PERFORMANCE REPORTING FACILITY

Version 1.0

EXTERNAL REFERENCE SPECIFICATION

Canadian Development Division
Performance Evaluation Unit

October 25, 1982
Revised
December 9, 1983
Revised
February 24,1984

1.0 PRF OVERVIEW

1.0 PRF OVERVIEW

The Performance Reporting Facility (PRF) is an integrated performance reporting package for NOS/VE. Some portions of PRF run on NOS/VE in 180 state while the rest of the package currently runs in 170 state under NOS/170. PRF is intended to be a modular system which can be extended into a full short and long term performance tracking system.

The current PRF is based on the PMF hardware option for CYBER 180. In the longer term, it will accept performance data from other sources and maintain a performance database.

1.1 PMF_HARDWARE

The Performance Monitoring Facility (PMF) is a CYBER 180 hardware option which provides a set of hardware counters and event buffers designed to collect performance data about the system. Significant hardware states and/or events may be timed and counted through PMF counters. Data on software events within NOS/VE and its product set may be obtained by inserting keypoint instructions in the software which may be collected through the PMF keypoint buffers. Because PMF uses a combination of hardware and software monitoring techniques, it is a good example of an integrated hybrid performance monitor.

1.1.1 PMF COUNTERS

PMF hardware contains eight 32-bit counters which may count hardware events and time system states. The list of events and states supported is model dependent and there is a selection code associated with each one. A limited amount of signal combination is allowed through A/B counter pairings and appropriate control codes. Event/state selection and signal combination are controlled through settings in register 22.

1.1.1.1 Event/State_List

To be supplied.

1.1.2 KEYPOINT INSTRUCTIONS

Keypoints are special 180 instructions which place an indication of significant software events into the FIFO buffer in PMF hardware or cause a keypoint trap to allow the data to be collected by software. These instructions act as NO-OPs to the NOS/VE software unless keypoint traps are enabled. In any system with the PMF hardware option, hardware collection rather than software collection should be used as there is almost no system overhead involved in gathering the performance data.

By convention, keypoints are used to mark the entry and exit

points of major O/S and product set routines, to flag unusual events of high importance, and to indicate significant events for debugging. This mechanism provides a very low overhead system tracing mechanism as well as a performance tracking facility.

1.1.3 PMF KEYPOINT BUFFER

PMF has a circular event buffer used to hold keypoint data for collection along with an associated time stamp. The exact size of this buffer is model-dependent as shown below:

825	16 keypoints
835	16 keypoints
855	16 keypoints
Theta	1024 keypoints

1.2 PMF DATA COLLECTION

PMF counter and keypoint data may be collected by the CPKPNOS program on NOS/170. CPKPNOS operates in one of two modes depending on whether keypoint collection is selected or not. If keypoint collection is not selected, CPKPNOS calls up a PP program (PMF) to initialize counter collection through register 22 and then the PP program drops out. CPKPNOS must then be called up again when the counters are to be read out. If keypoint collection is selected, the PP program will remain resident to continuously collect the contents of the keypoint buffer as it is being filled.

1.3 PMF DATA REDUCTION

There are currently two components of PMF data reduction, the Primary Data Reduction Package and the Call Analysis System. The Primary Data Reduction Package is designed to give overview information about system behaviour during the keypoint collection and the Call Analysis System gives microlevel timing information on the operation of procedures within NOS/VE and its product set.

2.0 KEYPOINT CONVENTIONS AND GUIDELINES

2.0 KEYPOINT CONVENTIONS AND GUIDELINES

The CYBER 180 System Interface Standard (SIS) describes conventions for the placement of keypoints within the O/S and product set members. Further documentation is found in the latest NOS/VE Helpful Hints document.

2.0 KEYPOINT CONVENTIONS AND GUIDELINES
2.1 KEYPOINT CLASSES

2.1 KEYPOINT CLASSES

The 180 hardware allows 16 keypoint classes and collection of keypoints is controlled by a keypoint mask in the task's exchange package. Only those keypoints whose class matches one of the classes set in the keypoint mask will be collected by either hardware or software collection. This allows the keypoint classes to define a collection hierarchy. The following table shows the keypoint class assignments.

Class	Used for
0	O/S data
1	O/S unusual events
2	O/S procedure entry
3	O/S procedure exit
4	O/S debug
5	O/S PAS (Performance Analysis Sevices)
6	Product set data
7	Product set unusual events
8	Product set procedure entry
9	Product set procedure exit
10	Product set debug
11-14	Reserved for end users
15	PMF hardware start/stop control

2.2 KEYPOINT IDENTIFIERS AND DATA FIELDS

Each keypoint instruction executed which provides data for hardware or software keypoint collection leaves a 32-bit data portion in the event in the collection buffer. By convention, this is divided into two sections, a 12-bit keypoint identifier and a 20-bit data field. The keypoint identifier is used in conjunction with the keypoint class to determine the actual event which triggered the keypoint instruction. The data field may be used to provide any information of interest associated with the keypoint event (e.g. the PTL ordinal of a task being dispatched). If twenty bits are not sufficient for the desired information, the DATA class allows this to be extended by up to thirty-two more bits to give a total of fifty-two bits of data. The DATA class should be used with caution as it may cause keypoint clustering resulting in an overflow of the 16 event FIFO buffer in PMF hardware. Also note that DATA class keypoints are not allowed in conjunction with UNUSUAL class keypoints.

2.3 CYBIL_KEYPOINT_INTRINSIC

CYBIL provides an intrinsic to generate keypoint instructions directly from CYBIL code. This intrinsic is used in conjunction with standard keypoint common deck declarations to generate the actual keypoint instructions present in NOS/VE and product set code. The form of the intrinsic is as follows:

```
#KEYPOINT(keypoint_class, osk$m * data, keypoint_id);
```

2.4 KEYPOINT COMMON DECK STRUCTURE

There are a standard set of common decks provided for use by O/S and product set developers in instrumenting their code. In addition, there are standard rules for adding keypoint description common decks for other areas which are being keypointed. The currently defined common decks are found in NOSVEPL and OSLPI for NOS/VE and in OSLPI and PSCOMPL for the product set.

2.4.1 KEYPOINT CLASS COMMON DECKS

There are three main common decks used to define the classes for a NOS/VE system, OSDKEYS in NOSVEPL, OSDKEYC in OSLPI, and PSDKEYC in PSCOMPL. OSDKEYS contains the master range declaration and OSDKEYC and PSDKEYC contain the actual class declarations for the O/S and product set respectively.

Listings of these three common decks follow:

OSDKEYS

```
  CONST

    {  Keypoint Classes :
    {
    {       The 16 keypoint classes supported by the hardware are partitioned
    {  between the System, Product Set and User as follows.

    osk$system_class = 0      {  0 ..  5 },
    osk$product_set_class = 6 {  6 .. 10 },
    osk$user_class = 11       { 11 .. 14 },
    osk$pmf_control = 15;


    {  Keypoint Multiplier:
    {
    {       By convention, the 32 bit keypoint code supported by the hardware
    {  is split into two fields.  The right field contains a keypoint
    {  identifier which is used to identify a function within a keypoint class.
    {  For example, if a particular keypoint class represents exit from a
    {  procedure, then the keypoint identifier might identify exit from
    {  procedure A versus exit from procedure B.
    {       The left field is used as a data parameter appropriate to the
    {  function identified by the keypoint identifier.  In the procedure exit
```

2.0 KEYPOINT CONVENTIONS AND GUIDELINES
2.4.1 KEYPOINT CLASS COMMON DECKS

```
{  example above, the data parameter field might be used to ind  ate the
{  status of the procedure call.
{       The keypoint multiplier is used to partition the keypoint code
{  into the two fields.  The data parameter should be multiplied by the
{  keypoint multiplier to prevent it from overlapping the keypoint
{  identifier field.

  CONST
    osk$m = 4096;
```

OSDKEYC

```
{Define KEYPOINT CLASS Codes.

  CONST
    osk$data = osk$system_class + 0, { OS - DATA keypoint}
    osk$unusual = osk$system_class + 1, {U OS - Unusual keypoint class.}
    osk$entry = osk$system_class + 2, {E OS - Standard keypoint (gated cr major internal)}
    osk$exit = osk$system_class + 3, {X OS - Exit keypoint}
    osk$debug = osk$system_class + 4, {D OS - Debug keypoint.}
    osk$reserved = osk$system_class + 5; {R OS - Reserved keypoint.}
```

PSDKEYC

To be supplied.

