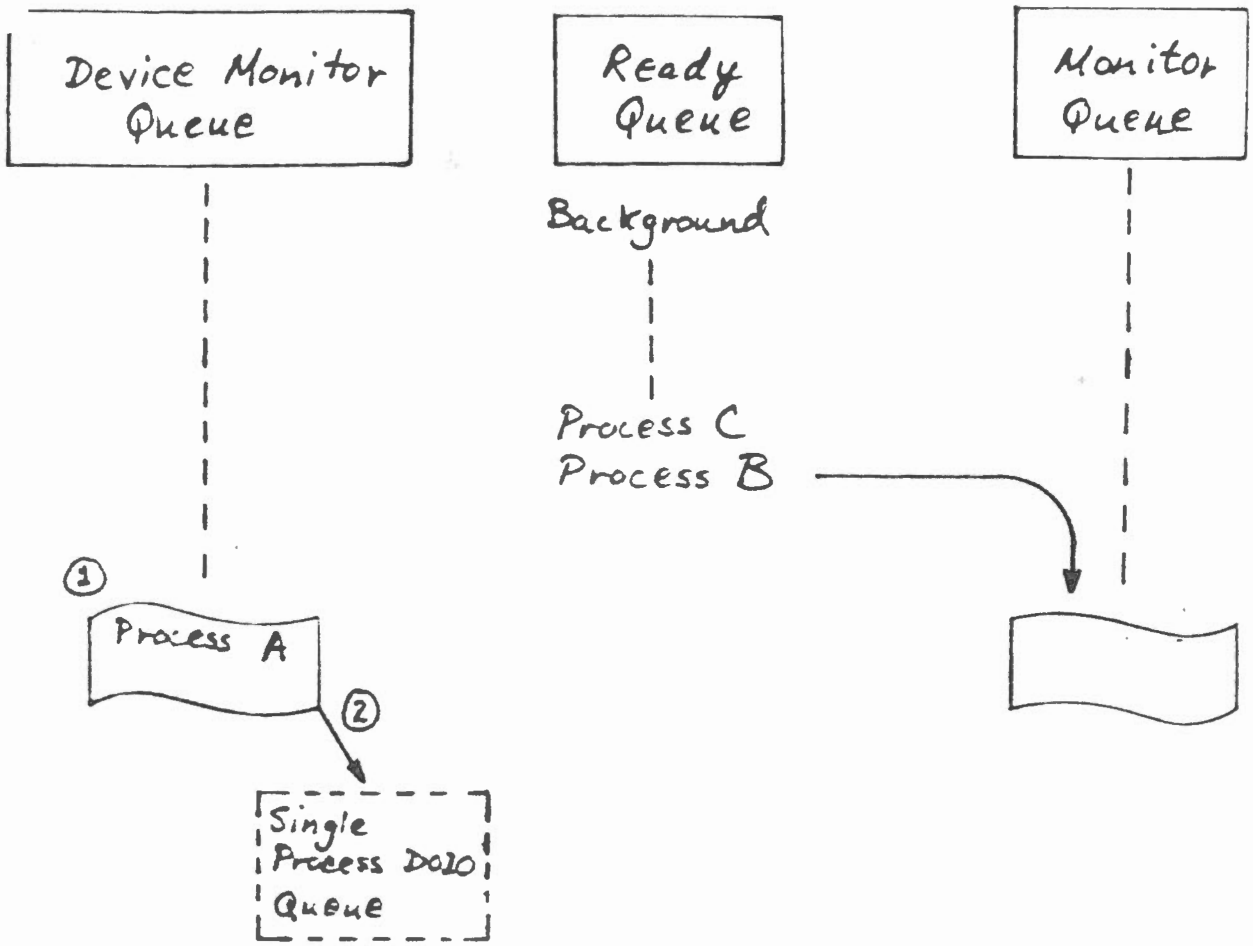
The interrupt handling system in mCP is based on the mCP DOIO P-code instruction. This instruction may only occur in a device monitor. On execution, it causes the currently running process to be preempted leaving that process to wait for an interrupt to occur. The kernel will then fetch the next process on the ready queue and allow it to run. When an interrupt occurs, the kernel identifies the source of interrupt, preempts the current running process, and resumes operation of the process that had been preempted and was waiting for the interrupt (see fig. 1).

In the present system up to 32 levels of interrupts can be handled, each having a unique priority, and these priorities may be altered dynamically.

The ERTC18 program operates a real time clock. The operator may set the current time from a terminal and view the current time on the terminal display. The program updates the time on receiving a 1 hertz interrupt from a clock source. In total, three interrupts must be recognized by the operating system: clock interrupt, UART receive and UART transmit interrupts, with the clock interrupt having the highest priority.

The program consists of two processes: the clock process and the operator process. These processes are continuous loops. The clock process handles the clock interrupts in the clock device monitor, and encapsulates the time data structure in the clock monitor. The operator process allows interaction between the user and the system. It accesses the Uart read and write device monitors and allows the user to access the time through the clock monitor. The clock process is continuously running, responding to clock interrupts and updating the time data in the clock monitor. The clock monitor can be accessed by the operator process and thus serves as the bridge between the two processes. The operator process continually waits for operator input to read or change the current time in the clock monitor.

FIGURE 1.(a) Simple System Response to a DOIO Instruction

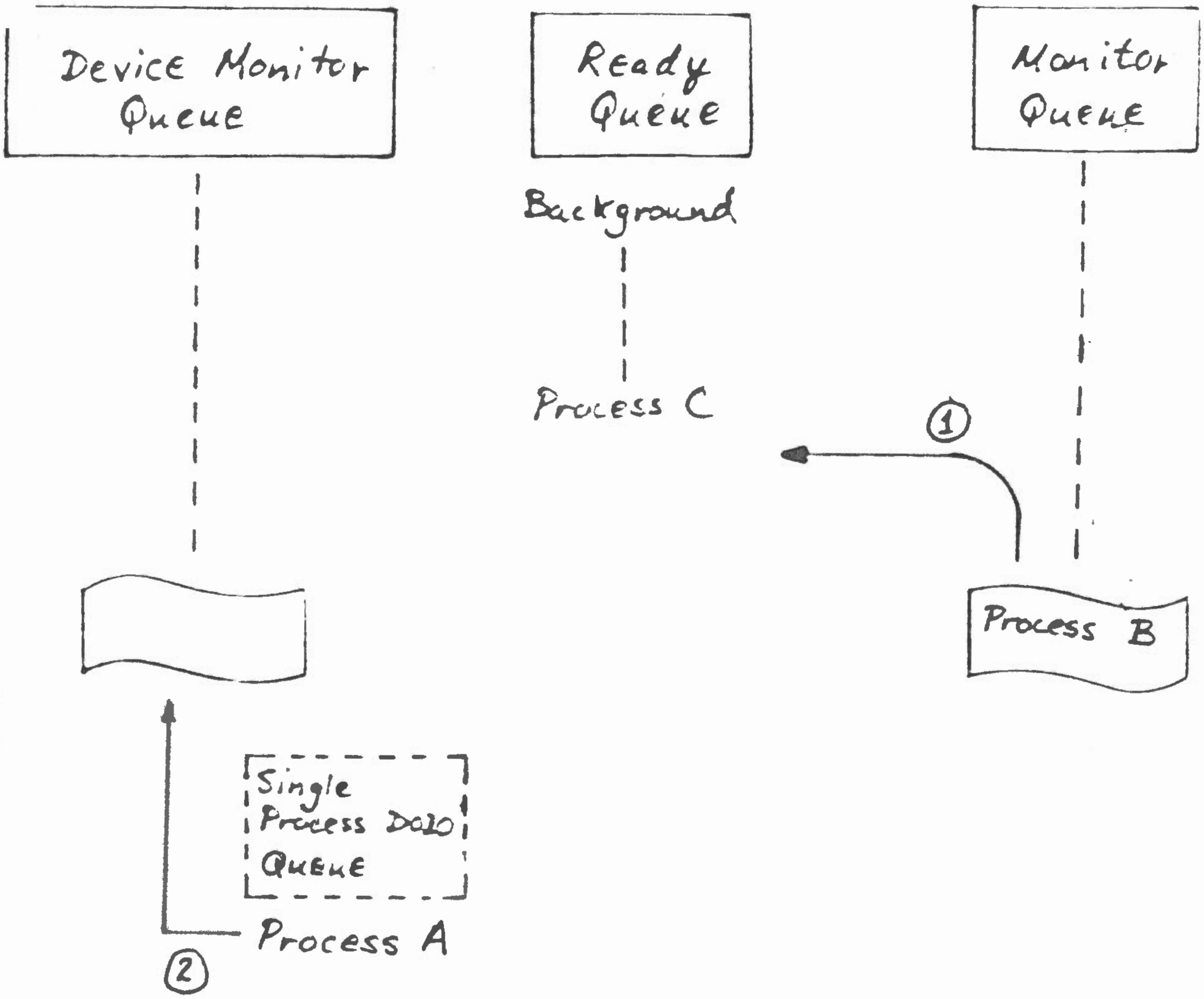


1. Process A active in Device Monitor issues a DOIO statement.

2. Process A preempted and placed on the device monitor's Single Process DOIO Queue.

3. Process at head of Ready Queue is run (i.e. Process B enters Monitor).

FIGURE 1.(b)



1. Interrupt occurs, currently running process (Process B) is preempted and placed at head of Ready Queue.

2. Process waiting for interrupt, Process A, restarted (in device monitor).

The program flow is shown in figure 2. When the program is run, the time is reset to 00:00:00. In the clock device monitor a DOIO statement preempts the clock process and the clock process then waits for an interrupt. The other process, the operator process, is then scheduled, initializing the UART and sending a prompt (">") to the terminal. The operator process is then preempted on execution of a DOIO statement in the UART read device monitor, and the process then waits for the operator to enter a command on the terminal.

