```
*****************************************************************************************************************
*                                                                                                               *
* FILE: @DMACCO-FILE*SSD*MD*USERS.D*KB.D*LOADER.D*VS0601.D*BOOTSTRAP.P                                          *
*                                                                                                               *
* DATE: 82:12:03                                                                                                *
*                                                                                                               *
* TIME: 10:38:09                                                                                                *
*                                                                                                               *
*****************************************************************************************************************
*                                                                                                               *
*                                                                                                               *
*                                                                                                               *
* BBBB     000     000   TTTTT   SSS   TTTTT  RRRR    AAA   PPPP          PPPP                                 *
* B   B   0   0   C   0     T    S   S    T    R   R  A   A  P   P         P   P                                *
* B   B   0   C   C   0     T    S        T    R   R  A   A  P   P         P   P                                *
* BBBB    0   C   C   C     T     SSS     T    RRRR   A   A  PPPP          PPPP                                 *
* B   B   0   C   C   0     T        S    T    R R    AAAAA  P             P                                    *
* B   B   0   C   C   0     T    S   S    T    R  R   A   A  P             P                                    *
* BBBB     000     000      T     SSS     T    R   R  A   A  P             P                                    *
*                                                                                                               *
*                                                                                                               *
*                                                                                                               *
*  SSS    Y   Y    SSS                                                                                          *
* S   S   Y   Y   S   S                                                                                         *
* S        Y Y    S                                                                                             *
*  SSS      Y      SSS                                                                                          *
*     S     Y         S                                                                                        *
* S   S     Y    S    S                                                                                        *
*  SSS      Y     SSS                                                                                          *
*                                                                                                               *
*                                                                                                               *
*                                                                                                               *
*  888     222          1     222         000   33333                  1     000         33333   888   *
* 8   8   2   2        11    2   2       0   0      3                11    0   0   ::        3   8   8  *
* 8   8       2        1       2        0   0     3                  1    0   0   ::       3    8   8  *
*  888       2         1      2         0 0 0    33                  1    0 0 0           33     888   *
* 8   8     2          1     2          0   0     3                  1    0   0  .::       3    8   8  *
* 8   8    2           1     2          0   0   3   3                1    0   0   ::    3   3   8   8  *
*  888    22222   ..   111   22222   ..   000    333                111    000         333     888   *
*                                                                                                               *
*                                                                                                               *
*                                                                                                               *
*****************************************************************************************************************
```

```
CR80 ASSEMBLER VERSION 800114

ASSEMBLY OF FILE:  BOOTSTRAP.S      AT  82:10:20 16:48:02
```

```
C0000001 0 0000 LIST                                                                ;
C0000002 0 0CC0 BEGIN MODULE BMPM                                                   ;
C0000003 0 00C0 AREASWITCH= 1;                                                      ;
C0000004 0 0CC0 USE BASE                                                            ;
C0000005 0 00C0 ABASE=  #FFE0          ;  ABS BASE OF THIS PROCESS                  ;
C0000006 0 0CC0 FIRST=  #F800          ;  ABSOLUTE START OF PROM LOADER             ;
C0000007 0 0CC0 EBASE=  #20            ;  ABS BASE OF LOADER WHEN EMERGENCY;
C0000008 0 0CC0 HDRSZ=  26             ;  SIZE OF DATA HEADER                       ;
C0000009 0 0CC0 SIZE=   1              ;  OFFSET TO SIZE FIELD IN HEADER            ;
C0000010 0 0CC0 START=  7              ;  OFFSET TO START ADDRESS IN HEADER;
C0000011 0 00C0 XBASE=  8              ;                                            ;
C0000012 0 0CC0 XPROG=  10             ;                                            ;
C0000013 0 00C0 XPRPC=  11             ;                                            ;
C0000014 0 0CC0 XPSW=   13             ;                                            ;
C0000015 0 0CC0 LOC= LOC-2                                                          ;
C0000016 0 FFFE ; LOADER PROCESS DESCRIPTOR                                         ;
C0000017 0 FFFE 1                      ;  LEVEL                                     ;
C0000018 0 FFFF -1                     ;  BOUND                                     ;
C0000019 0 00C0 RPRPC                  ;  REG 0  USED BY ROOT AS PRPC               ;
         0 00C1 0                      ;  REG 1  (0,1,2) CAUSE OF ACTIVATIO;
C0000020 0 00C2 0                      ;  REG 2                                     ;
C0000021 0 00C3 0                      ;  REG 3                                     ;
C0000022 0 00C4 0                      ;  REG 4                                     ;
C0000023 0 00C5 FIRST                  ;  REG 5  1 + LAST COPY LOCATION             ;
C0000024 0 00C6 0                      ;  REG 6                                     ;
C0000025 0 00C7 0                      ;  REG 7                                     ;
C0000026 0 00C8 ABASE                  ;  BASE                                      ;
C0000027 0 00C9 ABASE                  ;  MOD                                       ;
C0000028 0 00CA APROG                  ;  PRG                                       ;
C0000029 0 00CB APRPC                  ;  PRPC                                      ;
C0000030 0 00CC 0                      ;  TIMER                                     ;
C0000031 0 00CD #E000                  ;  PSW                                       ;
C0000032 0 00CE 0                      ;  LINK BASE                                 ;
C0000033 0 00CF 0                      ;  LOCAL ACTION                              ;
C0000034 0 0010 0 REPEAT 13                                                         ;
C0000035 0 001E EMERGENCY              ;  POINTER TO EMERGENCY PROCEDURE            ;
C0000036 0 001F EMERGENCY              ;  POINTER TO EMERGENCY PROCEDURE            ;
C0000037 0 0020 ;                                                                   ;
```

```
C0000038 0 0020 ;PROGRAM PART                                                    ;
C0000039 0 0020 ;                                                                ;
C0000040 0 0020 USE PROG                                                         ;
C0000041 1 0000 ROOT:                                                            ;
C0000042 1 0000         MOVC      1             R1  ;  SET ROOT CAUSE            ;
C0000043 1 0001         MOVC  FIRST             R5  ;  SET 1 + LAST COPY LOCATION ;
C0000044 1 0003         JMP               INIT      ;                            ;
C0000045 1 0004 EMERG:                                                           ;
C0000046 1 0004         MOVC      2             R1  ;  SET EMERGENCY CAUSE       ;
C0000047 1 0005         MOVC  EBASE             R5  ;  SET BASE OF LOADER        ;
C0000048 1 0006 INIT:                                                            ;
C0000049 1 0006         MOV   XBASE             R7  ;  BASE OF PROCESS CAUSING EMERGENCY;
C0000050 1 0007         MOV           R7        R4  ;                            ;
C0000051 1 0008         NEG                     R4  ;  - BASE                    ;
C0000052 1 0009         MOVC  #EC00             R0  ;                            ;
C0000053 1 000B    →    LDS           R0            ;  SWITCH TO SECTION 0       ;
C0000054 1 000C ; AT THIS POINT THE REGISTERS MUST CONTAIN                       ;
C0000055 1 000C ; R1  ACTIVATION CAUSE: 0 = MASTERCLEAR, 1 = ROOT, 2 = EMERGENCY ;
C0000056 1 000C ; R4  - BASE                                                     ;
C0000057 1 000C ; R5  IF EMERGENCY THEN COPY BASE ELSE 1+LAST COPY LOCATION      ;
C0000058 1 000C ; R7 BASE OF PROCESS GENERATING EMERGENCY                        ;
C0000059 1 000C         MOVC  FIRST             R3  ;                            ;
C0000060 1 000E         ADD           R3        R4  ;                            ;
C0000061 1 000F         MOV   START. X4         R2  ;  ENTRY POINT IN BOOT LOADER ;
C0000062 1 0010         ADD   SIZE.  X4         R4  ;  REF TO DATA PART          ;
C0000063 1 0011         MOV   SIZE.  X4         R0  ;  SIZE OF DATA TO BE MOVED  ;
C0000064 1 0012         SNE           R1  2         ;  IF NOT EMERGENCY          ;
C0000065 1 0013         JMP                 COPY    ;  THEN                      ;
C0000066 1 0014         SUB           R0        R5  ;    RESERVE SPACE TO DATA  ;
C0000067 1 0015         ADDC  HDRSZ             R5  ;    LOCATION OF BASE       ;
C0000068 1 0016         SRL           R5  4         ;    PAGE ALLIGNMENT        ;
C0000069 1 0017         SLL           R5  4         ;                           ;
C0000070 1 0018 COPY:                                                            ;
C0000071 1 0018         SUB           R7        R5  ;  ABS -> REL                ;
C0000072 1 0019         MOV           R5        R6  ;  R6 = REL ADDR OF LOADER BASE ;
C0000073 1 001A         ADDC  -HDRSZ            R5  ;  DEST OF LOADER PROCESS    ;
C0000074 1 001B MOVE:                                                            ;
C0000075 1 001B         MOV           X4        X5  ;                            ;
C0000076 1 001C         INCD          R4        R5  ;                            ;
C0000077 1 001D         SOB           R0  MOVE      ;                            ;
C0000078 1 001E ;                                                                ;
C0000079 1 001E ; INSTALL BASE,PROG,PRPC,PSW                                     ;
C0000080 1 001E ;                                                                ;
C0000081 1 001E         MOV           R7        R0  ;                            ;
C0000082 1 001F         ADD           R6        R0  ;  R0 = ABS ADDR OF LOADER BASE ;
C0000083 1 0020         MOV           R0  XBASE. X6 ;                            ;
C0000084 1 0021         MOV           R3  XPROG. X6 ;                            ;
C0000085 1 0022         MOV           R3  XPRPC. X6 ;                            ;
```

```
C0000084 1 0021          MOV            R3   XPRPC. X6  ;                                    ;
C0000085 1 0022          MOV            R3   XPRPC. X6  ;                                    ;


C0000086 1 0023          ADD            R2   XPRPC. X6  ;                                    ;
C0000087 1 0024          MOV            R7   1.     X6  ;  STORE EMERGENCY ACTION BASE IN R1;
C0000088 1 0025          MOV            R1   0.     X6  ;  STORE ACTIVATION CAUSE IN R0      ;
C0000089 1 0026          MOVC  #EC00                R7  ;                                    ;
C0000090 1 0028          MOV            R7   XPSW.  X6  ;                                    ;
C0000091 1 0029          LDP            R0              ;                                    ;
C0000092 1 002A          0 REPEAT 46-LOC                                                     ;
C0000093 1 002E USE BASE                                                                     ;
C0000094 0 0C20 BMPM=    #FFB0                                                               ;
C0000095 0 0020 APROG=   BMPM                                                                ;
C0000096 0 0020 APRPC=   APROG+INIT                                                          ;
C0000097 0 0020 EMERGENCY= APROG+EMERG                                                       ;
C0000098 0 0C20 RPRPC= APROG+ROOT                                                            ;
C0000099 0 0C20 END                                                                          ;
```

LOS RQ FIRST FETCH

```
ZOFFB0 T0000
P
FFB0L 0149 F856 004D 02D8 0249 2C4D 086F 7CBB
FFB8L F4BC E056 C048 E0BC F856 0C4B 348A 071A
FFC0L 0134 0118 927C 04D8 058B 1ACD D424 D4A0
FFC8L 758B 5EBB 1AAD CDFB 458E 03C0 78BB 608A
FFD0L 8880 8A83 8B83 880A 8187 9EBB E056 004F
FFD8L 8D87 00BC C000 0000 0000 0C00 0001 FFFF
FFE0L FFB0 0CC0 C000 0000 0000 F800 0000 0000
FFE8L FFE0 FFE0 FFB0 FFB6 0000 EC00 0000 0000
FFF0L 0000 0CC0 C000 0000 0C00 0C00 0000 0000
FFF8L 0000 0000 C000 0000 0000 0000 FFB4 FFB4
      S0
```