2.4.2 KEYPOINT IDENTIFIER COMMON DECKS

There are two common decks which declare the identifier ranges for the O/S and product sets. These are OSDKEYD in OSLPI and PSDKEYD in PSCOMPL for the O/S and product sets respectively. There is a required convention for all range definitions that they be of the form XXk$base for the O/S and YYk$psbase for the product set where XX and YY are the two character area designators for the O/S and product set respectively.

Listings of these common decks follow:

2.0 KEYPOINT CONVENTIONS AND GUIDELINES
2.4.2 KEYPOINT IDENTIFIER COMMON DECKS

OSDKEYD

```
{This deck defines constants for use with KEYPOINTS.

{Define base keypoint procedure identifiers for each area of the OS.

  CONST
    amk$base = 100, {100 - 149}
    bak$base = 150, {150 - 249}
    clk$base = 250, {250 - 299}
    cmk$base = 300, {300 - 349}
    dbk$base = 350, {350 - 399}
    dmk$base = 400, {400 - 549}
    fmk$base = 550, {550 - 599}
    ick$base = 600, {600 - 649}
    ifk$base = 650, {650 - 699}
    iik$base = 700, {700 - 749}
    ink$base = 750, {750 - 799}
    jmk$base = 800, {800 - 849}
    lgk$base = 850, {850 - 899}
    llk$base = 900, {900 - 949}
    lok$base = 950, {950 - 999}
    luk$base = 1000, {1000 - 1049}
    mik$base = 1050, {1050 - 1099}
    mmk$monitor_base = 1100, {1100 - 1149}
    mmk$job_base = 1150, {1150 - 1199}
    msk$base = 1200, {1200 - 1249}
    mtk$base = 1250, {1250 - 1299}
    ock$base = 1300, {1300 - 1349}
    ofk$base = 1350, {1350 - 1399}
    osk$base = 1400, {1400 - 1449}
    pfk$base = 1500, {1500 - 1549}
    pmk$base = 1600, {1600 - 1699}
    rhk$base = 1750, {1750 - 1799}
    srk$base = 1800, {1800 - 1819}
    stk$base = 1850, {1850 - 1899}
    tmk$monitor_base = 1900, {1900 - 1949}
    tmk$job_base = 1950, {1950 - 1999}
    jsk$monitor_base = 2000, {2000 - 2049}
    jsk$job_base = 2050, {2050 - 2099}
    avk$base = 2100, {2100 - 2149}
    sfk$base = 2150, {2150 - 2199}
    iok$base = 2200, {2200 - 2249}
    rmk$base = 2250, {2250 - 2299}
    dmk$tape_base = 2300, {2300 - 2349}
    syk$job_base = 2350, {2350 - 2399}
    mtk$assembly_language_base = 4000; {4000 - 4095}
{   OS assembly language      4000 - 4095}
```

2.0 KEYPOINT CONVENTIONS AND GUIDELINES
2.4.2 KEYPOINT IDENTIFIER COMMON DECKS

PSDKEYD

To be supplied.

2.4.3 KEYPOINT DEFINITION COMMON DECKS

Each area identified in OSDKEYD and PSDKEYD should have a keypoint definition common deck to give the exact keypoint codes and data format as well as special flags for the data reduction software. Each area has a two character string which identifies it. For the product set, this is the two character product identifier as described in the SIS. The standard convention for naming O/S keypoint definition common decks is xxDKEY where xx is the two character area name. The convention for those for the product set is xxDKEYP in a similar vein.

The following O/S and product set common deck listings show how these entries are set up.

SIDKEY

```
{ PURPOSE:
{   This deck contains all of the set manager keypoint constants.

  CONST
    {ENTRY/EXIT CLASS KEYPOINTS }
    {gated entry points for users}

    stk$create_set = stk$base + 1,
      {E  'stp$create_set' 'ring    ' H }
      {X  'stp$create_set' 'status  ' I20 }

    stk$purge_set = stk$base + 2,
      {E  'stp$purge_set' }
      {X  'stp$purge_set' 'status  ' I20 }

    stk$add_member_vol_to_set = stk$base + 3,
      {E  'stp$add_member_vol_to_set' }
      {X  'stp$add_member_vol_to_set' 'status  ' I20 }

    stk$remove_member_vol_from_set = stk$base + 4,
      {E  'stp$remove_member_vol_from_set' }
      {X  'stp$remove_member_vol_from_set' 'status  ' I20 }

    stk$change_access_to_set = stk$base + 5,
      {E  'stp$change_access_to_set' 'access  ' H20 }
```

```
    {X  'stp$change_access_to_set' 'status  ' I20 }

  {interfaces into sets from other functional areas }
  {calls from device management }

  stk$get_jobs_scratch_volumes = stk$base + 10,
    {E  'stp$get_jobs_scratch_volumes' }
    {X  'stp$get_jobs_scratch_volumes' 'status  ' I20 }

  stk$get_volumes_in_set = stk$base + 11,
    {E  'stp$get_volumes_in_set' 'setinit ' H20 }
    {X  'stp$get_volumes_in_set' 'status  ' I20 }

  stk$get_volumes_by_set_ordinal = stk$base + 12,
    {E  'stp$get_volumes_by_set_ordinal' }
    {X  'stp$get_volumes_by_set_ordinal' 'status  ' I20 }

  stk$get_volumes_set_name = stk$base + 13,
    {E  'stp$get_volumes_set_name' 'vsn     ' A32 }
    {X  'stp$get_volumes_set_name' 'status  ' I20 }

  stk$is_volume_in_set = stk$base + 14,
    {E  'stp$is_volume_in_set' }
    {X  'stp$is_volume_in_set' 'status  ' H20 }

  stk$disk_volume_active = stk$base + 15,
    {E  'stp$disk_volume_active' 'avtindx ' H20 }
    {X  'stp$disk_volume_active' 'status  ' I20 }

  stk$disk_volume_inactive = stk$base + 16,
    {E  'stp$disk_volume_inactive' 'setord  ' H20 }
    {X  'stp$disk_volume_inactive' 'status  ' I20 }

  {calls from permanent files }

  stk$get_pf_root_size = stk$base + 20,
    {E  'stp$get_pf_root_size' }
    {X  'stp$get_pf_root_size' 'status  ' I20 }

  stk$get_pf_root = stk$base + 21,
    {E  'stp$get_pf_root' }
    {X  'stp$get_pf_root' 'status  ' I20 }

  stk$store_pf_root = stk$base + 22,
    {E  'stp$store_pf_root' }
    {X  'stp$store_pf_root' 'status  ' I20 }

  stk$purge_pf_root = stk$base + 23,
    {E  'stp$purge_pf_root' }
```

```
    {X  'stp$purge_pf_root' 'status  ' H20 }

  stk$get_set_owner = stk$base + 24,
    {E  'stp$get_set_owner' }
    {X  'stp$get_set_owner' 'status  ' I20 }

  stk$set_pf_lock = stk$base + 25,
    {E  'stp$set_pf_lock' }
    {X  'stp$set_pf_lock' 'status  ' I20 }

  stk$clear_pf_lock = stk$base + 26,
    {E  'stp$clear_pf_lock' }
    {X  'stp$clear_pf_lock' 'status  ' I20 }

  {call from job management}

  stk$set_end_job = stk$base + 27,
    {E  'stp$set_end_job' }
    {X  'stp$set_end_job' 'status  ' I20 }

  {call during deadstart sequence }

  stk$initialize_sets = stk$base + 29,
    {E  'stp$initialize_sets' }
    {X  'stp$initialize_sets' 'status  ' I20 }

  stk$get_active_set_list = stk$base + 30,
    {E  'stp$get_active_set_list' 'setlis  ' I20 }
    {X  'stp$get_active_set_list' 'numset  ' I20 }

{UNUSUAL CLASS keypoints }

  stk$cant_dm_store_set_ord = stk$base,
    {U  'cant dmp$store_avt_set_ordinal' 'avtindx ' H20 }


{DEBUG keypoints }

  stk$set_exclusive_lock = stk$base,
    {D  'set exclusive access' }

  stk$clear_exclusive_lock = stk$base + 1,
    {D  'clear exclusive access' }

  stk$ast_index_assigned = stk$base + 2,
    {D  'active set table index assigned' 'astindx ' H20 }

  stk$new_job_accessing_set = stk$base + 3,
    {D  'first of access of sets within job' }
```