At this point the only process which is running is the background process. Both clock and operator processes have been preempted and are placed on the clock single process DOIO queue and the UART single process DOIO queue, respectively. The system in this state will from now on be referred to as the "minimum state". It is in the minimum state because all monitor queues are empty and the only process on the ready queue is the background process (see figure 3).

FIGURE 2 ERTC18 Program Flow

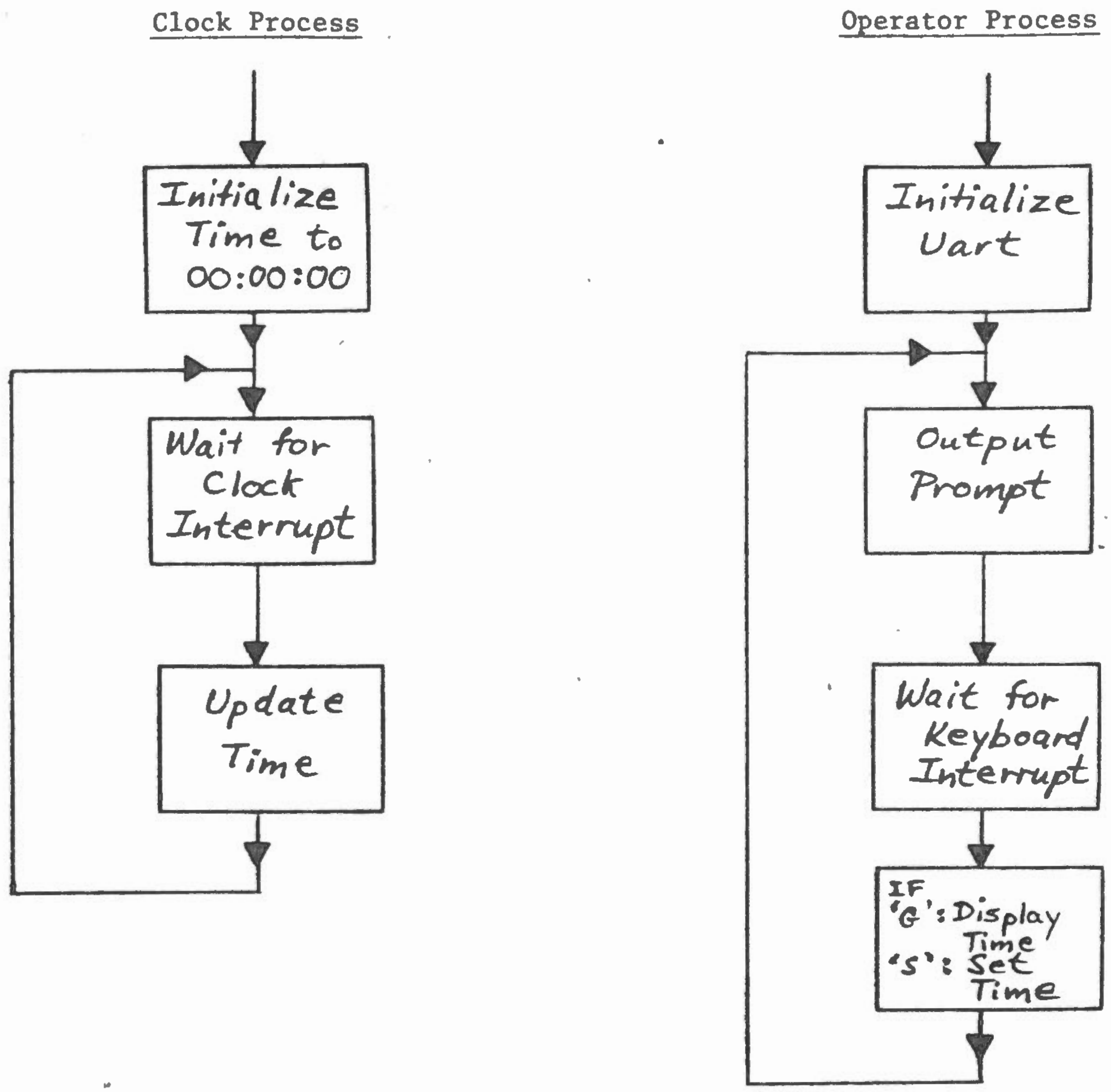
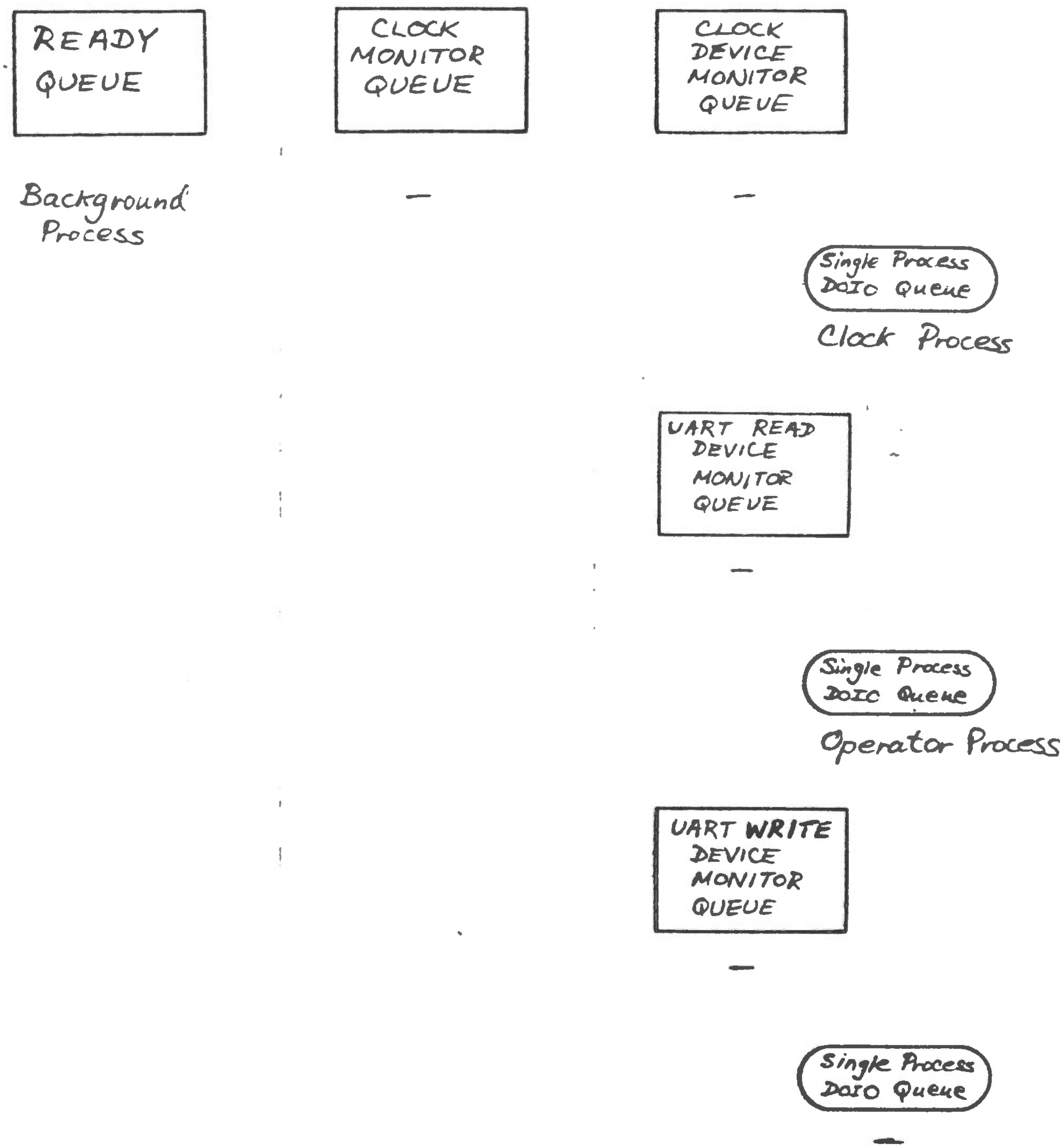


Figure 3. Minimum State of ERTC18 Operating System



Clock Interrupt Servicing

During the execution of a process in mCP the kernel will disable interrupts as a P-code instruction is fetched, and enable interrupts after a P-code instruction is completed. Hence there is a "window" through which interrupts are allowed to interrupt the running process. An interrupt could occur just after a P-code was fetched, and the kernel would then prevent the CPU from being interrupted until the P-code was interpreted and executed. This reduces the efficiency of interrupt handling. Although this problem cannot be directly avoided, it points to the major contributing factor in the efficiency of interrupt servicing: the speed of executing P-codes.

In the minimum state of the ERTC18 program, the running process is the background process. This process consists of a mCP HALT instruction which puts the CPU in an idle state and waits for interrupts. Thus the problem of an interrupt occurring while the kernel is executing a P-code is avoided. As soon as any interrupt occurs in the minimum state, control is transferred immediately to the kernel's interrupt handling routines. The time taken by the kernel to handle interrupts from this state will be the minimum.

The code contained in the Clock device monitor must be analyzed to determine the interrupt handling efficiency. This code is listed below,

```
BEGIN
        DOIO;
        OUT( #00,CLOCK_WORD );
END
```

The DOIO statement suspends the clock process to wait for an interrupt to occur. The operator process is run, and eventually suspended by a DOIO statement. The system is in the minimum state with both processes waiting for interrupts. When the clock interrupt occurs the Kernel's interrupt servicing follows these steps:

1: The source of Interrupt is determined.

2: The presently running process (the background process) is preempted.

3: The process waiting for the interrupt (i.e. clock process) is restarted.

The source of interrupt is determined by examining the event flag associated with the interrupting device (see Appendix: Clock Hardware). When the clock process is restarted, the next P-code fetched and executed will be the OUT instruction. This instruction will cause a pulse to be transmitted which will reset the clock hardware. The clock device monitor will then be exitted on execution of the END statement. The clock process then updates the software time variable by one (i.e. one second) and eventually the process cycles back to the minimum state upon execution of the DOIO instruction in the clock device monitor. Every second, on clock interrupts, this sequence is repeated.

