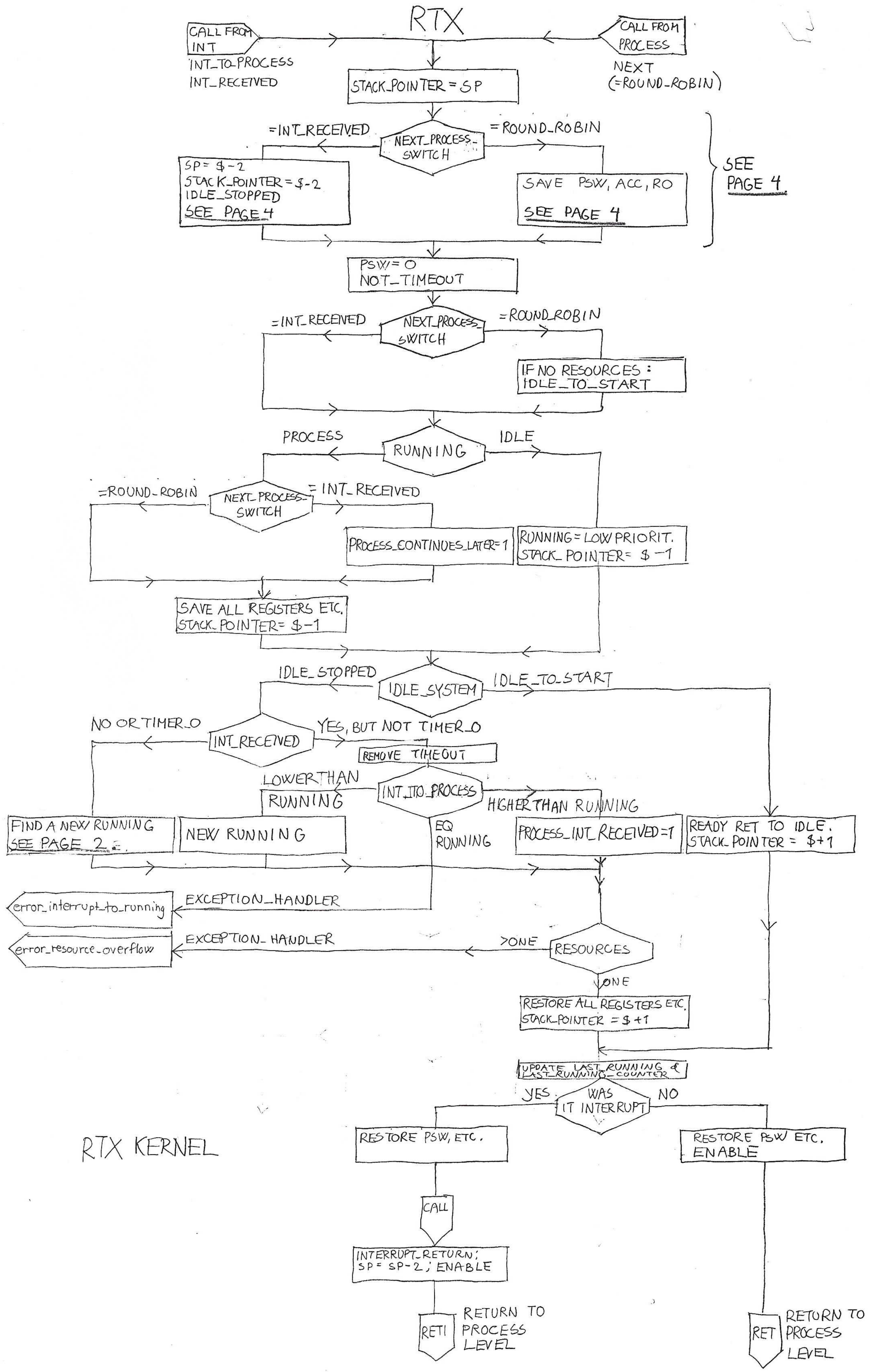
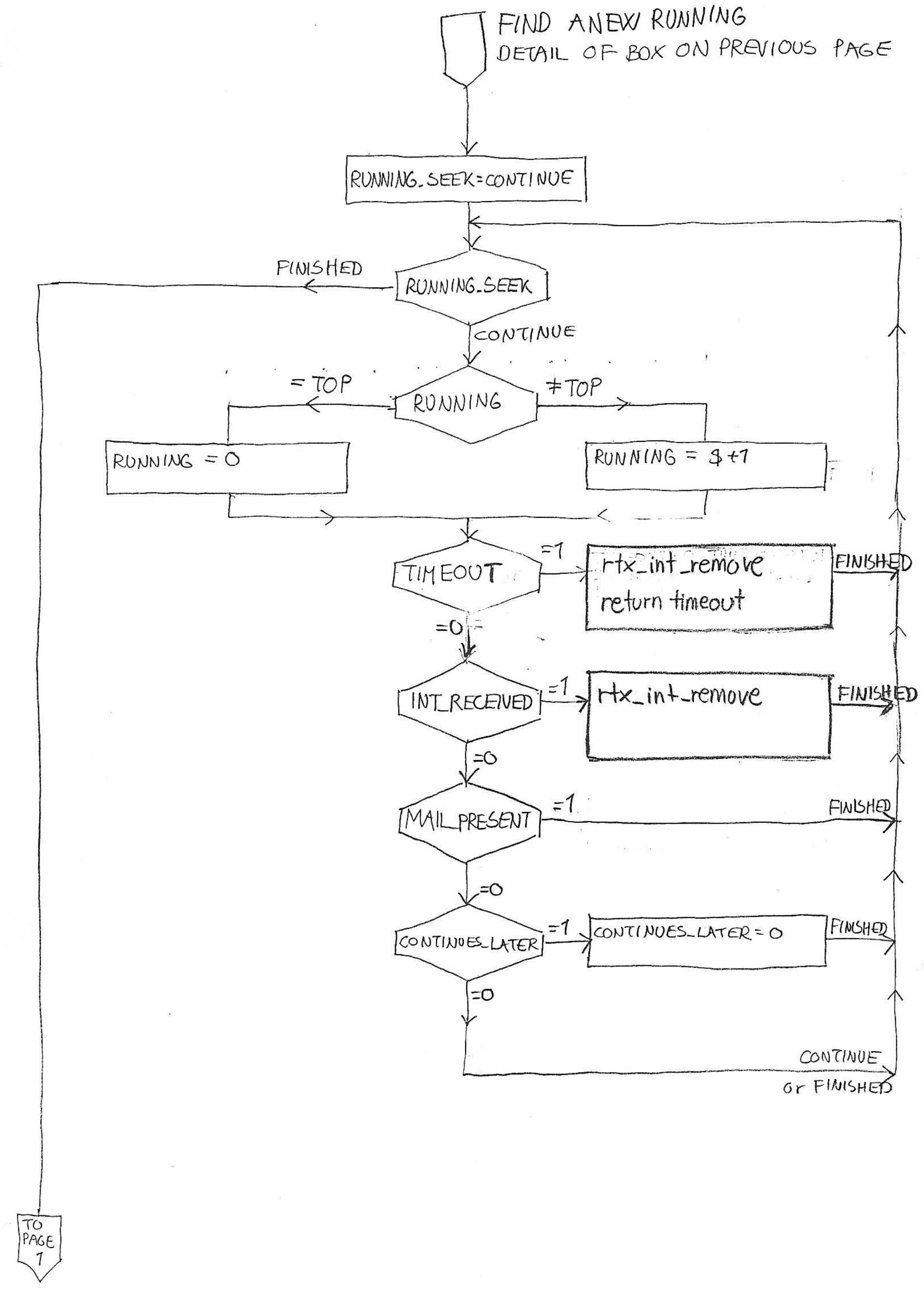
1988.3

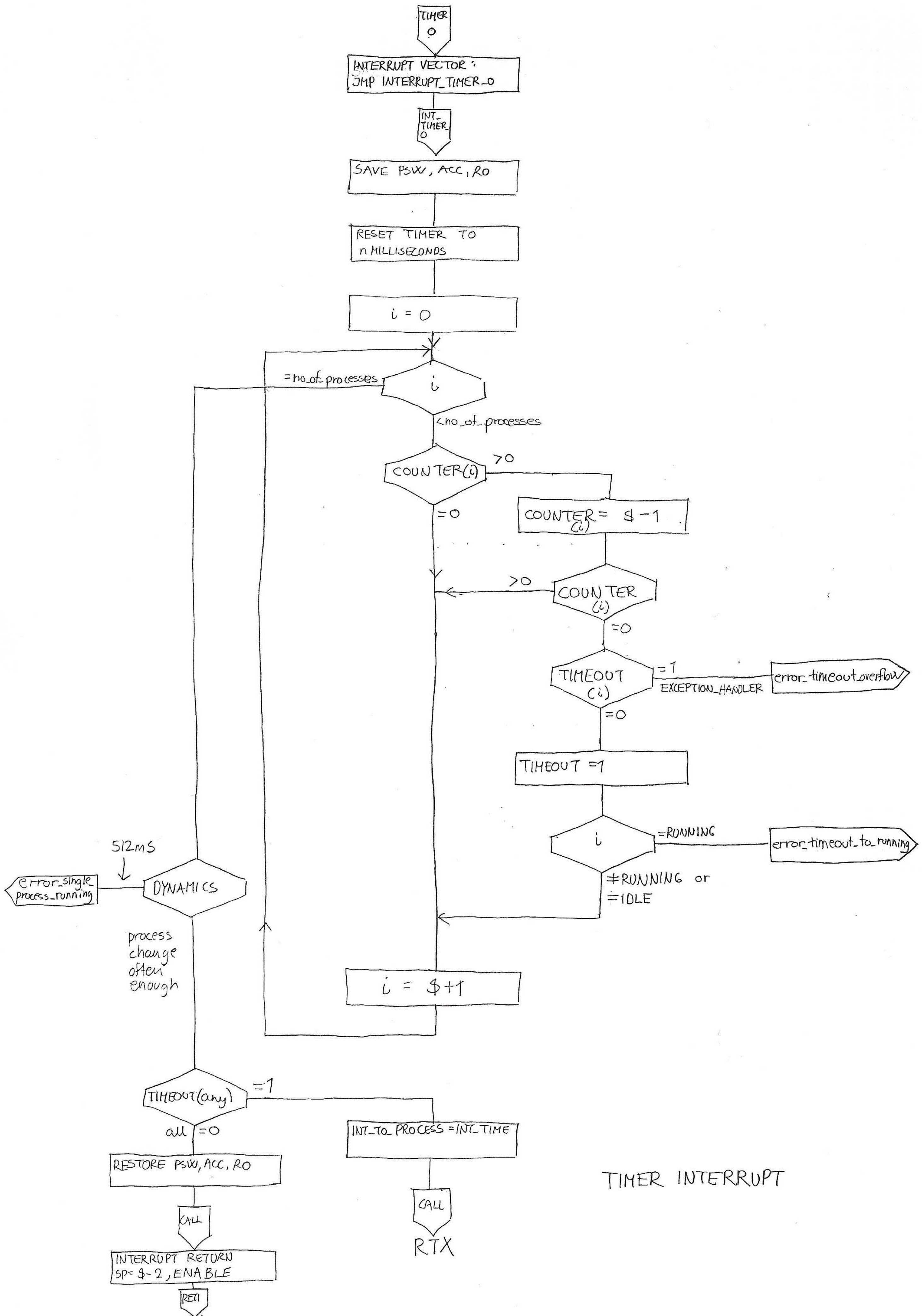
```
    DOS 3.30 (038-N) PL/M-51 V1.2
    COMPILER INVOKED BY:  C:\MCS51\PLM\PLM51.EXE RX51-080.PLM


                   $  optimize (3)
                   $  debug
                   $  symbols

                   /*
Autronica A/S.
Realtime executive for MCS51.

Observe       : The author does not guarantee that this program is free from
              : error. However, every attempt has been made to make it error-
              : free. In case an error is found, the user MUST report to the
              : original author.
Parts         : 1. Introduction and user part of RX51.
              : 2. Kernel part of RX51.
              : 3. User part of RX51.
              : 4. Power-up part of RX51.