2.0 KEYPOINT CONVENTIONS AND GUIDELINES
2.4.3 KEYPOINT DEFINITION COMMON DECKS

```
      stk$mei_index_assigned = stk$base + 4,
        {D  'mei index assigned - ast index V' 'status  ' I20 }

      stk$pf_root_size = stk$base + 5;
        {D  'pf_root_size' 'rootsiz ' H20 }

?? PUSH (LIST := OFF) ??
```

EMDKEYP

To be supplied.

2.5 BNF KEYPOINT DESCRIPTION

```
<analyzer_descriptor_input> ::= <keypoint_class_allocation_deck>
                                [ <definition_deck> ... ]

<keypoint_class_allocation_deck> ::= <cybil code and/or comments>
                                     [ <class_base_definitions> ... ]
                                     <cybil code and/or comments>

<class_base_definitions> ::= <class_base_id> <spc> = <spc> <base>

<class_base_id> ::= osk$system_class ! osk$product_set_class !
                    osk$user_class ! osk$pmf_control

<spc> ::= <space> [ <space> ... ]          (one or more)

<base> ::= <integer>

<definition_deck> ::= <class_definition_deck> !
                      <base_definition_deck> !
                      <keypoint_definition_deck>

<class_definition_deck> ::= {$$$ START KEYPOINT CLASSES $$$}
                            <cybil code and/or comments>
                            [ <class_definitions> ... ]
                            <cybil code and/or comments>
                            {$$$ END KEYPOINT CLASSES $$$}

<class_definitions> ::= <keypoint_class> <spc> = <spc>
                        <class_base_id>  <offset>
                        { <keypoint_class_id> <cybil comment>

<keypoint_class> ::= <identifier>

<offset> ::= + <spc> <integer> <delimiter>

<delimiter> ::= , ! ;

<keypoint_class_id> ::= <character>

<base_definition_deck> ::=
                  {$$$ START KEYPOINT IDENTIFIER BASES $$$}
                  <cybil code and/or comments>
                  [ <range_base_definitions> ... ]
                  <cybil code and/or comments>
                  {$$$ END KEYPOINT IDENTIFIER BASES $$$}

<range_base_definitions> ::= <keypoint_base> <delimiter>
```

2.0 KEYPOINT CONVENTIONS AND GUIDELINES
2.5 BNF KEYPOINT DESCRIPTION

```
                                    <base_range>

    <keypoint_base> ::= <spc> <base_id> <spc> = <spc> <base>

    <base_id> ::= <identifier>

    <base_range> ::= <spc> { <low_base> <sp> - <high_base> [ } ]

    <low_base> ::= <integer>

    <high_base> ::= <integer>

    <keypoint_definition_deck> ::=
                               {$$$ START KEYPOINT DESCRIPTIONS $$$}
                               <cybil code and/or comments>
                               [ <xxdkey_deck> ... ]
                               <cybil code and or comments>
                               {$$$ END KEYPOINT DESCRIPTIONS $$$}

    <xxdkey_deck> ::= [ <cybil code and/or comments> ]
                      [ <keypoint_info> ... ]

    <keypoint_info> ::= <keypoint_constant_line> <delimiter> <eol>
                        [ <keypoint_descriptor> ... ]
                        [ <blank lines> ]

<keypoint_constant_line>  ::= <keypoint_constant> <spc> = <spc>
                                 <keypoint_base> <spc> [ <offset> ] <spc>

<keypoint_constant> ::= <identifier>

<keypoint_base> ::= <identifier>

<keypoint_descriptor> ::= { <keypoint_descriptor_list>
                            [ <spc> ] [ } ] <eol>

<keypoint_descriptor_list> ::= <keypoint_class_id>
                               [ <special_case_code> ]
                               [ <spc> <sub_id_field> ]
                               <spc> <keypoint_label>
                               [ <spc> <data_field> ]

<special_case_code> ::= M ! N ! S ! T
        (M = Mtr, N = Nos, S = task Switch, and T = Trap)

<sub_id_field> ::= <sub_id_length> . <sub_id_match>

<sub_id_length> ::= <field_length>
```

2.0 KEYPOINT CONVENTIONS AND GUIDELINES
2.5 BNF KEYPOINT DESCRIPTION

<field_length> ::= 0..52 (in bits)

<sub_id_match> ::= <small_integer>

<small_integer> ::= 0..0 FFFFFFFFFFFF(16)

<keypoint_label> ::= <label>

<label> ::= ' <character_string> '

<character_string> ::= [<alphabet> ...] (zero or more)

<data_field> ::= <data_label> [<spc> <data_field_descriptor>]

<data_label> ::= <label>

<data_field_descriptor> ::= <data_format> [<data_field_length>]

<data_format> ::= A | H | I
(A = Alphanumeric, H = Hex, I = Integer)

<data_field_length> ::= <field_length>

NOTE: <sub_id_length> + <data_field_length> must be <= 52 bits)

Operating system <keypoint_class_id> = {D,E,U,X})

<keypoint_class_id> for any keypoint used for additional information to previous keypoints must be a space)

A <definition_deck> remains in effect until superceeded by a deck which redefines the area to which it pertains)

3.0 NOS/VE KEYPOINT COMMANDS

There are three NOS/VE commands supplied to control keypoint processing and to issue specific keypoints at a command level. These commands are also defined in the Helpful Hints document for the current cycle of NOS/VE.

3.0 NOS/VE KEYPOINT COMMANDS
3.1 CONTROL COMMANDS

3.1 CONTROL_COMMANDS

These two commands control the setting of the keypoint masks in the job and monitor mode exchange packages and the type of keypoint collection.

3.1.1 ACTIVATE_KEYPOINT

The activate_keypoint command (abbreviated actk) initiates keypoint recording and collection. This is done directly for software keypoint collection but hardware collection also requires the 170 state keypoint collection program CPKPNOS to be initiated.

The parameters for the actk command are as follows:

KEYWORD	DESCRIPTION	VALUE	DEFAULT
mode m	collection mode	software|s|hardware|h	parameter required
environment e	recording environment	job|j|system|s	parameter required
monitor_mask mm	list of integers	(0,1,..,15)	(0..15)
job_mask jm	list of integers	(0,1,..,15)	(0..15)
start_class start	integer	0|1|..|15	15
stop_class stop	integer	0|1|..|15	15
keypoint_file kf	name of file for software collection	file name	$LOCAL. KEYFILE
collector_buffer _size cbs	integer	1000..100000	10000

3.0 NOS/VE KEYPOINT COMMANDS
3.1.1 ACTIVATE_KEYPOINT

collector_delay cd	delay period in milliseconds	10..100000	50

3.1.2 DEACTIVATE_KEYPOINT

The deactivate_keypoint command (abbreviated deak) is used to terminate the issuing of keypoints by clearing the keypoint masks in the job and monitor mode exchange packages and ending software collection (if enabled).

KEYWORD	DESCRIPTION	VALUE	DEFAULT
environment e	recording environment	job|j|system|s	job

3.2 KEYPOINT_ISSUE_COMMANDS

3.2.1 EMIT_KEYPOINT

The emit_keypoint command (abbreviated emik) is used to issue a keypoint at the SCL level. Its main use is to issue class 15 keypoints to start and stop keypoint collection runs.