RUN Base = #F6E0
PROC = #F800

MC-CLEAR Descriptor.

```
MEMORY MAP:
  AREA 1  FFB0
  AREA 0  FFDE
  AREA 3  0000
  AREA 2  0000
  AREA 5  0000
  AREA 4  0000
  AREA 7  0000
  AREA 6  0000
  AREA 9  0000
  AREA 8  0000
  AREA B  0000
  AREA A  0000
  AREA D  0000
  AREA C  0000
  AREA F  0000
  AREA E  0000

    4 MODIFIES INSERTED
   80 WORDS OUTPUT
```

```
    80 WORDS ASSEMBLED
ASSEMBLY OK
```

```
FILE: @DMA000-FILE*SSD*MD*USERS.D*KB.D*LOADER.D*VS0601.D*LOADER.P

DATE: 82:12:03

TIME: 10:38:42


 L       OOO     AAA    DDDD    EEEEE  RRRR           PPPP
 L      O   C   A   A   D   D   E      R   R          P   P
 L      O   C   A   A   D   D   E      R   R          P   P
 L      O   C   A   A   D   D   EEEE   RRRR           PPPP
 L      O   C   AAAAA   D   D   E      R R            P
 L      O   C   A   A   D   D   E      R  R    ..     P
 LLLLL   OOO    A   A   DDDD    EEEEE  R   R   ..     P


  SSS    Y   Y    SSS
 S   S   Y   Y   S   S
 S        Y Y    S
  SSS      Y      SSS
      S    Y         S
 S    S    Y    S    S
  SSS      Y     SSS


  888     222            1     222           000    33333                1     000          33333    888
 8   8   2   2          11    2   2         0   0       3              11    0   0   ::         3   8   8
 8   8      2            1        2         0   0     3                 1    0   0   ::        3    8   8
  888      2            1       2          0 0 0    33                 1    0 0 0            33     888
 8   8    2             1      2           0   0      3                1    0   0   ::         3   8   8
 8   8   2         ..    1     2     ..    0   0  3   3               1    0   0   ::     3   3   8   8
  888    22222     ..   111    22222   ..     000    333             111    000            333    888
```

```
FILE: @DMACCO-FILE*SSD*MD*USERS.D*KB.D*LOADER.D*VS0601.D*LOADER.P

DATE: 82:12:03

TIME: 10:38:46
```

```
 L        OOO     AAA    DDDD    EEEEE  RRRR           PPPP
 L       O   O   A   A   D   D   E      R   R          P   P
 L       O   O   A   A   D   D   E      R   R          P   P
 L       O   O   A   A   D   D   EEEE   RRRR           PPPP
 L       O   O   AAAAA   D   D   E      R R            P
 L       O   O   A   A   D   D   E      R  R     ..    P
 LLLLL    OOO    A   A   DDDD    EEEEE  R   R    ..    P


  SSS    Y   Y    SSS
 S   S   Y   Y   S   S
 S        Y Y    S
  SSS      Y      SSS
     S     Y         S
 S   S     Y     S   S
  SSS      Y      SSS


  888     222            1     222           000    33333                  1     000          33333    888
 8   8   2   2          11    2   2         0   0       3                11    0   0   ::         3   8   8
 8   8       2           1         2        0   0      3                  1    0   0   ::        3    8   8
  888       2            1        2         0 0 0     33                 1    0 0 0            33     888
 8   8     2             1       2          0   0       3                1    0   0   ::         3   8   8
 8   8   2         ..    1     2      ..    0   0   3   3                1    0   0   ::     3   3   8   8
  888    22222     ..   111    22222  ..      000     333                111    000           333    888
```

```
======== LOADER.S

C0.00001 %PRINT
C0.00002 %LIST
C0.00003 %SOURCE

======== HEADER.S

01.00001 "-----------------------------------------------------------------
01.00002 "
01.00003 "  PROJECT:                X
01.00004 "
01.00005 "  MODULE NAME:            LOADER
01.00006 "  MODULE ID NMB:          CSS/123
01.00007 "  MODULE VERSION:         06.01
01.00008 "  MODULE TYPE:            PROM
01.00009 "  MODULE FILES:           NONE
01.00010 "  MERGE FILES:            @**GENS.D*SWELLPREFIX.D*GENERALPARAMS.S
01.00011 "                          @**GENS.D*SWELLPREFIX.D*FMSPARAMS.S
01.00012 "                          @**GENS.D*SWELLPREFIX.D*VOL_STRUCTURES.S
01.00013 "                          @**GENS.D*SWELLPREFIX.D*X2GENPARAMS.S
01.00014 "
01.00015 "  SPECIFICATIONS:         CSS/123/USM/0056
01.00016 "  AUTHOR/DATE:            KB/810801
01.00017 "
01.00018 "  DELIVERABLE:            YES
01.00019 "  SOURCE LANGUAGE:        SWELL
01.00020 "  COMPILE COMPUTER:       CR80
01.00021 "  TARGET COMPUTER:        CR80
01.00022 "  OPERATING SYSTEM:       AMOS/XAMOS
01.00023 "
01.00024 "-----------------------------------------------------------------
01.00025 "
01.00026 "  CHANGE RECORD
01.00027 "
01.00028 "  VERSION    AUTHOR/DATE   DESCRIPTION OF CHANGE
01.00029 "  -------    -----------   --------------------
```

```
01.00030 "  04.01      KB/810801     THE LOADER IS ABLE TO COPE
01.00031 "                           WITH TIME ERROR ON CMD DISKS.
01.00032 "
01.00033 "  04.02      KB/811218     THE LOADER USES LESS THEN 2K WORD MEMORY,
01.00034 "                           EVEN WHEN THE COMPILE TIME CONDITION
01.00035 "                           S = TRUE IS TRUE.
01.00036 "
01.00037 "  05.01      KB/820215     AUTO LOADING IS IMPLEMENTED WHEN
01.00038 "                           A MASTERCLEAR OR AN EMERGENCY ACTION HAPPENS.
01.00039 "                           IF NO INPUT IS GIVEN WITHIN 30 S, THE LOADER
01.00040 "                           WILL TRY TO LOAD WITH THE PARAMETERS:
01.00041 "                             DISK CONTR. I/O ADDRESS       = #32
01.00042 "                             DISK UNIT                     = 0
01.00043 "                             KIND OF DRIVE                 = SEE OPTION.S
01.00044 "                             CONTR. MEMORY SECTION         = 3 IF M
01.00045 "                                                           = 1 IF S
01.00046 "                             CONTR. MEMORY ADDRESS         = 0
01.00047 "                             BFD NO OF BOOT FILE           = SEE OPTION.S
01.00048 "
01.00049 "  05.02      KB/820401     LEVEL AND BOUND FIELD ARE INITIALIZED
01.00050 "                           IN THE PROCESS DESCRIPTER, TO COPE WITH
01.00051 "                           THE XAMOS OPERATING SYSTEM
01.00052 "
01.00053 "  06.01      KB/821101     AN INFINITE LOOP IS CORRECTED.
01.00054 "
01.00055 "                           THE ERROR MESSAGE FORMAT IS ALTERED.
01.00056 "
01.00057 "                           A SET PARITY COMMAND IS OFFERED.
01.00058 "
01.00059 "                           THE FOUR I/O COMMANDS ARE OFFERED.
01.00060 "
01.00061 "                           WHEN AUTO LOADING IS PERFORMED, THE
01.00062 "                           LOAD PARAMETERS ARE PRINTED.
01.00063 "
01.00064 "                           IT IS NOW POSSIBLE TO ACTIVATE THE LOADER
01.00065 "                           FROM ROOT, AND RETURN BY MEANS OF A RETURN
01.00066 "                           COMMAND.
01.00067 "
01.00068 "                           THE BASE OF THE LOADER IS PRINTED AT
01.00069 "                           ACTIVATION.
01.00070 "
01.00071 "                           IF A PARITY ERROR IS DISCOVERED DURING
```

```
C1.00072 "                          SETTING PARITY, A MESSAGE IS PRINTED.
C1.00073 "
01.00074 "-------------------------------------------------------------------
01.00075 "
```

```
01.00076
01.00076

======== LOADER.S

00.00003
00.00004 %NOLIST
00.00005 MAINMODULE LOADER;
00.00006 %SOURCE

======== GENERALPARAMS.S


======== LOADER.S

00.00006
00.00007 %SOURCE

======== FMSPARAMS.S


======== LOADER.S

00.00007
00.00008 %SOURCE

======== VOL_STRUCTURES.S


======== LOADER.S

00.00008
00.00009 %SOURCE

======== X2GENPARAMS.S


======== LOADER.S

00.00009
00.00010 %LIST
00.00011 %SOURCE

======== OPTIONS.S
```

```
06.00001 "
```

```
06.00002
06.00002 "==================== O P T I O N S ====================
06.00003
06.00004
06.00005
06.00006 TYPE
06.00007    OS_TYPE = (AMOS, XAMOS);
06.00008    DISKIND=
06.00009      (MMD12M,MMD12F,MMD24M,MMD24F,MMD80M,MMD81F,MMD82F,MMD160M,
06.00010      SMD40R,SMD80R,SMD150R,SMD300R,SMD600R,
06.00011      CMD32F,CMD32R,CMD64F,CMD64R,CMD96F,CMD96R);
06.00012
06.00013 CONST
06.00014    HEADER = 'XAMOS BOOT LOADER V5.0601(:0:)';  "FIRST HEAD LINE
06.00015    OS = AMOS;                                  "OPERATING SYSTEM
06.00016    S = FALSE;                                  "DISK CONTROLLER IS OF
06.00017                                                "TYPE S OR M
06.00018    AUTO_LOAD_KIND_OF_DRIVE = SMD80R;
06.00019    AUTO_LOAD_BFD_ENTRY_NO = #5;
06.00020    TRACE_ON = FALSE;                           "TRACE OF PROCEDURE CALLS ON
06.00021

======== LOADER.S

00.00011
00.00012 "
```

```
00.00013
00.00013 VAR
00.00014    SAVEREGS: ARRAY[0..7] OF INTEGER;
00.00015
00.00016 %SOURCE

======== CONSOLE.S

07.00001 "
```

```
07.00002
07.00002 "==================== C O N S O L E ====================
07.00003
07.00004
07.00005
07.00006
07.00007
07.00008 CONST "*****  GENERAL STATUS CONSTANTS  *****"
07.00009
07.00010       CONSOLEADDR        =  1;     "INTERFACE DEVICE ADDRESS"
07.00011       CONSOLEENABLE      =  3;     "ENABLE CONSOLE (USED BY CIO)"
07.00012       CONSOLEPARITY      = 14;
07.00013
07.00014       "***** AV 24 I/F STATUS CONSTANTS *****"
07.00015       INPUTREADY         = 10;     "CONSOLE INPUT READY"
07.00016       OUTPUTREADY        = 11;     "CONSOLE OUTPUT READY"
07.00017       CONSOLEBREAK       = 13;     "BREAK PRESSED AT CONSOLE"
07.00018
07.00019       "*****  SCM I/F STATUS CONSTANTS  *****"
07.00020
07.00021       SCM_INPUTREADY     = 1;      "CONSOLE INPUT READY"
07.00022       SCM_OUTPUTREADY    = 0;      "CONSOLE OUTPUT READY"
07.00023       SCM_CONSOLEBREAK   = 5;      "BREAK PRESSED AT CONSOLE"
07.00024
07.00025       "*****  SCM I/F CONTROL CONSTANTS  *****"
07.00026
07.00027       SETUP_MODE_REG1 = #88;
07.00028       SETUP_MODE_REG2 = #35;
07.00029       SETUP_CMD_REG   = #37;
07.00030       LCAD_MODE_REG1  = #200;
07.00031       LCAD_MODE_REG2  = #600;
07.00032       LCAD_CMD_REG    = #300;
07.00033       CLEAR_SCM       = #37;
07.00034       SENSE_STATUS    = #100;
07.00035
07.00036       L_MAX = 78;  "MAX LINE LENGTH"
07.00037
07.00038       NRSIGN = FALSE;
07.00039
07.00040 TYPE
07.00041       TYPE_OF_DEVICE     = (SCM,V24);
07.00042
```

```
07.00043        PATCHARRAY = ARRAY[0..0] OF INTEGER;
07.00044
07.00045 VAR
07.00046        L_LENGTH : INTEGER;       "LINE LENGTH OF CURRENT LINE"
07.00047
07.00048        LINE     : ARRAY[1..L_MAX] OF CHAR;
07.00049
07.00050        DEVICE_TYPE : TYPE_OF_DEVICE;
07.00051
07.00052        CONSOLE_STATUS: INTEGER;
07.00053
07.00054 EXPORT VAR
07.00055
07.00056        BREAKADDR : INTEGER;     "ADDRESS TO JUMP TO IF A BREAK
07.00057                                 "IS PRESSED DURING NORMAL INPUT
07.00058        CH_COUNT : INTEGER;       "POINTER TO CURRENT CHARACTER
07.00059
07.00060        LAST_IN  : INTEGER;       "LAST INPUT CHARACTER READ
07.00061
07.00062        TIMER : INTEGER;          "MILLISECONDS COUNTER
07.00063
07.00064 INIT
07.00065        L_LENGTH    = 0;
07.00066        CH_COUNT    = 0;
07.00067        CONSOLE_STATUS = 0;
07.00068 "
```

```
07.00069
07.00069 EXPORT PROCEDURE SET_DEVICE_TYPE
07.00070 "============================="
07.00071                    (R6);  "LINK
07.00072 VAR
07.00073    S5, S6, S7: INTEGER;
07.00074 BEGIN
07.00075    R5 => S5; R6 => S6; R7 => S7;
07.00076    CONSCLEADDR => R7;
07.00077    SIO(R5, R7);
07.00078    RIO(R6, R7);
07.00079    R5 EXTRACT 7;
07.00080    R6 EXTRACT 7;
07.00081    IF R5 <> R6
07.00082    THEN
07.00083       V24 => R6 => DEVICE_TYPE
07.00084    ELSE
07.00085       SCM => R6 => DEVICE_TYPE;
07.00086    S5 => R5; S6 => R6; S7 => R7;
07.00087    EXIT(R6);
07.00088 END; "SET_DEVICE_TYPE"
07.00089
```

```
07.00090
07.00090 EXPORT PROCEDURE INIT_IF
07.00091 "======================"
07.00092                          (R6);            "LINK"
07.00093 VAR S6, S7 : INTEGER;
07.00094 BEGIN
07.00095    R6 => S6;  R7 => S7;
07.00096    SET_DEVICE_TYPE(R6);
07.00097    IF DEVICE_TYPE => R6 = V24
07.00098    THEN
07.00099    BEGIN
07.00100       CIO(CONSOLEENABLE => R6, CONSOLEADDR => R7);
07.00101       SIO(R6, R7);
07.00102       RIO(R6, R7);
07.00103    END
07.00104    ELSE      " SCM "
07.00105    BEGIN
07.00106       WIO(SETUP_MODE_REG1=>R6,CONSOLEADDR+LOAD_MODE_REG1=>R7);
07.00107       WIO(SETUP_MODE_REG2=>R6,CONSOLEADDR+LOAD_MODE_REG2=>R7);
07.00108       WIO(SETUP_CMD_REG=>R6,CONSOLEADDR+LOAD_CMD_REG=>R7);
07.00109    END;
07.00110    S7 => R7;
07.00111    EXIT(S6);
07.00112 END; "INIT_IF"
07.00113
```

```
07.00114
07.00114 EXPORT PROCEDURE SENSE
07.00115 "==================="
07.00116                (R7;              "RETURNS STATUS FROM I/F IN V24 LAYOUT"
07.00117                 R6);             "LINK"
07.00118 VAR S6, S2: INTEGER;
07.00119 BEGIN
07.00120    R6 => S6;   R2 => S2;
07.00121    IF DEVICE_TYPE=>R7=V24
07.00122    THEN
07.00123    BEGIN
07.00124       CIO(CONSOLEENABLE => R7, CONSOLEADDR => R6);
07.00125       SIO(R7, R6);
07.00126    END
07.00127    ELSE
07.00128    BEGIN
07.00129       RIO(R2,CONSOLEADDR=>R2+SENSE_STATUS);
07.00130       0=>R7;
07.00131       IF R2[SCM_INPUTREADY] THEN SETS(R7,INPUTREADY);
07.00132       IF R2[SCM_OUTPUTREADY] THEN SETS(R7,OUTPUTREADY);
07.00133       IF R2[SCM_CONSOLEBREAK] THEN
07.00134       BEGIN
07.00135          SETS(R7,CONSOLEBREAK);
07.00136          WIO(CLEAR_SCM=>R6,CONSOLEADDR+LOAD_CMD_REG=>R2);
07.00137       END;
07.00138    END;
07.00139    S2 => R2;
07.00140    EXIT(S6);
07.00141 END; "SENSE"
07.00142
```

```
07.00143
07.00143 PROCEDURE OUTBYTE
07.00144 "================"
07.00145                    (R3;              "CHARACTER"