The steps in handling the clock interrupt from the minimum state are outlined in figure 4. Labels identify the routines used in the kernel. From measurements made on the logic analyzer, it was found that it takes 238 microseconds for the kernel to recognize the highest priority Legal interrupt (i.e. the clock interrupt, Step 1).

At this point additional time is required to preempt the current running process and switch control to the process waiting for the interrupt. The time it takes between switching processes up to the point of fetching the OUT P-code in the device monitor following the DOIO instruction is 694 microseconds (Steps 2 and 3).

Execution of the OUT instruction to reset the clock takes 2621 microseconds. Up to this point the total time taken from the time the clock interrupt occurred to the time when the clock hardware is reset is 3552 microseconds.

Finally, the time taken for the kernel to exit from the device monitor and return to the clock process (where the time variable is to be updated) is 4493 microseconds.

FIGURE 4.(a) Kernel Steps in Handling Interrupts

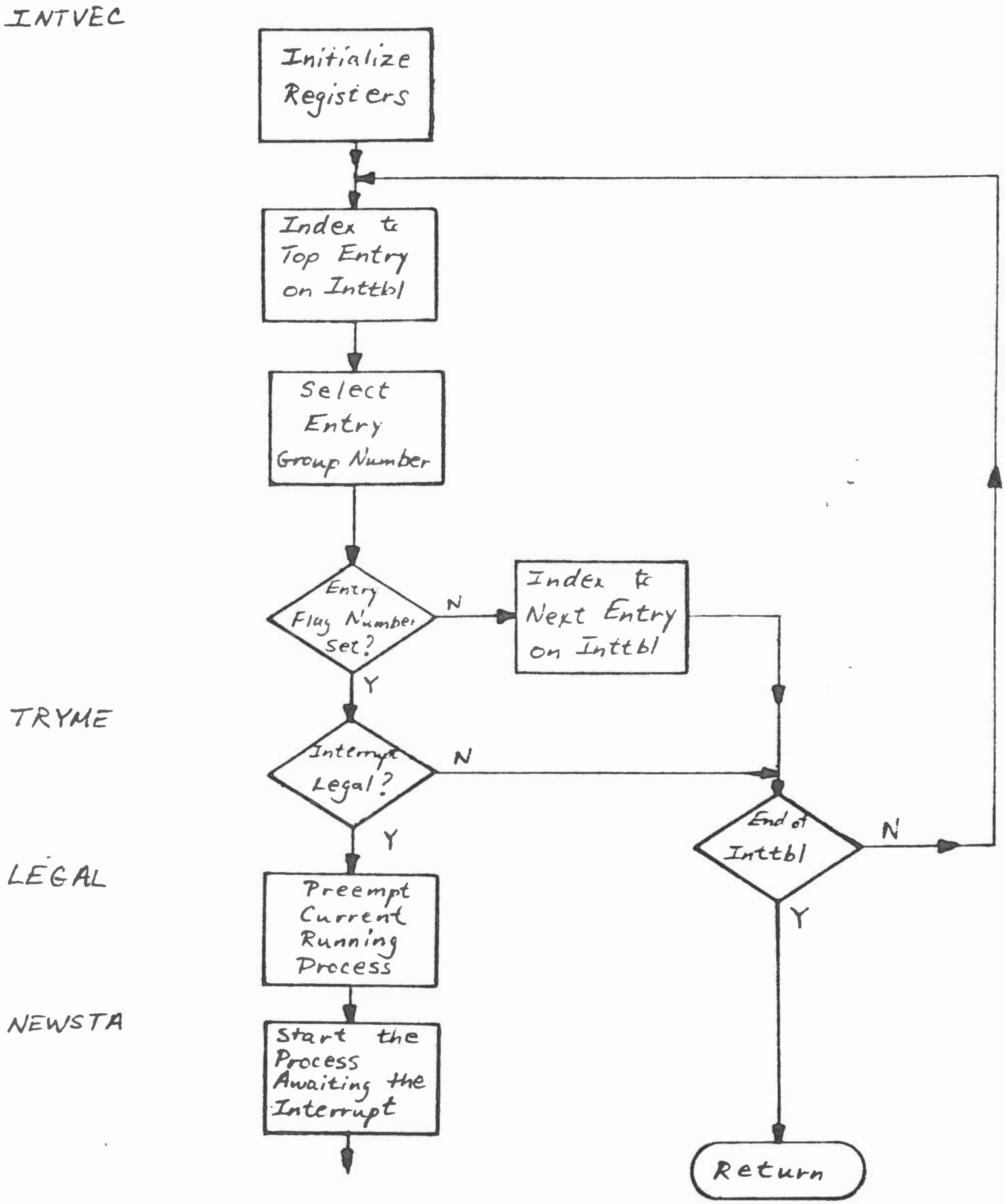
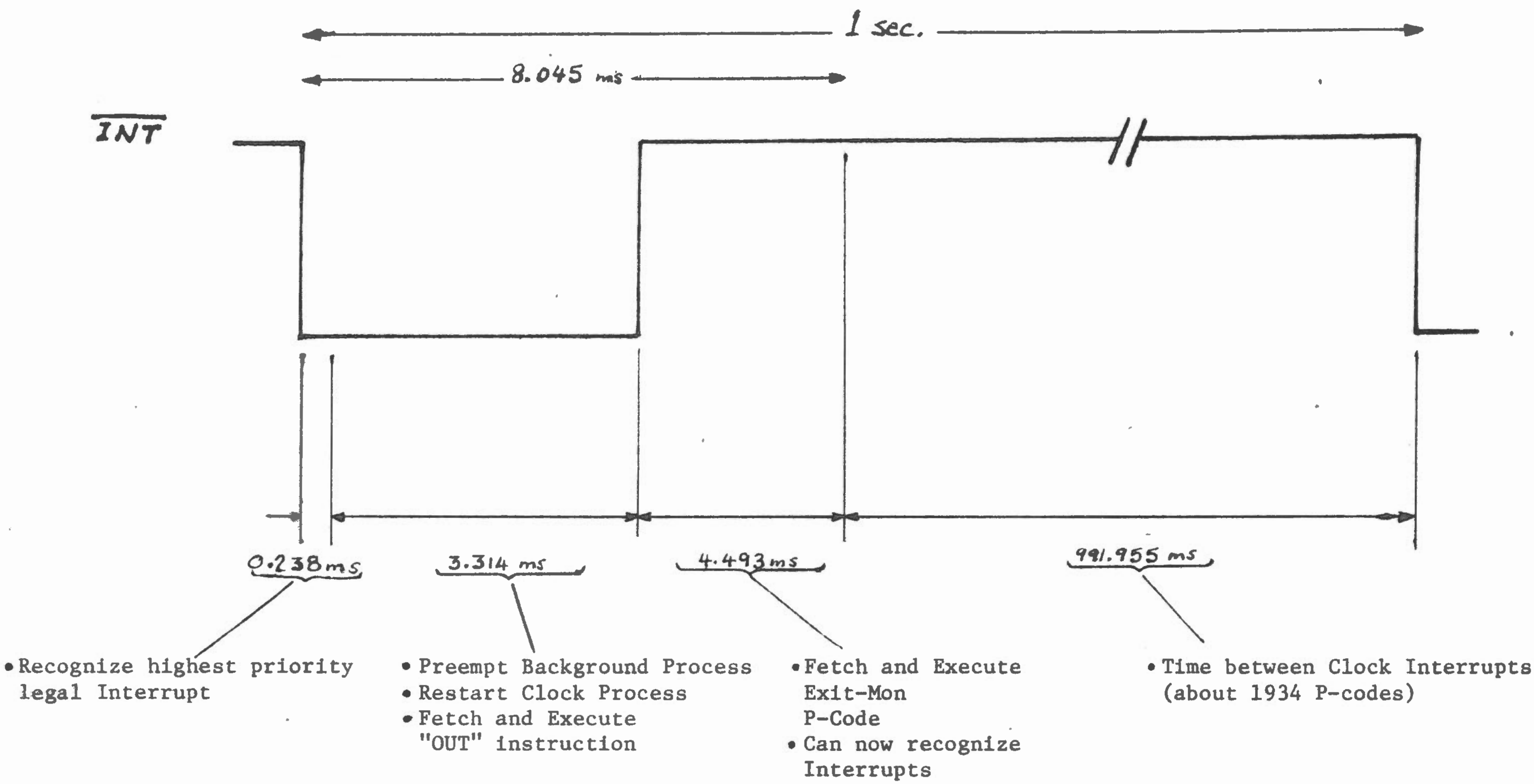


Figure 4.(b) Clock Interrupt Servicing



MCP Interrupt Handling Problems: Summary

The organization of the ERTC18 program is satisfactory for the intended purpose - providing a simple operating system allowing a real time clock to be set or viewed via a terminal. The system can handle the 1 Hz clock interrupt effectively. Since the interrupt period is sufficiently long there is enough time after the clock interrupt has been handled for the system to execute up to 1934 P-codes. However, if we wanted to run the clock at higher speeds problems would arise.

A clock frequency of 200 Hz for example could not be handled since it takes 8.045 milliseconds to recognize the interrupt, reset the clock, and exit from the clock device monitor. A clock frequency of 120 Hz could be handled, but there would not remain any time between interrupts to run other processes. At 5 Hz, though, there would be 192 milliseconds between interrupts, and 374 P-codes could be executed. This clock frequency would be the most appropriate for the present 1802 system.