KEYWORD	DESCRIPTION	VALUE	DEFAULT
class	keypoint class	0¦1¦..¦15	15
code	keypoint code	0¦1¦..¦FFFFFFFF(16)	0

4.0 PMF SOFTWARE OVERVIEW

4.0 PMF SOFTWARE OVERVIEW

The 800 series machines have a performance monitoring facility (PMF) built into the hardware. Various types of data are made available for monitoring system performance. This introduction will not deal with the details of what is available or the details of how to collect the data.

The collection is initiated from the system console for systems which, at this time, are running in dual state. Keypoint instructions have been inserted at the gated entries and exits of significant procedures in the operating system (OS) and product set (PS). There may also be other keypoints inserted for data collection or other purposes. In addition, some processor state data is available. This user guide is directed towards the users of the keypoint data.

The keypoint data must be analyzed to obtain useful information. The set of programs in this user guide is directed toward describing the tools available to analysts for transforming data into information.

The keypoint collection is done from the operator console while NOSVE is running the job(s) to be analyzed. The output is the raw keypoints data file (KPTS). This is run each time a job(s) is to be analyzed.

The following is an example of the logical sequence and intercation of programs and data:

```
Data Collection ---> raw-data

KPPRE           ---> preprocss-data

                     !--> KPRED ---> report(s)
KPPAR ---> KPPRE --!
                     !--> KPTRUN ---> report

raw-data        --!
                  !--> KPQRY ---> queries
preprocss-data  --!

raw-data ---> KPCNT ---> report
```

The overall process is shown in figure 1.

4.0 PMF SOFTWARE OVERVIEW

4.0 PMF SOFTWARE OVERVIEW

4.0 PMF SOFTWARE OVERVIEW

KPPAR is run to produce the PARSE file. This process modifies the keypoint data on the common decks into a form suitable for the preprocessor (KPPRE). This process is rerun only if the common decks are changed. The output, PARSFL may be used by all of the PMF users.

KPPRE transforms the raw data into a common format for the subsequent analysis programs. It can do some editing of the file using the directives from the KLUG file. It can also select portions of the raw data file using the start (STR) end stop (STP) directives in the parameter input. The output from the preprocessor are the preprocessed keypoint file, the descriptor file and several reports.

KPQRY allows scanning of either the raw keypoints data file or the preprocessed data file to determine some of the data in the keypoint files. This is intended to make quick, ad hoc inquiries about the nature of the keypoints.

KPRED uses the data from the descriptor file and the preprocessed keypoints to make more extensive, but predefined, reports of the contents of the keypoint runs.

KPTRUN is a process (Fig. 2) for analyzing the entries and exits from gated keypoints. It reconstructs the stack to determine the execution times within each gated keypoint. It is composed of 3 CYBIL programs and 2 sorts. CALLAN reads the preprocessed keypoint file and builds the network of caller-callee relationships that exist on the keypoint file. To keep the memory requirements low, and to decrease the processing time, the networks are are dumped periodically (every 3000 keypoints) to an intermediate file. The sort program simply bunches all the like keypoints from the interval dumps together for input to the COMBINE program. The COMBINE program sums the data from all the keypoints with the same parent and/or child into one record. It outputs these records together with the keypoint descriptions on a file in ascii format on the file specified by the FIRST parameter. Each record contains a parent and usually a child in ascii format. The processing to this point needs only to be done once for any given keypoint file. If reports are desired in several different sort orders, the NEWSORT parameter allows the processing to start at this point for the second and subsequent reports. The intermediate file is now sorted using the SORTKEY parameters specified to order the records for the desired report. Finally, the PRINT process reads this ordered file and splits the combined caller-callee records into records suitable for printing.

5.0 PMF SOFTWARE INSTALLATION

5.0 PMF SOFTWARE INSTALLATION

MATERIALS PMFPL
PMFCOR
XULIB

Add any local mods to PMFCOR, the file containing changes to the PMF programs.

Add the following statements to the PROFILE file.

```
\ PLIB= 'PMFPLIB'

\ TOOLLIB='XULIB'

\ TOOLACT= the PMF user name,ie., 'KEYPERF'

\ SEARCH = (PMFPLIB,the PMF user name)
```

After PMFPL, PMFCOR and XULIB have been loaded do

```
SES.GENCOMP PMFINST C=PMFCOR B=PMFPL

SES.REPPROC G=COMPILE B=PMFPLIB

SES.PMFINST ALL
```

The calls to GENCOMP and REPPROC need only be done when the PMF programs are being installed for the first time. From then on, only PMFINST needs to be called to install the particular modules which are to be altered.

5.0 PMF SOFTWARE INSTALLATION
5.1 PMFINST

5.1 PMFINST

This procedure installs the various components of the PMF system. It installs the object code on a library file, and the SES procedures on a PLIB file. Prior to installation a file, PMFPL, containing the various modules must exist in the installation user number (UN). A second file, PMFCOR, containing the correction set must also be available. The PROFILE of the user must contain the definitions given above.

Any of the parameters defined in the SES module JOBPARM may be used. These include JOBTL, JOBFL, JOBCN, JOBPR, JOBUN, JOBPW, JOBFMLY, JOBCN, JOBPN, LOCAL, BATCHN, BATCH, DEFER, NODAYF, DAYFILE, AND DF. (See SES Procedure Writer's Guide, Appendix A)

Installation Parameters:

CALLAN Install the object for the call analysis portion of KPTRUN.

COMBINE Install the object code for the program which combines the intermediate segments produced by CALLAN (Part of KPTRUN).

PRINT Install the object code for the routine which prints the REPORT(s) (Part of KPTRUN)

CPKPNOS Installs the CPU portion of the data collection routines.

PPKPNOS Installs the PP portion of the data collection routines.

KPPRE Install the object code for the Preprocessor.

KPPAR Install the object code for the keypoint descriptions parser program.

KPRED Install the object code for the primary data reduction program.

KPQRY Install the object code for the KPQRY program.

KPCNT Install the object code for the KPCNT program.

All of the above programs are stored on the file XULIB.

PRPMF Install the SES proc for running the keypoint collection.

5.0 PMF SOFTWARE INSTALLATION
5.1 PMFINST

PRKPPRE Install the SES proc for running KPPRE.

PRKPPAR Install the SES proc for running KPPAR.

PRKPRED Install the SES proc for running KPRED.

PRKPQRY Install the SES proc for running KPQRY.

PRKPCNT Install the SES proc for running KPCNT.

KPTRUN Install the SES proc for running the call analysis programs.

PMFINST Install the SES proc for installing the installation program. This will usually be done only when the installation program has been modified in PMFCOR.

ALL Calls for installation of all of the above programs and procedures.

The above procedures are installed on the SES proc library, PMFPLIB.

PLC This parameter is used to change the name of system CPL, if it is under a different name.

ID This parameter is used to specify the account where PLC resides, if different than LIBRARY.

After PMFINST has completed batch execution, all binaries are found in XULIB, and all outputs are found in LIST. The PP data collection program is under the name PMF. The sequence of commands necessary to place the code in the system is:

```
SES.GETMEM PMF PP B=XULIB

SYSEDIT,B=GROUP.
```

These commands must be entered from the system console.

5.2 PROGRAM MAINTENANCE

All PMF related material is stored on PMFPL. In order to maintain an orderly system of changing or correcting the programs and procedures, all changes are entered into a correction file. This file has two areas for entering changes. General changes are entered in the correction ID called PMFCOR. It is recognized that some sites will have special need to cater to their equipment or operational procedures. Accordingly there is a correction ID for site local modifications to be stored. If these changes are of general use, then they may be moved into the PMFCOR ID. Presently, the site local changes reflect the use of group encoding used on tape equipment rather than phase encoding. In addition, it may be desirable for some sites to change the default conditions on some of the procs and these may also go into the site local mods.