07.00146                     R6);             "LINK"
07.00147 VAR S3, S6, S7: INTEGER;
07.00148 BEGIN
07.00149    R3 => S3;  R6 => S6;  R7 => S7;
07.00150    REPEAT
07.00151       SENSE(R7, R6);                  "SENSE DEVICE STATUS"
07.00152    IF  R7[CONSOLEBREAK] THEN BEGIN
07.00153       RIO(R7, CONSOLEADDR => R7); "READ CHARACTER"
07.00154       1000 => R7 => CH_COUNT;
07.00155       S7 => R7;
07.00156       EXIT(BREAKADDR => R6);   "BREAK HAS BEEN PRESSED"
07.00157    END;
07.00158    UNTIL R7[OUTPUTREADY];
07.00159    CONSOLEADDR => R7;
07.00160    R3 EXTRACT(8);                  "EXTRACT CHARACTER"
07.00161    WIO(R3, R7);
07.00162    S3 => R3;  S7 => R7;
07.00163    EXIT(S6);
07.00164 END; "OUTBYTE"
07.00165 "
```

```
07.00166
07.00166 EXPORT PROCEDURE·OUTCHAR
07.00167 "======================"
07.00168                    (R3;          "CHARACTER"
07.00169                     R6);         "LINK"
07.00170 VAR S3, S6 : INTEGER;
07.00171 BEGIN
07.00172    R6 => S6;
07.00173    OUTBYTE(R3, R6);
07.00174    IF R3 = CR THEN
07.00175    BEGIN
07.00176       R3 => S3;
07.00177       0=>R3;
07.00178       OUTBYTE(R3, R6);
07.00179       OUTBYTE(R3, R6);
07.00180       OUTBYTE(R3, R6);
07.00181       OUTBYTE(R3, R6);
07.00182       OUTBYTE(R3, R6);
07.00183    S3 => R3;
07.00184    END;
07.00185    EXIT(S6);
07.00186 END; "OUTCHAR"
07.00187
```

```
07.00188
07.00188 EXPORT PROCEDURE OUTTEXT
07.00189 "======================"
07.00190                     (R3;     "ADDRESS OF TEXT"
07.00191                      R6);    "LINK"
07.00192 "-------------------------------------------------------
07.00193 " PRINT A STRING OF CHARACTERS
07.00194 " TERMINATED BY (:0:)
07.00195 "-------------------------------------------------------
07.00196 TYPE
07.00197    BYTEARR = ARRAY [0..1] OF CHAR;
07.00198 VAR
07.00199    S2, S3, S4, S6: INTEGER;
07.00200 BEGIN
07.00201    R2 => S2; R3 => S3; R4 => S4; R6 => S6;
07.00202    R3 => R2;
07.00203    0=>R4;
07.00204    REPEAT
07.00205       OUTCHAR(R2@BYTEARR[R4] => R3, R6);
07.00206       R4 + 1;
07.00207    UNTIL R3 = 0;
07.00208    S2 => R2; S3 => R3; S4 => R4; S6 => R6;
07.00209    EXIT(R6);
07.00210 END; "OUTTEXT"
07.00211
```

```
07.00212
07.00212 EXPORT PROCEDURE OUTNEWLINE
07.00213 "=========================="
07.00214                              (R6);   "LINK"
07.00215 "-------------------------------------------------------
07.00216 " PRINT A CR AND A LF CHARACTER ON THE CONSOLE
07.00217 "-------------------------------------------------------
07.00218 VAR S3, S6 : INTEGER;
07.00219 BEGIN
07.00220    R3 => S3;   R6 => S6;
07.00221    OUTCHAR(CR => R3, R6);
07.00222    OUTCHAR(LF => R3, R6);
07.00223    S3 => R3;
07.00224    EXIT(S6);
07.00225 END; "OUTNEWLINE"
07.00226
```

```
07.00227
07.00227 EXPORT PROCEDURE OUTHEX
07.00228 "====================="
07.00229                  (R3;        "INTEGER TO BE PRINTED"
07.00230                   R6);       "LINK"
07.00231 "------------------------------------------------------
07.00232 " OUTPUT AN INTEGER IN ITS HEX.
07.00233 " REPRESENTATION
07.00234 "------------------------------------------------------
07.00235 VAR
07.00236    S3, S4, S5, S6: INTEGER;
07.00237 BEGIN
07.00238    R3 => S3; R4 => S4; R5 => S5; R6 => S6;
07.00239
07.00240 %WHEN NRSIGN = FALSE SKIP
07.00243    S3 => R4;
07.00244    0 => R5;
07.00245    REPEAT
07.00246       R4 SHIFTLC 4;
07.00247       R4=>R3 EXTRACT 4;
07.00248       IF R3 < 10
07.00249       THEN
07.00250          R3 + ('0' - 'A' + 10);
07.00251       R3 + ('A' -10);
07.00252       OUTCHAR(R3, R6);
07.00253    UNTIL R5+1 = 4;
07.00254
07.00255    S3 => R3; S4 => R4; S5 => R5; S6 => R6;
07.00256    EXIT(R6);
07.00257 END; "OUTHEXA"
07.00258
```

```
07.00259
07.00259 EXPORT PROCEDURE IS_THERE_ANY_INPUT
07.00260 "=============================
07.00261                  (R3;  "<TIME OUT IN MILLISECONDS: INTEGER;
07.00262                   R6)  "LINK
07.00263                  : BOOLEAN;
07.00264 VAR
07.00265    S6, S7: INTEGER;
07.00266 BEGIN
07.00267    R6 => S6; R7 => S7;
07.00268    0 => R6 => TIMER;
07.00269
07.00270    REPEAT
07.00271       SENSE(R7, R6);
07.00272    UNTIL R7[INPUTREADY] LOGOR TIMER => R6 >= R3;
07.00273
07.00274    R7 => CONSOLE_STATUS;
07.00275
07.00276    IF R7[INPUTREADY]
07.00277    THEN
07.00278       S6 => R6 + TRUE
07.00279    ELSE
07.00280       S6 => R6 + FALSE;
07.00281
07.00282    S7 => R7;
07.00283    EXIT(R6);
07.00284 END; "PROCEDURE IS_THERE_ANY_INPUT"
07.00285
```

```
07.00286
07.00286 EXPORT PROCEDURE GETBYTE
07.00287 "======================"
07.00288                    (R3;     "CHARACTER (RETURN)"
07.00289                     R6);    "LINK"
07.00290 "------------------------------------------------------------
07.00291 " Gets the next character from console
07.00292 "------------------------------------------------------------
07.00293 VAR S6, S7 : INTEGER;
07.00294 BEGIN
07.00295    R6 => S6;   R7 => S7;
07.00296    CONSOLE_STATUS => R7;
07.00297
07.00298    WHILE NOT R7[INPUTREADY] DO
07.00299       SENSE(R7, R6);            "GET CONSOLE STATUS"
07.00300
07.00301    0 => R6 => CONSOLE_STATUS;
07.00302
07.00303    IF R7[CONSOLEBREAK] THEN BEGIN
07.00304       1000 => R7 => CH_COUNT;
07.00305       RIO(R7, CONSOLEADDR => R6);
07.00306       S7 => R7;
07.00307       EXIT(BREAKADDR => R6);
07.00308    END;
07.00309    RIO(R7,CONSOLEADDR=>R6);
07.00310    R7 EXTRACT 7 => R3;
07.00311    OUTCHAR(R3, R6);
07.00312    IF R3 = CR THEN OUTCHAR(LF => R3, R6);
07.00313    R7 => R3;
07.00314    S7 => R7;
07.00315    EXIT(S6);
07.00316 END; "GETBYTE"
07.00317
```

```
07.00318
07.00318 EXPORT PROCEDURE IN_BYTE
07.00319 "======================"
07.00320                         (R3;       "CHARACTER VALUE (RETURN)"
07.00321                         R6);       "LINK"
07.00322 TYPE  BYTEARR = ARRAY[0..1] OF CHAR;
07.00323 VAR S4, S6, S7 : INTEGER;
07.00324 BEGIN
07.00325    R4 => S4;  R6 => S6;  R7 => S7;
07.00326    ADDRESS(LINE) => R7;
07.00327    IF CH_COUNT => R4 >= L_LENGTH => R6 THEN BEGIN
07.00328       0 => R4 => CH_COUNT;
07.00329       REPEAT
07.00330          R4 + 1;            "CHARACTER COUNTER"
07.00331          GETBYTE(R3, R6);
07.00332          R3 => R7@BYTEARR[R4];
07.00333          IF R3 = 127 "DEL" THEN BEGIN
07.00334             OUTTEXT(ADDRESS(' *DEL(:0:)') => R3, R6);
07.00335             OUTNEWLINE(R6);
07.00336             OUTCHAR('>' => R3, R6);
07.00337             0 => R4;
07.00338          END;
07.00339       UNTIL R3 = CR LOGOR R4 = L_MAX;
07.00340       R4 => L_LENGTH;
07.00341    END;
07.00342    CH_COUNT => R4 + 1 => CH_COUNT;
07.00343    R7@BYTEARR[R4] => R3;
07.00344    S4 => R4;  S7 => R7;
07.00345    EXIT(S6);
07.00346 END; "IN_BYTE"
07.00347
```

```
07.00348
07.00348 EXPORT PROCEDURE GET_NEXT_NON_BLANK(R3;R6);
07.00349 "==================================="
07.00350                                    " R6  LINK
07.00351                                    " R3   RETURN CHARACTER VALUE
07.00352                                    " GLOBAL: LAST_IN
07.00353
07.00354 " WHILE IGNORING THE LAST CHARACTER READ IN,THE
07.00355 "-------------------------------------------------------------
07.00356 " PROCEDURE GETS THE NEXT NEITHER BLANK NOR COMMA
07.00357 " INPUT CHARACTER AND RESERVES IT IN LAST_IN
07.00358 "-------------------------------------------------------------
07.00359
07.00360 VAR
07.00361   S6 : INTEGER;
07.00362
07.00363 BEGIN
07.00364   R6=>S6;
07.00365  REPEAT
07.00366    IN_BYTE(R3,R6)
07.00367   UNTIL R3<>' ';
07.00368   R3=>LAST_IN;
07.00369   EXIT(S6);
07.00370 END" GET_NEXT_NON_BLANK ";
07.00371
```

```
07.00372
07.00372 EXPORT PROCEDURE GET_HEXA(R2;R1;R6);
07.00373                                " R2   HEX LEGAL NO
07.00374                                " R1   INVALIDITY FLAG
07.00375                                " R6   LINK
07.00376                                " GLOBAL: LAST_IN
07.00377
07.00378 " WILL GET THE FIRST SEQUENCE OF NEITHER SPACE NOR COMMA DATA,
07.00379 " AND CHECK HEX VALIDITY.
07.00380 " IF LESS THAN 4 CONSECUTIVE HEX CHARACTERS ENTERED, WILL PAD W/ ZEROES
07.00381 " ON LEFT EXCEPT IF NO HEX DATA ENTERED.
07.00382 " THE FIRST DIGIT IS EXPECTED IN LAST_IN
07.00383
07.00384 TYPE
07.00385   CHARCLASS=(HEXALFA,NUMERIC,NON_HEX);
07.00386
07.00387 VAR
07.00388   SAVEREGS : ARRAY [0..7] OF INTEGER;
07.00389
07.00390 BEGIN
07.00391   R7 => SAVEREGS[7];
07.00392   STC(6,ADDRESS(SAVEREGS[7])=>R7);
07.00393   TRUE=>R1;
07.00394   0 => R2 => R5;
07.00395   LAST_IN=>R3;
07.00396   IF R3 = '#'
07.00397   THEN
07.00398     GET_NEXT_NON_BLANK(R3,R6);
07.00399
07.00400   REPEAT
07.00401     IF R3>='0' LOGAND R3<'9'+1
07.00402     THEN
07.00403       NUMERIC=>R4
07.00404     ELSE
07.00405       IF R3>='A' LOGAND R3<'F'+1
07.00406       THEN HEXALFA=>R4
07.00407       ELSE NON_HEX=>R4;
07.00408     IF R4 <> NON_HEX
07.00409     THEN
07.00410     BEGIN
07.00411       FALSE=>R1;                        " AT LEAST ONE HEX_CHAR READ IN
07.00412       IF R4=NUMERIC
```

```
07.00413        THEN
07.00414          R3 - ('0'-'A'+10);
07.00415        R3 - ('A'-10);
07.00416        R5+1;                          "UPDATE DIGIT COUNTER
07.00417        R2 SHIFTLL 4;
07.00418        R2+R3;
07.00419        IN_BYTE(R3,R6);
07.00420      END;
07.00421    UNTIL R4 = NON_HEX;
07.00422
07.00423    R3 => LAST_IN;
07.00424    IF R5 >= 5
07.00425    THEN
07.00426      TRUE => R1;
07.00427    R1=>SAVEREGS[1];
07.00428    R2=>SAVEREGS[2];
07.00429    UNS (7,ADDRESS(SAVEREGS[0])=>R7);
07.00430    EXIT(R6);
07.00431 END" GET_HEXA ";
07.00432

======== LOADER.S

00.00016
00.00017 %SOURCE

======== TRACE.S

08.00001 "
```

```
08.00002
08.00002 "==================== T R A C E ====================
08.00003
08.00004
08.00005
08.00006
08.00007
08.00008 %SKIP
08.00067

======== LOADER.S

00.00017
00.00018 %SOURCE

======== ERROR.S

09.00001 "
```

```
09.00002
09.00002 "==================== E R R O R ====================
09.00003
09.00004
09.00005
09.00006
09.00007
09.00008 EXPORT VAR
09.00009        ERRADDR : INTEGER;
09.00010 "
```

```
C9.00011
C9.00011 EXPORT PROCEDURE ERROR(R7;R6);
C9.00012                      " R7 ILLEGAL INPUT PARAMETER INDEX
C9.00013                      " R6 LINK
09.00014 BEGIN
C9.00015 %SKIP
C9.00019   OUTNEWLINE(R6);
09.00020   OUTTEXT(ADDRESS('*** ERROR: (:0:)') => R3, R6);
09.00021   OUTHEX(R7=>R3,R6);
C9.00022   EXIT(ERRADDR=>R6);
09.00023 END" error ";
09.00024

======== LOADER.S

00.00018
00.00019 %SOURCE

======== DISK.S

10.00001 "
```

```
10.00002
10.00002 "==================== D I S K ====================
10.00003
10.00004
10.00005
10.00006
10.00007
10.00008 CONST
10.00009   OK = 12;
10.00010   LIMIT = 10;
10.00011   ERRECOVOPCOD = #C250;
10.00012
10.00013 TYPE
10.00014   PTR  =  INTEGER;
10.00015   BUFPTR      = ARRAY[1..15] OF ABSPTR;
10.00016   TRACK_SELECT = (NO_CHANGE_OF_TRACK, NO_CHANGE, HEAD_SELECT, CYLINDER_AND_HEAD_SELECT);
10.00017
10.00018   CTRLB = RECORD
10.00019           RETURN_INFORMATION : ARRAY[0..2] OF INTEGER;
10.00020           OP_CODE_UNIT       : INTEGER;              " 12 MSBs are OP_CODE, 4 LSBs are xxUNIT
10.00021           CF_CYLINDER        : INTEGER;              " Bit 13 is CF, 10 LSBs - CYLINDER
10.00022           SECTOR_HEAD        : INTEGER;              " 8 MSBs = SECTOR, the rest are HEAD
10.00023           NO_OF_SECTORS_TRACKS_PER_CYLINDER : INTEGER;" 8 MSBs are NO OF SECTORS, the rest - TRACKS PER CYLINDER
10.00024           OFFSET             : INTEGER;
10.00025           PTR_TO_BUFFER0     : PTR;    "REL / ABS
10.00026         END;
10.00027
10.00028
10.00029   BUFFER = RECORD
10.00030            POINTER : INTEGER;
10.00031            PROLOGUE : ARRAY[0..15] OF INTEGER;
10.00032            SECTOR   : ARRAY[0..255] OF INTEGER;
10.00033            EPILOGUE : ARRAY[0..1] OF INTEGER;
10.00034          END;
10.00035
10.00036 VAR
10.00037   FIRST_SECTOR : LONG;
10.00038   CYLINDER     : INTEGER;
10.00039
10.00040 EXPORT VAR
10.00041   FIRST_CYLINDER : INTEGER;
10.00042   DISK_TYPE : DISKIND;
```

```
10.00043    DISK_IO_ADDRESS : INTEGER;
10.00044    DISK_PAGE       : INTEGER;
10.00045    UNIT : INTEGER;
10.00046    CF   : INTEGER;
10.00047    BOOT_STRAP_BFD_ENTRY_NO : INTEGER;
10.00048    CONTROLLER_ADDRESS_MEMORY_SECTION : INTEGER;
10.00049    CONTROLLER_MEMORY_WORD_ADDRESS : INTEGER;
10.00050    SECTORS_PER_CYLINDER : INTEGER ;
10.00051
10.00052 "
```

```
10.00053
10.00053    "##########################################"
10.00054    "#                                        #"
10.00055    "#   PARAMETER      HANDLING             #"
10.00056    "#                                        #"
10.00057    "##########################################"
10.00058
10.00059 PROCEDURE UPDATE_DISK_PARAM(R4;R6);
10.00060                              " R4 DISK TYPE (RUINED)
10.00061                              " R6 LINK
10.00062                              " GLOBALS: DISK_TYPE,SECTORS_PER_CYLINDER
10.00063
10.00064 VAR
10.00065    LINK: INTEGER;
10.00066    SEC_CYLS  : ARRAY [DISKIND] OF INTEGER;
10.00067 INIT
10.00068    SEC_CYLS = 64,128,128,128,160,160,160,320,160,160,608,608,1280,
10.00069               32, 32, 96, 32,160, 32;
10.00070 BEGIN
10.00071   R6=>LINK;
10.00072 %SKIP
10.00076   SEC_CYLS[R4]=>R6=>SECTORS_PER_CYLINDER;
10.00077
10.00078   IF R4 >= #D THEN R4 EXTRACT 1 => CF
10.00079      ELSE 0 => R4=>CF;
10.00080
10.00081   EXIT(LINK);
10.00082 END"  UPDATE DISK PARAMETERS  ";
10.00083 "
```

```
10.00084