              : 5. Compiler appendix.
Copying       : No copying to a third part. All copying via original author.
              : COPYRIGHT (C) 1988 BY AUTRONICA.
Used in       : BSU-50.
Orig. author  : Øyvind Teig.

This file     : rx51-nnn.plm (nnn: see next page)
Files         : A. rx51-nnn.plm -> this file.
              : B. note-nnn.inc -> included in this file. (PcWrite 2.71 file size max. is 60 K)
              : C. rx51-nnn.inc -> must be included in user process files.
              : D. rxiv-nnn.asm -> interrupt vectors and interrupt return.
PUBLICS       : No PUBLIC vars, EXTERNAL vars must be PUBLIC in user file.
Text editor   : PCWRITE 2.71. PLM list file through PR.
non-ascii     : Alt E (heart)            : Is carriage return and boldface.
              : Alt B (face)             : Is boldface.
              : Alt G dot R colon alt C  : Is compressed print mode (top of this page).
Ruler files   : in directory pcwr
nnn next page : Version of this file.
              */
```


RTX
CALL FROM INT
INT_TO_PROCESS
INT_RECEIVED
CALL FROM PROCESS
NEXT (=ROUND_ROBIN)
STACK_POINTER = SP
=INT_RECEIVED
NEXT_PROCESS_SWITCH
=ROUND_ROBIN
SP = $-2
STACK_POINTER = $-2
IDLE_STOPPED
SEE PAGE 4
SAVE PSW, ACC, R0
SEE PAGE 4
SEE PAGE 4
PSW = 0
NOT_TIMEOUT
=INT_RECEIVED
NEXT_PROCESS_SWITCH
=ROUND_ROBIN
IF NO RESOURCES :
IDLE_TO_START
PROCESS
RUNNING
IDLE
=ROUND_ROBIN
NEXT_PROCESS_SWITCH
=INT_RECEIVED
PROCESS_CONTINUES_LATER=1
RUNNING = LOW PRIORIT.
STACK_POINTER = $ -1
SAVE ALL REGISTERS ETC.
STACK_POINTER = $-1
IDLE_STOPPED
IDLE_SYSTEM
IDLE_TO_START
NO OR TIMER_0
INT_RECEIVED
YES, BUT NOT TIMER_0
REMOVE TIMEOUT
LOWER THAN RUNNING
INT_TO_PROCESS
HIGHER THAN RUNNING
EQ RUNNING
FIND A NEW RUNNING
SEE PAGE 2
NEW RUNNING
PROCESS_INT_RECEIVED=1
READY RET TO IDLE.
STACK_POINTER = $+1
EXCEPTION_HANDLER
error_interrupt_to_running
EXCEPTION_HANDLER
error_resource_overflow
>ONE
RESOURCES
ONE
RESTORE ALL REGISTERS ETC.
STACK_POINTER = $+1
UPDATE LAST_RUNNING & LAST_RUNNING_COUNTER
YES
WAS IT INTERRUPT
NO
RESTORE PSW, ETC.
RESTORE PSW ETC.
ENABLE
CALL
INTERRUPT_RETURN;
SP = SP-2; ENABLE
RETI
RETURN TO PROCESS LEVEL
RET
RETURN TO PROCESS LEVEL

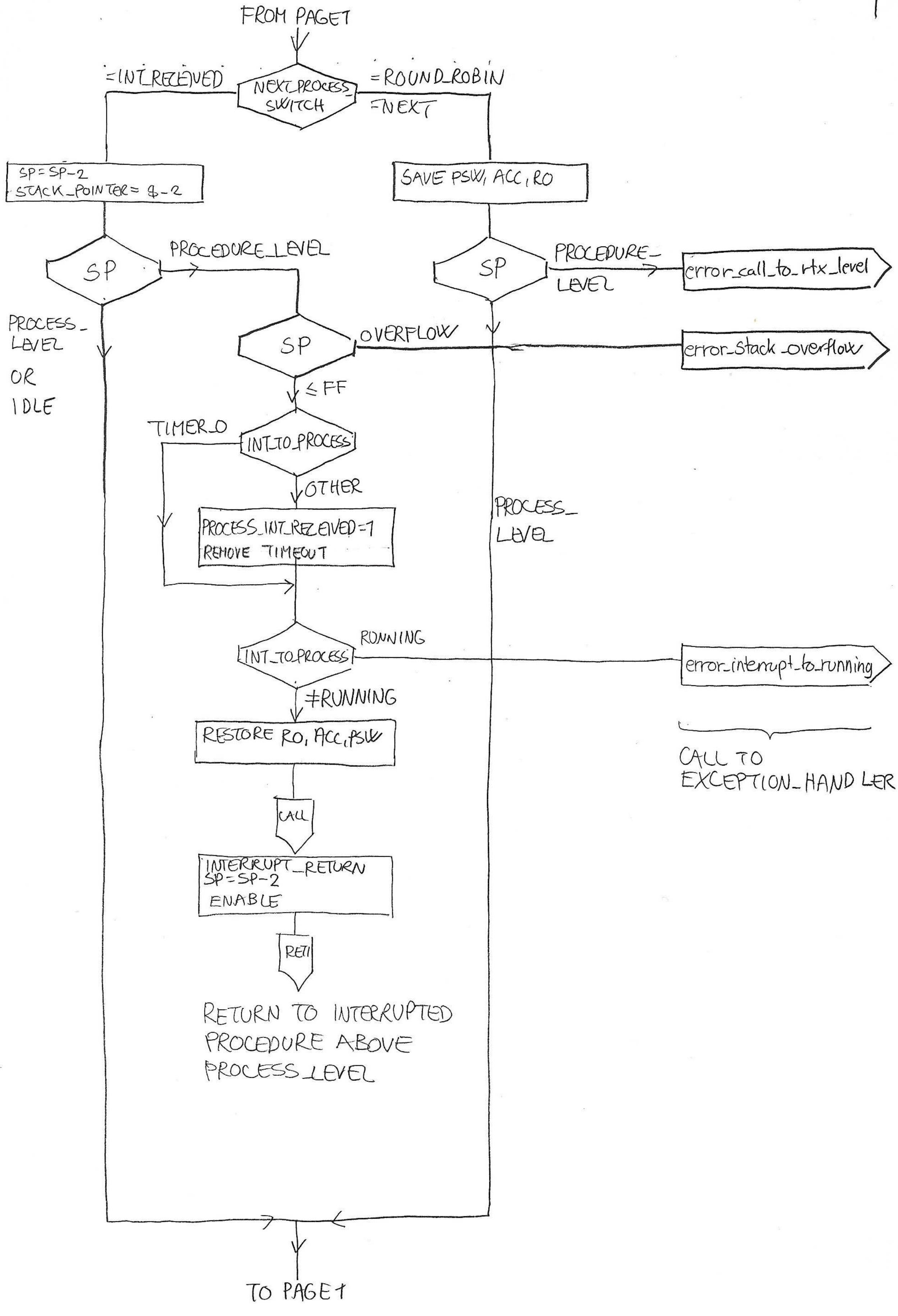
RTX KERNEL



DETAIL OF
RTX KERNEL



TIMER INTERRUPT



DETAIL OF RTX KERNEL

```
                 $  eject
                 /*
nnn Date:     Part:   Description                            Author:  Reader:
--- --------- ------- -------------------------------------- -------  -------
028 07.03.88 1,2,3,4 Ready for first new user.              Teig
    09.03.88 1       share_ram_a,b                          -"-
    05.04.88 1       protect_parameters not necessary       -"-
034 14.04.88 3       interrupt_ser_chan, wait (250,prc0)    -"-
             2       time_bit only set to 1, must be        -"-
                     cleared by process after use.
035 15.04.88 2       interrupt at process level priority    -"-
    18.04.88 1,2     interrupt priority scheeme             -"-
041 24.05.88         after software walkthrough by:         Teig     Solli
                                                                     Skogstad
             1       Increase readability and               -"-
                     process_mail_present not to 0 when     -"-
                     process_timeout ?running.              -"-
                     trap_before_idle removed.              -"-
050 50.09.88 4       internal ramtest at power-up           Teig
    14.09.88 2       Mailbox_empty_and_free renamed to      Teig
                     Mailbox_emptied
061 03.10.88 1,2     AMD_87C541 watchdog trigger            Teig
080 04.11.88 1,2     disable_if_timeout_inwait is
                     protection_code                        Teig
                 */

                 rx51__:
     1   1       DO;
```

```
          $  eject
          $  include (note-050.inc)
       =  /*
       =  Notes page 1:
       =
       =   1. Realtime multitasking executive. Up to 8 parallel processes may be
       =      implemented. Process 0 has highest priority when an interrupt occurs.
       =      When all interrupts have been served the processes are run in a round-
       =      robin fashion. Knowledge about the processes must be compiled into
       =      this file. (Name of the processes, how many there are, and the
       =      procedure rx51_init).
       =
       =   2. No time-slice mechanism is implemented. This has been considered not
       =      necessary because of the limited amount of processes possible and
       =      because at least one process must have periodic calls to the RX.
       =
       =   3. All IO ports, all internal registers in any registerbank, and all
       =      interrupts are available to the process level. The only hardware resource
       =      that the RX uses is the TIMER 0 which is reserved for the time-mechanism.
       =      (Also one bit of PSW is used, see time, page 12).
       =
       =   4. Special interrupt scheme and interrupt-to-process priority.
       =      The PL-M/51 INTERRUPT n facility therefore should not be used in the
       =      context of the rx51 kernel.
       =
       =   5. Inter-process communication and synchronization is realized with mail
       =      and protected mailboxes. (See page 10-11)
       =
       =   6. Waiting for mail may be done with a time-out mechanism.
       =
       =   7. Waiting for interrupt may be done with a time-out mechanism.
       =
       =   8. Continuation of a process again after a delay is realized.
       =
       =   9. Use "DISABLE" before any call to any PUBLIC procedure in this file.
       =      (Parameters must not be disturbed).
       =
       =  10. Use "ENABLE" again when appropriate.
       =      (When parameters have been moved to process-space).
       =
       =  11. Use "DISABLE" if R2, R3, R4 or R5 is used at process level,
       =      then "ENABLE" again. (Critical transaction at process level).
       =      It is not necessary with "DISABLE" if R2, R3, R4 or R5 is used
       =      at procedure level.
       =  */
```

```
        =   $  eject
        =   /*
        =   Notes page 2:
        =
        =   12. The RX is the only procedure within this file that returns with
        =       interrupts enabled. (That modifies the "ENABLE"/"DISABLE" to "ENABLE").
        =       Here is an example:
        =
        =       disable_within_do_end;      (=literally 'disable')
        =       DO;                         (extra DO - END to increase readability)
        =         call wait (200);
        =         call rx (next);           (rx ENABLES on return to next line)
        =       END;                        (here the interrupts are ENABLED)
        =
        =   13. All EXTERNAL processes must use register bank 0, with no USING.
        =
        =   14. The system will go idle if no processes have a reason to run.
        =
        =   15. There is one common stack for all the processes. The elements of the
        =       processor stack is addressed relative to SP. A process that uses
        =       the stack will have exclusive access to it. This implies that if a
        =       process has called a procedure, this procedure will finish before any
        =       other process will be started. This gives some advantages:
        =       1. Any procedure may be PUBLIC, and may be called from any process.
        =          Thus it will look reentrant, even if it actually is not so.
        =          However, take care to protect the input and output parameters.
        =       2. The stacksize will be given by the one process that needs the
        =          most stack, not by the sum of the processes' needs.
        =       3. Interrupts need not be disabled from procedure level.
        =       4. The internal variables in procedures called from process level may
        =          be at the same addresses in different files. Parameters used
        =          inside the procedure file must be declared at the top level.
        =          Several files will use the same ram, much the same way as a
        =          register bank. However, the parameters used at process level when
        =          invoking the procedure must not use this mechanism. The conclusion
        =          is that the only ram-resource that the procedure will need
        =          exclusively will be the parameters and local bit- variables.
        =
        =   16. The RX may be called from process level only.
        =       The other PUBLIC procedures may be called from any (stack-) level.
        =
        =   17. Study the error messages.