There are two methods of generating the corrections. One is to get the module from PMFPL applying the PMFCOR changes, and then changing the resultant module to fix the bug or introduce the change. Then after testing, make an SES GENCOR run to produce the changes required to go into PMFCOR. The other way is to make a GENCOMP run with the SEQ parameter to get the line numters, then enter the proper insert and delete entries in PMFCOR.

6.0 PMF DATA COLLECTION (PPKPNOS, CPKPNOS)

The PMF data collection program was written to provide a means for the collection of data from the CYBER 180 Performance Monitoring Facility (PMF). The program uses the PMF hardware to collect information on program execution and to collect various machine statistics.

Full use of the PMF hardware is possible with the program. In addition, the program provides keypoint selection options and the capability for the periodic sampling of keypoints.

The PMF data collection program is written in PP COMPASS and runs under the A170 NOS operating system. The program will function on either a strictly A170 mode machine or on the 170 side of a dual state machine. While running on the 170 side of a dual state machine, the program can collect information about both the A170 and the 180 sides (through both the counter and keypoint mechanisms for 180 state). A170 state collection is limited to counter data.

This document assumes that the reader is familiar with the CYBER 180 Performance Monitoring Facility and CYBER 170 COMPASS.

6.0 PMF DATA COLLECTION (PPKPNOS, CPKPNOS)
6.1 PROGRAM CONTROL AND COMMUNICATION

6.1 PROGRAM CONTROL AND COMMUNICATION

The program uses several areas in central memory as communication areas. These areas are shown in figure 1.

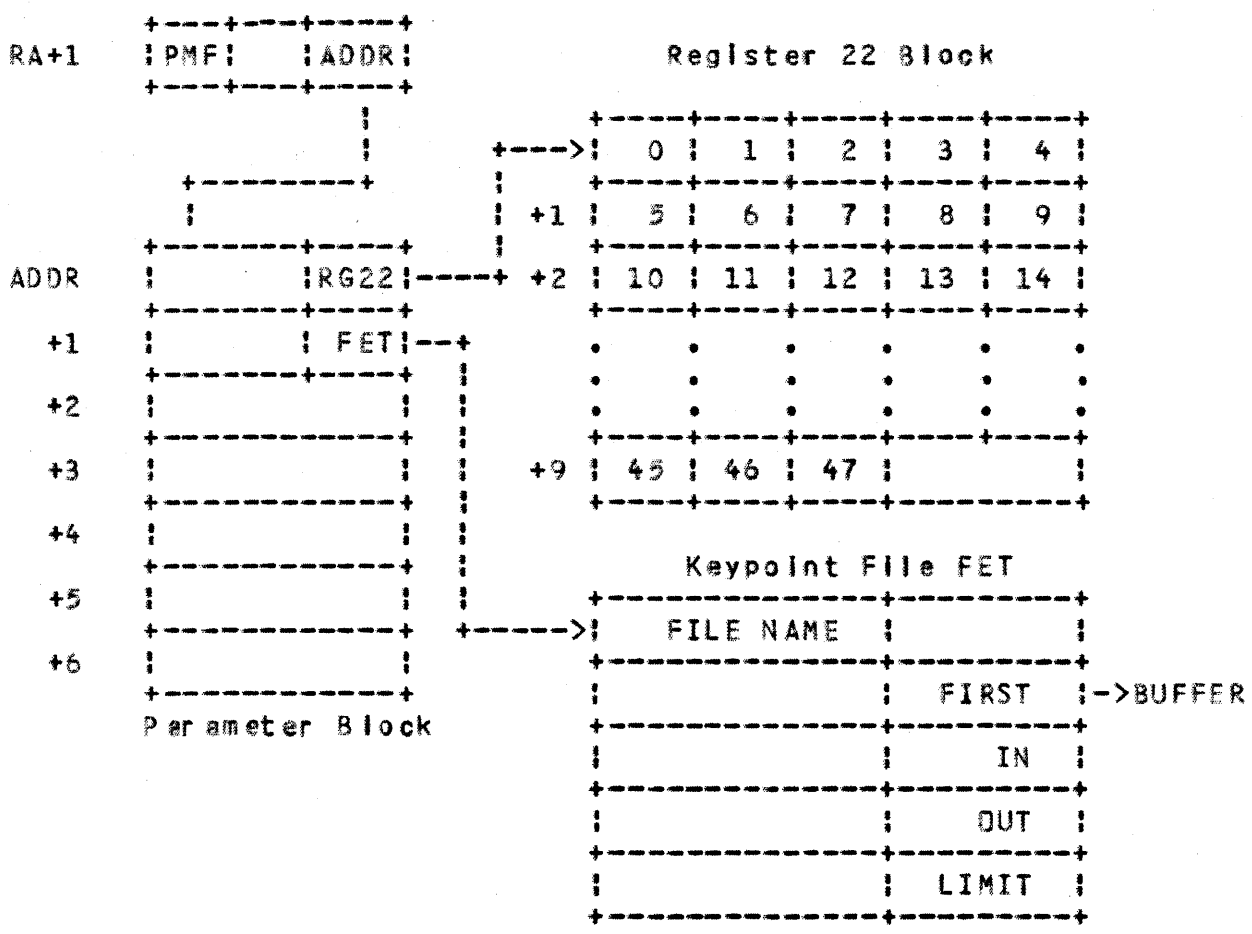


Figure 1

Program PMF is initiated by a RA+1 call which contains in its lower 18 bits the address of the parameter block. This block contains the keypoint selection and sampling parameters, a pointer to the register 22 block, and a pointer to a File Environment Table (FET). The FET in turn describes a circular buffer into which the keypoints are collected.

6.2 PROGRAM PPKPNOS

6.2.1 INITIALIZATION

The program PMF is loaded into a PP in response to a RA+1 call. Once initiated, it will insure that the CPU is a CYBER 170 model 825, 835, or 855 and that the PMF hardware option is installed. If the program detects any other copies of PMF running, it will abort the calling program and then drops out.

If no other copies of PMF are running, the program will read the seven word parameter block pointed to by the lower 18 bits of the RA+1 call. The address of the register 22 block is then checked to insure that the block lies within the users field length.

If the keypoint collection flag is not set in word two of the parameter block, PMF will ignore the remainder of the block. This enables the user to omit the FET and circular buffer if keypoint collection is not desired.

PMF then checks the origin type of the calling program. If an interactive program has requested keypoint collection, it will be aborted with an error message. (Keypoints may not be collected interactively as the CP program must be locked into CM for the duration of the collection period.) It is possible for register 22 to be loaded and dumped interactively for counter mode collection only.

6.2.2 LOADING AND DUMPING REGISTER 22

To start the PMF hardware, register 22 must be loaded with 48 bytes of data. This initializes the hardware and begins PMF operation as specified by the control fields in the data. To stop PMF hardware operation and to collect the eight 32-bit event counters contained in register 22, the register must be read and dumped to CM.

After initialization, the program PMF determines from the parameter block whether register 22 should be loaded or dumped. If the user has selected loading, the area in CM designated as the register 22 block is read and its contents transferred to the PMF hardware. If dumping is selected, PMF is stopped and the contents of register 22 are transferred back to the register 22 block in CM.

6.0 PMF DATA COLLECTION (PPKPNOS, CPKPNOS)
6.2.2 LOADING AND DUMPING REGISTER 22

If keypoint collection is not required, the program PMF will restart its calling program and drop out after loading register 22. This permits the user to initialize PMF and to collect data in the counters without having the program PMF running at the same time. When the test is complete, the user then calls PMF to dump the contents of register 22 to CM.

After a dump of register 22, the program PMF always restarts the calling program and drops.

6.2.3 KEYPOINT COLLECTION

After loading register 22, the user may have the program PMF remain in the PP to collect keypoints. When collecting keypoints, the program enters a loop which consists of the following steps:

1. Test for any condition that requires the program PMF to stop. This includes a PMF hardware stop, an error in the CP program, an operator drop, or the operator command "n.CFO.STOP.". If the condition is not fatal to the CP program, termination processing is begun.