10.00084 EXPORT PROCEDURE GENERATE_PSW_IN_PAGE(R3;   " INPUT-PAGE
10.00085                                       R4;   " RETURNED PSW
10.00086                                       R6);  " LINK
10.00087 "-------------------------------------------------------------
10.00088 " GENERATE PSW for given memory section
10.00089 " disabling timer interrupts
10.00090 "-------------------------------------------------------------
10.00091 VAR
10.00092   S6  : INTEGER;
10.00093 BEGIN
10.00094   R6=>S6;
10.00095 %SKIP
10.00099   R3 => R4 SHIFTLL 2 + #E000;
10.00100   EXIT(S6);
10.00101 END" Generate PSW in page ";
10.00102 "
```

```
10.00103
10.00103 PROCEDURE GENERATE_BASIC_CTRLB(R0;    " CF
10.00104                                R1;    " UNIT
10.00105                                R2;    " TRACKS PER CYLINDER
10.00106                                R3;    " CONTROLLER MEMORY SECTION
10.00107                                R4;    " PSW FOR DISK
10.00108                                R5;    " CONTROLLER base relative WORD ADDRESS
10.00109                                R6);   " LINK
10.00110 "-------------------------------------------------------------
10.00111 " Insert disk unit characteristic data in control block
10.00112 " Read (R0) sectors beginning at (R01).Return Completion Code
10.00113 "-------------------------------------------------------------
10.00114 BEGIN
10.00115   R7 => SAVEREGS[7];
10.00116   STC (6,ADDRESS(SAVEREGS[7])=>R7);
10.00117 %SKIP
10.00121   DISK_PAGE => R4;
10.00122   SVS (R7);
10.00123   LDS (R4);
10.00124   R1=>R5@CTRLB.OP_CODE_UNIT;
10.00125   R2=>R5@CTRLB.NO_OF_SECTORS_TRACKS_PER_CYLINDER;
10.00126   0=>R3=>R5@CTRLB.OFFSET;
10.00127   IF R0=1 THEN SETS (R3,13);
10.00128   R3=>R5@CTRLB.CF_CYLINDER;
10.00129   LDS (R7);
10.00130 "* OUTNEWLINE(R6);
10.00131 "* OUTHEX(R3,R6);
10.00132 "* OUTTEXT(ADDRESS(' AT GEN_BASIC_CTRLB (:0:)')=>R3,R6);
10.00133 "* OUTHEX(R0=>R3,R6);
10.00134 "* OUTHEX(R1=>R3,R6);
10.00135 "* OUTHEX(R2=>R3,R6);
10.00136 "* OUTHEX(S4=>R3,R6);
10.00137 "* OUTHEX(R5=>R3,R6);
10.00138 "* OUTHEX(S6=>R3,R6);
10.00139 "* OUTHEX(R7=>R3,R6);
10.00140   UNS (7,ADDRESS(SAVEREGS[0])=>R7);
10.00141   EXIT(R6);
10.00142 END" GENERATE BASIC " ;
10.00143 "
```

```
10.00144
10.00144 PROCEDURE GEN_ERRECOV_CTRLB(R1;  " Unit
10.00145                             R3;  " Controller memory section
10.00146                             R5;  " Controller base relative memory address
10.00147                             R6); " Link
10.00148 "-------------------------------------------------------------
10.00149 " Generate an error recovery control block
10.00150 " to perform CLEAR FAULT, RETURN TO ZERO
10.00151 "-------------------------------------------------------------
10.00152 BEGIN
10.00153   R7 => SAVEREGS[7];
10.00154   STC (6,ADDRESS(SAVEREGS[7])=>R7);
10.00155 %SKIP
10.00159   R5 + SIZE(CTRLB);
10.00160   DISK_PAGE=>R4;
10.00161   SVS (R6);
10.00162   LDS (R4);
10.00163   CLR(R5@CTRLB.OFFSET);
10.00164   CLR(R5@CTRLB.RETURN_INFORMATION[2]);
10.00165   R1 EXTRACT 4 + ERRECOVOPCOD =>R5@CTRLB.OP_CODE_UNIT;
10.00166   LDS (R6);
10.00167 "* OUTNEWLINE(R6);
10.00168 "* OUTHEX(R3,R6);
10.00169 "* OUTTEXT(ADDRESS(' AT GEN ERRECOV CTRLB (:0:)')=>R3,R6);
10.00170 "* OUTHEX(R0=>R3,R6);
10.00171 "* OUTHEX(R1=>R3,R6);
10.00172 "* OUTHEX(R2=>R3,R6);
10.00173 "* OUTHEX(R4=>R3,R6);
10.00174 "* OUTHEX(R5=>R3,R6);
10.00175 "* OUTHEX(S6=>R3,R6);
10.00176 "* OUTHEX(R7=>R3,R6);
10.00177   UNS (7,ADDRESS(SAVEREGS[0])=>R7);
10.00178   EXIT(R6);
10.00179 END " Generating error recovery control block " ;
10.00180
```

```
10.00181
10.00181 PROCEDURE GENERATE_BUFFER_CHAIN
10.00182                                (R3;     " DISK CONTROLLER MEMORY SECTION
10.00183                                 R5;     " DISK CONTROLLER BASE RELATIVE  WORD_ADDRESS IN MEMORY -
10.00184                                 R6);    " LINK
10.00185 "-------------------------------------------------------------
10.00186 " Generate a buffer chain for disk I/O (16 buffers)
10.00187 "-------------------------------------------------------------
10.00188 BEGIN
10.00189    R7 => SAVEREGS[7];
10.00190    STC (6,ADDRESS(SAVEREGS[7])=>R7);
10.00191 %SKIP
10.00195    G@PROCESS_DESCRIPTOR.BASE=>R3+R5;                    " Assign ctrlb absolute address to R3
10.00196    DISK_PAGE=>R4;              "ADJUST PSW TO RIGHT MEMORY SECTION
10.00197    SVS (R6);
10.00198    LDS (R4);
10.00199    R3=>R4+(2*SIZE(CTRLB));                               " POINT AT 1ST LOCATION AFTER SECOND CONTROL BLOCK
10.00200    R4=>R5@CTRLB.PTR_TO_BUFFER0;
10.00201    R5+18;
10.00202    0=>R3;
10.00203    REPEAT
10.00204      R3+1;
10.00205      R4+275=>R5@BUFFER.POINTER;
10.00206      R5+275;
10.00207    UNTIL R3=15;
10.00208    LDS (R6);
10.00209 "* OUTNEWLINE(R6);
10.00210 "* OUTHEX(R3,R6);
10.00211 "* OUTTEXT(ADDRESS(' AT GEN BUFFER CHAIN (:0:)')=>R3,R6);
10.00212 "* OUTHEX(R0=>R3,R6);
10.00213 "* OUTHEX(R1=>R3,R6);
10.00214 "* OUTHEX(R2=>R3,R6);
10.00215 "* OUTHEX(R4=>R3,R6);
10.00216 "* OUTHEX(R5=>R3,R6);
10.00217 "* OUTHEX(S6=>R3,R6);
10.00218 "* OUTHEX(R7=>R3,R6);
10.00219    UNS (7,ADDRESS(SAVEREGS[0])=>R7);
10.00220    EXIT(R6);
10.00221 END " generate buffer chain " ;
10.00222 "
```

```
10.00223
10.00223 PROCEDURE SET_READ_OP_CODE(R7;  " Returned operation code
10.00224                            R2;  " Track or Cylinder select: (no change, no change of track,
10.00225                                 "                            Head select, Cylinder & Head select)
10.00226                            R6); " Link
10.00227 "-------------------------------------------------------------
10.00228 " Generate read operation code with one of three possible track
10.00229 " select choices
10.00230 "-------------------------------------------------------------
10.00231 VAR
10.00232   S6: INTEGER;
10.00233 BEGIN
10.00234   R6 => S6;
10.00235 %SKIP
10.00239   #1200=>R7;     "   Read & Release
10.00240   IF R2=HEAD_SELECT THEN SETS (R7,11);
10.00241   IF R2=CYLINDER_AND_HEAD_SELECT THEN BEGIN
10.00242                                        SETS (R7,11);
10.00243                                        SETS (R7,10);
10.00244                                      END;
10.00245   S6 => R6;
10.00246   EXIT(R6);
10.00247 END " Set read op_code ";
10.00248
```

```
10.00249
10.00249 PROCEDURE CYL_HEAD_SECT(R01;    " First sector
10.00250                         R2;     " Sectors per cylinder
10.00251                                 " Returns: Cylinder
10.00252                       " R0;                Head
10.00253                       " R1;                Sector
10.00254                         R3;     " First cylinder (0 or 896)
10.00255                         R6);    " Link
10.00256 "-------------------------------------------------------------
10.00257 " Return physical sector, head & cylinder in correspondance
10.00258 " with disk type and first logical cylinder
10.00259 "-------------------------------------------------------------
10.00260 VAR
10.00261    S6 : INTEGER;
10.00262 BEGIN
10.00263   R6=>S6;
10.00264 %SKIP
10.00268   R01 / R2;
10.00269   R0 + R3 => R2;
10.00270   R1=>R0;
10.00271   0=>R1;
10.00272   R01 / (32=>R6);      " SECTOR=>R1 & HEAD=>R0
10.00273 "* OUTNEWLINE(R6);
10.00274 "* OUTHEX(R3,R6);
10.00275 "* OUTTEXT(ADDRESS(' AT CYL HEAD SECT (:0:)')=>R3,R6);
10.00276 "* OUTHEX(R0=>R3,R6);
10.00277 "* OUTHEX(R1=>R3,R6);
10.00278 "* OUTHEX(R2=>R3,R6);
10.00279 "* OUTHEX(R4=>R3,R6);
10.00280 "* OUTHEX(R5=>R3,R6);
10.00281 "* OUTHEX(S6=>R3,R6);
10.00282 "* OUTHEX(R7=>R3,R6);
10.00283   EXIT(S6);
10.00284 END " cyl_head_sect ";
10.00285
```

```
10.00286
10.00286 PROCEDURE ASSIGN_CTRLB(R0;   " Head
10.00287                        R1;   " Sector
10.00288                        R2;   " Cylinder
10.00289                        R3;   " Controller memory section
10.00290                        R4;   " NO of sectors
10.00291                        R5;   " Controller base relative word address
10.00292                        R7;   " Operation  (12 MSBs)
10.00293                        R6);  " Link
10.00294 "-------------------------------------------------------------
10.00295 " Assign disk parameters & operation code to control block
10.00296 "-------------------------------------------------------------
10.00297 BEGIN
10.00298    R7 => SAVEREGS[7];
10.00299    STC (6,ADDRESS(SAVEREGS[7])=>R7);
10.00300    SAVEREGS[7] => R7;
10.00301 %SKIP
10.00305    DISK_PAGE=>R4;
10.00306    SAVEREGS[4]=>R3;     " NO of sectors
10.00307    SVS (R6);
10.00308    LDS (R4);
10.00309    0=>R4=>R5@CTRLB.RETURN_INFORMATION[2];
10.00310    R5@CTRLB.OP_CODE_UNIT=>R4 EXTRACT 4;
10.00311    R7 + R4 => R5@CTRLB.CP_CODE_UNIT;
10.00312    R1 SHIFTLL 8 + R0 => R5@CTRLB.SECTOR_HEAD;
10.00313    R5@CTRLB.CF_CYLINDER=>R4 AND (#FC00=>R1);
10.00314                              " Clear cylinder parameter
10.00315    R2 + R4 => R5@CTRLB.CF_CYLINDER;
10.00316    R5@CTRLB.NO_OF_SECTORS_TRACKS_PER_CYLINDER => R4 EXTRACT 8;
10.00317    R3 SHIFTLL 8 + R4 => R5@CTRLB.NO_OF_SECTORS_TRACKS_PER_CYLINDER;
10.00318    LDS (R6);
10.00319 "* OUTNEWLINE(R6);
10.00320 "* OUTHEX(R3,R6);
10.00321 "* OUTTEXT(ADDRESS(' AT ASSIGN CTRLB (:0:)')=>R3,R6);
10.00322 "* OUTHEX(R0=>R3,R6);
10.00323 "* OUTHEX(R1=>R3,R6);
10.00324 "* OUTHEX(R2=>R3,R6);
10.00325 "* OUTHEX(R4=>R3,R6);
10.00326 "* OUTHEX(R5=>R3,R6);
10.00327 "* OUTHEX(S6=>R3,R6);
10.00328 "* OUTHEX(R7=>R3,R6);
10.00329   UNS (7,ADDRESS(SAVEREGS[0])=>R7);
```

```
10.00330    EXIT(R6);
10.00331 END " Assign control block ";
10.00332
```

```
10.00333
10.00333 PROCEDURE CLEAR_WORD_2(R3;      " Section
10.00334                        R5;      " Base relative
10.00335                        R6);
10.00336 VAR
10.00337   S4,  S6 : INTEGER;
10.00338 BEGIN
10.00339   R4=>S4;   R6=>S6;
10.00340 %SKIP
10.00344   DISK_PAGE=>R4;
10.00345   SVS (R6);
10.00346   LDS (R4);
10.00347   0=>R4=>R5@CTRLB.RETURN_INFORMATION[2];
10.00348   LDS (R6);
10.00349 "* OUTNEWLINE(R6);
10.00350 "* OUTHEX(R3,R6);
10.00351 "* OUTTEXT(ADDRESS(' AT CLR WORD2 (:0:)')=>R3,R6);
10.00352 "* OUTHEX(R0=>R3,R6);
10.00353 "* OUTHEX(R1=>R3,R6);
10.00354 "* OUTHEX(R2=>R3,R6);
10.00355 "* OUTHEX(R4=>R3,R6);
10.00356 "* OUTHEX(R5=>R3,R6);
10.00357 "* OUTHEX(S6=>R3,R6);
10.00358 "* OUTHEX(R7=>R3,R6);
10.00359   S4=>R4;
10.00360   EXIT(S6);
10.00361 END " ";
10.00362
```

```
10.00363
10.00363 PROCEDURE CONTROL_IO(R3;  " Controller memory section
10.00364                      R5;  " Control block relative word address
10.00365                      R4;  " Controller I/O address
10.00366                      R2;  " Returned Completion Code
10.00367                      R6); " Link
10.00368 "GLOBALS: UNIT
10.00369
10.00370 "-------------------------------------------------------------
10.00371 " Transfer a control block to the disk controller
10.00372 "-------------------------------------------------------------
10.00373
10.00374 CONST
10.00375    WAIT_CYCLES = 20000;
10.00376
10.00377 BEGIN
10.00378    R7 => SAVEREGS[7];
10.00379    STC (6,ADDRESS(SAVEREGS[7])=>R7);
10.00380 %SKIP
10.00384
10.00385    IF R3[1]                                        "Set page bits
10.00386    THEN SETS(R4, 6);
10.00387
10.00388    IF R3[0]
10.00389    THEN SETS(R4, 7);
10.00390
10.00391    SETS(R4, 15);                                   "Mask out OPC interrupt
10.00392    SETS(R4, 9);                                    "Command code:
10.00393                                                    "Load instruction pointer,
10.00394                                                    "initiate operation
10.00395    R5 + (0@PROCESS_DESCRIPTOR.BASE => R6);         "Rel to abs address
10.00396    CIO(R5, R4);
10.00397    CLRS(R4, 9 => R6);                              "Command code:
10.00398                                                    "Read status
10.00399
10.00400    REPEAT                                          "Await DIF not busy
10.00401       SIO(R0, R4);
10.00402    UNTIL NOT R0[15];
10.00403
10.00404 %WHEN S = TRUE SKIP
10.00405 "====== Patch for coping with time error on CMD disks"
10.00406
```

```
10.00407    #C3FF " = 1100.0011.1111.1111 " => R6;
10.00408
10.00409    IF R6 AND R0 = #0309
10.00410    THEN
10.00411    BEGIN
10.00412       WAIT_CYCLES => R6;
10.00413       WHILE R6 - 1 >= 0 DO;
10.00414
10.00415       SETS(R4, 9);                          "Repeat command
10.00416       CIO(R5, R4);
10.00417       CLRS(R4, 9 => R6);
10.00418
10.00419       REPEAT
10.00420          SIO(R0, R4);
10.00421       UNTIL NOT R0[15];
10.00422    END;
10.00423
10.00424 "====== end of patch"
10.00425
10.00426    R0 => R1 EXTRACT 10 SHIFTRL 8;           "R1 = Phase
10.00427    R0 => R5 EXTRACT  6 SHIFTRL 4;           "R5 = Result
10.00428    R0 => R2 EXTRACT  4;                     "R2 = Condition
10.00429
10.00430    IF NOT R0[14] LOGAND                     "Not illegal command
10.00431       NOT R0[7]  LOGAND                     "Seek not initiated
10.00432       R5 >= 2                               "Result = normal ending
10.00433    THEN
10.00434       OK => R6 => SAVEREGS[2]
10.00435    ELSE
10.00436       IF R5 = 0 LOGAND R2 < 2 LOGOR         "Drive busy or seeking
10.00437          R5 = 1 LOGAND R1 = 0 LOGOR         "Drive busy
10.00438          R0[7]                              "DIF has initiated a seek
10.00439       THEN
10.00440       BEGIN
10.00441          UNIT => R7 + 10;                   "Await unit flag
10.00442
10.00443          WHILE NOT R0[R7] DO
10.00444          BEGIN
10.00445             SIO(R0, R4);