Other problems will arise in multiple interrupting applications. MCP does not enable interrupts after I/O instructions. In the ERTC18 program although the clock interrupt is recognized and reset in 3.55 milliseconds, the kernel will only enable interrupts when the P-code following the OUT P-code has been completed. This means that the CPU is tied up for another 391 microseconds where it cannot service interrupts from other sources. Furthermore, the time taken to exit from the device monitor that resets the interrupting device and switch processes back to the main running process will reduce the available time to perform background processing (i.e. data manipulations, math operations, etc.).

For minimum-processing applications like the ERTC18 program where the frequency and number of interrupts are low, then the present mCP interrupt handling scheme is acceptable. A clock interrupt frequency of 5 Mhz. would allow enough time between interrupt servicing for 374 P-codes, for example. For applications requiring higher clock interrupt frequencies, with several sources of interrupts, and more processing, the mCP interrupt handling scheme will not work effectively and may have to be altered.

Hardware/Software Solutions

The simplest way to increase the clock interrupt handling speed is to remove the mCP OUT instruction that resets the clock hardware. Knowing that it takes 238 microseconds to recognize a legal interrupt, we could reduce the clock interrupt pulse to 500 microseconds. This would allow sufficient time for the kernel to identify the interrupt while ensuring that the interrupt line will be reset before the clock device monitor is exitted. By removing the OUT instruction a saving of 2621 microseconds can be achieved. The total time taken to handle this interrupt, from the moment the interrupt occurs to the time interrupts are enabled after exitting from the device monitor will be 5.424 ms compared to the previous 8.045 ms.

An alternate safer approach would require modifying the kernel. When the appropriate interrupt is identified, a "Set Q", "Reset Q" sequence could be performed within the kernel, thus sending a pulse via the CPU Q output. This output would reset the clock hardware before the next P-code (i.e. EXIT) was fetched. This would replace the OUT instruction, and thus the time savings would be similar to the previous approach.

These methods are simple, yet would not provide any real great improvement in general interrupt handling. At most an additional 5 P-codes could be executed between the clock interrupts. Any need for faster handling would require modifications to the basic mCP program design, necessitating kernel modifications and a departure from the DOIO "wait-for-interrupt" approach. Some ideas include interrupt vectoring or a dedicated clock interrupt service routine.

Interrupt vectoring would require some external hardware such as the RCA 1877 Interrupt Controller. Modifications to the mCP kernel and the DOIO section of the interpreter would have to be made.

The other possibility is in applications where a clock interrupt plays a major role in the scheduling of processes and control. Use of the RCA 1804 CPU with its internal counter-timer could provide accurate presettable internal clock interrupts. Modifications of the kernel would allow conditional Branch on Internal Interrupts to transfer control to certain processes or simply update a software counter.

Conclusion

The aim of this report was to evaluate the 1802 micro-Concurrent Pascal system. After running the Enertec ERTC18 real-time program it was found that the clock interrupt servicing was not very efficient. This report examined the mCP interrupt handling scheme particular to the 1802 system, using the ERTC18 program as an example.

An analysis of the steps taken by the mCP kernel in servicing interrupts showed that for the ERTC18 program, handling of the clock interrupt was from a minimum state. From this state, CPU servicing of the interrupt was the quickest possible. The time taken to recognize this interrupt, reset the clock interrupt device, and exit from the clock device monitor was 8.045 milliseconds.

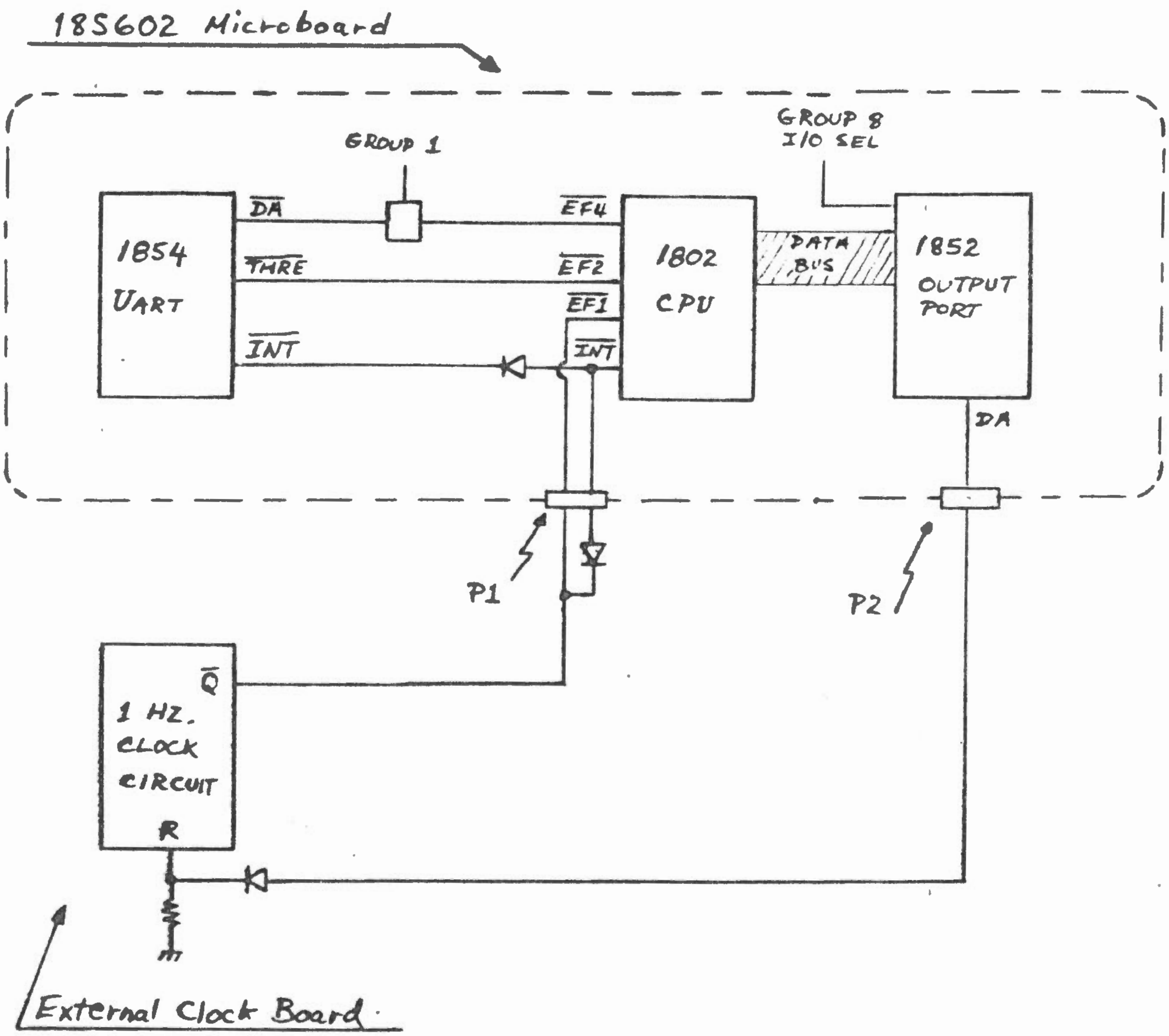
For applications such as the ERTC18 real-time clock, where the frequency and number of interrupting devices is low, the present mCP system is suitable. The time available between the 1 hertz clock interrupt was enough to allow for updating of the data and examining the UART device for operator input. A suitable typical application could be a data acquisition system that requires a terminal control and performs measurements on a 5 Hz clock interrupt. In this case there would be enough time for about 374 P-codes to be executed between clock interrupt servicing. If the data processing overhead is not too severe then the present system will perform well for this type of application.

Applications involving several, frequent interrupting devices or heavy mathematical computation and data processing between interrupts, then the present system will not work well. In such a system requiring quick response to an interrupt, certain hardware or software modifications will have to be made. A modification of the kernel could allow a faster acknowledgement of a clock interrupt and avoid the necessity of having a clock device monitor and its software overhead to control the clock. MCP's provision for accessing external assembly language routines can be used advantageously, or hardware controlling of interrupts could benefit certain higher speed applications.

In general, though, these modifications would require a modification of the mCP interrupt handling scheme. This would entail a departure from the structured, portable, high-level software to a more dedicated, less portable system. With an interpretive pseudo-coded language such as mCP, code execution speed will not be as fast as a similar assembly language program. The fundemental problem of interrupt handling for real-time applications lies not in the mCP interrupt servicing scheme, but in the speed of P-code execution. Attempts to modify the kernel can improve a system, but the overall performance of the system is limited by the speed of P-code execution. Modifications of the system should be incorporated within the basic mCP programming constructs (i.e. processes, monitors, and device monitors). In this way the fundamental reasons for mCP programming can be realized. A real-time program can be created using the clarity and flexibility of a structured high-level language with a programming environment that reduces the possibility of real-time errors.