2. Read keypoints from the PMF FIFO buffer to the selection buffer in the PP. If a sampling interval has been given in the parameters, one keypoint is read after each interval.

3. Select keypoints from the selection buffer and place them into the keypoint buffer. Selection is based upon the keypoint selection mask given in the parameters. The user may also collect every nth keypoint of any given class according to the keypoint selection counts (see section 4.1).

4. Write keypoints from the keypoint buffer to the circular buffer in central memory.

The CPU program may read the keypoints directly from the circular buffer or may periodically issue CIO requests to empty the buffer to a file.

The keypoints are placed into a circular buffer with one 8-bit byte per 12-bit PP parcel. The upper four bits of each parcel are set to zero. Since each keypoint is eight bytes in length, a keypoint will occupy one and three-fifths CM words in the buffer. Exactly forty keypoints will fit into a single PRU.

6.2.4 END PROCESSING

End processing consists of dumping the last keypoints collected to the circular buffer, and dumping the contents of register 22 to the register 22 block.

Keypoints are transferred from the PP buffer to the circular buffer on a PRU basis. When collection is complete, 8 bytes (one keypoint) of ones are added after the last keypoint if there is not exactly one PRU of keypoints left in the PP buffer. This enables the user to determine how many keypoints in the last PRU are valid and serves as a delimiter.

After dumping the keypoints, the PMF hardware is stopped and the contents of register 22 is placed into the register 22 block in CM. Bit 2**0 of byte 0 of register 22 is set to one when the register is dumped to CM. This bit is checked during keypoint collection to determine if PMF operation has stopped.

6.3 COMMUNICATION_BLOCKS

6.3.1 PARAMETER BLOCK

The parameter block is seven CM words in length. The following pseudo-COMPASS describes its format:

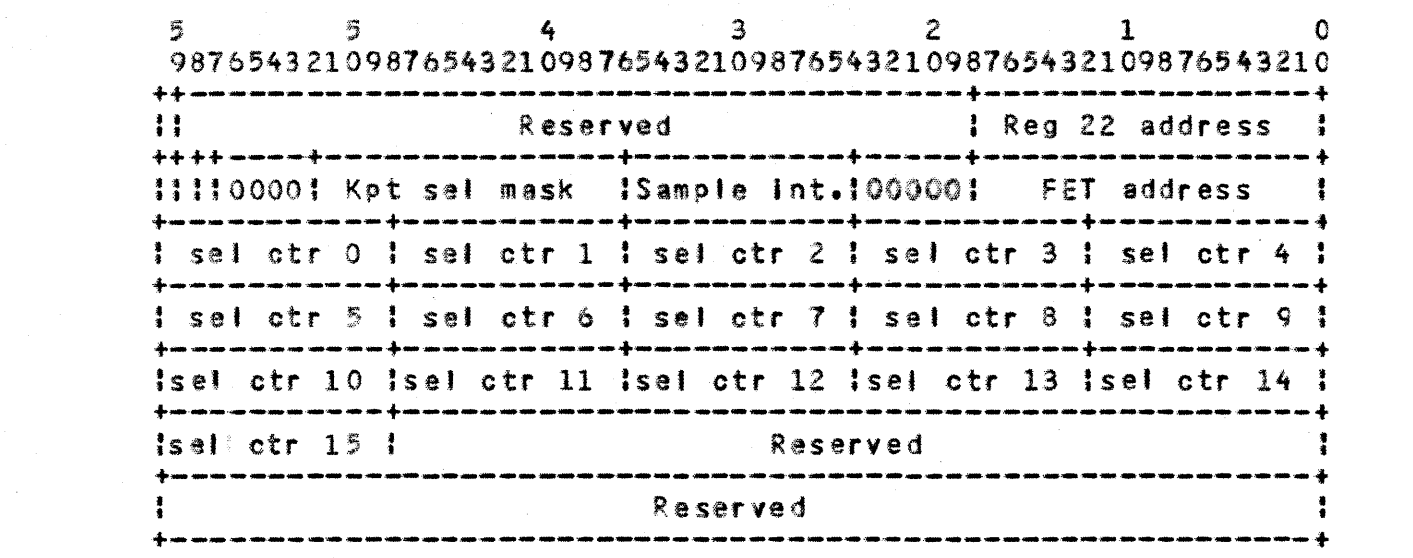
```
PARBLK     VFD     1/Load_Dump_Flag
           VFD     41/0   (reserved for future use)
           VFD     18/Reg22_Block_Address

PARBLK+1   VFD     1/Keypoint_Collection_Flag
           VFD     1/Keypoint_Sample_Flag
           VFD     1/Keypoint_Selection_Flag
           VFD     1/Central_Memory_PMF_Buffer
           VFD     4/0   (reserved)
           VFD     16/Keypoint_Selection_Mask
           VFD     12/Keypoint_Sample_Interval
           VFD     6/0   (reserved)
           VFD     18/FET_Address

PARBLK+2   VFD     12/Selection_Count_0
           VFD     12/Selection_Count_1
            .               .
            .               .
            .               .
           VFD     12/Selection_Count_15
           VFD     48/0   (reserved)

PARBLK+6   VFD     60/0   (reserved)
```

6.0 PMF DATA COLLECTION (PPKPNOS, CPKPNOS)
6.3.1 PARAMETER BLOCK



Note that the selection count parameters occupy words PARBLK+2 through PARBLK+5.

The parameters are:

Load_Dump_Flag	If set to 1, indicates that register 22 should be loaded from the register 22 block. If set to 0, indicates that the contents of register 22 should be dumped to the register 22 block in CM.
Reg22_Block_Address	CM address of the register 22 block.
Keypoint_Collection_Flag	Set to 1 if the user desires the program PMF to remain in the PP and collect keypoints.
Keypoint_Sample_Flag	If set to 1 the program will read a single keypoint from the hardware at intervals specified by the Keypoint_Sample_Interval. Otherwise, all the keypoints are read.

6.0 PMF DATA COLLECTION (PPKPNOS, CPKPNOS)
6.3.1 PARAMETER BLOCK

Keypoint_Selection_Flag	Indicates that the selection of keypoints by keypoint class should be performed. Selection is based upon the Keypoint Selection Mask. A one in the mask at the position corresponding to class n indicates that the keypoints of class n should be collected. A zero in that position indicates that all keypoints of class n should be discarded. Note that the selection mask performs the same function as the Keypoint Mask in a CYBER 180 exchange package. If possible, the exchange package mask should be used to minimize overhead.
Central_Memory_PMF_Buffer	If set indicate that the Central Memory PMF FIFO buffer will be used instead of the hardware FIFO buffer. The CM buffer is one page long versus the 16 words of the hardware FIFO buffer. This buffer is driven directly by the microcode.
Keypoint_Selection_Mask	This mask determines which classes of keypoints are collected, and which classes are discarded (see the description of the Keypoint_Selection_Flag). The leftmost bit in this mask corresponds to keypoint class 0, the rightmost bit to keypoint class 15.
Keypoint_Sample_Interval	In milliseconds, the amount of time to wait between the collection of keypoints (see the Keypoint_Sample_Flag).
FET_address	The CM address of the FET for keypoint collection.

6.0 PMF DATA COLLECTION (PPKPNOS, CPKPNOS)
6.3.1 PARAMETER BLOCK

Selection_Count_n	For a value m, every mth keypoint of class n is collected (e.g. if Selection_Count_7 has a value of 5, every 5th keypoint of class 7 will be collected). If a value of 0 or 1 is specified, all keypoints of that class will be collected.

6.3.2 REGISTER 22 BLOCK

This block is the image of register 22 as it is read or written from the PMF hardware. Register 22 consists of 48 eight bit bytes. To simplify the handling of these bytes, they are left unpacked in the register 22 block, one byte in every 12 bit parcel of a CM word. When loading register 22, the upper 4 bits of each parcel is ignored, and when dumping the register the upper 4 bits are set to zero.