10.00446          END;
10.00447
10.00448          OK => R6 => SAVEREGS[2];
```

```
10.00449        END
10.00450        ELSE
10.00451        BEGIN
10.00452           IF RO[14]                              "Illegal command
10.00453           THEN OUTCHAR('M' => R3, R6)
10.00454           ELSE
10.00455              IF RO[6]                            "Write protected
10.00456              THEN OUTCHAR('W' => R3, R6)
10.00457              ELSE
10.00458                 IF R5 = 1                        "Operation terminated
10.00459                 THEN OUTCHAR('T' => R3, R6)
10.00460                 ELSE OUTCHAR('C' => R3, R6);
10.00461           OUTHEX(R2 => R3, R6);
10.00462           OUTCHAR('D' => R3, R6);
10.00463           OUTHEX(RO => R3, R6);
10.00464           OUTNEWLINE(R6);
10.00465           R2 => SAVEREGS[2];
10.00466        END;
10.00467 %COMPILE
10.00468 %WHEN S = FALSE SKIP
10.00531
10.00532     UNS(7, ADDRESS(SAVEREGS[0]) => R7);
10.00533     EXIT(R6);
10.00534 END; "PROCEDURE CONTROL_IO"
10.00535
```

```
10.00536
10.00536 EXPORT PROCEDURE INITIATE_DISK_IO(R1;  " Unit
10.00537                                   R3;  " Disk controller memory section
10.00538                                   R4;  " Disk type
10.00539                                   R5;  " Disk controller absolute word address in memory
10.00540                                   R6);  " Link
10.00541 "-------------------------------------------------------------
10.00542 " Set disk unit characteristic values
10.00543 "-------------------------------------------------------------
10.00544 VAR
10.00545   SAVEREGS: ARRAY [0..7] OF INTEGER;
10.00546 BEGIN
10.00547   R7 => SAVEREGS[7];
10.00548   STC (6,ADDRESS(SAVEREGS[7])=>R7);
10.00549 %SKIP
10.00553   UPDATE_DISK_PARAM(R4,R6);
10.00554   R5-(0&PROCESS_DESCRIPTOR.BASE=>R6);
10.00555   R5 => CONTROLLER_MEMORY_WORD_ADDRESS;       "  make R5 base relative
10.00556   SECTORS_PER_CYLINDER=>R6;
10.00557   0=>R7;
10.00558   R67 SHIFTRL 5;       " Compute
10.00559   R6 => R2;        " Tracks per cylinder
10.00560   GENERATE_PSW_IN_PAGE(R3,R4,R6);
10.00561   R4 => DISK_PAGE;
10.00562   GENERATE_BASIC_CTRLB(CF=>R0,R1,R2,R3,R4,R5,R6);
10.00563   GEN_ERRECOV_CTRLB(R1,R3,R5,R6);
10.00564   GENERATE_BUFFER_CHAIN(R3,R5,R6);
10.00565    DISK_IO_ADDRESS=>R4;
10.00566    SETS (R4,15);       " Disable OPC interrupt
10.00567    IF R3[1] THEN SETS (R4,6);
10.00568    IF R3[0] THEN SETS (R4,7);
10.00569    CIO (R5,R4);       " Reset
10.00570   UNS (7,ADDRESS(SAVEREGS[0])=>R7);
10.00571   EXIT(R6);
10.00572 END " Disk I/O initiation ";
10.00573
```

```
10.00574
10.00574 EXPORT PROCEDURE READ_SECTORS(R01;  " First sector
10.00575                               R2;   " Returned Completion Code
10.00576                               R4;   " NO of sectors
10.00577                                   " Returns : PSW for disk memory
10.00578                               R7;   " Returns : base relative address of first buffer
10.00579                               R6);  " Link
10.00580 "-------------------------------------------------------------
10.00581 " Read (R4) sectors beginning at (R01). Return : Controller Completion Code,
10.00582 " PSW for disk memory section and Start address of transfered data
10.00583 "-------------------------------------------------------------
10.00584 VAR
10.00585   SAVEREGS: ARRAY [0..7] OF INTEGER;
10.00586 BEGIN
10.00587   R7 => SAVEREGS[7];
10.00588   STC (6,ADDRESS(SAVEREGS[7])=>R7);
10.00589 %SKIP
10.00593   SET_READ_OP_CODE(R7,CYLINDER_AND_HEAD_SELECT=>R2,R6);
10.00594   CYL_HEAD_SECT(R01,SECTORS_PER_CYLINDER=>R2,FIRST_CYLINDER=>R3,R6);
10.00595   ASSIGN_CTRLB(R0,R1,R2,CONTROLLER_ADDRESS_MEMORY_SECTION=>R3,R4,
10.00596                CONTROLLER_MEMORY_WORD_ADDRESS=>R5,R7,R6);
10.00597   DISK_PAGE=>R4;
10.00598   R4 => SAVEREGS[4];  "SAVE PSW
10.00599   CONTROL_IO(R3,R5,DISK_IO_ADDRESS=>R4,R2,R6);   <- start disk I/F
10.00600   0 => R7;          " Counter
10.00601   WHILE  R7 < LIMIT  LOGAND  R2 <> OK
10.00602          DO BEGIN
10.00603             CONTROLLER_MEMORY_WORD_ADDRESS => R5 + SIZE(CTRLB);
10.00604             REPEAT
10.00605               CLEAR_WORD_2(R3,R5,R6);
10.00606               CONTROL_IO(R3,R5,R4,R2,R6);
10.00607               R7 + 1;
10.00608             UNTIL  R7 = LIMIT  LOGOR  R2 = OK;
10.00609             CONTROLLER_MEMORY_WORD_ADDRESS => R5;
10.00610             CLEAR_WORD_2(R3,R5,R6);
10.00611             CONTROL_IO(R3,R5,R4,R2,R6);          <-
10.00612             END;
10.00613   R5 + (2*SIZE(CTRLB)) => SAVEREGS[7];       " Address of first buffer
10.00614   R2 => SAVEREGS[2];  "SAVE CC
10.00615-> UNS (7,ADDRESS(SAVEREGS[0])=>R7);
10.00616   EXIT(R6);
10.00617 END " Read sectors ";
```

```
10.00618

======== LOADER.S

00.00019
00.00020 %SOURCE

======== BOOT.S

11.00001 "
```

```
11.00002
11.00002 "==================== B O O T ====================
11.00003
11.00004
11.00005
11.00006
11.00007
11.00008 LABEL
11.00009   NEXT_WORD;
11.00010
11.00011 CONST
11.00012   SLP = #108D;
11.00013   PARAM_NO =  5;
11.00014
11.00015 TYPE
11.00016   BOOT_HEADER = RECORD
11.00017                 SIZE_OF_LOAD_MODULE_IN_WORDS: INTEGER;  " Discluding header
11.00018                 BASE_OF_PROCESS_TO_BE_LOADED: INTEGER;
11.00019                 LOAD_ADDRESS                : INTEGER;
11.00020                 LOAD_PAGE                   : INTEGER;
11.00021                 CHECK_SUM                   : INTEGER;
11.00022                 NOTUSED                     : ARRAY[6..256] OF INTEGER;
11.00023             END;
11.00024
11.00025
11.00026 VAR
11.00027   COUNTER  : INTEGER;
11.00028   LAST_SECTORS : INTEGER;
11.00029   VALID_WORDS  : INTEGER;
11.00030   BUFFER_LINK             : INTEGER;
11.00031   START_OF_LOAD_MODULE    : INTEGER;
11.00032   BS_PSW                  : INTEGER;
11.00033   NEXT_BS_SECTOR          : LONG;
11.00034   BS_CHECK_SUM            : INTEGER;
11.00035   CHECK_SUM               : INTEGER;
11.00036
11.00037 EXPORT VAR
11.00038   SET_PAR                 : BOOLEAN;
11.00039   PARITY_ERROR_FLAG       : BOOLEAN;
11.00040
11.00041 "
```

```
11.00042
11.00042 PROCEDURE INBOOT(R7;R6);
11.00043                        " R7 INPUT PARAMETER INDEX
11.00044                        " R6 LINK
11.00045 "-------------------------------------------------------------------
11.00046 " The routine handles input parameters to the boot procedure,
11.00047 " one at a time.
11.00048 "-------------------------------------------------------------------
11.00049
11.00050 TYPE
11.00051     INPUT_PARAMETER = 1..5;
11.00052     KIND_OF_DRIVE   = 0..18;
11.00053 VAR
11.00054   BLANK : BOOLEAN;          " FLAG INDICATING HEX INPUT DELIMITER
11.00055 BEGIN
11.00056   R7 => SAVEREGS[7];
11.00057   STC (6,ADDRESS(SAVEREGS[7])=>R7);
11.00058   SAVEREGS[7] => R7;
11.00059 %SKIP
11.00063   TRUE=>R3=>BLANK;
11.00064   CASE R7: INPUT_PARAMETER OF
11.00065        1: BEGIN     "disk controller i/o address
11.00066           GET_HEXA(R2,R1,R6);
11.00067           IF R2>>=#003F LOGOR R1=TRUE THEN ERROR(R7,R6);
11.00068           R2=>DISK_IO_ADDRESS;
11.00069           IF LAST_IN=>R3<>' ' THEN FALSE=>R3=>BLANK;
11.00070           END;
11.00071        2: BEGIN             " UNIT  "
11.00072           LAST_IN=>R3;
11.00073           IF R3-'0' >>= 4 THEN ERROR(R7,R6);
11.00074           R3=>UNIT;
11.00075           END;
11.00076        3: BEGIN             " KIND OF DRIVE "
11.00077           GET_HEXA(R2,R1,R6);
11.00078           IF R2>>=#0013 LOGOR R1=TRUE THEN ERROR(R7,R6);
11.00079           IF LAST_IN=>R3<>' ' THEN FALSE=>R3=>BLANK;
11.00080           0 => R4 => FIRST_CYLINDER;
11.00081           IF R2 = 0 LOGOR R2 = 2 LOGOR R2 = 4 LOGOR R2 = 6 THEN 896 => R4 => FIRST_CYLINDER;
11.00082           R2 => DISK_TYPE;
11.00083           END;
11.00084        4: BEGIN             " CONTROLLER MEMORY ADDRESS : PAGE
11.00085           LAST_IN=>R3-'C';
```

```
11.00086            IF R3>>=4 THEN ERROR(R7,R6);
11.00087            R3=>CONTROLLER_ADDRESS_MEMORY_SECTION;
11.00088            END;
11.00089         OTHERWISE BEGIN            " CONTROLLER MEMORY ADDRESS : WORD ADDRESS
11.00090            GET_HEXA(R2,R1,R6);
11.00091            R2 => R3 EXTRACT 12;
11.00092            IF R1=TRUE LOGOR R2 <> R3 THEN ERROR(R7,R6);
11.00093            IF LAST_IN=>R3<>' ' THEN FALSE=>R3=>BLANK;
11.00094            R2=>CONTROLLER_MEMORY_WORD_ADDRESS;
11.00095            END;
11.00096    IF BLANK=>R3=TRUE THEN GET_NEXT_NON_BLANK(R3,R6);
11.00097    UNS (7,ADDRESS(SAVEREGS[0])=>R7);
11.00098    EXIT(R6);
11.00099 END"  INBOOT  ";
11.00100 "
```

```
11.00101
11.00101 PROCEDURE SET_PARITY(R6);
11.00102 BEGIN
11.00103   R7 => SAVEREGS[7];
11.00104   STC (6,ADDRESS(SAVEREGS[7])=>R7);
11.00105 %SKIP
11.00109   0=>R3;                                  " R3 MEMORY SECTION
11.00110   -(C@PROCESS_DESCRIPTOR.BASE=>R5);       " R5 = POINTER TO MEMORY
11.00111   R5=>R1;
11.00112     TRUE => R7 => SET_PAR;
11.00113   SVS (R7);
11.00114   REPEAT
11.00115     R3 => R4 SHIFTLL 2 + #E000;
11.00116     LDS (R4);
11.00117     REPEAT
11.00118       R5@INTEGER=>R6;
11.00119       R5+1;
11.00120 NEXT_WORD:
11.00121     UNTIL R5=R1;
11.00122     R3+1;
11.00123   UNTIL R3=4;
11.00124   LDS (R7);
11.00125   FALSE => R7 => SET_PAR;
11.00126   IF PARITY_ERROR_FLAG => R6 = TRUE
11.00127   THEN
11.00128   BEGIN
11.00129     CUTTEXT(ADDRESS('PARITY ERROR(:0:)') => R3, R6);
11.00130     CUTNEWLINE(R6);
11.00131   END;
11.00132   UNS (7,ADDRESS(SAVEREGS[0])=>R7);
11.00133   EXIT(R6);
11.00134 END  " SET_PARITY ";
11.00135 "
```

```
11.00136
11.00136 PROCEDURE MOVE_SECTOR(
11.00137                        R0;  " Check sum
11.00138                        R1;  " NO of words to be moved ( destroyed )
11.00139                        R4;  " PSW for input page
11.00140                        R7;  " Relative pointer to beginning of input sector
11.00141                             " At return points at 1'st location after sector
11.00142                        R2;  " PSW for output page
11.00143                        R3;  " Base relative pointer to output location
11.00144                             " At return points at next output location
11.00145                        R6); " Link
11.00146 VAR
11.00147   S5, S6   : INTEGER;
11.00148 BEGIN
11.00149   R5 => S5;
11.00150   R6 => S6;
11.00151 %SKIP
11.00155   SVS (R6);
11.00156   WHILE R1<>0 DO BEGIN
11.00157      R1 - 1;
11.00158      LDS (R4);
11.00159      R7&INTEGER => R5;
11.00160      LDS (R2);
11.00161      R5 => R3&INTEGER;
11.00162      R0 + R5;
11.00163      INCD(R3,R7);
11.00164    END;
11.00165   LDS (R6);
11.00166   S5 => R5;
11.00167   EXIT(S6);
11.00168 END "  Move sectors  " ;
11.00169
```

```
11.00170
11.00170 PROCEDURE NEXT_BUFFER (R3;  " Base relativ output location,
11.00171                         " at return base rel. addr. of next location
11.00172                     R4;  " PSW for disk memory section
11.00173                     R5;  " Number of buffers to be moved
11.00174                     R7;  " Base relativ buffer address
11.00175                     R6); " Link
11.00176 BEGIN
11.00177    R7 => SAVEREGS[7];
11.00178    STC(6, ADDRESS(SAVEREGS[7]) => R7);
11.00179    SAVEREGS[7] => R7;
11.00180 %SKIP
11.00184    REPEAT
11.00185       SVS (R6);
11.00186       LDS (R4);
11.00187       R7@BUFFER.POINTER => R2;          " Absolute address of next buffer
11.00188       ADDRESS(R7@BUFFER.SECTOR) => R7;
11.00189       LDS (R6);
11.00190       R2 - (0@PROCESS_DESCRIPTOR.BASE => R6) => BUFFER_LINK;     " Compensate base
11.00191       BS_PSW => R2;
11.00192       CHECK_SUM => R0;
11.00193       MOVE_SECTOR(R0,256=>R1,R4,R7,R2,R3,R6);
11.00194       R0 => CHECK_SUM;
11.00195       BUFFER_LINK => R7;                          " Relative address
11.00196       R5 - 1;
11.00197    UNTIL R5 = 0;
11.00198    R3 => SAVEREGS[3];
11.00199    UNS(7, ADDRESS(SAVEREGS[0]) => R7);
11.00200    EXIT(R6);
11.00201 END;
11.00202 "
```

```
11.00203
11.00203 PROCEDURE READ_BODY(R01;R2;R3;R4;R7;R5;R6);
11.00204                     " R4  PSW for disk memory section
11.00205                     " R7  Points at beginning of input buffer
11.00206                     " R2  Return code from READ SECTOR.
11.00207                     " RC1 Returns body address
11.00208                     " R5  Returns random file area size
11.00209 "-------------------------------------------------------------
11.00210 " The procedure understands the structure of a BFD entry, and reads in
11.00211 " the first sector of a (random i.e. index) file
11.00212 "-------------------------------------------------------------
11.00213 VAR
11.00214     LINK: INTEGER;
11.00215     "* S3: INTEGER;
11.00216 BEGIN
11.00217#324  R6 =>LINK;
11.00218 %SKIP
11.00222             SVS (R6);
11.00223             LDS (R4);
11.00224             ADDRESS(R7@BUFFER.SECTOR) => R7;
11.00225             R7@BFDENTRY.ORGANIZATION=>R3;
11.00226             R7@BFDENTRY.AREASIZE=>R5;
11.00227             R7@BFDENTRY.BODYADDR=>R01;
11.00228             LDS (R6);
11.00229             "* R3 => S3;
11.00230             "* OUTHEX(R3,R6);
11.00231             "* OUTHEX(R5=>R3,R6);
11.00232             "* OUTHEX(R1=>R3,R6);
11.00233             "* OUTHEX(R0=>R3,R6);
11.00234             "* OUTNEWLINE(R6);
11.00235             "* S3 => R3;
11.00236             1=>R4;       " NO OF SECTORS
11.00237             READ_SECTORS(R01,R2,R4,R7,R6);
11.00238     EXIT(LINK);
11.00239 END;
11.00240 "
```