        =
        =   18. MCS-51 stack grows up (to a higher address) during a call.
        =
        =   */
```

```
=   $  eject
=   /*
=   Notes page 3:
=
=   19. Before a call, SP points to the address below the ram where the return
=       address will later be stored.
=       During the call SP will be incremented by 1 and LSB of the return
=       address will be stored at *SP.
=       Then the SP will be incremented by 1 and MSB of the return
=       address will be stored at *SP.
=       So then, inside the called procedure, SP will point to MSB of the
=       return address, and the cell below the LSB of the return address is
=       stored.
=       The stack may reside in MAIN or IDATA. All ram may be
=       indirectly addressable with respect to R0 or R1 or SP.
=
=       addresses:  00C0H   00C1H   00C2H  00C3H  00C4H  00C5H
=       sp = 00C1H            *
=       call                        PCL>   PCH>
=       inside                      lsb    msb
=       sp = 00C3H                          *
=       return                      PCL<   PCH<
=       sp = 00C1H            *
=
=   20. The MCS-51 series processors do not have any WORD arithmetic. The
=       compiler writers have assigned word values to MSB (lowest address)
=       and LSB (highest address). This is contrary to usual Intel standard,
=       but more readable during the debug phase.
=
=   21. RETI instruction resets the interrupt-logic. This is done at the
=       EXTERNAL label interrupt_return.
=
=   22. The variables assigned to AT (00,01,06,07) MUST be at those locations.
=
=   23. Also, BASED variables are used to gain some speed. The STRUCTURE
=       solution is (programming-wise) easier and safer, but unfortunately
=       much slower. The IDATA suffix with BASED variables removes the
=       "4 DEFAULTED BASED VARIABLES" warning. The IDATA suffix also saves
=       MAIN ram (but not total ram). Indirectly addressable ram (by *R0)
=       does not need to be located in MAIN.
=
=   24. At a clock frequency of 12 MHz, the process switching time is in the
=       250 uS area.
=
=   */
=   $  eject
=   /*
=   Notes page 4:
=
=   25. RAM resources, example with 4 processes and 1 mailbox:
=       (EXTERNAL ram must be added to these values to get the complete picture).
=       MODULE INFORMATION:                (STATIC+OVERLAYABLE)
=           CODE SIZE                      = ----H       ----D
=           CONSTANT SIZE                  = 000CH        12D
=           DIRECT VARIABLE SIZE           =   0EH+04H    14D+  4D
=           INDIRECT VARIABLE SIZE         =   32H+00H    50D+  0D
=           BIT SIZE                       =   02H+03H     2D+  3D
=           BIT-ADDRESSABLE SIZE           =   00H+00H     0D+  0D
=           AUXILIARY VARIABLE SIZE        = 0000H         0D
=           MAXIMUM STACK SIZE             = 000BH        11D
=           REGISTER-BANK(S) USED:           0
=
=       REGISTER used is R0 only.
=       R1,R2,R3,R4,R5,R6,R7 are never used.
=       EXCEPTION: R1 is used at rx51 power-up, in the internal ram test.
=
=   26. This point is referenced at the start of all the interrupt procedures.
=       The MICEII or the OKI 80C51VS 5709 or 5621 has the following error:
=       address.    Instruction                       *  This
=       000B        LJMP  03F2   Interrupt branch     *  works
=       03F2        MOV   01,01  Dummy move           *  just
=       03F5        MOV   10,PSW Save psw             *  fine
=
=       address.    Instruction                       *  This
=       000B        LJMP  03F5   Interrupt branch     *  works
=       03F2        MOV   01,01  Dummy move           *  just
=       03F5        MOV   10,PSW Save psw             *  fine
=
=       address.    Instruction                       *  This does not
=       000B        LJMP  03F2   Interrupt branch     *  work. It moves
=       03F2        MOV   10,PSW Save psw             *  0FFH to
=       03F5        MOV   01,01  Dummy move           *  address 10
=
=       The situation gives rise to error only after an IDLE.
=       (PCON = PCON or 00000001B). Conclusion: error with saving of PSW
=       when this is the first non-jump instruction if it is located
=       at a particular address.
=       Fix: do not save PSW as the first save in interrupt procedures.
=       Also see note 27.
=
=    */
=   $  eject
=   /*
=   Notes page 5:
=
=   27. In the serial channel procedure (outside the scope of this text), the
=       "p" (parity-bit in psw) did not work correctly. After some investigation
=       it was found that this was because the emulation processor on the MICE-II
=       just had left the "idle" state.
=       This is probably the same problem as the one described in note 26.
=       The MICE-II hardware (CEP 8051 REV-D) does not function
=       properly when the "idle" state is left.
=       Fix: The debug flag.
=            The initialization is done outside the scope of this program:
=            It is initalized to "1" if MICE-II is to be used.
=            It is initalized to "0" if program is running on the single-chip uP.
=       Use: Here the debug flag is used in the "idle" procedure to inhibit idle.
=            It is also used in the "interrupt_timer_0" procedure to inhibit the
=            "error_timer_interrupt_lost" if debug. This is another MICE-II
=            error, it does not turn off the timers while stopped.
=
=   28. Please bear in mind that the strongest force when writing the RX kernel
=       was to make it "simple". The user has to know how to use only 5 procedures:
=         1. call wait         see note 9
=         2. Mailbox_reserved  see note 9
=         3. Mailbox_filled    see note 9
=         4. Mailbox_emptied   see note 9
=         5. call rx           see note 9
=       No queueing at all is being done.
=       The user is urged to make an exception handler.
=       If no error messages comes through to the exception_handler,
=       the "user" should be relatively well assured that the rx51 is used
=       correctly.
=   */
=   $  eject
=   /*
=       State diagrams of the four main variables.
=       All have one bit for each process. (bit0 -> process 0).
=       ?running means "the test that finds a new running".
=       ?idle    means "the test that decides if idle is to be run".
=
=       rx51_init                ._________________________.
=       _________________.0  process_int_received  |
=       rx_int_remove_   |                         |   in kernel when
=       at_new_running   |                         |    int. received
=       _________________.0                       1._________________
=                        |                         |       (3 places)
=                        |          test           |
=                        .___________._____________.
=       when into ?idle              |        in kernel when ?running
=       _____________________________._______________________________
=
=       rx51_init                ._________________________.   Mailbox_filled
=       _________________.0  process_mail_present 1._________________
=                        |                         |
=                        |                         |
=       Mailbox_empty..  |0         test           |
=       ________________.___________._____________.
=                                    |        in kernel when ?running
=       when into ?idle              |               Mailbox_emptied
=       _____________________________._______________________________
=
=
=                        ._________________________.
=                        | process_continues_later |
=       in kernel when   |                         |        rx51_init
=       mail -> running  |                        1._________________
=       _________________.0                       1|   in kernel when
=       in kernel when   |                         | prior. interrupt
=       cont -> running  |0         test          1| wait (0 or 1)
=       _________________.___________._____________._________________
=       when into ?idle              |        in kernel when ?running
=       _____________________________._______________________________
=
=       rx51_init                ._________________________.
=       _________________.0   process_timeout      |
=       rx_int_remove_   |                         |
=       at_new_running   |                         |
=       _________________.0                        |
=       in kern. when int|                         |
=       _________________.0                        |
=       in kern. when int|                         |
=       & procedure level|0         test          1|    in timer int.
=       _________________.___________._____________._________________
=                                    |        in kernel when ?running
=       when into ?idle              |                  in timer int.
=       _____________________________._______________________________
=   */
=   $  eject
=   /*
=       Rudimentary transition diagram:
=
=
=                   ._____________. ._____________. ._____________.
=                   |    Idle.    |   Running.    | | Interrupted |
=                   |             ._.             ._. or wait(0). |
=                   |             | |             | | (continues_ |
=                   |             | |             | | later.) (*) |
=                   ._____________. .______.______. ._____________.
=                                          |
=        ._________________________________._________________________________.
=        |                           Ready to run.                           |
=        ._._______________._______________._______________.____________.____.
=          |               |               |               |   .________.____.
=          |               |               |               |   |Waiting for  |
=          |               |        .______.______.        |   |int. without |
=          |               | Init:__. Waiting for |        |   |timeout (*)  |
=          |               |        |  timeout.   |        |   .________.____.
=   ._______._____. .______.______. .______.______. .______.______.     |
=   |Waiting for  | | Waiting for |        |        | Waiting for |     |
=   |mail without | | mail with   |        |        | int. with   |     |
=   |timeout.     | | timeout.    |        |        | timeout.(*) |     |
=   ._________.___. .__._______.__.        |        .__._______.__.     |
=             |        |       |           |           |       |        |
=             |        |       |           |           |       |        |
=        .____.________.__. .__.___________.___________.__. .__.________._.
=        | Mailbox_filled | |     Timer 0 interrupt.      | | Interrupt(*)|
=        .________________. ._____________________________. ._____________.
=
=        .___________________________________________________________________.
=        |     Interrupt(*) or timeout that was not waited for is an error.  |
=        .___________________________________________________________________.
=
=         (*) in this case "interrupted" means that the process has been
=         running at process level (not subroutine level), and another
=         process has been rescheduled, leaving the "interrupted" process
=         unaware of the fact that it was suspended. It must, of course,
=         continue later.
=   */
=   $  eject
=   /*
=       Example of synchronization and process communication:
=
=      Process n: this is the process that is being signalled to:
=      DO FOREVER
=        disable;
=        DO WHILE not Mailbox_reserved (box_0); (Performs a wait for the byte-mailbox)
=          call wait (2);
=          call rx (next);
=          disable;
=        END;
=        call wait (timeout_counter) (optional)
=        call rx (next);
=        Sleep here until wakeup. (Performs a wait for data in the byte-mailbox)
=        .
=        wakeup done.
=        .
=        disable;
=        IF Mailbox_emptied (box_0, .mail) THEN
=          mail received because another process has sent mail via the
=          "Mailbox_filled" procedure.
=          (Performs a get mail, free the mailbox - operation)
=        ELSE
=        DO;
=         IF time_bit = timeout THEN
=            timeout because no other process has sent mail within the legal time
=            interval (=timeout_counter millisecond)
=          ELSE
=            interrupt received. It is actually not necessary to wait for an
=            interrupt via a mailbox.
=        END;
=        time_bit = false
=
=        Process on the mail if necessary.
=        During this phase the mail named box_0 is empty and any other process
=        may signal to it. An example of this is shown below:
=      END;
=
=   */
=   $  eject
=   /*
=      Process m (any process except process n). The process is signalling.
=      DO FOREVER
=
=        process data and make it ready to be mail.
=
=        disable;
=        DO WHILE not Mailbox_filled (box_0, mail);
=        (Performs a filling of the byte-mailbox when any is waiting for data in it)
=          call wait (2);
=          call rx (next);
=          disable;
=        END;
=        enable;
=
=        Here the mailbox has been filled since obviously some other process was
=        waiting for data in it.
=        Continue any other processing.
=      END;
=   */
```

```
             =   $  eject
                 /*
EXTERNAL references (that the user may modify):
                 */
     2   2          Assp_interrupt: PROCEDURE bit EXTERNAL;
                      /* Declare PUBLIC in \assp-nnn.plm */
     3   1          END Assp_interrupt;

     4   2          exception_handler: PROCEDURE (class_reason) EXTERNAL;
     5   2            DECLARE
                        class_reason byte;
                      /* Also declare EXTERNAL in \error-??.inc.
                      Declare PUBLIC in \excep-??.PLM */
     6   1          END exception_handler;

     7   2          interrupt_return: PROCEDURE EXTERNAL;
                      /* Declare PUBLIC in c:\bs100\slave\rx\rxiv-nnn.asm */
     8   1          END interrupt_return;

     9   2          test_exram: PROCEDURE EXTERNAL;
                      /* Declare PUBLIC in \iic2-nnn.plm */
    10   1          END test_exram;

    11   1          process_0: PROCEDURE EXTERNAL; END process_0;
    13   1          process_1: PROCEDURE EXTERNAL; END process_1;
    15   1          process_2: PROCEDURE EXTERNAL; END process_2;
    17   1          process_3: PROCEDURE EXTERNAL; END process_3;

    19   1          DECLARE
                                                            /* See note 25. */
                      debug                   bit  EXTERNAL, /* See note 27. */
                      milliseconds            byte EXTERNAL, /* max. 255     */
                      milliseconds_2          byte EXTERNAL, /* overflows    */
                      milliseconds_512        byte EXTERNAL, /* max. 255     */
                      stack_pointer_test_value byte EXTERNAL;

    20   1          DECLARE
                      TIME_BIT bit at (0D5H) register, /* flag 0 of PSW    */
                      ti       bit at (099H) register; /* transmitter int. */
                 $  eject
                 /*
DECLARE "public" references (also declared in RX51-nnn.inc).