Appendix 1

System Device Interrupt Connections



Interrupt Signal	Group Number	Associated Event Flag
1 Hz Clock	8	$\overline{EF1}$
$\overline{DA}$	1	$\overline{EF4}$
$\overline{THRE}$	None	$\overline{EF2}$

Appendix 2

```
{$LIST=LONG, DEBUG ON                                  $}
{##############################################
#                                             #
#   Micro Concurrent Pascal (mCP) Program     #
#      ENERTEC Real-Time for the 1802         #
#               4/13/81                       #
#                                             #
#          (c)  Copyright   1981              #
#                                             #
#           ENERTEC, Inc.                     #
#           19 Jenkins Avenue                 #
#           Lansdale, Pa.  19446              #
#              (215) 362-0966                 #
#                                             #
##############################################}

CONST CLOCK_GROUP = 8;
      CLOCK_FLAG = 1;
      CLOCK_SELECTOR = 6;
      UART_GROUP = 1;
      UART_RECV_FLAG = 4;
      UART_RECV_SELECTOR = 4;
      UART_XMIT_FLAG = 2;
      UART_XMIT_SELECTOR = 3;
      INTTBL_TERMINATOR = -1;

STRUC_CON INTTBL: ARRAY[1..10] OF -1..8 =
 [CLOCK_GROUP, CLOCK_FLAG, CLOCK_SELECTOR,
  UART_GROUP, UART_RECV_FLAG, UART_RECV_SELECTOR,
  UART_GROUP, UART_XMIT_FLAG, UART_XMIT_SELECTOR,
  INTTBL_TERMINATOR];

CONST  DATA_WORD = ADR(#0102);
       CTRL_WORD = ADR(#0103);
       CLOCK_WORD = ADR(#0802);
       LINELENGTH = 74;
       NUL='(:0:)';  BS='(:8:)';  LF='(:10:)';  CR='(:13:)';
       NUL2='(:0:)(:0:)';

TYPE  TIME = RECORD
               HOUR : 0..24;
               MIN, SEC : 0..60
             END;
      LINE_DISP =(PROMPT, NEWLINE, STAY);
      LINE = ARRAY[1..LINELENGTH] OF CHAR;
      INT = 0..127;

{###############
#  UART_WRITE  #
###############}

TYPE UART_WRITE = DEVICE_MON (SELECTOR: INT);

PROCEDURE ENTRY WRITE(MESSAGE: LINE; DISP: LINE_DISP);
  VAR I: INT;
      THROWAWAY: INTEGER;
  BEGIN
    I:=1;
    OUT(#BD, CTRL_WORD) { XMIT REQ., INT. EN., 8 DATA, 2 STOP, NO PARITY
    DOIO;
    WHILE (MESSAGE[I] <> NUL)  AND  (I<LINELENGTH) DO
    BEGIN
      OUT(ORD(MESSAGE[I]), DATA_WORD);  {SEND A CHARACTER}
```

```
      INC(I);
    END;
    IF (DISP=PROMPT) OR (DISP=NEWLINE) THEN
      BEGIN OUT(ORD(CR), DATA_WORD); DOIO;
            OUT(ORD(LF), DATA_WORD); DOIO;
      END;
    IF DISP=PROMPT THEN
    BEGIN OUT(ORD('>'), DATA_WORD); DOIO; END;
     OUT(#3D, CTRL_WORD); {TRANSMIT INHIBIT OTHERWISE SAME AS ABOVE}
     THROWAWAY := INN(CTRL_WORD);
  END;


PROCEDURE ENTRY ECHO(CHRS:CHAR);
  BEGIN
    OUT(#BD, CTRL_WORD);
    OUT(ORD(CHRS), DATA_WORD);
    OUT(#3D, CTRL_WORD);
  END;

BEGIN
  OUT(#3D, CTRL_WORD);
END;

{##############
#  UART_READ  #
##############}

TYPE UART_READ = DEVICE_MON (TERM_OUT: UART_WRITE;
                             SELECTOR: INT);

  VAR THROWAWAY: INT;

PROCEDURE ENTRY READ( VAR MESSAGE: LINE;  VAR LENGTH: INT);
  VAR LASTCHAR: CHAR;
  BEGIN
    LENGTH:=0;
    REPEAT
      IF LENGTH<LINELENGTH THEN INC(LENGTH);
      DOIO;
      LASTCHAR := CHR(INN(DATA_WORD));   {GET A CHARACTER}
      IF LASTCHAR=BS THEN
        BEGIN
        IF LENGTH >= 2 THEN DEC(LENGTH);
        DEC(LENGTH);
        END
      ELSE MESSAGE[LENGTH] := LASTCHAR;
      IF LASTCHAR <> CR THEN
          BEGIN
          TERM_OUT.ECHO(LASTCHAR);
          END;
    UNTIL (LASTCHAR = CR);
    TERM_OUT.WRITE(NUL2, NEWLINE);
  END;

BEGIN
  THROWAWAY:=INN(DATA_WORD);
  THROWAWAY:=INN(DATA_WORD);
END;

{###################
#  CLOCK_MONITOR  #
###################}

TYPE CLOCK_MONITOR = MONITOR (TERM_OUT: UART_WRITE);
```

```
PROCEDURE ENTRY TICK;
  BEGIN
  WITH CLOCKTIME DO
    BEGIN
    INC(SEC);
    IF SEC=60 THEN
      BEGIN
        SEC:=0;  INC(MIN);
        IF MIN=60 THEN
          BEGIN
            MIN:=0;  INC(HOUR);
            IF HOUR=24 THEN
              BEGIN
                HOUR:=0;
                TERM_OUT.WRITE('IT IS A NEW DAY(:0:)', NEWLINE);
              END;
          END
      END
    END;
  END;

PROCEDURE ENTRY SETTIME(T: TIME);
  BEGIN
    CLOCKTIME := T;
  END;

PROCEDURE ENTRY GETTIME(VAR T: TIME);
  BEGIN
    T := CLOCKTIME;
  END;

BEGIN
  WITH CLOCKTIME DO BEGIN HOUR:=0; MIN:=0; SEC:=0; END;
END;

(****************
*  CLOCKPULSE  *
****************)

TYPE CLOCKPULSE = DEVICE_MON (SELECTOR: INT);

PROCEDURE ENTRY TICK;
  BEGIN
    DOIO;
    OUT(#00, CLOCK_WORD);
  END;

BEGIN
  ( SET UP TO START THE CLOCK )
END;

(******************
*  CLOCKPROCESS  *
******************)

TYPE CLOCKPROCESS = PROCESS (PULSE: CLOCKPULSE;
                             CLOCK: CLOCK_MONITOR);

BEGIN
  CYCLE CLOCK.TICK; PULSE.TICK; END;
END;

(*********************
*  OPERATORPROCESS  *
```

```
*********************}

TYPE OPERATORPROCESS = PROCESS ( CLOCK: CLOCK_MONITOR;
                                 TERM_IN: UART_READ;
                                 TERM_OUT: UART_WRITE);

  VAR BUFFER: LINE;
      LENGTH: INT;
      T: TIME;

  FUNCTION NUMBER(CHARACTER: CHAR): INT;
    BEGIN  NUMBER:=ORD(CHARACTER) - ORD('0');  END;

  FUNCTION ASCII(NUMBER: INT): CHAR;
    BEGIN  ASCII:=CHR(NUMBER + ORD('0'));  END;

BEGIN
  CYCLE
    TERM_OUT.WRITE(NUL2, PROMPT);
    TERM_IN.READ(BUFFER, LENGTH);
    CASE BUFFER[1] OF
'S' (SET TIME) :
      BEGIN
        WITH T DO
          BEGIN
          HOUR:=23;
          MIN :=50;
          SEC :=10;
          END;
        CLOCK.SETTIME(T);
       END;
'G' (GET TIME) :
      BEGIN
        CLOCK.GETTIME(T);;
        WITH T DO
          BEGIN
          BUFFER[1]:=ASCII(HOUR DIV 10); BUFFER[2]:=ASCII(HOUR MOD 10);
          BUFFER[3]:=':';
          BUFFER[4]:=ASCII(MIN DIV 10);  BUFFER[5]:=ASCII(MIN MOD 10);
          BUFFER[6]:=':';
          BUFFER[7]:=ASCII(SEC DIV 10);  BUFFER[8]:=ASCII(SEC MOD 10);
          BUFFER[9]:=NUL;
          END;
        TERM_OUT.WRITE(BUFFER, NEWLINE);
      END;
():   BEGIN END;
    END;
  END;
END;

(*********************
*  INITIAL PROCESS  *
*********************)

VAR CLOCK: CLOCK_MONITOR;                    (MONITOR)
    TICKTOCK: CLOCKPROCESS;                  (PROCESS TO CYCLE CLOCK)
    PULSE: CLOCKPULSE;                       (DEVICE MONITOR FOR TIMER)
    OPERATOR: OPERATORPROCESS;               (PROCESS)
    TERM_IN: UART_READ;                      (DEVICE MON FOR TERMINAL)
    TERM_OUT: UART_WRITE;                    (DEVICE MON FOR TERMINAL)

BEGIN
  INIT TERM_OUT(UART_XMIT_SELECTOR);
  INIT CLOCK(TERM_OUT);
  INIT PULSE(CLOCK_SELECTOR);
  INIT TERM_IN(TERM_OUT, UART_RECV_SELECTOR);
```

```
T TICKTOCK(PULSE, CLOCK);
T OPERATOR(CLOCK, TERM_IN, TERM_OUT);
```