~ pg. 3 60 X C

r3 Rekn organisation

PSW := E00C 943

732

MOD R7
MOVC 17 R7
MOV 3.X7 → R3

946 + BASE = 26 pg 3.

Her loeses #0006

r5 = 2 ? = #30.

r0 = 1 , r1=0 men for r0=0, r1=0

```
11.00241
11.00241 PROCEDURE LOAD
11.00242 "============"
11.00243           (R6); "LINK
11.00244 VAR
11.00245    SAVEREGS: ARRAY[0..7] OF INTEGER;
11.00246
11.00247 BEGIN
11.00248    R7 => SAVEREGS[R7];
11.00249    STC(6, ADDRESS(SAVEREGS[7])=> R7);
11.00250 %SKIP
11.00254                                        "* OUTNEWLINE(R6);
11.00255                                        "* OUTTEXT(ADDRESS('CTRLR  UNIT  KIND-OF   CTRLR-MEM-ADR  BOOT-STRAP(:0:)')=>R3,R6);
11.00256                                        "* OUTNEWLINE(R6);
11.00257                                        "* OUTTEXT(ADDRESS('IO-ADR        DRIVE    PAGE  WRD-ADR  BFD-ENTRY(:0:)')=>R3,R6);
11.00258                                        "* OUTNEWLINE(R6);
11.00259                                        "* OUTHEX(DISK_IO_ADDRESS=>R3,R6);
11.00260                                        "* OUTTEXT(ADDRESS('    (:0:)')=>R3,R6);
11.00261                                        "* OUTHEX(UNIT=>R3,R6);
11.00262                                        "* OUTTEXT(ADDRESS('      (:0:)')=>R3,R6);
11.00263                                        "* OUTHEX(DISK_TYPE=>R3,R6);
11.00264                                        "* OUTTEXT(ADDRESS('      (:0:)')=>R3,R6);
11.00265                                        "* OUTHEX(CONTROLLER_ADDRESS_MEMORY_SECTION=>R3,R6);
11.00266                                        "* OUTTEXT(ADDRESS('    (:0:)')=>R3,R6);
11.00267                                        "* OUTHEX(CONTROLLER_MEMORY_WORD_ADDRESS=>R3,R6);
11.00268                                        "* OUTTEXT(ADDRESS('    (:0:)')=>R3,R6);
11.00269                                        "* OUTHEX(BOOT_STRAP_BFD_ENTRY_NO=>R3,R6);
11.00270                                        "* OUTNEWLINE(R6);
11.00271    SET_PARITY(R6);
11.00272    UNIT => R1;
11.00273    CONTROLLER_ADDRESS_MEMORY_SECTION => R3;
11.00274    CONTROLLER_MEMORY_WORD_ADDRESS => R5;
11.00275    DISK_TYPE=>R4;
11.00276    INITIATE_DISK_IO(R1,R3,R4,R5,R6);
11.00277    0=>R0=>R1;  " 1'ST SECTOR
11.00278    1=>R4;      " NO OF SECTORS
11.00279    READ_SECTORS(R01,R2,R4,R7,R6);
11.00280    IF R2 <> OK THEN ERROR(#101=>R7,R6);
11.00281                                            "* OUTNEWLINE(R6);
11.00282                                            "* OUTCHAR('K'=>R3,R6);
11.00283                                            "* OUTTEXT(ADDRESS(' R7 = (:0:)')=>R3,R6);
11.00284                                            "* OUTHEX(R7=>R3,R6);
```

r7 = #932 = #12 abs.

read homeblock.

```
11.00285            SVS (R6);
11.00286            LDS (R4);
11.00287            ADDRESS(R7@BUFFER.SECTOR) => R7;
11.00288            R7@HOMEBLOCK.BFDADDR=>R01;
11.00289            LDS (R6);
11.00290
11.00291
11.00292
11.00293
11.00294 →          1=>R4;      " NO OF SECTORS
11.00295            READ_SECTORS(R01,R2,R4,R7,R6);
11.00296  IF R2 <> OK THEN ERROR(#102=>R7,R6);
11.00297
11.00298
R3,R6);
11.00299            "* OUTNEWLINE(R6);
11.00300            READ_BODY(R01,R2,R3,R4,R7,R5,R6);
11.00301  IF R2 <> OK THEN ERROR(#103=>R7,R6);
11.00302            "* OUTNEWLINE(R6);
11.00303            "* OUTTEXT(ADDRESS(' BFD INDEX SECTOR =  (:0:)')=>R3,R6);
11.00304            "* OUTTEXT(ADDRESS(' WHICH AREA, WHICH SECTOR IN AREA,  =  (:0:)')=>R3,R6);
11.00305            BOOT_STRAP_BFD_ENTRY_NO=>R0;
11.00306            0 => R1;
11.00307            R01 / R5;          " R0 <= which area.
11.00308                               " R1 <= which sector in area.
11.00309            "* OUTHEX(R0=>R3,R6);
11.00310            "* OUTHEX(R1=>R3,R6);
11.00311            "* OUTNEWLINE(R6);
11.00312            "* OUTTEXT(ADDRESS(' POINTING AT SECTOR NO  (:0:)')=>R3,R6);
11.00313            SVS (R6);
11.00314            LDS (R4);
11.00315            ADDRESS(R7@BUFFER.SECTOR) => R7;
11.00316            R7@INDEXSECTOR.INDEX[R0].LEAST=>R2;
11.00317            R7@INDEXSECTOR.INDEX[R0].MOST=>R3;
11.00318            LDS (R6);
11.00319  "*          OUTHEX(R3,R6);
11.00320  "*          R3 => R0;
11.00321  "*          OUTHEX(R2=>R3,R6);
11.00322  "*          R0 => R3;
11.00323  "*          OUTNEWLINE(R6);
11.00324            R1 => R0;
11.00325            0 => R1;
11.00326            READ_SECTORS(R01+R23,R2,1=>R4,R7,R6);
```

r7 = #943 = #23 abs ✓ OK her i Pkg HB.

check vord. af R7 = #26 pg 3

HOVL ⇒ 0 ⇒ r1 2 ⇒ R0

"* OUTTEXT(ADDRESS(' R7= ,BFDADDR = (:0:)')=>R3,R6);
"* OUTHEX(R7=>R3,R6);
"* OUTHEX(R1=>R3,R6);
"* OUTHEX(R0=>R3,R6);

r0 = 2, r1 = 0, r4 = 1

r7 = #932 ok

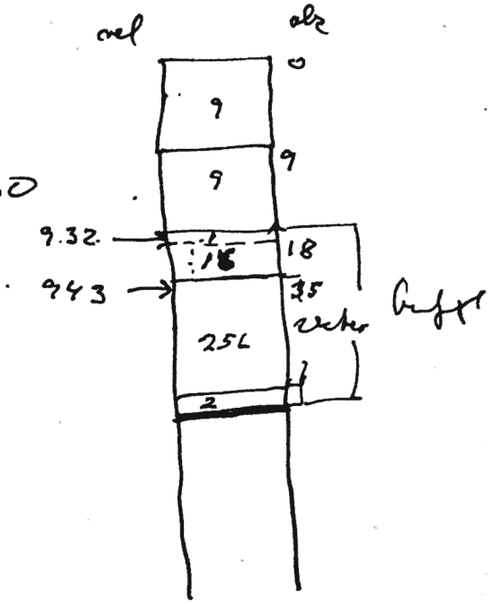
"* OUTNEWLINE(R6);
"* OUTTEXT(ADDRESS(' ORGANIZATION AREA SIZE BODY ADDRESS (:0:)')=>

r7 = #932, r3 = #E, r0 = 1, r1 = 0, r5 = #30

læsning af bfd index

IDENTIFIKALD R7:=

r0 = 0, r1 = 5, r5 = #30



check R7 := 943

r2 = 2, r3 = 0

Her læses code = E.