                 */
   21   1          DECLARE
                     assp_to_rx  literally '1',
                     box_0       literally '0',
                     next        literally '0',
                     not_timeout literally '0',
                     timeout     literally '1';
                 /*
DECLARE literally for user configuration:
                 */
   22   1          DECLARE
                     processes_no_of literally '4',
                     mailboxes_no_of literally '1',
                     timer0_preset   literally '0F830H',
                     ramsize         literally '256';

   23   1          DECLARE
                     protection_code (processes_no_of) byte constant
                       (01101111B, /* To zero: EA, ES */
                        01111111B, /* To zero: EA     */
                        01111111B, /* To zero: EA     */
                        01111111B);/* To zero: EA     */
```

```
                $  eject
                /*
DECLARE literally except error messages:
                */
   24   1         DECLARE
                    call_no_return            literally 'call',
                    cycles_stopped            literally '16',
                    false                     literally '0',
                    idle_to_start             literally '1',
                    idle_stack_fix            literally '1',
                    idle_stopped              literally '0',
                    idle_is_running           literally '0FFH',
                    input_rx_int_received     literally '1',
                    input_rx_round_robin      literally 'next',
                    int_time                  literally 'processes_no_of',
                    int_to_process0           literally '0',
                    int_to_process1           literally '1',
                    int_to_process2           literally '2',
                    int_to_process3           literally '3',
                    int_to_process0_mask      literally '00000001B',
                    int_to_process1_mask      literally '00000010B',
                    int_to_process2_mask      literally '00000100B',
                    int_to_process3_mask      literally '00001000B',
                    no_owner                  literally 'processes_no_of',
                    running_search_finished   literally '1',
                    running_search_continues  literally '0',
                    timer_start               literally '1',
                    timer_stop                literally '0',
                    true                      literally '1';
                $  eject
                /*
DECLARE literally error messages:
                */
    25   1            DECLARE
                        error_call_to_rx_level        literally '001H',
                        error_interrupt_to_running    literally '002H',
                        error_no_timebase             literally '003H',
                        error_process_reg_bank        literally '004H',
                        error_return_to_idle          literally '005H',
                        error_single_process_running  literally '006H',
                        error_stack_overflow          literally '007H',
                        error_timeout_overflow        literally '008H',
                        error_timeout_to_running      literally '009H',
                        error_timer_interrupt_lost    literally '010H', /* note 27 */
                        error_idle_never_running      literally '011H',
                        error_interrupt_ext_0         literally '012H',
                        error_interrupt_ext_1         literally '013H',
                        error_interrupt_timer_1       literally '014H',
                        error_interrupt_timer_2       literally '015H',
                        error_disable_missing         literally '016H',
                        error_no_mailbox              literally '017H',
                        error_internal_ram            literally '018H';
```

```
                 $  eject
                 /*
DECLARE AT data, see note 22:
                 */
    26   1          DECLARE
                      process_and_rx_R0 byte at (00H), /* see note 25 */
                      process_R1        byte at (01H), /* see note 25 */
                      process_R6        byte at (06H),
                      process_R7        byte at (07H),
                      share_ram_a       byte,
                        int_to_process  byte at (.share_ram_a),
                        int_aux         byte at (.share_ram_a),
                      share_ram_b       byte,
                        int_mask        byte at (.share_ram_b),
                        int_aux_timer0  byte at (.share_ram_b);
                 /*
DECLARE data and constants:
                 */
    27   1          DECLARE
                      bit_mask (8) byte constant (01H, 02H, 04H, 08H, 10H, 20H, 40H, 80H),

                      copy_of_ACC_in_process   byte,
                      copy_of_PSW_in_process   byte,
                      copy_of_R0_in_process    byte,
                      process_int_received     byte,
                      process_mail_present     byte,
                      process_continues_later  byte,
                      process_timeout          byte,
                      running                  byte,
                      dot_idle_high            byte at (.running),
                      running_not_idle_counter byte,
                      running_last             byte,
                      running_last_counter     byte,
                      running_mask             byte,
                      dot_idle_low             byte at (.running_mask),

                      process_stack (processes_no_of)        structure
                       ((B, ACC, PSW, R1, R6, R7, DPH, DPL)  byte,
                        (pc_hl)                              word,
                         R0                                  byte) IDATA,

                      delay_counter (processes_no_of) byte IDATA,
                      mailbox       (mailboxes_no_of) byte IDATA,
                      mailbox_owner (mailboxes_no_of) byte IDATA,

                      idle_system    bit,
                      running_search bit,

                      ram BASED process_and_rx_R0 byte IDATA; /* note 23 */
```

```
                $  eject

                /*
Standard Intel declarations used in this context:
                */
   28   1         DECLARE
                    ACC  byte at (0E0H) register,
                    B    byte at (0F0H) register,
                    DPH  byte at (083H) register,
                    DPL  byte at (082H) register,
                    IE   byte at (0A8H) register,
                    IP   byte at (0B8H) register,
                    EA   bit  at (0AFH) register,
                    PCON byte at (087H) register,
                    PSW  byte at (0D0H) register,
                    SP   byte at (081H) register,
                    TH0  byte at (08CH) register,
                    TL0  byte at (08AH) register,
                    TMOD byte at (089H) register;

   29   1         DECLARE
                    CY  bit at (0D7H) register;

   30   1         DECLARE
                    ES  bit at (0ACH) register,
                    ET0 bit at (0A9H) register;

   31   1         DECLARE
                    TF0 bit at (08DH) register,
                    TR0 bit at (08CH) register;
                /*
AMD 87C541 watchdog trigger:
                */
   32   1         DECLARE
                    AMD_87C541_watchdog_key byte at (0AAH) register;
                $  eject
                /*
rx51_init:

process (0) will be the highest prioritized, process (3) the lowest.
delay_counter (n) = 1 will activate processes at first timer 0 interrupt.
                */

    33   2          rx51_init: PROCEDURE;

                /*
                !    Not compiled, garbage is allowed:
                !    DO i = 0 to (processes_no_of - 1);
                !      process_stack (i).ACC = 0;
                !      process_stack (i).B   = 0;
                !      process_stack (i).R0  = 0;
                !      process_stack (i).R1  = 0;
                !      process_stack (i).R6  = 0;
                !      process_stack (i).R7  = 0;
                !    END;

                !    Not compiled, all ram is already zeroed at RX51 system level:
                !    running_not_idle_counter = 0;
                !    running_last_counter = 0;
                !    running_mask = 0;
                !    process_int_received = 0;
                !    process_mail_present = 0;
                !    process_continues_later = 0;
                !    process_timeout = 0;
                !    process_stack (n).PSW = 0;
                */
    34   2           mailbox_owner (0)       = no_owner;
    35   2           process_stack (0).pc_hl = .process_0;
    36   2           delay_counter (0)       = 1;
    37   2           process_stack (1).pc_hl = .process_1;
    38   2           delay_counter (1)       = 1;
    39   2           process_stack (2).pc_hl = .process_2;
    40   2           delay_counter (2)       = 1;
    41   2           process_stack (3).pc_hl = .process_3;
    42   2           delay_counter (3)       = 1;

    43   2           running = idle_is_running;
    44   2           running_last = idle_is_running;
    45   1          END rx51_init;
```

```
                $  eject
                /*
time_unit_init:
                */
    46   2          time_unit_init: PROCEDURE;

                /*   Timer 0 mode 1.
                     16 bit timer/counter with program reload.
                     Internal xtal1.2 division.
                     TR0 to start and stop.
                */
    47   2           TMOD = TMOD and 11110000B;
    48   2           TMOD = TMOD or  00000001B;

                /*   A. Clock frequency  : 12 MHz
                     B. Divide by 12
                     C. Internal node S3 : 1 MHz
                     D. Internal node S3 : 1 S
                     E. Count 2000 to overflow
                     D. CY to TF0        : 2 mS
                     F. timer0_preset    : 10000H - 2000D = 0FB30H
                     G. error_timer_interrupt_lost : interrupt_timer_0
                */
    49   2           TL0 = low  (timer0_preset);
    50   2           TH0 = high (timer0_preset);

                /*   start timer 0 and enable interrupt from it: */

    51   2           TF0 = 0;
    52   2           TR0 = timer_start;
    53   2           ET0 = 1;
    54   1         END time_unit_init;
```

```
                 $  eject
                 /*
idle:
This procedure is never called, but started as if it were a process.

On the BS-50 hardware, a pulse will trigger the watchdog circuit 80 uS after
idle is left. The watchdog will reset the processor 1.2 seconds after last trigger puls.

The internal watchdog on the AMD_87C541 processor may also be used.
It is triggered in rx when idle was left.
                 */
    55   2          idle: PROCEDURE;

    56   3            DO WHILE 1 = 1;
    57   3              running_not_idle_counter = 0;
    58   3              IF not debug THEN /* See note 27. */
    59   4              DO;
    60   4                PCON = PCON or 00000001B;
    61   4              END;
    62   3            END;
    63   2            call_no_return exception_handler (error_return_to_idle);

    64   1          END idle;
```

```
                 $  eject
                 /*
interrupt_init:
                 */
  65   2            interrupt_init: PROCEDURE;

                    /*
                    When one interrupt may interrupt another interrupt, it is
                    called "multilevel interrupt". Within the context of the RX,
                    multilevel interrupts are not allowed.

                    However, the RX allows different bits in the ip-register
                    to be set to 1. In the file rxiv-nnn.asm, the first
                    instruction of every interrupt routine is "disable" or
                    clr 0A8H.7. This allows for different priority, but still
                    multilevel interrupts are avoided.

                    The idea is that all interrupts should be served so fast
                    that it is not necessary for one interrupt to interrupt
                    another.

                    The rx51 is able to serve at least one serial channel at a
                    baud rate of 9600 and at the same time have timer 0
                    interrupts every 2 mS and at the same time have 4 living
                    processes.

                    The ip is reset to 0 during the interrupt_timer_0 if
                    error_timer_interrupt_lost is attempted. Then the differ-
                    ent bits must be set again outside the context the RX.

                    For the OKI 83C154, the ip.bit7 must be 0.
                    */

  66   2              IE = 00000000B;
  67   2              IP = 00010000B;

  68   1            END interrupt_init;
```

```
                 $  eject
                 /*
wait:
Assign a new value to the delay counter in order to start or stop delay,
wait for interrupt with timeout or mail with timeout:

IF wait_milliseconds = 0   THEN process_continues_later.
IF wait_milliseconds = 1   THEN process_continues_later.
IF wait_milliseconds = 2   THEN timeout within 2   to   4 mS.
IF wait_milliseconds = 3   THEN timeout within 2   to   4 mS.
IF wait_milliseconds = 4   THEN timeout within 4   to   6 mS.
IF wait_milliseconds = 5   THEN timeout within 4   to   6 mS.

IF wait_milliseconds = 250 THEN timeout within 250 to 252 mS.

IF wait_milliseconds = 254 THEN timeout within 254 to 256 mS.
IF wait_milliseconds = 255 THEN timeout within 254 to 256 mS.