RO CONCURRENT PASCAL   VERSION 2.4
 COPYRIGHT   ENERTEC, INC.   LANSDALE, PA   1979

PROGRAM LISTING

RLEN .H

ELENGTH: 643
STLENGTH: 44

```
( 0004)
( 0004)        ($LIST=LONG, DEBUG ON                                    $)
( 0004)        (*****************************************************
( 0004)        *                                                        *
( 0004)        *    Micro Concurrent Pascal (mCP) Program               *
( 0004)        *       ENERTEC Real-Time for the 1802                   *
( 0004)        *                  4/13/81                               *
( 0004)        *                                                        *
( 0004)        *             (c)  Copyright   1981                      *
( 0004)        *                                                        *
( 0004)        *              ENERTEC, Inc.                             *
( 0004)        *              19 Jenkins Avenue                         *
( 0004)        *              Lansdale, Pa.  19446                      *
( 0004)        *                 (215) 362-0966                         *
( 0004)        *                                                        *
( 0004)        *****************************************************)
( 0004)
' 0004)  CONST CLOCK_GROUP = 8;
  0004)        CLOCK_FLAG = 1;
( 0004)        CLOCK_SELECTOR = 6;
( 0004)        UART_GROUP = 1;
( 0004)        UART_RECV_FLAG = 4;
( 0004)        UART_RECV_SELECTOR = 4;
( 0004)        UART_XMIT_FLAG = 2;
( 0004)        UART_XMIT_SELECTOR = 3;
( 0004)        INTTBL_TERMINATOR = -1;
( 0004)
( 0004)  STRUC_CON INTTBL: ARRAY[1..10] OF -1..8 =
( 0004)   [CLOCK_GROUP, CLOCK_FLAG, CLOCK_SELECTOR,
( 0004)    UART_GROUP, UART_RECV_FLAG, UART_RECV_SELECTOR,
( 0004)    UART_GROUP, UART_XMIT_FLAG, UART_XMIT_SELECTOR,
( 0004)    INTTBL_TERMINATOR];
( 0004)
( 0004)  CONST  DATA_WORD = ADR(#0102);
( 0004)         CTRL_WORD = ADR(#0103);
( 0004)         CLOCK_WORD = ADR(#0802);
( 0004)         LINELENGTH = 74;
( 0004)         NUL='(:0:)';  BS='(:8:)';  LF='(:10:)';  CR='(:13:)';
( 0004)         NUL2='(:0:)(:0:)';
( 0004)
( 0004)  TYPE  TIME = RECORD
( 0004)                HOUR : 0..24;
' 0004)                MIN, SEC : 0..60
  0004)              END;
( 0004)        LINE_DISP =(PROMPT, NEWLINE, STAY);
( 0004)        LINE = ARRAY[1..LINELENGTH] OF CHAR;
( 0004)        INT = 0..127;
( 0004)
( 0004)  (***************
( 0004)  * UART_WRITE  *
( 0004)  ***************)
```

```
    END;
 432E  ,  .           JUMP              -210
  END;
 D6                   END_PROCESS

   (*********************
   *   NITIAL PROCESS  *
   *********************)

   VAR CLOCK: CLOCK_MONITOR;                        (MONITOR)
       TICKTOCK: CLOCKPROCESS;                      (PROCESS TO CYCLE CLOCK)
       PULSE: CLOCKPULSE;                           (DEVICE MONITOR FOR TIMER)
       OPERATOR: OPERATORPROCESS;                   (PROCESS)
       TERM_IN: UART_READ;                          (DEVICE MON FOR TERMINAL)
       TERM_OUT: UART_WRITE;                        (DEVICE MON FOR TERMINAL)

)  BEGIN
)    INIT TERM_OUT(UART_XMIT_SELECTOR);
) 8AF4                PS_AD_G_1        -12
) 03                  CONSTANT    3
) C600000231FE        INIT_MON           0       2    -463
)    INIT CLOCK(TERM_OUT);
) 8AFE                PS_AD_G_1         -2
) 82F4                PS_VW_G_1        -12
) C6030002DAFE        INIT_MON           3       2    -294
)    INIT PULSE(CLOCK_SELECTOR);
) 8AFA                PS_AD_G_1         -6
) 06                  CONSTANT    6
) C6000002EEFE        INIT_MON           0       2    -274
)    INIT TERM_IN(TERM_OUT, UART_RECV_SELECTOR);
) 8AF6                PS_AD_G_1        -10
) 87 `                PS_VW_G_1        -12
) 04                  CONSTANT    4
) C601000461FE        INIT_MON           1       4    -415
)    INIT TICKTOCK(PULSE, CLOCK);
) 8AFC                PS_AD_G_1         -4
) 82FA                PS_VW_G_1         -6
) 82FE                PS_VW_G_1         -2
) CB4100000004D9FE INIT_PROCESS         65       0        4    -295
)    INIT OPERATOR(CLOCK, TERM_IN, TERM_OUT);
) 8AF8                PS_AD_G_1         -8
) 82FE                PS_VW_G_1         -2
) 82F6                PS_VW_G_1        -10
) 82F4                PS_VW_G_1        -12
) CB44004E0006E8FE INIT_PROCESS         68      78        6    -280
)  END.
) D6                  END_PROCESS
) CB2C000C0000B3FF INIT_PROCESS         44      12        0     -77
) FC                  HALT

LE:
      00 00 00 00 00 00 00 00 00 00 00 00 00 [    .............]
0 00 08 01 06 01 04 04 01 02 03 FF 00 00 49 [...............I]
0 49 53 20 41 20 4E 45 57 20 44 41 59 00    [T IS A NEW DAY. ]
OK
```

```
   ( 01C1) B0                  FUNC_WORD
   ( 01C2) 96FE01              PS_VW_L_1_FD     -2       1
   ( 01C5) 91                  PS_INDB
   ( 01C6) 0A                  CONSTANT  10
   ( 01C7) EA                  DIV_WORD
   ( 01C8) CF8CFF              CALL_ROUTINE    -116
   ( 01CB) A3                  COPY_BYTE
     01CC) 8AB6                PS_AD_G_1        -74
   ( 01CE) 05                  CONSTANT   5
   ( 01CF) A7                  INDEX_11
   ( 01D0) B0                  FUNC_WORD
   ( 01D1) 96FE01              PS_VW_L_1_FD     -2       1
   ( 01D4) 91                  PS_INDB
   ( 01D5) 0A                  CONSTANT  10
   ( 01D6) EC                  MOD
   ( 01D7) CF7DFF              CALL_ROUTINE    -131
   ( 01DA) A3                  COPY_BYTE
35: ( 01DB)               BUFFER[6]:=':';
   ( 01DB) 8AB6                PS_AD_G_1        -74
   ( 01DD) 06                  CONSTANT   6
   ( 01DE) A7                  INDEX_11
   ( 01DF) 9A3A                PS_CONST_1        58
   ( 01E1) A3                  COPY_BYTE
36: ( 01E2)               BUFFER[7]:=ASCII(SEC DIV 10);  BUFFER[8]:=ASCII(SEC MOD 10
   ( 01E2) 8AB6                PS_AD_G_1        -74
   ( 01E4) 07                  CONSTANT   7
   ( 01E5) A7                  INDEX_11
   ( 01E6) B0                  FUNC_WORD
   ( 01E7) 96FE02              PS_VW_L_1_FD     -2       2
   ( 01EA) 91                  PS_INDB
   ( 01EB) 0A                  CONSTANT  10
   ( 01EC) EA                  DIV_WORD
     01ED) CF67FF              CALL_ROUTINE    -153
   ( 01F0) A3                  COPY_BYTE
   ( 01F1) 8AB6                PS_AD_G_1        -74
   ( 01F3) 08                  CONSTANT   8
   ( 01F4) A7                  INDEX_11
   ( 01F5) B0                  FUNC_WORD
   ( 01F6) 96FE02              PS_VW_L_1_FD     -2       2
   ( 01F9) 91                  PS_INDB
   ( 01FA) 0A                  CONSTANT  10
   ( 01FB) EC                  MOD
   ( 01FC) CF58FF              CALL_ROUTINE    -168
   ( 01FF) A3                  COPY_BYTE
37: ( 0200)               BUFFER[9]:=NUL;
   ( 0200) 8AB6                PS_AD_G_1        -74
   ( 0202) 09                  CONSTANT   9
   ( 0203) A7                  INDEX_11
   ( 0204) A0                  COPY_ZERO
38: ( 0205)               END;
   ( 0205) FE02                POP                2
39: ( 0207)             TERM_OUT.WRITE(BUFFER, NEWLINE);
   ( 0207) 51                  FPS_VW_G+06
   ( 0208) 8AB6                PS_AD_G_1        -74
   ( 020A) 01                  CONSTANT   1
   ( 020B) CFF7FD              CALL_ROUTINE    -521
40: ( 020E)         END;
     020E) 4422                JUMP              34
41: ( 0210)  <>:    BEGIN END;
   ( 0210) 4420                JUMP              32
42: ( 0212)        END;
   ( 0212) CE470C              CASE_JUMP         71      12
   ( 0215) FAFF7AFFF6FFF4FF                      -6    -134     -10     -12
   ( 021D) F2FFF0FFEEFFECFF                     -14     -16     -18     -20
   ( 0225) EAFFE8FFE6FFE4FF                     -22     -24     -26     -28
```

```
    ( 0175) 449C              JUMP             156
?:  ( 0177)  'S' (SET TIME) :
3:  ( 0177)         BEGIN
9:  ( 0177)           WITH T DO
    ( 0177) 8AB2              PS_AD_G_1        -78
0:  ( 0179)             BEGIN
1:  (  179)             HOUR:=23;
    ( 0179) 86FE              PS_VW_L_1         -2
    ( 017B) 17                CONSTANT  23
    ( 017C) A3                COPY_BYTE
2:  ( 017D)             MIN :=50;
    ( 017D) 96FE01            PS_VW_L_1_FD      -2        1
    ( 0180) 9A32              PS_CONST_1        50
    ( 0182) A3                COPY_BYTE
3:  ( 0183)             SEC :=10;