The following COMPASS code will generate the register 22 block:

```
REG22   VFD     12/Byte_0
        VFD     12/Byte_1
         .          .
         .          .
         .          .
        VFD     12/Byte_47
        VFD     24/Reserved
```

```
 5           5           4           3           2           1           0
 987654321098765432109876543210987654321098765432109876543210
+-----------+-----------+-----------+-----------+-----------+
!  Byte 0   !  Byte 1   !  Byte 2   !  Byte 3   !  Byte 4   !
+-----------+-----------+-----------+-----------+-----------+
!  Byte 5   !  Byte 6   !  Byte 7   !  Byte 8   !  Byte 9   !
+-----------+-----------+-----------+-----------+-----------+
!  Byte 10  !  Byte 11  !  Byte 12  !  Byte 13  !  Byte 14  !
+-----------+-----------+-----------+-----------+-----------+
!  Byte 15  !  Byte 16  !  Byte 17  !  Byte 18  !  Byte 19  !
+-----------+-----------+-----------+-----------+-----------+
!  Byte 20  !  Byte 21  !  Byte 22  !  Byte 23  !  Byte 24  !
+-----------+-----------+-----------+-----------+-----------+
!  Byte 25  !  Byte 26  !  Byte 27  !  Byte 28  !  Byte 29  !
+-----------+-----------+-----------+-----------+-----------+
!  Byte 30  !  Byte 31  !  Byte 32  !  Byte 33  !  Byte 34  !
+-----------+-----------+-----------+-----------+-----------+
!  Byte 35  !  Byte 36  !  Byte 37  !  Byte 38  !  Byte 39  !
+-----------+-----------+-----------+-----------+-----------+
!  Byte 40  !  Byte 41  !  Byte 42  !  Byte 43  !  Byte 44  !
+-----------+-----------+-----------+-----------+-----------+
!  Byte 45  !  Byte 46  !  Byte 47  !        Reserved       !
+-----------+-----------+-----------+-----------------------+
```

Note that the register 22 block occupies 10 CM words, with the lower 2 parcels in the last word being reserved for future use.

6.0 PMF DATA COLLECTION (PPKPNOS, CPKPNOS)
6.3.2 REGISTER 22 BLOCK

The CYBER 180 MIGDS contains a detailed description of each byte in register 22.

6.3.3 FET AND CIRCULAR BUFFER

The FET and circular buffer used for keypoint collection are identical to those used when performing normal I/O transfers. They may be created using the following COMPASS macros:

```
BUFL    EQU     6000B
FET     FILEB   BUFF,BUFL
BUFF    BSS     BUFL
```

The keypoint collection program was written to provide fast transfers of keypoints to the CM buffer. To facilitate this, the program assumes that the CM buffer is always (n * 100B) words in length (where n >= 3). If the buffer length does not conform to this restriction, keypoints may be lost and data in or after the buffer may be destroyed. The PP will never write data outside the user's field length. The optimum buffer size for a particular application will depend upon the rate at which keypoints are issued and the amount of time needed to empty the buffer.

It is the responsibility of the CP program to insure that the buffer is emptied. The program may do this by periodically issuing a CIO write request. Alternately, the program may read the keypoints directly out of the buffer and process them as they are being collected. If the CP program is to read the keypoints out of the buffer, it must read only integer multiples of a PRU at a time.

If keypoints are being issued at too great, a rate the circular buffer may become full. If this buffer is not emptied before the PP transfers the next PRU of keypoints, all the keypoints in the CM buffer will be lost. Special checks are made in the data collection software to detect such data loss and to flag its occurrence.

6.4 PROGRAM CPKPNOS

The program CPKPNOS has been written to provide the user with a means to collect counter and/or keypoint data from the PMF hardware. CPKPNOS, which is written in FORTRAN 5 and COMPASS, uses the PP program PMF to read and write register 22, the PMF control register. It then produces a listing which contains hardware and software statistics.

The default information provided by CPKPNOS is:

1. Central Memory Reads
2. Cache Misses
3. Cache Hit Rate
4. Central Memory Writes
5. Total Memory Accesses
6. Instructions Executed
7. Run Time
8. MIPS Rate
9. Transfers to 180 State
10. Time in 180 State
11. Percentage of Time in 180 State

In order to give the user additional control over the control fields in PMF register 22, a set of parameter keywords are provided. These also allow the user to define the start and stop times for the PMF collection period. The following pages summarize the different control parameters for the data collection program.

6.4.1 CONTROL PARAMETERS

All control parameters may be specified either by their full names or by a one to three character abbreviation. In the following descriptions, the abbreviation is shown directly beneath the full name of the parameter. Note that all tokens in the parameters supplied to CPKPNOS must be separated by either a blank or a comma.

TYPE = <BEGIN¦END¦COLLECT> Default = END

BEGIN = start counter collection
END = stop counter collection
COLLECT = start keypoint collection

This parameter determines the type of data collection run.

SAMPLETIME = <n> Default = 0
SMT

Counter mode -- n is in seconds
Keypoint mode -- n is in milliseconds

This parameter gives the sampling interval. The scale for the interval given depends on whether counters or keypoints are being collected.

STARTIME = <hh.mm> Default = 00.00
STT

This parameter is used to start collection at a specified time of day. If its value is "00.00", collection starts immediately.

STOPTIME = <hh.mm> Default = 00.00
SPT

This parameter is used to stop collection at a specified time of day. If its value is "00.00", there is no stop time processing.

MACHINEID = <n> Default = 2
MID

This parameter is used to specfiy the machine model on which the collection is running (S1=1, S2=2, S3=3).

6.4.2 KEYPOINT COLLECTION PARAMETERS

These parameters are used only for keypoint collection. Thus, the Load_Dump_Flag and the Keypoint_Collection_Flag are set by default.

CMB = <YES¦NO¦Y¦N> Default = NO
Central Memory PMF buffer.

SMF = <YES¦Y¦NO¦N> Default = NO
Keypoint Sample Flag.

SLF = <YES¦Y¦NO¦N> Default = NO
Keypoint Selection Flag.

SLM = <n> Default = FFFF
This parameter specifies the keypoint selection mask to be used in conjunction with the SLF parameter.

SCi = <n> Default = 0
This parameter gives the selection count for class i (where i is in the range 0..15). Note that i must be specified as a two digit number (e.g. 00 as in SC00).

More information on these parameters is found in the Parameter Block section.

6.4.3 REGISTER 22 PARAMETERS

The register 22 parameters allow full control of the various bytes in the PMF control register. Normally these do not need to be used as appropriate default values are selected; A170 state data for counter collection and 180 state data for keypoint collection.

Each parameter is used to set an 8-bit control byte in register 22. A complete description of the effect of the various control options is given in the MIGDS. Note that there are no parameters to set bytes 0, 6, and 7 of register as these are always preset to zero.

CTL = n1+n2+n3
- n3 Stop on keypoint class
- n2 Start keypoint operation
- n1 PMF keypoint request

Byte 1 control
Defaults:
- counter mode = 00
- keypoint mode = 50

SKC = n1+n2
- n1 Start keypoint class
- n2 Stop keypoint class

Byte 2 control
Defaults:
- counter mode = 00
- keypoint mode = FF

SCO = n

Byte 3 control
Defaults:
- counter mode = FF
- keypoint mode = FF

Each bit set in SCO will cause PMF to terminate processing if the corresponding counter overflows.

PIA = n

Byte 4 control
Defaults:
- counter mode = 02
- keypoint mode = 04

This parameter gives the Processor Instruction Argument for instruction counting.

PIM = n

Byte 5 control
Defaults:
- counter mode = FF
- keypoint mode = FF

This parameter gives the Processor Instruction Mask for instruction counting.

6.0 PMF DATA COLLECTION (PPKPNOS, CPKPNOS)
6.4.3 REGISTER 22 PARAMETERS

SA0 = n — Byte 8 control
Defaults:
counter mode = 01
keypoint mode = 01

This parameter gives the event/state selection code for counter A0 and the control bit to AND counters A0 and B0.

SB0 = n — Byte 9 control
Defaults:
counter mode = 00
keypoint mode = 00

This parameter gives the event/state selection code for counter B0.