                 */
   69   2          wait: PROCEDURE (wait_milliseconds) PUBLIC;

   70   2          DECLARE
                     wait_milliseconds byte,
                     timer0_tics       byte;

   71   2            IF EA THEN call_no_return exception_handler (error_disable_missing);

   73   2            timer0_tics = shr (wait_milliseconds, 1); /* div 2 */
   74   2            IF timer0_tics = 0 THEN
   75   3            DO;
   76   3              process_continues_later = process_continues_later or bit_mask (running);
   77   3              timer0_tics = 0;
   78   3            END;
   79   2            ELSE timer0_tics = timer0_tics + 1;

   80   2            delay_counter (running) = timer0_tics;

   81   1          END wait;
```

```
                 $  eject
                 /*
Mailbox_reserved:
Process synchronization primitive: the mailbox.
                 */
    82   1          Mailbox_reserved:
                    PROCEDURE (mailbox_num) bit PUBLIC;

    83   2          DECLARE
                      mailbox_num byte;

    84   2            IF EA THEN call_no_return exception_handler (error_disable_missing);

    86   2            IF mailbox_num >= mailboxes_no_of THEN
    87   2              call_no_return exception_handler (error_no_mailbox);

    88   2            IF mailbox_owner (mailbox_num) = no_owner THEN
    89   3            DO;
    90   3              mailbox_owner (mailbox_num) = running;
    91   3              RETURN (true);
    92   3            END;
    93   2            ELSE RETURN (false);

    94   1          END Mailbox_reserved;
```

```
                  $  eject
                  /*
Mailbox_filled:
Process synchronization primitive: the mailbox.
                  */
   95   1            Mailbox_filled:
                     PROCEDURE (mailbox_num, mail) bit PUBLIC;

   96   2            DECLARE
                       mailbox_num       byte,
                       mail              byte,
                       mailbox_owned_by  byte,
                       mask              byte;

   97   2              IF EA THEN call_no_return exception_handler (error_disable_missing);

   99   2              IF mailbox_num >= mailboxes_no_of THEN
  100   2                call_no_return exception_handler (error_no_mailbox);

  101   2              mailbox_owned_by = mailbox_owner (mailbox_num);

  102   2              IF mailbox_owned_by = no_owner THEN RETURN (false);
  104   2              ELSE
                       DO;
  105   3                mailbox (mailbox_num) = mail;
  106   3                mask = bit_mask (mailbox_owned_by);
  107   3                process_mail_present = process_mail_present or mask;
  108   3                RETURN (true);
  109   3              END;

  110   1            END Mailbox_filled;
```

```
                 $  eject
                 /*
Mailbox_emptied:
Process synchronization primitive: the mailbox.
                 */
   111   1          Mailbox_emptied:
                    PROCEDURE (mailbox_num, mail_ptr) bit PUBLIC;

   112   2          DECLARE
                      mailbox_num byte,
                      mail_ptr    byte,
                      mail_return BASED mail_ptr byte IDATA; /* note 23 */

   113   2            IF EA THEN call_no_return exception_handler (error_disable_missing);

   115   2            IF mailbox_num >= mailboxes_no_of THEN
   116   2              call_no_return exception_handler (error_no_mailbox);

   117   2            IF (process_mail_present and running_mask) = 0 THEN RETURN (false);
   119   2            ELSE
                      DO;
   120   3              process_mail_present = process_mail_present and not running_mask;
   121   3              mail_return = mailbox (mailbox_num);
   122   3              mailbox_owner (mailbox_num) = no_owner;
   123   3              RETURN (true);
   124   3            END;

   125   1          END Mailbox_emptied;
```

```
                 $  eject
                 /*
Procedure called from within the rx kernel:
rx_int_remove_at_new_running:
                 */
   126   2          rx_int_remove_at_new_running: PROCEDURE;
   127   2          DECLARE
                      mask byte;

   128   2            mask                 = not running_mask;
   129   2            process_timeout      = process_timeout and mask;
   130   2            process_int_received = process_int_received and mask;
   131   2            running_search       = running_search_finished;
   132   2            IE                   = IE and protection_code (running);

   133   1          END rx_int_remove_at_new_running;
```

```
                $  eject
                /*
rx: (one call from process level)
                */
   134   2          rx: PROCEDURE (next_process_switch) PUBLIC;

   135   2          DECLARE
                      next_process_switch bit,
                      copy_of_CY          bit,
                      return_from_rx      bit;

   136   2          DECLARE
                      process_B             byte,
                      stack_pointer         byte,
                      stack_data            BASED stack_pointer byte IDATA, /* note 23 */
                      stack_data_dot_idle   BASED stack_pointer word IDATA, /* note 23 */
                                            /* word not to use high/low that use R6,R7 */
                      process_stack_pointer byte,
                      stack_data_indirect   BASED process_stack_pointer byte IDATA, /* note 23 */

                /*    never used as such: */
                      process_stack_element BASED process_stack_pointer structure
                        ((B, ACC, PSW, R1, R6, R7, DPH, DPL, PCH, PCL, R0) byte) IDATA;
                /*
Take copy of stack pointer:
                */
   137   2            stack_pointer = SP;
                /*
Interrupt received, remove call from interrupt procedure:
                */
   138   2            IF next_process_switch = input_rx_int_received THEN
   139   3            DO;
   140   3              stack_pointer = stack_pointer - 2;
   141   3              SP            = stack_pointer;
   142   3              idle_system   = idle_stopped;
```

```
                $  eject
                /*
Test for stack overflow from OFFH to zero:
                */
   143   3             IF stack_pointer_test_value <> SP THEN
   144   4             DO;
   145   4               IF stack_pointer_test_value > SP THEN
   146   5               DO;
   147   5                 SP = stack_pointer_test_value;
   148   5                 call_no_return exception_handler (error_stack_overflow);
   149   5               END;
                /*
A procedure called from process level was interrupted:
Make note of interrupt, remove timeout, return to interrupted point:
                */
   150   4               IF (int_to_process <> int_time) THEN
   151   5               DO;
   152   5                 process_int_received = process_int_received or int_mask;
   153   5                 delay_counter (int_to_process) = 0;
   154   5                 process_timeout = process_timeout and not int_mask;

   155   5                 IF int_to_process = running THEN
   156   6                 DO;
   157   6                   call_no_return exception_handler (error_interrupt_to_running);
   158   6                 END;
   159   5               END;

   160   4               process_and_rx_R0 = copy_of_R0_in_process;
   161   4               ACC               = copy_of_ACC_in_process;
   162   4               PSW               = copy_of_PSW_in_process;
   163   4               call_no_return interrupt_return;
   164   4             END;
```

```
                $  eject
                /*
Process level of high priority was interrupted:
Make note of interrupt, remove timeout, return to interrupted point:
                */
  165   3            ELSE IF (running <= int_to_process) THEN
  166   4            DO;
  167   4              IF (int_to_process <> int_time) THEN
  168   5              DO;
  169   5                process_int_received = process_int_received or int_mask;
  170   5                delay_counter (int_to_process) = 0;
  171   5                process_timeout = process_timeout and not int_mask;

  172   5                IF int_to_process = running THEN
  173   6                DO;
  174   6                  call_no_return exception_handler (error_interrupt_to_running);
  175   6                END;
  176   5              END;

  177   4              process_and_rx_R0 = copy_of_R0_in_process;
  178   4              ACC               = copy_of_ACC_in_process;
  179   4              PSW               = copy_of_PSW_in_process;
  180   4              call_no_return interrupt_return;
  181   4            END;
  182   3          END;
                /*
No interrupt, save important resources:
                */
  183   2          ELSE
                   DO;
  184   3            copy_of_PSW_in_process = PSW;/* Incl. input par. */
  185   3            copy_of_ACC_in_process = acc;
  186   3            copy_of_R0_in_process  = process_and_rx_R0;
                /*     No use of acc,R0 or PSW (except CY) yet. */
                /*
Call to rx from a procedure called from process level is not legal:
                */
  187   3            IF stack_pointer_test_value <> SP THEN
  188   4            DO;
  189   4              call_no_return exception_handler (error_call_to_rx_level);
  190   4            END;
  191   3          END;
```

```
                 $  eject
                 /*
Register bank 0:
                 */
  192   2            PSW = 0;
                 /*
Reset return parameter.
                 */
  193   2            return_from_rx = not_timeout;
                 /*
Test for correct register bank in process:
                 */
  194   2            IF (copy_of_PSW_in_process and 00011000B) <> 0 THEN
  195   3            DO;
  196   3              call_no_return exception_handler (error_process_reg_bank);
  197   3            END;
                 /*
Return to idle if no resources:
                 */
  198   2            IF next_process_switch     = input_rx_round_robin and
                        process_timeout         = 0 and
                        process_int_received    = 0 and
                        process_continues_later = 0 and
                        process_mail_present    = 0 THEN
  199   3            DO;
  200   3              idle_system = idle_to_start;
  201   3            END;
                 /*
Idle was running, the highest prioritized process will be started.
Trigger the AMD_87C541 watchdog circuit now:
                 */
  202   2            IF running = idle_is_running THEN
  203   3            DO;
  204   3              running = processes_no_of - 1;
  205   3              stack_pointer = stack_pointer - idle_stack_fix;
  206   3              AMD_87C541_watchdog_key = 0A5H;
  207   3              AMD_87C541_watchdog_key = 05AH;
  208   3            END;
                 /*
Process was running, save its resources, suspend running to run later on if interrupted:
                 */
  209   2            ELSE
                     DO;
  210   3              IF next_process_switch = input_rx_int_received THEN
  211   4              DO;
  212   4                process_continues_later = process_continues_later or running_mask;
  213   4              END;
```

```
                 $  eject
                 /*
Save status of running process and reset stack pointer:
                 */
         3              process_B             = B;
         3              process_stack_pointer = .process_stack (running);

         3              stack_data_indirect   = process_B;
         3              process_stack_pointer = process_stack_pointer + 1;
         3              stack_data_indirect   = copy_of_ACC_in_process;
         3              process_stack_pointer = process_stack_pointer + 1;
         3              stack_data_indirect   = copy_of_PSW_in_process;
         3              process_stack_pointer = process_stack_pointer + 1;
         3              stack_data_indirect   = process_R1;
         3              process_stack_pointer = process_stack_pointer + 1;
         3              stack_data_indirect   = process_R6;
         3              process_stack_pointer = process_stack_pointer + 1;
         3              stack_data_indirect   = process_R7;
         3              process_stack_pointer = process_stack_pointer + 1;
         3              stack_data_indirect   = DPH;
         3              process_stack_pointer = process_stack_pointer + 1;
         3              stack_data_indirect   = DPL;

         3              process_stack_pointer = process_stack_pointer + 1;
         3              stack_data_indirect   = stack_data;
         3              process_stack_pointer = process_stack_pointer + 1;
         3              stack_pointer         = stack_pointer - 1;
         3              stack_data_indirect   = stack_data;
         3              process_stack_pointer = process_stack_pointer + 1;
         3              stack_data_indirect   = copy_of_R0_in_process;
         3            END; /* ELSE */
```

```
              $  eject
              /*
Return to idle, idle does not have any resources to restore:
Much work in order not to use R6, R7.
These registers may have been used here, since idle_to_start anyway.
The effort is made, then, to make the system "clean and neat", never to
use any register but R0.