r0 = 7, r1 = 0, r7 = #932

```
11.00327   IF R2 <> OK THEN ERROR(#104=>R7,R6);
11.00328 " ASSUMING CONTIG B S
11.00329              "* OUTNEWLINE(R6);
11.00330              "* OUTTEXT(ADDRESS(' BOOT STRAP B F D ENTRY (:0:)')=>R3,R6);
11.00331              "* OUTNEWLINE(R6);
11.00332              "* OUTTEXT(ADDRESS(' ORGANIZATION     AREA SIZE      BODY ADDRESS  (:0:)')=>R3,R6);
11.00333              "* OUTNEWLINE(R6);
11.00334              READ_BODY(R01,R2,R3,R4,R7,R5,R6);
11.00335   IF R2 <> OK THEN ERROR(#105=>R7,R6);
11.00336              IF R3 <> CONTIGUOUS THEN ERROR(#C=>R7,R6);
11.00337              1 => R2;          ↑ contiguous = #C
11.00338              0 => R3;
11.00339              R01 + R23;
11.00340              R0 => NEXT_BS_SECTOR.LEAST;
11.00341              R1 => NEXT_BS_SECTOR.MOST;
11.00342              SVS (R6);
11.00343              LDS (R4);
11.00344              ADDRESS(R7@BUFFER.SECTOR) => R7;
11.00345 """          R7@BOOT_HEADER.BASE_OF_PROCESS_TO_BE_LOADED=>R3;
11.00346              R7@BOOT_HEADER.LOAD_ADDRESS=>R0;     " ( absolute )
11.00347              R7@BOOT_HEADER.LOAD_PAGE=>R1;
11.00348              R7@BOOT_HEADER.CHECK_SUM=>R2;
11.00349              R7@BOOT_HEADER.SIZE_OF_LOAD_MODULE_IN_WORDS=>R4;
11.00350              LDS (R6);
11.00351 """          R3 => BASE_OF_BS_TO_BE_LOADED ;
11.00352              R2 =>BS_CHECK_SUM;
11.00353              0 => R5;
11.00354              R45 / (256=>R3);      " R4 = NO of full sectors in file
11.00355                                    " R5 = NO of valid words in last sector
11.00356              R5 => VALID_WORDS;
11.00357              0 => R5;
11.00358              R45 / (16 =>R2);      " R4 = NO of rounds of 16 full sectors
11.00359                                    " R5 = NO of full sectors to be transferred next (0..15)
11.00360              R4 => COUNTER;
11.00361              R5 => LAST_SECTORS;
11.00362              R1 => R3;      " Page
11.00363              GENERATE_PSW_IN_PAGE(R3,R4,R6);
11.00364              R4 => BS_PSW;
11.00365              R0 => R3 - (0@PROCESS_DESCRIPTOR.BASE => R6)
11.00366                 => START_OF_LOAD_MODULE;
11.00367              0 => R0 => CHECK_SUM ;
11.00368              NEXT_BS_SECTOR.LEAST => R0;
```

```
11.00369              NEXT_BS_SECTOR.MOST => R1;
11.00370
11.00371              WHILE (COUNTER => R6 >= 1) DO BEGIN
11.00372                READ_SECTORS(R01,R2,16=>R4,R7,R6);
11.00373                IF R2 <> OK THEN ERROR(#106=>R7,R6);
11.00374                16 => R5;
11.00375                 0 => R6;
11.00376                R01 + R56;
11.00377                NEXT_BUFFER(R3,R4,R5,R7,R6);
11.00378                COUNTER => R6 - 1 => COUNTER;
11.00379              END;
11.00380
11.00381              LAST_SECTORS => R5;
11.00382
11.00383              IF R5<>0 THEN BEGIN
11.00384                R5 => R4;
11.00385                READ_SECTORS(R01,R2,R4,R7,R6);
11.00386                IF R2 <> OK THEN ERROR(#107=>R7,R6);
11.00387                 0 => R6;
11.00388                R01 + R56;
11.00389                NEXT_BUFFER(R3,R4,R5,R7,R6);
11.00390              END;
11.00391
11.00392              IF VALID_WORDS => R6 >= 1 THEN BEGIN
11.00393                READ_SECTORS(R01,R2,1=>R4,R7,R6);
11.00394                IF R2 <> OK THEN ERROR(#108=>R7,R6);
11.00395              END;
11.00396
11.00397              SVS (R6);
11.00398              LDS (R4);
11.00399              ADDRESS(R7&BUFFER.SECTOR) => R7;
11.00400              LDS (R6);
11.00401              BS_PSW => R2;
11.00402              CHECK_SUM => R0;
11.00403              MOVE_SECTOR(R0,VALID_WORDS=>R1,R4,R7,R2,R3,R6);
11.00404              "* OUTNEWLINE(R6);
11.00405              "* OUTTEXT(ADDRESS('CHECK SUM =(:0:)')=>R3,R6);
11.00406              "* OUTHEX(R0=>R3,R6);
11.00407              IF R0 = (BS_CHECK_SUM => R1)
11.00408              THEN BEGIN
11.00409                 START_OF_LOAD_MODULE=>R6;
11.00410                 WHILE R6&GENERAL_HEAD.KIND <> KPROCESS DO
```

```
11.00411                    R6 + R6@GENERAL_HEAD.WORD_SIZE;
11.00412                 R6 + SIZE(PROCESS_HEAD);
11.00413                 R6 + 0@PROCESS_DESCRIPTOR.BASE;
11.00414 %SKIP
11.00416                 LDP(R6);
11.00417 %SKIP
11.00421              END;
11.00422              ERROR(#111=>R7,R6);
11.00423   UNS (7,ADDRESS(SAVEREGS[0])=>R7);
11.00424   EXIT(R6);
11.00425 END; "PROCEDURE LOAD"
11.00426 "
```

```
11.00427
11.00427 EXPORT PROCEDURE BOOT(
11.00428                R6);     "  LINK
11.00429 VAR
11.00430   SAVEREGS: ARRAY [0..7] OF INTEGER;
11.00431
11.00432 BEGIN
11.00433   R7 => SAVEREGS[7];
11.00434   STC (6,ADDRESS(SAVEREGS[7])=>R7);
11.00435 %SKIP
11.00439   0=>R7 => UNIT;
11.00440   50=>R3=>DISK_IO_ADDRESS;
11.00441   SMD80R=>R4=>DISK_TYPE;
11.00442   R7=>CONTROLLER_MEMORY_WORD_ADDRESS;
11.00443 %WHEN S=TRUE SKIP
11.00444   3 => R3;
11.00445 %COMPILE
11.00446 %WHEN S=FALSE SKIP
11.00449   R3=>CONTROLLER_ADDRESS_MEMORY_SECTION;
11.00450   GET_NEXT_NON_BLANK(R3,R6);
11.00451   REPEAT
11.00452     R7+1;
11.00453     IF LAST_IN=>R6=CR THEN PARAM_NO=>R7              " ASSIGN ALL DEFAULT VALUES
11.00454      ELSE IF LAST_IN=>R6=',' THEN BEGIN
11.00455                GET_NEXT_NON_BLANK(R3,R6);
11.00456                IF R3=CR THEN PARAM_NO=>R7
11.00457                 ELSE IF R3<>',' THEN INBOOT(R7,R6);
11.00458               END
11.00459            ELSE INBOOT(R7,R6);
11.00460    UNTIL R7=PARAM_NO;
11.00461   WHILE LAST_IN=>R3=',' LOGOR R3=CR  DO
11.00462                GET_NEXT_NON_BLANK(R3,R6);       " BFD entry
11.00463           GET_HEXA(R2,R1,R6);
11.00464           IF R1=TRUE THEN ERROR(6=>R7,R6);
11.00465           R2=>BOOT_STRAP_BFD_ENTRY_NO;
11.00466    LOAD(R6);
11.00467    UNS(7, ADDRESS(SAVEREGS[0]) => R7);
11.00468    EXIT(R6);
11.00469 END; "PROCEDURE BOOT"
11.00470 "
```

```
11.00471
11.00471 EXPORT PROCEDURE PERFORM_AUTO_LOAD
11.00472 "================================"
11.00473                    (R6); "LINK
11.00474 CONST
11.00475    AUTO_LOAD_DISK_IO_ADDRESS           = #32;
11.00476    AUTO_LOAD_DISK_UNIT                 = 0;
11.00477    AUTO_LOAD_CONTROLLER_MEMORY_SECTION
11.00478 %WHEN S = TRUE SKIP
11.00479                                        = 3;
11.00480 %COMPILE
11.00481 %WHEN S = FALSE SKIP
11.00484    AUTO_LOAD_CONTROLLER_MEMORY_ADDRESS = 0;
11.00485 VAR
11.00486    S3, S6: INTEGER;
11.00487
11.00488 BEGIN
11.00489    R3 => S3; R6 => S6;
11.00490 %SKIP
11.00494    OUTCHAR('B' => R3, R6); OUTCHAR(',' => R3, R6);
11.00495    AUTO_LOAD_DISK_IO_ADDRESS           => R6 => DISK_IO_ADDRESS;
11.00496    OUTHEX(R6 => R3, R6); OUTCHAR(',' => R3, R6);
11.00497    AUTO_LOAD_DISK_UNIT                 => R6 => UNIT;
11.00498    OUTHEX(R6 => R3, R6); OUTCHAR(',' => R3, R6);
11.00499    AUTO_LOAD_KIND_OF_DRIVE             => R6 => DISK_TYPE;
11.00500    OUTHEX(R6 => R3, R6); OUTCHAR(',' => R3, R6);
11.00501    AUTO_LOAD_CONTROLLER_MEMORY_SECTION => R6 => CONTROLLER_ADDRESS_MEMORY_SECTION;
11.00502    OUTHEX(R6 => R3, R6); OUTCHAR(',' => R3, R6);
11.00503    AUTO_LOAD_CONTROLLER_MEMORY_ADDRESS => R6 => CONTROLLER_MEMORY_WORD_ADDRESS;
11.00504    OUTHEX(R6 => R3, R6); OUTCHAR(',' => R3, R6);
11.00505    AUTO_LOAD_BFD_ENTRY_NO              => R6 => BOOT_STRAP_BFD_ENTRY_NO;
11.00506    OUTHEX(R6 => R3, R6); OUTNEWLINE(R6);
11.00507    LOAD(R6);
11.00508    S3 => R3; S6 => R6;
11.00509    EXIT(R6);
11.00510 END; "PROCEDURE PERFORM_AUTO_LOAD"
11.00511

======== LOADER.S

00.00020
00.00021 %SOURCE

======== MAIN.S
```