    ( 0183) 96FE02            PS_VW_L_1_FD      -2        2
    ( 0186) 0A                CONSTANT  10
    ( 0187) A3                COPY_BYTE
4:  ( 0188)             END;
    ( 0188) FE02              POP                2
5:  ( 018A)           CLOCK.SETTIME(T);
    ( 018A) 53                FPS_VW_G+10
    ( 018B) 8AB2              PS_AD_G_1        -78
    ( 018D) CF7DFF            CALL_ROUTINE    -131
26: ( 0190)          END;
    ( 0190) 44A0              JUMP             160
27: ( 0192)  'G' (GET TIME) :
28: ( 0192)         BEGIN
29: ( 0192)           CLOCK.GETTIME(T);;
    ( 0192) 53                FPS_VW_G+10
    ( 0193) 8AB2              PS_AD_G_1        -78
      0195) CF7FFF            CALL_ROUTINE    -129
30:   0198)           WITH T DO
    ( 0198) 8AB2              PS_AD_G_1        -78
31: ( 019A)             BEGIN
32: ( 019A)             BUFFER[1]:=ASCII(HOUR DIV 10); BUFFER[2]:=ASCII(HOUR MOD 1
    ( 019A) 8AB6              PS_AD_G_1        -74
    ( 019C) 01                CONSTANT   1
    ( 019D) A7                INDEX_11
    ( 019E) B0                FUNC_WORD
    ( 019F) 86FE              PS_VW_L_1         -2
    ( 01A1) 91                PS_INDB
    ( 01A2) 0A                CONSTANT  10
    ( 01A3) EA                DIV_WORD
    ( 01A4) CFB0FF            CALL_ROUTINE     -80
    ( 01A7) A3                COPY_BYTE
    ( 01A8) 8AB6              PS_AD_G_1        -74
    ( 01AA) 02                CONSTANT   2
    ( 01AB) A7                INDEX_11
    ( 01AC) B0                FUNC_WORD
    ( 01AD) 86FE              PS_VW_L_1         -2
    ( 01AF) 91                PS_INDB
    ( 01B0) 0A                CONSTANT  10
    ( 01B1) EC                MOD
    ( 01B2) CFA2FF            CALL_ROUTINE     -94
    ( 01B5) A3                COPY_BYTE
33:   01B6)             BUFFER[3]:=':';
      01B6) 8AB6              PS_AD_G_1        -74
    ( 01B8) 03                CONSTANT   3
    ( 01B9) A7                INDEX_11
    ( 01BA) 9A3A              PS_CONST_1        58
    ( 01BC) A3                COPY_BYTE
34: ( 01BD)             BUFFER[4]:=ASCII(MIN DIV 10);  BUFFER[5]:=ASCII(MIN MOD 1
    ( 01BD) 8AB6              PS_AD_G_1        -74
                              CONSTANT   4
```

```
( 0141)
( 0141)  (******************
( 0141)  *  CLOCKPROCESS  *
( 0141)  ******************)
( 0141)
( 0141)  TYPE CLOCKPROCESS = PROCESS (PULSE: CLOCKPULSE;
( 0141)                                 CLOCK: CLOCK_MONITOR);
 )141)
( 0141)  BEGIN
( 0141)    CYCLE CLOCK.TICK; PULSE.TICK; END;
( 0141) 51                FPS_VW_G+06
( 0142) CF89FF            CALL_ROUTINE   -119
( 0145) 52                FPS_VW_G+08
( 0146) CFEBFF            CALL_ROUTINE    -21
( 0149) 43F7              JUMP             -9
( 014B)  END;
( 014B) D6                END_PROCESS
( 014C)
( 014C)  (*********************
( 014C)  *  OPERATORPROCESS  *
( 014C)  *********************)
( 014C)
( 014C)  TYPE OPERATORPROCESS = PROCESS ( CLOCK: CLOCK_MONITOR;
( 014C)                                   TERM_IN: UART_READ;
( 014C)                                   TERM_OUT: UART_WRITE);
( 014C)    VAR BUFFER: LINE;
( 014C)        LENGTH: INT;
( 014C)        T: TIME;
( 014C)
( 014C)    FUNCTION NUMBER(CHARACTER: CHAR): INT;
( 014C)      BEGIN  NUMBER:=ORD(CHARACTER) - ORD('0');  END;
( 014C) C10A00            ENTER              10       0
 )14F) 72                FPS_AD_L+12
( 0150) 31                FPS_VW_L+10
( 0151) 9A30              PS_CONST_1         48
( 0153) E6                SUB_WORD
( 0154) A2                COPY_WORD
( 0155) D1                EXIT
( 0156)
( 0156)    FUNCTION ASCII(NUMBER: INT): CHAR;
( 0156)      BEGIN  ASCII:=CHR(NUMBER + ORD('0'));  END;
( 0156) C10A00            ENTER              10       0
( 0159) 72                FPS_AD_L+12
( 015A) 31                FPS_VW_L+10
( 015B) 9A30              PS_CONST_1         48
( 015D) E4                ADD_WORD
( 015E) A2                COPY_WORD
( 015F) D1                EXIT
( 0160)
( 0160)  BEGIN
( 0160)    CYCLE
( 0160)      TERM_OUT.WRITE(NUL2, PROMPT);
( 0160) 51                FPS_VW_G+06
( 0161) 891A00            PS_AD_CN_2         26
( 0164) 00                CONSTANT    0
( 0165) CF9DFE            CALL_ROUTINE   -355
( 0168)      TERM_IN.READ(BUFFER, LENGTH);
  0168) 52                FPS_VW_G+08
( 0169) 8AB6              PS_AD_G_1         -74
( 016B) 8AB5              PS_AD_G_1         -75
( 016D) CF08FF            CALL_ROUTINE   -248
( 0170)      CASE BUFFER[1] OF
( 0170) 8AB6              PS_AD_G_1         -74
( 0172) 01                CONSTANT    1
( 0173) A7                INDEX 11
```

```
( 0106) CFFCFE              CALL_ROUTINE    -260
(.0109)                  END;
(.0109)              END
( 0109)          END
( 0109)        END;
( 0109) FE02                POP               2
( 010B)      END;
  10B) D3                   EXIT_MON
( 010C)
( 010C)   PROCEDURE ENTRY SETTIME(T: TIME);
( 010C)      BEGIN
( 010C) C30C00              ENTER_MON        12       0
( 010F)        CLOCKTIME := T;
( 010F) 8AFD                PS_AD_G_1        -3
( 0111) 31                  FPS_VW_L+10
( 0112) A40300              COPY_STRUC        3
( 0115)      END;
( 0115) D3                  EXIT_MON
( 0116)
( 0116)   PROCEDURE ENTRY GETTIME(VAR T: TIME);
( 0116)      BEGIN
( 0116) C30C00              ENTER_MON        12       0
( 0119)         T := CLOCKTIME;
( 0119) 31                  FPS_VW_L+10
( 011A) 8AFD                PS_AD_G_1        -3
( 011C) A40300              COPY_STRUC        3
( 011F)      END;
( 011F) D3                  EXIT_MON
( 0120)
( 0120)   BEGIN
( 0120) C20A00              BEGIN_MON        10       0
( 0123)      WITH CLOCKTIME DO BEGIN HOUR:=0; MIN:=0; SEC:=0; END;
 0123) 8AFD                 PS_AD_G_1        -3
( 0125) 86FE                PS_VW_L_1        -2
( 0127) A0                  COPY_ZERO
( 0128) 96FE01              PS_VW_L_1_FD     -2       1
( 012B) A0                  COPY_ZERO
( 012C) 96FE02              PS_VW_L_1_FD     -2       2
( 012F) A0                  COPY_ZERO
( 0130) FE02                POP               2
( 0132)   END;
( 0132) D2                  END_MON
( 0133)
( 0133)   (***************
( 0133)   *  CLOCKPULSE  *
( 0133)   ***************)
( 0133)
( 0133)   TYPE CLOCKPULSE = DEVICE_MON (SELECTOR: INT);
( 0133)
( 0133)   PROCEDURE ENTRY TICK;
( 0133)      BEGIN
( 0133) C30A00              ENTER_MON        10       0
( 0136)        DOIO;
( 0136) DA                  DOIO
( 0137)        OUT(#00, CLOCK_WORD);
( 0137) 00                  CONSTANT    0
( 0138) 940208              PS_CONST_2     2050
 013B) B7                   OUT
( 013C)      END;
( 013C) D3                  EXIT_MON
( 013D)
( 013D)   BEGIN
( 013D) C20A00              BEGIN_MON        10       0
( 0140)      { SET UP TO START THE CLOCK }
( 0140)   END;
```

```
( 00BE) 8AFF              PS_AD_G_1       -1
( 00C0) 940201            PS_CONST_2      258
( 00C3) B6                INN
( 00C4) A3                COPY_BYTE
( 00C5)    THROWAWAY:=INN(DATA_WORD);
( 00C5) 8AFF              PS_AD_G_1       -1
( 00C7) 940201            PS_CONST_2      258
( 00CA) B6                INN
( 00CB) A3                COPY_BYTE
( 00CC)  END;
( 00CC) D2                END_MON
( 00CD)
( 00CD)  (******************
( 00CD)  *  CLOCK_MONITOR  *
( 00CD)  ******************)
( 00CD)
( 00CD)  TYPE CLOCK_MONITOR = MONITOR (TERM_OUT: UART_WRITE);
( 00CD)    VAR CLOCKTIME: TIME;
( 00CD)
( 00CD)  PROCEDURE ENTRY TICK;
( 00CD)    BEGIN
( 00CD) C30A00            ENTER_MON       10       0
( 00D0)    WITH CLOCKTIME DO
( 00D0) 8AFD              PS_AD_G_1       -3
( 00D2)      BEGIN
( 00D2)      INC(SEC);
( 00D2) 96FE02            PS_VW_L_1_FD    -2       2
( 00D5) 9F                INC_BYTE
( 00D6)      IF SEC=60 THEN
( 00D6) 96FE02            PS_VW_L_1_FD    -2       2
( 00D9) 91                PS_INDB
( 00DA) 9A3C              PS_CONST_1      60
( 00DC) EE                EQ_WORD
( 00DD) 4C2B              FALSEJUMP       43
( 00DF)        BEGIN
( 00DF)          SEC:=0;  INC(MIN);
( 00DF) 96FE02            PS_VW_L_1_FD    -2       2
( 00E2) A0                COPY_ZERO
( 00E3) 96FE01            PS_VW_L_1_FD    -2       1
( 00E6) 9F                INC_BYTE
( 00E7)          IF MIN=60 THEN
( 00E7) 96FE01            PS_VW_L_1_FD    -2       1
( 00EA) 91                PS_INDB
( 00EB) 9A3C              PS_CONST_1      60
( 00ED) EE                EQ_WORD