              */
                  IF idle_system = idle_to_start THEN
                  DO;
                    stack_data_dot_idle = .idle; /* high byte is at lowest address, swap high/low bytes: */

                    DO;
                      dot_idle_high = stack_data;
                      stack_pointer = stack_pointer + 1;
                      dot_idle_low  = stack_data;

                      stack_data    = dot_idle_high;
                      stack_pointer = stack_pointer - 1;
                      stack_data    = dot_idle_low;
                    END;

                    stack_pointer = stack_pointer + idle_stack_fix;
                    running       = idle_is_running;
                    running_mask  = idle_is_running;
                  END;
                  $  eject
                  /*
Enter process later, interrupt other than timer interrupt, remove its timeout counter:
                  */
   254   2              ELSE
                        DO;
   255   3                IF (int_to_process <> int_time) and (next_process_switch = input_rx_int_received) THEN
   256   4                DO;
   257   4                  delay_counter (int_to_process) = 0;
   258   4                  process_timeout = process_timeout and not int_mask;
                  /*
Make note of less prioritized int. to be served at a later call to rx:
                  */
   259   4                  IF (int_to_process >= running) THEN
   260   5                  DO;
   261   5                    process_int_received = process_int_received or int_mask;

                  /*
Interrupt to running is an error:
                  */
   262   5                    IF int_to_process = running THEN
   263   6                    DO;
   264   6                      call_no_return exception_handler (error_interrupt_to_running);
   265   6                    END;
   266   5                  END;
                  /*
Prioritized interrupt to be served now:
                  */
   267   4                  ELSE
                            DO;
   268   5                    running = int_to_process;
   269   5                    running_mask = int_mask;
   270   5                  END;
   271   4                END;
```

```
                $  eject
                /*
No interrupt or timer interrupt, assign new running according to status:
                */
   272   3             ELSE
                       DO;
   273   4               running_search = running_search_continues;
                /*
Look for a candidate for running:
                */
   274   5               DO WHILE running_search = running_search_continues;
                /*
Round-robin search for running:
                */
   275   5                 IF running = (processes_no_of - 1) THEN
   276  6                 DO;
   277   6                   running = 0;
   278   6                   running_mask = 00000001B;
   279   6                 END;
   280   5                 ELSE
                           DO;
   281   6                   running = running + 1;
   282   6                   running_mask = shl (running_mask, 1);
   283   6                 END;
```

```
                    $  eject
                    /*
Running found if timeout,
set return parameter and disable the corresponding interrupts and zero process_mail_present:
                    */
   284   5                   IF ((process_timeout and running_mask) <> 0) THEN
   285   6                   DO;
   286   6                     call rx_int_remove_at_new_running;
   287   6                     return_from_rx = timeout;
   288   6                   END;
   289   5                   ELSE
                             DO;
                    /*
Running found if an earlier interrupt was not recognized:
                    */
   290   6                     IF ((process_int_received and running_mask) <> 0) THEN
   291   7                     DO;
   292   7                       call rx_int_remove_at_new_running;
   293   7                     END;
                    /*
Running found if mail has been sent: (Mailbox_emptied later). Remove its timeout counter:
                    */
   294   6                     ELSE
                               DO;
   295   7                       IF ((process_mail_present and running_mask) <> 0) THEN
   296   8                       DO;
   297   8                         process_continues_later = process_continues_later and not running_mask;
   298   8                         running_search          = running_search_finished;
   299   8                         delay_counter (running) = 0;
   300   8                       END;
   301   7                       ELSE
                                 DO;
                    /*
Running found if process has been suspended earlier:
                    */
   302   8                         IF ((process_continues_later and running_mask) <> 0) THEN
   303   9                         DO;
   304   9                           process_continues_later = process_continues_later and not running_mask;
   305   9                           running_search          = running_search_finished;
   306   9                         END; /* IF    */
   307   8                       END; /* ELSE  */
   308   7                     END; /* ELSE  */
   309   6                   END; /* ELSE  */
   310   5                 END; /* WHILE */
   311   4               END; /* ELSE  */
```

```
                $ eject
                /*
Interrupt or no interrupt, assign values to next running process:
                */
        3              process_stack_pointer = .process_stack (running);

        3              B                     = stack_data_indirect;
        3              process_stack_pointer  = process_stack_pointer + 1;
        3              copy_of_ACC_in_process = stack_data_indirect;
        3              process_stack_pointer  = process_stack_pointer + 1;
        3              copy_of_PSW_in_process = stack_data_indirect;
        3              process_stack_pointer  = process_stack_pointer + 1;
        3              process_R1            = stack_data_indirect;
        3              process_stack_pointer  = process_stack_pointer + 1;
        3              process_R6            = stack_data_indirect;
        3              process_stack_pointer  = process_stack_pointer + 1;
        3              process_R7            = stack_data_indirect;
        3              process_stack_pointer  = process_stack_pointer + 1;
        3              DPH                   = stack_data_indirect;
        3              process_stack_pointer  = process_stack_pointer + 1;
        3              DPL                   = stack_data_indirect;

        3              process_stack_pointer  = process_stack_pointer + 1;
        3              stack_pointer         = stack_pointer + 1;
        3              stack_data /* pc_h */  = stack_data_indirect;
        3              process_stack_pointer  = process_stack_pointer + 1;
        3              stack_pointer         = stack_pointer - 1;
        3              stack_data /* pc_l */  = stack_data_indirect;

        3              process_stack_pointer  = process_stack_pointer + 1;
        3              process_and_rx_R0     = stack_data_indirect;
        3              ACC                   = copy_of_ACC_in_process;
        3            END; /* ELSE */
```

```
               $  eject
               /*
Update test parameters to make timer interrupt able to test for error_single_process_running:
               */
   338   2          running_last = running;
   339   2          running_last_counter = 0;
               /*
Return to idle or process, restore PSW and return to running process:
Process must reset TIME_BIT to false when used.
               */
   340   2          IF next_process_switch = input_rx_int_received THEN
   341   3          DO;
   342   3            PSW        = copy_of_PSW_in_process;
   343   3            copy_of_CY = CY;
   344   3            TIME_BIT   = TIME_BIT or return_from_rx; /* Involves CY */
   345   3            cy         = copy_of_CY;
   346   3            call_no_return interrupt_return;
   347   3          END;
   348   2          PSW        = copy_of_PSW_in_process;
   349   2          copy_of_CY = CY;
   350   2          TIME_BIT   = TIME_BIT or return_from_rx; /* Involves cy */
   351   2          CY         = copy_of_CY;
   352   2          enable;

   353   1        END rx;
```

```
                  $  eject
                  /*
interrupt_timer_0: (vector at address 0BH). Priority after interrupt_external_0 only.
                  */
   354   2           interrupt_timer_0: PROCEDURE PUBLIC;

   355   2            DECLARE
                       /* int_aux is used both alone (int_aux) or with: */
                       counter BASED int_aux byte IDATA; /* note 23 */
                  /*
Save status:
                  */
   356   2             copy_of_R0_in_process  = process_and_rx_R0;
   357   2             copy_of_ACC_in_process = ACC;
   358   2             copy_of_PSW_in_process = PSW; /* See note 26. */
   359   2             int_aux_timer0         = 00000001B;
                  /*
Reset timer 0 counter. Do not introduce time-skew.
                                                                                       cycles
   360   2        */   TR0     = timer_stop;                                         /*     */
   361   2             int_aux = TL0 + low (timer0_preset + cycles_stopped);         /* 3+ */
   362   2             TL0     = int_aux;                                            /* 2+ */

   363   2             If CY = 1 THEN                                                /* 4+ */
   364   3             DO;                                                           /* 0  */
   365   3               TH0 = TH0 + 1;                                              /* 1. */
   366   3               TH0 = TH0 + high (timer0_preset + cycles_stopped);          /* 3. */
   367   3             END;                                                          /* 2. */
   368   2             ELSE /* The same number of cycles as THEN above:                   */
                       DO;                                                           /* 0  */
   369   3               int_aux_timer0 = 00000001B;                                 /* 1. */
   370   3               int_aux_timer0 = 00000001B;                                 /* 1. */
   371   3               int_aux_timer0 = 00000001B;                                 /* 1. */
   372   3               TH0            = TH0 + high (timer0_preset + cycles_stopped); /* 3. */
   373   3             END;                                                          /* 0  */
   374   2             TR0 = timer_start;                                            /* 1+ */
                  /*
Test to see if a scheduled timer interrupt will be lost:
The timer 0 CY signal is deprived if the "+" operation on the TH0 gives
CY equal to 1, eventually causing a timer interrupt to be lost.
                  */
   375   2             IF CY = 1 THEN
   376   3             DO;
   377   3               IF IP <> 0 THEN IP = 0; /* see proc. interrupt_init */
   379   3               ELSE
                         IF not debug THEN call_no_return /* See note 27. */
   380   3                 exception_handler (error_timer_interrupt_lost);
   381   3             END;
```

```
                 $  eject
                 /*
Count milliseconds to max. 255:
                 */
  382   2             milliseconds = milliseconds + 2;
  383   2             IF CY = 1 THEN milliseconds = 255;
                 /*
Count milliseconds_512 to max. 255 (130 seconds 560 milliseconds):
milliseconds_2 overflows and never stops.
                 */
  385   2             int_aux = 1;
  386   2             milliseconds_2 = milliseconds_2 + int_aux;
  387   2             IF CY = 1 THEN
  388   3             DO;
  389   3               milliseconds_512 = milliseconds_512 + int_aux;
  390   3               IF CY = 1 THEN milliseconds_512 = 255;
  392   3             END;
                 /*
Count all the delay_counter that are greater than zero:
Here the counter BASED int_aux is used, int_aux is not use alone any more:
                 */
  393   2             int_aux = .delay_counter (0);
  394   3             DO WHILE (int_aux < .delay_counter (processes_no_of));
  395   3               IF counter > 0 THEN
  396   4               DO;
  397   4                 counter = counter - 1;
                 /*
Make note of timeout to process:
                 */
  398   4                 IF counter = 0 THEN
  399   5                 DO;
  400   5                   IF (process_timeout and int_aux_timer0) = 0 THEN
  401   6                   DO;
  402   6                     process_timeout = process_timeout or int_aux_timer0;

                 /*
Timeout to running is an error:
                 */
  403   6                     IF int_aux_timer0 = running_mask THEN
  404   7                     DO;
  405   7                       call_no_return exception_handler (error_timeout_to_running);
  406   7                     END;
  407   6                   END;
                 /*
Not used previous timeout is an error:
                 */
  408   5                   ELSE call_no_return exception_handler (error_timeout_overflow);
  409   5                 END;
  410   4               END;
  411   3               int_aux_timer0 = shl (int_aux_timer0, 1);
  412   3               int_aux = int_aux + 1;
  413   3             END;
```

```
                   $  eject
                   /*
Test to see if a single process or idle is running continuously for 512 mS:
                   */
   414   2              IF running_last = running THEN
   415   3              DO;
   416   3                running_last_counter = running_last_counter + 1;
   417   3                IF running_last_counter = 255 THEN
   418   4                DO;
   419   4                  call_no_return exception_handler (error_single_process_running);
   420   4                END;
   421   3              END;
                   /*
Test to see if IDLE has not been allowed to run at least once within the last 512 mS:
                   */
   422   2              running_not_idle_counter = running_not_idle_counter + 1;
   423   2              IF running_not_idle_counter = 255 THEN
   424   3              DO;
   425   3                call_no_return exception_handler (error_idle_never_running);
   426   3              END;
                   /*
Switch to rx if timeout of any delay_counter:
                   */
   427   2              IF process_timeout <> 0 THEN
   428   3              DO;
   429   3                int_to_process = int_time;
   430   3                call_no_return rx (input_rx_int_received);
   431   3              END;
                   /*
Return to running process or idle if no timeout:
                   */
   432   2              process_and_rx_R0 = copy_of_R0_in_process;
   433   2              ACC               = copy_of_ACC_in_process;
   434   2              PSW               = copy_of_PSW_in_process;
   435   2              call_no_return interrupt_return;
   436   1            END interrupt_timer_0;
```

```
                 $  eject
                 /*
interrupt_external_0: (vector at address 03H). Priority is highest.
                 */
  437   2           interrupt_external_0: PROCEDURE PUBLIC;

  438   2             call_no_return exception_handler (error_interrupt_ext_0);

  439   1           END interrupt_external_0;
                 /*
interrupt_external_1: (vector at address 13H)
                 */
  440   2           interrupt_external_1: PROCEDURE PUBLIC;

  441   2             call_no_return exception_handler (error_interrupt_ext_1);

  442   1           END interrupt_external_1;
                 /*
interrupt_timer_1: (vector at address 1BH)
                 */
  443   2           interrupt_timer_1: PROCEDURE PUBLIC;

  444   2             call_no_return exception_handler (error_interrupt_timer_1);

  445   1           END interrupt_timer_1;
```

```
                $  eject
                /*
interrupt_serial_channel: (vector at address 23H)
IF ps = 1 THEN top priority, ELSE priority after IE0, TF0, IE1, TF1.