12.00001 "

```
12.00002
12.00002 "==================== M A I N ====================
12.00003
12.00004
12.00005
12.00006
12.00007
12.00008 LABEL
12.00009   START;
12.00010
12.00011 CONST
12.00012   CPU_INTERRUPTS_ENABLED = 5;
12.00013   MASKED_PSW=#6000;
12.00014   DISABLE_INTERRUPTS = #E000;
12.00015   NOPRIT = TRUE;
12.00016   DEFAULTZERO = TRUE;
12.00017   TIMER_PRESET = 7;
12.00018   TIME_OUT = 30000;
12.00019
12.00020 TYPE
12.00021   SEMAPHORE = (FAILED,GOT_IT);
12.00022   ACTIVATION_CAUSES = (MASTERCLEAR, ROOT, EMERGENCY);
12.00023
12.00024 VAR
12.00025   RELEVANT_PSW: INTEGER;
12.00026   FIRST_TIME: BOOLEAN;
12.00027   ACTIVATION_CAUSE: ACTIVATION_CAUSES;
12.00028
12.00029 INIT
12.00030   FIRST_TIME = TRUE;
12.00031   BREAKADDR  = LOCATION(START);
12.00032   ERRADDR    = LOCATION(START);
12.00033
12.00034 "
```

```
12.00035
12.00035 PROCEDURE INTERRUPT_HANDLING(R6);
12.00036
12.00037 "-------------------------------------------------
12.00038 " The procedure assumes no timeout nor parity interrupts in SET PARITY routine
12.00039 " between assigning SET_PAR & deassigning it, except in target loop.
12.00040 "-------------------------------------------------
12.00041 VAR
12.00042   SI1     : INTEGER;
12.00043 BEGIN
12.00044   R1=>SI1;
12.00045   CASE 0@PROCESS_DESCRIPTOR.LOCAL_CAUSE=>R1: ACTION_CAUSES   OF
12.00046       CTIMER:   BEGIN
12.00047                   SI1=>R1;
12.00048                   SVP;
12.00049                   TIMER => R1 + 1 => TIMER;
12.00050                   0@PROCESS_DESCRIPTOR.PSW=>R0;
12.00051                   CLRS (R0,11=>R4);
12.00052                   R0=>0@PROCESS_DESCRIPTOR.PSW;
12.00053                   0=>R4;
12.00054                   R4@PROCESS_DESCRIPTOR.LOCAL_RETURN=>R0+R4@PROCESS_DESCRIPTOR.PROG=>R4@PROCESS_DESCRIPTOR.PRPC;
12.00055                   R4@PROCESS_DESCRIPTOR.BASE=>R0;
12.00056                   LDN (R0);
12.00057                 END;
12.00058       CPARITY:  IF SET_PAR => R1 = TRUE
12.00059                 THEN
12.00060                 BEGIN
12.00061                   R1 => PARITY_ERROR_FLAG;
12.00062                   SVS (R1);
12.00063                   LDS (R4);
12.00064                   R6 => R5@INTEGER;
12.00065                   LDS (R1);
12.00066                 END
12.00067                 ELSE
12.00068                 BEGIN
12.00069                   LDS(MASKED_PSW => R7);
12.00070                   ERROR(#41 => R7, R6);
12.00071                 END;
12.00072       CTIMEOUT: IF SET_PAR => R1 = TRUE
12.00073                 THEN
12.00074                 BEGIN
12.00075                   0@PROCESS_DESCRIPTOR.BASE => R6;
```

```
12.00076                    R5 + R6;
12.00077                    R5 AND (#F800 => R1) + 2048;
12.00078                    R5 - R6;
12.00079                    LOCATION(NEXT_WORD)=>R1=>0@PROCESS_DESCRIPTOR.LOCAL_RETURN;
12.00080                  END
12.00081                  ELSE
12.00082                  BEGIN
12.00083                    LDS(MASKED_PSW => R7);
12.00084                    ERROR(#42 => R7, R6);
12.00085                  END;
12.00086    END;
12.00087    SI1=>R1;
12.00088    0@PROCESS_DESCRIPTOR.LOCAL_RETURN=>R6;
12.00089    LDS(R4);
12.00090    EXIT(R6);
12.00091 END " INTERRUPT HANDLING " ;
12.00092 "
```

```
12.00093
12.00093 PROCEDURE GET_FIRST_ADDRESS(R5;R6);
12.00094                                        " R6   LINK
12.00095                                        " R5   RETURNED START ADDRESS
12.00096                                        " GLOBAL : RELEVANT_PSW
12.00097
12.00098 " SETS R5 TO HOLD START ADDRESS ENTERED BY USER IN THE FORMAT <START ADDRESS>
12.00099 "<START ADDRESS>::=<HEX NO>[.<OFF SET>]   <OFF SET>::=<HEX NO>
12.00100 " SETS PAGE BITS IN PSW ACCORDING TO USER PAGE INPUT
12.00101
12.00102
12.00103 BEGIN
12.00104   R7 => SAVEREGS[7];
12.00105   STC(6,ADDRESS(SAVEREGS[7])=>R7);
12.00106 %SKIP
12.00110
12.00111 " GET PAGE
12.00112
12.00113 %WHEN DEFAULTZERO = TRUE SKIP
12.00117 %WHEN DEFAULTZERO = FALSE SKIP
12.00118   IN_BYTE(R3,R6);
12.00119   IF R3=' ' THEN '0'=>R3;
12.00120 %COMPILE
12.00121   IF R3-'0' >>= 4 THEN ERROR(#10=>R7,R6);
12.00122   GENERATE_PSW_IN_PAGE(R3,R4,R6);
12.00123   R4=>RELEVANT_PSW;                                    " RELEVANT PSW
12.00124
12.00125
12.00126 " GET ADDRESS
12.00127   REPEAT GET_NEXT_NON_BLANK(R3,R6) UNTIL R3 <> ',';
12.00128
12.00129    GET_HEXA(R2,R1,R6);
12.00130    IF R1=TRUE  "  Invalid hexadecimal number  "  THEN ERROR(#13=>R7,R6);
12.00131   R2=>R5;                                                " START ADDRESS
12.00132   R5-(0&PROCESS_DESCRIPTOR.BASE=>R4);                                  " COMPENSATE OWN BASE
12.00133
12.00134
12.00135 " GET OFFSET
12.00136
12.00137    LAST_IN=>R3;
12.00138    IF R3=' '  THEN GET_NEXT_NON_BLANK(R3,R6);
12.00139
```

```
12.00140   IF R3 = '.' THEN     " Offset
12.00141     BEGIN
12.00142       GET_NEXT_NON_BLANK(R3,R6);
12.00143       GET_HEXA(R2,R1,R6);
12.00144       IF R1=TRUE THEN ERROR(#14=>R7,R6);          " INVALID
12.00145 "*        OUTTEXT(ADDRESS('INVALID HEX OFFSET.REENTER.(:0:)')=>R3,R6);
12.00146       R5+R2;                        " UPDATE START ADDRESS
12.00147     END;
12.00148   R5 => SAVEREGS[5];
12.00149   UNS (7,ADDRESS(SAVEREGS[0]) => R7);
12.00150  EXIT(R6);
12.00151 END" GET_START_ADDRESS ";
12.00152 "
```

```
12.00153
12.00153 PROCEDURE GET_LAST_ADDRESS(R2;R5;R6);
                                    " R6 LINK
12.00154                            " R5  Start address (input)
12.00155                            " R2  Last address (output)
12.00156                            " GLOBAL: LAST_IN
12.00157
12.00158
12.00159
12.00160 BEGIN
12.00161   R7 => SAVEREGS[7];
12.00162   STC(6,ADDRESS(SAVEREGS[7])=>R7);
12.00163 %SKIP
12.00167    LAST_IN=>R3;
12.00168    IF R3=' ' LOGOR R3=',' THEN REPEAT GET_NEXT_NON_BLANK(R3,R6) UNTIL R3<>',';
12.00169    IF R3 = CR THEN R5 => R2 + 1          " DEFAULT
12.00170     ELSE BEGIN
12.00171       IF  R3<>':' LOGANC R3<>'+'  THEN ERROR(#15=>R7,R6);
12.00172       R3 => R4;
12.00173       GET_NEXT_NON_BLANK(R3,R6);
12.00174       GET_HEXA(R2,R1,R6);
12.00175       IF R1 = TRUE THEN ERROR(#16=>R7,R6);
12.00176       R2 + 1;
12.00177       IF R4=':' THEN R2-0@PROCESS_DESCRIPTOR.BASE
12.00178        ELSE R2+R5;
12.00179     END;
12.00180
12.00181   R2 => SAVEREGS[2];
12.00182   UNS (7,ADDRESS(SAVEREGS[0]) => R7);
12.00183  EXIT(R6);
12.00184 END" GET NUMBER OF WORDS ";
12.00185 "
```

```
12.00186
12.00186 PROCEDURE DUMP(R6);
12.00187                        " R6 LINK
12.00188    " R1 SERVES AS WORD COUNTER
12.00189    " R5 POINTS AT MEMORY ADDRESS BEING DUMPED
12.00190
12.00191 " DUMPS MEMORY CONTENTS ACCORDING TO ADDRESS DATA ENTERED BY USER IN THE FORMAT
12.00192 " '<START ADDRESS> :<ENDING ADDRESS>'     (<ENDING ADDRESS>::=<HEX NO>)
12.00193 " '<START ADDRESS> +<NO OF WORDS>'        (<NO OF WORDS>::=<HEX NO> )
12.00194
12.00195  VAR
12.00196   SAVEREGS: ARRAY[0..7] OF INTEGER;
12.00197
12.00198  BEGIN
12.00199   R7 => SAVEREGS[7];
12.00200   STC(6,ADDRESS(SAVEREGS[7])=>R7);
12.00201 %SKIP
12.00205
12.00206    GET_FIRST_ADDRESS(R5,R6);
12.00207
12.00208    GET_LAST_ADDRESS(R2,R5,R6);
12.00209
12.00210    R5 AND (#FFF8=>R3);                          " Round start address
12.00211
12.00212   REPEAT
12.00213     R5=>R0;                                                " EDITING
12.00214     R0 EXTRACT 3;                                          "
12.00215     IF R0=0 THEN BEGIN                                     "
12.00216        CUTNEWLINE(R6);                                     "
12.00217        C@PROCESS_DESCRIPTOR.BASE=>R6;                                  "
12.00218        R5=>R3+R6;                                          " R3 = REAL DUMP ADDRESS
12.00219        CUTHEX(R3,R6);                                       "
12.00220        CUTCHAR(':'=>R3,R6);
12.00221        CUTCHAR(' '=>R3,R6);
12.00222        END                                                  "
12.00223      ELSE IF R0=4 THEN CUTCHAR(' '=>R3,R6);                 "
12.00224     SVS (R7);                                              " SAVE OWN PSW
12.00225     LDS (RELEVANT_PSW=>R4);                                      " LOAD RELEVANT PSW
12.00226     R5@INTEGER=>R3;
12.00227     LDS (R7);                                              " REESTABLISH OWN PSW
12.00228     CUTHEX(R3,R6);
12.00229     CUTCHAR(' '=>R3,R6);
```

```
12.00230
12.00231      R5+1;                                   " UPDATE DUMPED ADDRESS
12.00232     UNTIL R5=R2;
12.00233
12.00234    UNS (7,ADDRESS(SAVEREGS[0]) => R7);
12.00235   EXIT(R6);
12.00236   END;  "DUMP
12.00237
```

```
12.00238
12.00238 PROCEDURE PATCH(R6);
12.00239                    " R6  LINK
12.00240                    " R5 HOLDS START ADDRESS
12.00241                    " R2 HOLDS PATCH DATA
12.00242 "    GLOBALS: PROMPT_MODE (BOOLEAN)
12.00243 "             USER'S_PSW  (INTEGER)
12.00244 "             PAGE        (INTEGER)
12.00245
12.00246  VAR
12.00247   SAVEREGS: ARRAY[0..7] OF INTEGER;
12.00248
12.00249  BEGIN
12.00250   R7 => SAVEREGS[7];
12.00251   STC(6,ADDRESS(SAVEREGS[7])=>R7);
12.00252 %SKIP
12.00256
12.00257    GET_FIRST_ADDRESS(R5,R6);
12.00258
12.00259 IF R3='I' THEN                                    "PATCH MULTIPLE
12.00260
12.00261    BEGIN
12.00262    GET_LAST_ADDRESS(R2,R5,R6);
12.00263    R2=>R7;                                          " LOOP CONTROL
12.00264    LAST_IN => R3;
12.00265    IF R3=' ' LOGOR R3=',' THEN REPEAT GET_NEXT_NON_BLANK(R3,R6) UNTIL R3<>',';
12.00266    GET_HEXA(R2,R1,R6);                             " R2 = PATCH PATTERN
12.00267    IF R1=TRUE "INVALID HEX DATA" THEN ERROR(#17=>R7,R6);
12.00268
12.00269 " CONFIRM
12.00270 %WHEN NOPRIT = TRUE SKIP
12.00282       REPEAT
12.00283         SVS (R6);                                  " SAVE OWN PSW
12.00284         RELEVANT_PSW=>R4;
12.00285         LDS (R4);                                  " LOAD USER'S PSW
12.00286         R2=>R5@INTEGER;                            " PATCH
12.00287         LDS (R6);                                  " LOAD OWN PSW
12.00288         R5+1;                                      " UPDATE PATCH ADDRESS
12.00289        UNTIL R5=R7;
12.00290   END
12.00291
12.00292 ELSE                              " P A T C H
```

```
12.00293  BEGIN
12.00294    REPEAT
12.00295       IF R3=' ' LOGOR R3=',' LOGOR R3=CR THEN REPEAT GET_NEXT_NON_BLANK(R3,R6) UNTIL R3<>',' LOGAND R3<>CR;
12.00296       GET_HEXA(R2,R1,R6);                              " R2 = PATCH PATTERN
12.00297       IF R1<>TRUE THEN
12.00298        BEGIN
12.00299              SVS (R6);
12.00300              LDS (RELEVANT_PSW=>R3);
12.00301 " EXPERIMENT NOTE COMMENT
12.00302              R5@INTEGER => R3;                          " STORE PATTERN WHERE R5 IS POINTING
12.00303              R2=>R5@INTEGER;                          " STORE PATTERN WHERE R5 IS POINTING
12.00304              LDS (R6);
12.00305              R5+1;
12.00306         END
12.00307        ELSE ERROR(#18=>R7,R6);
12.00308     UNTIL LAST_IN=>R3=CR;
12.00309   END;
12.00310   UNS (7,ADDRESS(SAVEREGS[0]) => R7);
12.00311  EXIT(R6);
12.00312 END;       " PATCH
12.00313 "
```

```
12.00314
12.00314 PROCEDURE EXECUTE(R6);
12.00315
12.00316 " X < PAGE > < BASE >
12.00317 " Execute a process in memory section PAGE, data start address BASE
12.00318
12.00319 "                                                    (PAGE,BASE) point to the location where the registers of the
12.00320 "                                                    new process are located, & taken from by the LDN instruction
12.00321
12.00322 BEGIN
12.00323 %SKIP
12.00327    GET_NEXT_NON_BLANK(R3,R6);
12.00328    GET_HEXA(R2,R1,R6);
12.00329    IF R1 = TRUE THEN ERROR(#11=>R7,R6)
12.00330     ELSE LDN (R2);
12.00331
12.00332 END;
12.00333 "
```

```
12.00334
12.00334 PROCEDURE RETURN_TO_ROOT(R6);
12.00335 VAR
12.00336   S3, S5, S6: INTEGER;
12.00337 BEGIN
12.00338   R3 => S3; R5 => S5; R6 => S6;
12.00339 %SKIP
12.00343   IF ACTIVATION_CAUSE => R6 = ROOT
12.00344   THEN
12.00345   BEGIN
12.00346     C@PROCESS_DESCRIPTOR.PROCESS_LINK => R5;
12.00347     R5 - O@PROCESS_DESCRIPTOR.BASE;
12.00348     R5@PROCESS_DESCRIPTOR.PROCESS_LINK => R5;
12.00349     OUTCHAR('R' => R3, R6);
12.00350     LON(R5);
12.00351   END;
12.00352
12.00353   S3 => R3; S5 => R5; S6 => R6;
12.00354   EXIT(R6);
12.00355 END; "RETURN TO ROOT"
12.00356 "
```

```
12.00357
12.00357 PROCEDURE IO_COMMAND(R6);
12.00358 BEGIN
12.00359   R7 => SAVEREGS[7];
12.00360   STC(6, ADDRESS(SAVEREGS[7]) => R7);
12.00361 %SKIP
12.00365   LAST_IN => R0;
12.00366   GET_NEXT_NON_BLANK(R3, R6);
12.00367
12.00368   IF (R0 = 'R') LOGAND (R3 = CR)
12.00369   THEN
12.00370     RETURN_TO_ROOT(R6)
12.00371   ELSE
12.00372   BEGIN
12.00373     1 => R5;      "NO OF REPEATS
12.00374     0 => R7;      "NO OF HEXNO ON CURRENT LINE
12.00375     GET_HEXA(R2, R1, R6);
12.00376     R2 => R4;
12.00377
12.00378     IF R1 = FALSE
12.00379     THEN
12.00380     BEGIN
12.00381       IF LAST_IN => R3 = ' '
12.00382       THEN
12.00383         GET_NEXT_NON_BLANK(R3, R6);
12.00384
12.00385       IF R3 = '+'
12.00386       THEN
12.00387       BEGIN
12.00388         GET_NEXT_NON_BLANK(R3, R6);
12.00389         GET_HEXA(R2, R1, R6);
12.00390         R2 => R5;
12.00391         IF LAST_IN => R3 = ' '
12.00392         THEN
12.00393           GET_NEXT_NON_BLANK(R3, R6);
12.00394       END;
12.00395
12.00396       IF R1 = FALSE
12.00397       THEN
12.00398       BEGIN
12.00399         IF R0 = 'R' LOGOR R0 = 'S'
12.00400         THEN
```

```
12.00401          BEGIN
12.00402            REPEAT
12.00403              R7 + 1 EXTRACT 3;
12.00404              IF R0 = 'R'
12.00405              THEN
12.00406                RIO(R2, R4)
12.00407              ELSE
12.00408                SIO(R2, R4);
12.00409
12.00410              OUTHEX(R2 => R3, R6);
12.00411              OUTCHAR(' ' => R3, R6);
12.00412              IF R7 = 0
12.00413              THEN
12.00414                OUTNEWLINE(R6);
12.00415            UNTIL R5 - 1 = 0;
12.00416          END
12.00417          ELSE
12.00418          BEGIN
12.00419            GET_HEXA(R2, R1, R6);
12.00420
12.00421            IF R1 = FALSE
12.00422            THEN
12.00423              REPEAT
12.00424                IF R0 = 'W'
12.00425                THEN
12.00426                  WIO(R2, R4)
12.00427                ELSE
12.00428                  CIO(R2, R4);
12.00429              UNTIL R5 - 1 = 0;
12.00430          END;
12.00431        END;
12.00432      END;
12.00433
12.00434      IF R1 = TRUE
12.00435      THEN
12.00436        ERROR(#7 => R7, R6);
12.00437    END;
12.00438
12.00439    UNS(7, ADDRESS(SAVEREGS[0]) => R7);
12.00440    EXIT(R6);
12.00441  END; "PROCEDURE IO_COMMAND"
12.00442  "
```

```
12.00443
12.00443 " MAIN
12.00444
12.00445
12.00446 BEGIN
12.00447   "R0 = ACTIVATION_CAUSE
12.00448   "R1 = EMERGENCY ACTION BASE
12.00449
12.00450   TIMER_PRESET => R3;
12.00451   R3 => 0@PROCESS_DESCRIPTOR.TIMER_PRESET;
12.00452   LDT(R3);
12.00453   LOCATION(INTERRUPT_HANDLING)=>R3=>0@PROCESS_DESCRIPTOR.LOCAL_ACTION;
12.00454   LDS (MASKED_PSW => R3);
12.00455
12.00456 " Await completion of masterclearing
12.00457   0 => R6;
12.00458   REPEAT UNTIL R6 - 1 = 0;.
12.00459
12.00460   7 => R6;
12.00461 " Reserve semaphore
12.00462   SWITCH RESS(R6@INTEGER,0):SEMAPHORE TO
12.00463     FAILED:  BEGIN
12.00464                LDM ( CPU_INTERRUPTS_ENABLED );
12.00465                WHILE R0 = R0 DO;
12.00466              END;
12.00467   END;
12.00468
12.00469   R0 => ACTIVATION_CAUSE;
12.00470
12.00471   IF R0 = ROOT
12.00472   THEN
12.00473     SET_DEVICE_TYPE(R6)
12.00474   ELSE
12.00475     INIT_IF(R6);
12.00476
12.00477   IF R0 = EMERGENCY
12.00478   THEN BEGIN
12.00479     CUTTEXT(ADDRESS('EMERGENCY ACTION BASE (:0:)')=>R3,R6);
12.00480     CUTHEX(R1=>R3,R6);
12.00481   END
12.00482   ELSE CUTTEXT(ADDRESS(HEADER) => R3, R6);
12.00483
```

```
12.00484   OUTNEWLINE(R6);
12.00485   OUTTEXT(ADDRESS('BASE: (:0:)') => R3, R6);
12.00486   0&PROCESS_DESCRIPTOR.BASE => R3;
12.00487   OUTHEX(R3, R6);
12.00488
12.00489 START:
12.00490 %SKIP
12.00494    OUTNEWLINE(R6);
12.00495    FALSE => R6 => SET_PAR;
12.00496    R6 => PARITY_ERROR_FLAG;
12.00497    OUTNEWLINE(R6);
12.00498    L_MAX => R6 => CH_COUNT;
12.00499    OUTCHAR('>'=>R3,R6);
12.00500
12.00501    IF (FIRST_TIME => R6 = TRUE) LOGAND
12.00502       (ACTIVATION_CAUSE => R0 <> ROOT)
12.00503    THEN
12.00504    BEGIN
12.00505       FALSE => R6 => FIRST_TIME;
12.00506
12.00507       SWITCH IS_THERE_ANY_INPUT(TIME_OUT => R3, R6)
12.00508              : BOOLEAN TO
12.00509          FALSE:
12.00510             PERFORM_AUTO_LOAD(R6);
12.00511       END; "OF SWITCH
12.00512
12.00513    END; "OF IF
12.00514
12.00515    GET_NEXT_NON_BLANK(R3,R6);
12.00516    IF R3 = 'D' THEN                              DUMP (R6)
12.00517    ELSE IF R3 = 'P' LOGOR R3 = 'I' THEN          PATCH(R6)
12.00518         ELSE IF R3 = 'B' THEN                    BOOT (R6)
12.00519              ELSE IF R3 = 'X' THEN               EXECUTE(R6)
12.00520                   ELSE IF R3 = 'Z' THEN          SET_PARITY(R6)
12.00521                        ELSE IF R3 = 'R' LOGOR
12.00522                                R3 = 'W' LOGOR
12.00523                                R3 = 'S' LOGOR
12.00524                                R3 = 'C' THEN     IO_COMMAND(R6)
12.00525                             ELSE                 ERROR(#99=>R7,R6);
12.00526 GOTO START;
12.00527
12.00528 END" MAIN ";
```

```
12.00529

======== LOADER.S

00.00021
C0.00022 ENDMODULE
```

LINES: 3175

CODESIZE	VARSIZE	CONSTSIZE	TEMPSIZE
#0659	#00CC	#002D	#0000

SWELL VARIABLES:

```
==========BLOCK: 00.00005 LOADER
 00DA 0000 ACTIVATION_CAUSE
 0089 0000 BOOT_STRAP_BFD_ENTRY_NO
 0055 0000 BREAKADDR
 00BE 0000 BS_CHECK_SUM
 00BB 0000 BS_PSW
 00B9 0000 BUFFER_LINK
 0088 0000 CF
 00BF 0000 CHECK_SUM
 0056 0000 CH_COUNT
 0054 0000 CONSOLE_STATUS
 008A 0000 CONTROLLER_ADDRESS_MEMORY_SECTION
 008B 0000 CONTROLLER_MEMORY_WORD_ADDRESS
 0086 0000 COUNTER
 0082 0000 CYLINDER
 0053 0000 DEVICE_TYPE
 0085 0000 DISK_IO_ADDRESS
 0086 0000 DISK_PAGE
 0084 0000 DISK_TYPE
 007F 0000 ERRADDR
 0083 0000 FIRST_CYLINDER
 0080 0000 FIRST_SECTOR
 00D9 0000 FIRST_TIME
 0057 0000 LAST_IN
 00B7 0000 LAST_SECTORS
 002C 0000 LINE
 002E 0000 L_LENGTH
 00BC 0000 NEXT_BS_SECTOR
 00C1 0000 PARITY_ERROR_FLAG
 00D8 0000 RELEVANT_PSW
 0023 0000 SAVEREGS
 008C 0000 SECTORS_PER_CYLINDER
 00C0 0000 SET_PAR
 00BA 0000 START_OF_LOAD_MODULE
```

```
0058 0000 TIMER
0087 0000 UNIT
0088 0000 VALID_WORDS
```

SWELL CROSS REFERENCES:

INPUT_PARAMETER

```
USERID
                02.00146  05.00106
USER_NAME
                03.00137  03.00230  03.00277  03.00286
USER_ON_CMD
                03.00078  03.00131
V24
                07.00041  07.00083  07.00097  07.00121
VALID_WORDS
                11.00029  11.00356  11.00392  11.00403
VOLUME_NAME
                03.00154  03.00247  04.00077
VOL_NAME_SIZE
                03.00148  03.00154
WAIT_CYCLES
                10.00375  10.00412
WORD_SIZE
                02.00154  11.00411

 TOTAL NUMBER OF IDENTIFIERS:    647
```