( 00EE) 4C1A              FALSEJUMP       26
( 00F0)            BEGIN
( 00F0)              MIN:=0;  INC(HOUR);
( 00F0) 96FE01            PS_VW_L_1_FD    -2       1
( 00F3) A0                COPY_ZERO
( 00F4) 86FE              PS_VW_L_1       -2
( 00F6) 9F                INC_BYTE
( 00F7)              IF HOUR=24 THEN
( 00F7) 86FE              PS_VW_L_1       -2
( 00F9) 91                PS_INDB
( 00FA) 18                CONSTANT  24
( 00FB) EE                EQ_WORD
( 00FC) 4C0C              FALSEJUMP       12
( 00FE)                BEGIN
( 00FE)                  HOUR:=0;
( 00FE) 86FE              PS_VW_L_1       -2
( 0100) A0                COPY_ZERO
( 0101)                   TERM_OUT.WRITE('IT IS A NEW DAY(:0:)', NEWLINE);
( 0101) 51                FPS_VW_G+06
( 0102) B91C00            PS_AD_CN_2      28
```

```
( 007D) 91                PS_INDB
( 007E) 9A4A              PS_CONST_1        74
( 0080) F0                LS_WORD
( 0081) 4C03              FALSEJUMP          3
( 0083) 31                FPS_VW_L+10
( 0084) 9F                INC_BYTE
( 0085)             DOIO;
( 0085) DA                DOIO
( 0086)             LASTCHAR := CHR(INN(DATA_WORD));   (GET A CHARACTER)
( 0086) 6F                FPS_AD_L-01
( 0087) 940201            PS_CONST_2        258
( 008A) B6                INN
( 008B) A3                COPY_BYTE
( 008C)             IF LASTCHAR=BS THEN
( 008C) 2F                FPS_VB_L-01
( 008D) 08                CONSTANT    8
( 008E) EE                EQ_WORD
( 008F) 4C0D              FALSEJUMP         13
( 0091)               BEGIN
( 0091)               IF LENGTH >= 2 THEN DEC(LENGTH);
( 0091) 31                FPS_VW_L+10
( 0092) 91                PS_INDB
( 0093) 02                CONSTANT    2
( 0094) F4                NL_WORD
( 0095) 4C03              FALSEJUMP          3
( 0097) 31                FPS_VW_L+10
( 0098) 9D                DEC_BYTE
( 0099)               DEC(LENGTH);
( 0099) 31                FPS_VW_L+10
( 009A) 9D                DEC_BYTE
( 009B)               END
( 009B)             ELSE MESSAGE[LENGTH] := LASTCHAR;
( 009B) 4407              JUMP               7
( 009D) 32                FPS_VW_L+12
( 009E) 31                FPS_VW_L+10
( 009F) 91                PS_INDB
( 00A0) A7                INDEX_11
( 00A1) 2F                FPS_VB_L-01
( 00A2) A3                COPY_BYTE
( 00A3)             IF LASTCHAR <> CR THEN
( 00A3) 2F                FPS_VB_L-01
( 00A4) 0D                CONSTANT   13
( 00A5) F6                NE_WORD
( 00A6) 4C06              FALSEJUMP          6
( 00A8)                 BEGIN
( 00A8)                 TERM_OUT.ECHO(LASTCHAR);
( 00A8) 52                FPS_VW_G+08
( 00A9) 2F                FPS_VB_L-01
( 00AA) CFACFF            CALL_ROUTINE     -84
( 00AD)                 END;
( 00AD)           UNTIL (LASTCHAR = CR);
( 00AD) 2F                FPS_VB_L-01
( 00AE) 0D                CONSTANT   13
( 00AF) EE                EQ_WORD
( 00B0) 4BCB              FALSEJUMP        -53
( 00B2)           TERM_OUT.WRITE(NUL2, NEWLINE);
( 00B2) 52                FPS_VW_G+08
( 00B3) 891A00            PS_AD_CN_2        26
( 00B6) 01                CONSTANT    1
( 00B7) CF4BFF            CALL_ROUTINE    -181
( 00BA)         END;
( 00BA) D3                EXIT_MON
( 00BB)
( 00BB)      BEGIN
( 00BB) C20A00            BEGIN_MON         10         0
```

```
( 0041) EE                  EQ_WORD
( 0042) 4C08                FALSEJUMP          8
( 0044)        BEGIN OUT(ORD('>'), DATA_WORD); DOIO; END;
( 0044) 9A3E                PS_CONST_1        62
( 0046) 940201              PS_CONST_2       258
( 0049) B7                  OUT
( 004A) DA                  DOIO
( 004B)          OUT(#3D, CTRL_WORD); (TRANSMIT INHIBIT OTHERWISE SAME AS ABOVE)
( 004B) 9A3D                PS_CONST_1        61
( 004D) 940301              PS_CONST_2       259
( 0050) B7                  OUT
( 0051)          THROWAWAY := INN(CTRL_WORD);
( 0051) 6D                  FPS_AD_L-03
( 0052) 940301              PS_CONST_2       259
( 0055) B6                  INN
( 0056) A2                  COPY_WORD
( 0057)       END;
( 0057) D3                  EXIT_MON
( 0058)
( 0058)
( 0058)     PROCEDURE ENTRY ECHO(CHRS:CHAR);
( 0058)       BEGIN
( 0058) C30C00              ENTER_MON          12        0
( 005B)         OUT(#BD, CTRL_WORD);
( 005B) 9ABD                PS_CONST_1       189
( 005D) 940301              PS_CONST_2       259
( 0060) B7                  OUT
( 0061)         OUT(ORD(CHRS), DATA_WORD);
( 0061) 31                  FPS_VW_L+10
( 0062) 940201              PS_CONST_2       258
( 0065) B7                  OUT
( 0066)         OUT(#3D, CTRL_WORD);
( 0066) 9A3D                PS_CONST_1        61
( 0068) 940301              PS_CONST_2       259
( 006B) B7                  OUT
( 006C)       END;
( 006C) D3                  EXIT_MON
( 006D)
( 006D)     BEGIN
( 006D) C20A00              BEGIN_MON          10        0
( 0070)       OUT(#3D, CTRL_WORD);
( 0070) 9A3D                PS_CONST_1        61
( 0072) 940301              PS_CONST_2       259
( 0075) B7                  OUT
( 0076)     END;
( 0076) D2                  END_MON
( 0077)
( 0077)     (***************
( 0077)     *  UART_READ  *
( 0077)     ***************)
( 0077)
( 0077)     TYPE UART_READ = DEVICE_MON (TERM_OUT: UART_WRITE;
( 0077)                                  SELECTOR: INT);
( 0077)
( 0077)       VAR THROWAWAY: INT;
( 0077)
( 0077)     PROCEDURE ENTRY READ( VAR MESSAGE: LINE;  VAR LENGTH: INT);
( 0077)       VAR LASTCHAR: CHAR;
( 0077)       BEGIN
( 0077) C30E01              ENTER_MON          14        1
( 007A)         LENGTH:=0;
( 007A) 31                  FPS_VW_L+10
( 007B) A0                  COPY_ZERO
( 007C)         REPEAT
( 007C)           IF LENGTH<LINELENGTH THEN INC(LENGTH);
```

```
( 0004)  TYPE UART_WRITE = DEVICE MON (SELECTOR: INT);
( 0004)
( 0004)  PROCEDURE ENTRY WRITE(MESSAGE: LINE; DISP: LINE_DISP);
( 0004)     VAR I: INT;
( 0004)         THROWAWAY: INTEGER;
( 0004)     BEGIN
( 0004) C30F03              ENTER_MON         14       3
( 0007)         I:=1;
( 0007) 6F                  FPS_AD_L-01
( 0008) A1                  COPY_ONE
( 0009)         OUT(#BD, CTRL_WORD) { XMIT REQ., INT. EN., 8 DATA, 2 STOP, NO PA
( 0009) 9ABD                PS_CONST_1       189
( 000B) 940301              PS_CONST_2       259
( 000E) B7                  OUT
( 000F)         DOIO;
( 000F) DA                  DOIO
( 0010)         WHILE (MESSAGE[I] <> NUL)  AND  (I<LINELENGTH) DO
( 0010) 32                  FPS_VW_L+12
( 0011) 2F                  FPS_VB_L-01
( 0012) A7                  INDEX_11
( 0013) 91                  PS_INDB
( 0014) 00                  CONSTANT    0
( 0015) F6                  NE_WORD
( 0016) 2F                  FPS_VB_L-01
( 0017) 9A4A                PS_CONST_1        74
( 0019) F0                  LS_WORD
( 001A) AD                  AND
( 001B) 4C0E                FALSEJUMP         14
( 001D)         BEGIN
( 001D)           OUT(ORD(MESSAGE[I]),DATA_WORD);  {SEND A CHARACTER}
( 001D) 32                  FPS_VW_L+12
( 001E) 2F                  FPS_VB_L-01
( 001F) A7                  INDEX_11
( 0020) 91                  PS_INDB
( 0021) 940201              PS_CONST_2       258
( 0024) B7                  OUT
( 0025)           DOIO;
( 0025) DA                  DOIO
( 0026)           INC(I);
( 0026) 6F                  FPS_AD_L-01
( 0027) 9F                  INC_BYTE
( 0028)         END;
( 0028) 43E7                JUMP             -25
( 002A)         IF (DISP=PROMPT) OR (DISP=NEWLINE) THEN
( 002A) 31                  FPS_VW_L+10
( 002B) 00                  CONSTANT    0
( 002C) EE                  EQ_WORD
( 002D) 31                  FPS_VW_L+10
( 002E) 01                  CONSTANT    1
( 002F) EE                  EQ_WORD
( 0030) AE                  OR
( 0031) 4C0D                FALSEJUMP         13
( 0033)           BEGIN OUT(ORD(CR), DATA_WORD); DOIO;
( 0033) 0D                  CONSTANT   13
( 0034) 940201              PS_CONST_2       258
( 0037) B7                  OUT
( 0038) DA                  DOIO
( 0039)                 OUT(ORD(LF), DATA_WORD); DOIO;
( 0039) 0A                  CONSTANT   10
( 003A) 940201              PS_CONST_2       258
( 003D) B7                  OUT
( 003E) DA                  DOIO
( 003F)           END;
( 003F)         IF DISP=PROMPT THEN
( 003F) 31                  FPS_VW_L+10
```