                */
 446   2           interrupt_serial_channel: PROCEDURE PUBLIC;

 447   2             copy_of_R0_in_process  = process_and_rx_R0;
 448   2             copy_of_ACC_in_process = ACC;
 449   2             copy_of_PSW_in_process = PSW; /* See note 26. */

                     /* call wait (250), see file prc0-nnn.plm: */
 450   2             IF TI THEN delay_counter (0) = 126;

                     /* Do observe that Assp_interrupt cannot use R1-R7. */
 452   2             IF Assp_interrupt = assp_to_rx THEN
 453   3             DO;
 454   3               int_to_process = int_to_process0;
 455   3               int_mask       = int_to_process0_mask;
 456   3               ES             = 0; /* process0 may not immediately run */
 457   3               call_no_return rx (input_rx_int_received);
 458   3             END;
 459   2             ELSE
                     DO;
 460   3               process_and_rx_R0 = copy_of_R0_in_process;
 461   3               ACC               = copy_of_ACC_in_process;
 462   3               PSW               = copy_of_PSW_in_process;
 463   3               call_no_return interrupt_return;
 464   3             END;

 465   1           END interrupt_serial_channel;
                /*
interrupt_timer_2: (vector at address 2BH)
(Used as baud rate generator for serial channel).
                */
 466   2           interrupt_timer_2: PROCEDURE PUBLIC;

 467   2             call_no_return exception_handler (error_interrupt_timer_2);
 468   2           END;
```

```
                $  eject
                /*
ram_error:
                */
   469   2         ram_error: PROCEDURE;
   470   2           process_and_rx_R0 = ramsize - 1;
                     /* zero ram first, because exception_handler uses publics. */
   471   3           DO WHILE process_and_rx_R0 <> 0;
   472   3             ram = 0;
   473   3             process_and_rx_R0 = process_and_rx_R0 - 1;
   474   3           END;
   475   2           call_no_return exception_handler (error_internal_ram);
   476   1         END ram_error;
                /*
SYSTEM level, run at reset:
Elapsed times:
   000 mS : +5V power is supplied to the processor and ALE goes high.
   500 mS : The watchdog is triggered because ALE goes from
          : high (same as IDLE) to continuos pulses.
          : This is just now. PSW is set to zero, SP has been initialized
          : to a value given by the linker, then a jump to this point.
          : IDLE is NOT entered until:
  1200 mS : This system level is finished, all procesess are initialized,
          : it will last 700 mS. The watchdog is then often triggered because the
          : IDLE state is entered often.

Test internal ram 256 bytes (MAIN and IDATA):
All bits are tested for zero and one:
                */
   477   1         process_R1 = 1;
   478   2         DO WHILE process_R1 <> 0; /* 1,2,4,8,16,32,64,128, (256=0) */
                     /* write to complete ram: */
   479   2           process_and_rx_R0 = ramsize - 1;
   480   3           DO WHILE process_and_rx_R0 > 1;
   481   3             ram = process_and_rx_R0 - process_R1;
   482   3             process_and_rx_R0 = process_and_rx_R0 - 1;
   483   3           END;
                     /* read from complete ram: */
   484   2           process_and_rx_R0 = ramsize - 1;
   485   3           DO WHILE process_and_rx_R0 > 1;
   486   3             IF (ram + process_R1) <> process_and_rx_R0 THEN call_no_return ram_error;
   488   3             process_and_rx_R0 = process_and_rx_R0 - 1;
   489   3           END;
   490   2           process_R1 = process_R1 + process_R1;
   491   2         END;
                /*
Zero internal ram 256 bytes (MAIN and IDATA):
                */
   492   1         process_and_rx_R0 = ramsize - 1;
   493   2         DO WHILE process_and_rx_R0 <> 0;
   494   2           ram = 0;
   495   2           process_and_rx_R0 = process_and_rx_R0 - 1;
   496   2         END;
```

```
                   $  eject
                   /*
Init and test:
Observe that in rx51_init at least one of the delay_counter (n) must be
set to 1, to make a switch to RX as fast as possible. (= on the first
timer interrupt).
                   */
                      call test_exram;
                      call interrupt_init;
                      call rx51_init;
                      call time_unit_init;
                   /*
Stack_pointer_test_value:
The complete stack is used by the system, even if a return to this procedure
is never to be done after the first timeout.
(A complete stack is the ram ABOVE *SP, not including *SP).
                   */
                      stack_pointer_test_value = SP + 2;
                   /*
If a single_step with the MICE-II is to be done, stop it on the line above,
and set the "%debug" bit to 1 here. Unless, the "error_timer_interrupt_lost"
will be encountered. (rbit %debug 1)

?stack
This is an idata segment allocated by the PL/M-51 that has a length of 1 byte.
If the value is, say 0C1H, that address is NEVER used at all, since the stack
will reside ABOVE this address. Therefore the SP is decremented by 1 to
make it used, making another 1 byte of ram available.
                   */
                      SP = SP - 1;
                      stack_pointer_test_value = stack_pointer_test_value - 1;
                   /*
Wait for timer 0 interrupt:
Tune "to-limit": breakpoint timer 0 interrupt and display process_R7,
then process_R7 is 181.
                   */
                      enable;
                      DO process_R7 = 0 to 182;
                      END;

                      disable;
                      call_no_return exception_handler (error_no_timebase);

                   END rx51__;
```

```
SYMBOLS LISTING
---------------


    DEFN  SPACE  SIZE  NAME                          ATTRIBUTES
    ----  -----  ----  ----                          ----------


      27  IDATA     1  ACC . . . . . . . . . . . . . BYTE MEMBER OF PROCESS_STACK ; OFFSET=1
      28               ACC . . . . . . . . . . . . . BYTE REGISTER AT(E0H)
     136  IDATA     1  ACC . . . . . . . . . . . . . BYTE MEMBER OF PROCESS_STACK_ELEMENT ; OFFSET=1
      32               AMD_87C541_WATCHDOG_KEY . . . BYTE REGISTER AT(AAH)
       2  CODE         ASSP_INTERRUPT. . . . . . . . PROCEDURE BIT EXTERNAL USING(0)
      21               ASSP_TO_RX. . . . . . . . . . LITERALLY
      27  IDATA     1  B . . . . . . . . . . . . . . BYTE MEMBER OF PROCESS_STACK ; OFFSET=0
      28               B . . . . . . . . . . . . . . BYTE REGISTER AT(F0H)
     136  IDATA     1  B . . . . . . . . . . . . . . BYTE MEMBER OF PROCESS_STACK_ELEMENT ; OFFSET=0
      27  CODE      8  BIT_MASK. . . . . . . . . . . BYTE ARRAY(8)
      21               BOX_0 . . . . . . . . . . . . LITERALLY
      24               CALL_NO_RETURN. . . . . . . . LITERALLY
          DATA      1  CLASS_REASON. . . . . . . . . BYTE PARAMETER
      27  DATA      1  COPY_OF_ACC_IN_PROCESS. . . . BYTE
     135  BIT       1  COPY_OF_CY. . . . . . . . . . BIT
      27  DATA      1  COPY_OF_PSW_IN_PROCESS. . . . BYTE
      27  DATA      1  COPY_OF_R0_IN_PROCESS . . . . BYTE
     355  IDATA     1  COUNTER . . . . . . . . . . . BYTE BASED(INT_AUX)
      29               CY. . . . . . . . . . . . . . BIT REGISTER AT(D7H)
      24               CYCLES_STOPPED. . . . . . . . LITERALLY
      19  BIT       1  DEBUG . . . . . . . . . . . . BIT EXTERNAL
      27  IDATA     4  DELAY_COUNTER . . . . . . . . BYTE ARRAY(4)
      27  DATA      1  DOT_IDLE_HIGH . . . . . . . . BYTE AT(.RUNNING)
      27  DATA      1  DOT_IDLE_LOW. . . . . . . . . BYTE AT(.RUNNING_MASK)
      27  IDATA     1  DPH . . . . . . . . . . . . . BYTE MEMBER OF PROCESS_STACK ; OFFSET=6
      28               DPH . . . . . . . . . . . . . BYTE REGISTER AT(83H)
     136  IDATA     1  DPH . . . . . . . . . . . . . BYTE MEMBER OF PROCESS_STACK_ELEMENT ; OFFSET=6
      27  IDATA     1  DPL . . . . . . . . . . . . . BYTE MEMBER OF PROCESS_STACK ; OFFSET=7
      28               DPL . . . . . . . . . . . . . BYTE REGISTER AT(82H)
     136  IDATA     1  DPL . . . . . . . . . . . . . BYTE MEMBER OF PROCESS_STACK_ELEMENT ; OFFSET=7
      28               EA. . . . . . . . . . . . . . BIT REGISTER AT(AFH)
      25               ERROR_CALL_TO_RX_LEVEL. . . . LITERALLY
      25               ERROR_DISABLE_MISSING . . . . LITERALLY
      25               ERROR_IDLE_NEVER_RUNNING. . . LITERALLY
      25               ERROR_INTERNAL_RAM. . . . . . LITERALLY
      25               ERROR_INTERRUPT_EXT_0 . . . . LITERALLY
      25               ERROR_INTERRUPT_EXT_1 . . . . LITERALLY
      25               ERROR_INTERRUPT_TIMER_1 . . . LITERALLY
      25               ERROR_INTERRUPT_TIMER_2 . . . LITERALLY
      25               ERROR_INTERRUPT_TO_RUNNING. . LITERALLY
      25               ERROR_NO_MAILBOX. . . . . . . LITERALLY
      25               ERROR_NO_TIMEBASE . . . . . . LITERALLY
      25               ERROR_PROCESS_REG_BANK. . . . LITERALLY
      25               ERROR_RETURN_TO_IDLE. . . . . LITERALLY
      25               ERROR_SINGLE_PROCESS_RUNNING. LITERALLY
      25               ERROR_STACK_OVERFLOW. . . . . LITERALLY
      25               ERROR_TIMEOUT_OVERFLOW. . . . LITERALLY
      25               ERROR_TIMEOUT_TO_RUNNING. . . LITERALLY
      25               ERROR_TIMER_INTERRUPT_LOST. . LITERALLY
```

```
   30                ES. . . . . . . . . . . . . . . BIT REGISTER AT(ACH)
   30                ET0 . . . . . . . . . . . . . . BIT REGISTER AT(A9H)
    4  CODE          EXCEPTION_HANDLER . . . . . . . PROCEDURE EXTERNAL USING(0)
   24                FALSE . . . . . . . . . . . . . LITERALLY
                     HIGH. . . . . . . . . . . . . . BUILTIN
   55  CODE      21  IDLE. . . . . . . . . . . . . . PROCEDURE USING(0) STACK=02H
   24                IDLE_IS_RUNNING . . . . . . . . LITERALLY
   24                IDLE_STACK_FIX. . . . . . . . . LITERALLY
   24                IDLE_STOPPED. . . . . . . . . . LITERALLY
   27  BIT        1  IDLE_SYSTEM . . . . . . . . . . BIT
   24                IDLE_TO_START . . . . . . . . . LITERALLY
   28                IE. . . . . . . . . . . . . . . BYTE REGISTER AT(A8H)
   24                INPUT_RX_INT_RECEIVED . . . . . LITERALLY
   24                INPUT_RX_ROUND_ROBIN. . . . . . LITERALLY
  437  CODE       7  INTERRUPT_EXTERNAL_0. . . . . . PROCEDURE PUBLIC USING(0) STACK=02H
  440  CODE       7  INTERRUPT_EXTERNAL_1. . . . . . PROCEDURE PUBLIC USING(0) STACK=02H
   65  CODE       7  INTERRUPT_INIT. . . . . . . . . PROCEDURE USING(0) STACK=02H
    7  CODE          INTERRUPT_RETURN. . . . . . . . PROCEDURE EXTERNAL USING(0)
  446  CODE      50  INTERRUPT_SERIAL_CHANNEL. . . . PROCEDURE PUBLIC USING(0) STACK=03H
  354  CODE     240  INTERRUPT_TIMER_0 . . . . . . . PROCEDURE PUBLIC USING(0) STACK=02H
  443  CODE       7  INTERRUPT_TIMER_1 . . . . . . . PROCEDURE PUBLIC USING(0) STACK=02H
  466  CODE       7  INTERRUPT_TIMER_2 . . . . . . . PROCEDURE PUBLIC USING(0) STACK=02H
   26  DATA       1  INT_AUX . . . . . . . . . . . . BYTE AT(.SHARE_RAM_A)
   26  DATA       1  INT_AUX_TIMER0. . . . . . . . . BYTE AT(.SHARE_RAM_B)
   26  DATA       1  INT_MASK. . . . . . . . . . . . BYTE AT(.SHARE_RAM_B)
   24                INT_TIME. . . . . . . . . . . . LITERALLY
   26  DATA       1  INT_TO_PROCESS. . . . . . . . . BYTE AT(.SHARE_RAM_A)
   24                INT_TO_PROCESS0 . . . . . . . . LITERALLY
   24                INT_TO_PROCESS0_MASK. . . . . . LITERALLY
   24                INT_TO_PROCESS1 . . . . . . . . LITERALLY
   24                INT_TO_PROCESS1_MASK. . . . . . LITERALLY
   24                INT_TO_PROCESS2 . . . . . . . . LITERALLY
   24                INT_TO_PROCESS2_MASK. . . . . . LITERALLY
   24                INT_TO_PROCESS3 . . . . . . . . LITERALLY
   24                INT_TO_PROCESS3_MASK. . . . . . LITERALLY
   28                IP. . . . . . . . . . . . . . . BYTE REGISTER AT(B8H)
                     LOW . . . . . . . . . . . . . . BUILTIN
       DATA       1  MAIL. . . . . . . . . . . . . . BYTE PARAMETER
   27  IDATA      1  MAILBOX . . . . . . . . . . . . BYTE ARRAY(1)
   22                MAILBOXES_NO_OF . . . . . . . . LITERALLY
  111  CODE      55  MAILBOX_EMPTIED . . . . . . . . PROCEDURE BIT PUBLIC USING(0) STACK=02H
   95  CODE      59  MAILBOX_FILLED. . . . . . . . . PROCEDURE BIT PUBLIC USING(0) STACK=02H
       DATA       1  MAILBOX_NUM . . . . . . . . . . BYTE PARAMETER
       DATA       1  MAILBOX_NUM . . . . . . . . . . BYTE PARAMETER
       DATA       1  MAILBOX_NUM . . . . . . . . . . BYTE PARAMETER
   96  DATA       1  MAILBOX_OWNED_BY. . . . . . . . BYTE
   27  IDATA      1  MAILBOX_OWNER . . . . . . . . . BYTE ARRAY(1)
   82  CODE      42  MAILBOX_RESERVED. . . . . . . . PROCEDURE BIT PUBLIC USING(0) STACK=02H
       DATA       1  MAIL_PTR. . . . . . . . . . . . BYTE PARAMETER
  112  IDATA      1  MAIL_RETURN . . . . . . . . . . BYTE BASED(MAIL_PTR)
   96  DATA       1  MASK. . . . . . . . . . . . . . BYTE
  127  DATA       1  MASK. . . . . . . . . . . . . . BYTE
   19  DATA       1  MILLISECONDS. . . . . . . . . . BYTE EXTERNAL
   19  DATA       1  MILLISECONDS_2. . . . . . . . . BYTE EXTERNAL
   19  DATA       1  MILLISECONDS_512. . . . . . . . BYTE EXTERNAL
   21                NEXT. . . . . . . . . . . . . . LITERALLY
       BIT        1  NEXT_PROCESS_SWITCH . . . . . . BIT PARAMETER
  21                NOT_TIMEOUT . . . . . . . . . . LITERALLY
  24                NO_OWNER. . . . . . . . . . . . LITERALLY
 136  IDATA      1  PCH . . . . . . . . . . . . . . BYTE MEMBER OF PROCESS_STACK_ELEMENT ; OFFSET=8
 136  IDATA      1  PCL . . . . . . . . . . . . . . BYTE MEMBER OF PROCESS_STACK_ELEMENT ; OFFSET=9
  28                PCON. . . . . . . . . . . . . . BYTE REGISTER AT(87H)
  27  IDATA      2  PC_HL . . . . . . . . . . . . . WORD MEMBER OF PROCESS_STACK ; OFFSET=8
  22                PROCESSES_NO_OF . . . . . . . . LITERALLY
  11  CODE          PROCESS_0 . . . . . . . . . . . PROCEDURE EXTERNAL USING(0)
  13  CODE          PROCESS_1 . . . . . . . . . . . PROCEDURE EXTERNAL USING(0)
  15  CODE          PROCESS_2 . . . . . . . . . . . PROCEDURE EXTERNAL USING(0)
  17  CODE          PROCESS_3 . . . . . . . . . . . PROCEDURE EXTERNAL USING(0)
  26  DATA       1  PROCESS_AND_RX_R0 . . . . . . . BYTE AT(0000H)
 136  DATA       1  PROCESS_B . . . . . . . . . . . BYTE
  27  DATA       1  PROCESS_CONTINUES_LATER . . . . BYTE
  27  DATA       1  PROCESS_INT_RECEIVED. . . . . . BYTE
  27  DATA       1  PROCESS_MAIL_PRESENT. . . . . . BYTE
  26  DATA       1  PROCESS_R1. . . . . . . . . . . BYTE AT(0001H)
  26  DATA       1  PROCESS_R6. . . . . . . . . . . BYTE AT(0006H)
  26  DATA       1  PROCESS_R7. . . . . . . . . . . BYTE AT(0007H)
  27  IDATA     44  PROCESS_STACK . . . . . . . . . STRUCTURE ARRAY(4)
 136  IDATA     11  PROCESS_STACK_ELEMENT . . . . . STRUCTURE BASED(PROCESS_STACK_POINTER)
 136  DATA       1  PROCESS_STACK_POINTER . . . . . BYTE
  27  DATA       1  PROCESS_TIMEOUT . . . . . . . . BYTE
  23  CODE       4  PROTECTION_CODE . . . . . . . . BYTE ARRAY(4)
  27  IDATA      1  PSW . . . . . . . . . . . . . . BYTE MEMBER OF PROCESS_STACK ; OFFSET=2
  28                PSW . . . . . . . . . . . . . . BYTE REGISTER AT(D0H)
 136  IDATA      1  PSW . . . . . . . . . . . . . . BYTE MEMBER OF PROCESS_STACK_ELEMENT ; OFFSET=2
  27  IDATA      1  R0. . . . . . . . . . . . . . . BYTE MEMBER OF PROCESS_STACK ; OFFSET=10
 136  IDATA      1  R0. . . . . . . . . . . . . . . BYTE MEMBER OF PROCESS_STACK_ELEMENT ; OFFSET=10
  27  IDATA      1  R1. . . . . . . . . . . . . . . BYTE MEMBER OF PROCESS_STACK ; OFFSET=3
 136  IDATA      1  R1. . . . . . . . . . . . . . . BYTE MEMBER OF PROCESS_STACK_ELEMENT ; OFFSET=3
  27  IDATA      1  R6. . . . . . . . . . . . . . . BYTE MEMBER OF PROCESS_STACK ; OFFSET=4
 136  IDATA      1  R6. . . . . . . . . . . . . . . BYTE MEMBER OF PROCESS_STACK_ELEMENT ; OFFSET=4
  27  IDATA      1  R7. . . . . . . . . . . . . . . BYTE MEMBER OF PROCESS_STACK ; OFFSET=5
 136  IDATA      1  R7. . . . . . . . . . . . . . . BYTE MEMBER OF PROCESS_STACK_ELEMENT ; OFFSET=5
  27  IDATA      1  RAM . . . . . . . . . . . . . . BYTE BASED(PROCESS_AND_RX_R0)
  22                RAMSIZE . . . . . . . . . . . . LITERALLY
 469  CODE      22  RAM_ERROR . . . . . . . . . . . PROCEDURE USING(0) STACK=02H
 135  BIT        1  RETURN_FROM_RX. . . . . . . . . BIT
  27  DATA       1  RUNNING . . . . . . . . . . . . BYTE
  27  DATA       1  RUNNING_LAST. . . . . . . . . . BYTE
  27  DATA       1  RUNNING_LAST_COUNTER. . . . . . BYTE
  27  DATA       1  RUNNING_MASK. . . . . . . . . . BYTE
  27  DATA       1  RUNNING_NOT_IDLE_COUNTER. . . . BYTE
  27  BIT        1  RUNNING_SEARCH. . . . . . . . . BIT
  24                RUNNING_SEARCH_CONTINUES. . . . LITERALLY
  24                RUNNING_SEARCH_FINISHED . . . . LITERALLY
 134  CODE     662  RX. . . . . . . . . . . . . . . PROCEDURE PUBLIC USING(0) STACK=02H
  33  CODE      55  RX51_INIT . . . . . . . . . . . PROCEDURE USING(0) STACK=02H
   1  CODE     124  RX51__. . . . . . . . . . . . . MODULE
 126  CODE      30  RX_INT_REMOVE_AT_NEW_RUNNING. . PROCEDURE USING(0) STACK=02H
  26  DATA       1  SHARE_RAM_A . . . . . . . . . . BYTE
  26  DATA       1  SHARE_RAM_B . . . . . . . . . . BYTE
                    SHL . . . . . . . . . . . . . . BUILTIN
                    SHR . . . . . . . . . . . . . . BUILTIN
  28                SP. . . . . . . . . . . . . . . BYTE REGISTER AT(81H)
 136  IDATA      1  STACK_DATA. . . . . . . . . . . BYTE BASED(STACK_POINTER)
```

```
      136  IDATA     2  STACK_DATA_DOT_IDLE . . . . . WORD BASED(STACK_POINTER)
      136  IDATA     1  STACK_DATA_INDIRECT . . . . . BYTE BASED(PROCESS_STACK_POINTER)
      136  DATA      1  STACK_POINTER . . . . . . . . BYTE
       19  DATA      1  STACK_POINTER_TEST_VALUE. . . BYTE EXTERNAL
        9  CODE         TEST_EXRAM. . . . . . . . . . PROCEDURE EXTERNAL USING(0)
       31               TF0 . . . . . . . . . . . . . BIT REGISTER AT(8DH)
       28               TH0 . . . . . . . . . . . . . BYTE REGISTER AT(8CH)
       20               TI. . . . . . . . . . . . . . BIT REGISTER AT(99H)
       21               TIMEOUT . . . . . . . . . . . LITERALLY
       22               TIMER0_PRESET . . . . . . . . LITERALLY
       70  DATA      1  TIMER0_TICS . . . . . . . . . BYTE
       24               TIMER_START . . . . . . . . . LITERALLY
       24               TIMER_STOP. . . . . . . . . . LITERALLY
       20               TIME_BIT. . . . . . . . . . . BIT REGISTER AT(D5H)
       46  CODE     25  TIME_UNIT_INIT. . . . . . . . PROCEDURE USING(0) STACK=02H
       28               TL0 . . . . . . . . . . . . . BYTE REGISTER AT(8AH)
       28               TMOD. . . . . . . . . . . . . BYTE REGISTER AT(89H)
       31               TR0 . . . . . . . . . . . . . BIT REGISTER AT(8CH)
       24               TRUE. . . . . . . . . . . . . LITERALLY
       69  CODE     44  WAIT. . . . . . . . . . . . . PROCEDURE PUBLIC USING(0) STACK=02H
           DATA      1  WAIT_MILLISECONDS . . . . . . BYTE PARAMETER



MODULE INFORMATION:                         (STATIC+OVERLAYABLE)
    CODE SIZE                               = 05B8H       1464D
    CONSTANT SIZE                           = 000CH         12D
    DIRECT VARIABLE SIZE                    =   0EH+04H    14D+  4D
    INDIRECT VARIABLE SIZE                  =   32H+00H    50D+  0D
    BIT SIZE                                =   02H+03H     2D+  3D
    BIT-ADDRESSABLE SIZE                    =   00H+00H     0D+  0D
    AUXILIARY VARIABLE SIZE                 = 0000H          0D
    MAXIMUM STACK SIZE                      = 000BH         11D
    REGISTER-BANK(S) USED:                    0
    1580 LINES READ
    0 PROGRAM ERROR(S)
END OF PL/M-51 COMPILATION
```