SA1 = n — Byte 10 control
Defaults:
counter mode = 05
keypoint mode = 05

This parameter gives the event/state selection code for counter A1 and the control bit to AND counters A1 and B1.

SB1 = n — Byte 11 control
Defaults:
counter mode = 0C
keypoint mode = 0C

This parameter gives the event/state selection code for counter B1.

SA2 = n — Byte 12 control
Defaults:
counter mode = 11
keypoint mode = 07

This parameter gives the event/state selection code for counter A2 and the control bit to AND counters A2 and B2.

SB2 = n — Byte 13 control
Defaults:
counter mode = 0E
keypoint mode = 03

This parameter gives the event/state selection code for counter B2.

SA3 = n — Byte 14 control
Defaults:
counter mode = 91
keypoint mode = 90

This parameter gives the event/state selection code for counter A3 and the control bit to AND counters A3 and B3.

6.0 PMF DATA COLLECTION (PPKPNOS, CPKPNOS)
6.4.3 REGISTER 22 PARAMETERS

SB3 = n — Byte 15 control
Defaults:
counter mode = 14
keypoint mode = 14

This parameter gives the event/state selection code for counter B3.

CA0 = n — Bytes 16-19 initializ.
Default = 100

This parameter initializes counter A0 to the value n.

CB0 = n — Bytes 20-23 initializ.
Default = 100

This parameter initializes counter B0 to the value n.

CA1 = n — Bytes 24-27 initializ.
Default = 100

This parameter initializes counter A1 to the value n.

CB1 = n — Bytes 28-31 initializ.
Default = 100

This parameter initializes counter B1 to the value n.

CA2 = n — Bytes 32-35 initializ.
Default = 100

This parameter initializes counter A2 to the value n.

CB2 = n — Bytes 36-39 initializ.
Default = 100

This parameter initializes counter B2 to the value n.

CA3 = n — Bytes 40-43 initializ.
Default = 100

This parameter initializes counter A3 to the value n.

CB3 = n — Bytes 44-47 initializ.
Default = 100

This parameter initializes counter B3 to the value n.

6.4.4 DESCRIPTION OF FILES USED

6.4.4.1 Input Files

PARMFL = <file name> Default = INPUT

Example:

```
TYPE = BEGIN
MID = 1
SAO = 7
SCO = 0
```

This file contains the data collection parameters.

6.4.4.2 Output Files

REPTFL = <file name> Default = OUTPUT

This output file contains the collection report.

DEBGFL = <file name> Default = FDBG

This file will contain debugging information such as the Reg.22 and the parameter block dumps.

KPTS = <KPTS> Default = KPTS

If keypoint collection is selected, the keypoints collected will be outputted on file KPTS which is currently a hard coded name.

6.0 PMF DATA COLLECTION (PPKPNOS, CPKPNOS)
6.5 OUTPUT REPORTS

6.5 OUTPUT REPORTS

The only output report issued by CPKPNOS is a counter summary if counter collection is selected. No report is issued for keypoint collection as these reports are produced by the PMF Data Reduction software.

6.5.1 COUNTER SUMMARY

SYSTEM STATISTICS - S2 82/11/16. 17.01.06.

CENTRAL MEMORY STATISTICS

CENTRAL MEMORY READS	23444598		
CACHE MISSES		1450885	
CACHE HIT RATE (PERCENT)			93.811

INSTRUCTION EXECUTION STATISTICS

INSTRUCTIONS EXECUTED	97096795		
RUN TIME (SECONDS)		43.449472	
MIPS RATE			2.235

EXECUTIVE PROGRAM STATISTICS

TRANSFERS TO EXECUTIVE	35516130		
AVG. TIME IN EXECUTIVE	.056		
TIME IN EXECUTIVE		1.988626	
RUN TIME (SEC.)			43.449472
PERCENTAGE IN EXECUTIVE			4.577

6.6 OPERATIONAL PROCEDURES

6.6.1 INSTALLATION

To install these programs three files are needed: PMFPL, XULIB and PMFPLIB. To compile and sysedit the PP binary type:

```
SES.PMFINST PPKPNOS
SES.GETMEM PMF PP B=XULIB
SYSEDIT,B=GROUP.
```

To compile and save the data collection program type:

```
SES.PMFINST CPKPNOS
```

6.6.2 RUNNING THE DATA COLLECTION

To begin the collection of statistics, a job must be submitted with the data card TYPE = BEGIN. This will initialize register 22, and start the event counters in the PMF hardware. When the test period is complete, resubmit the job with the TYPE = END data card. The event counters will be read from PMF and the program CPKPNOS will output a listing containing the statistics.

The program CPKPNOS may also be used for the collection of keypoints from the 180 side of the machine when Dual State is running.

While collecting keypoints, the job will normally remain at a control point until PMF hardware detects a stop condition (counter overflow, stop keypoint). The job can be terminated prior to this by the operator command "n.CFO.STOP". Normal end processing is begun by this entry and the keypoint file will be correctly produced. If the job is dropped, the keypoint file may not contain all the desired keypoints and will not be terminated by the record which contains register 22.

If keypoints are issued too rapidly, the buffer in CPKPNOS may become full. If this happens, keypoints may be lost. While this problem is unlikely when the file KPTS is assigned to disk, it is a possibility if the keypoints are being written to a tape. No error message is currently issued if keypoints are lost.

The keypoint file produced by CPKPNOS can be input directly to the keypoint pre-processor program (KPPRE).

6.6.3 CPKPNOS RUN PROCEDURES

An SES procedure 'PMF' is supplied. It is recommended that this proc be installed on user number SYSTEMX or LIBRARY in order to allow the operator to initiate a data collection by typing:

x.PMF(parameters)

where the parameter descriptions are as follows:

KEYWORD	DESCRIPTION	VALUE	DEFAULT
TY	type of collection	COLLECT;BEGIN;END	END
MI	machine type	S1;S2;S3	S2
IF	input parameter file	file name	none
OF	statistics output file	file name	OCPKPNS
TPF	tape number (if collection on tape, otherwise the disk file KPTS is assumed)	tape vsn	none
DE	tape density	PE,GE,etc.	PE
LIB	library containing XCPKPNS in ULIB format	lib name	XULIB
LIBUN	user number where library is located	user id	KEYPERF

Some examples of how to use PMF follow:

SES.PMF TY=BEGIN	This call initiates PMF counter collection.
SES.PMF	This call terminates PMF counter collection. The output will be found on file OCPKPNS.

6.0 PMF DATA COLLECTION (PPKPNOS, CPKPNOS)
6.6.3 CPKPNOS RUN PROCEDURES

SES.PMF TY=BEGIN IF=INFILE	This call will initiate a PMF counter collection using the input file INFILE to change some or all of the Reg.22 setting.
SES.PMF TY=COLLECT	This call initiates PMF keypoint collection. The collected keypoints will be found on file KPTS.
SES.PMF TY=COLLECT TPF=ABC001	This call initiates PMF keypoint collection. The collected keypoints will be found on tape ABC001.

7.0 KEYPOINT DESCRIPTION PARSER (KPPAR)

This program was originally imbedded into the pre-processor program. For obvious reasons of redundancy a stand alone program has been built. This approach was choosen because the need of parsing the common decks is only when changes to the common decks are done. Thus, every time the pre-processor is used, there is no need to re-parse all the common decks.

This program parse the keypoint common decks and genrates a more concise binary file which the pre-processor will use over and over.

7.0 KEYPOINT DESCRIPTION PARSER (KPPAR)
7.1 KPPAR GENERAL DESCRIPTION

7.1 KPPAR GENERAL DESCRIPTION

The program reads the keypoint common decks and creates a binary file summarizing the keypoint descriptions, for use by the pre-processor. It does also check the syntax of every description according to the language specification. Error messages will be printed accordingly.

7.0 KEYPOINT DESCRIPTION PARSER (KPPAR)
7.2 DESCRIPTION OF FILES USED

7.2 DESCRIPTION OF FILES USED

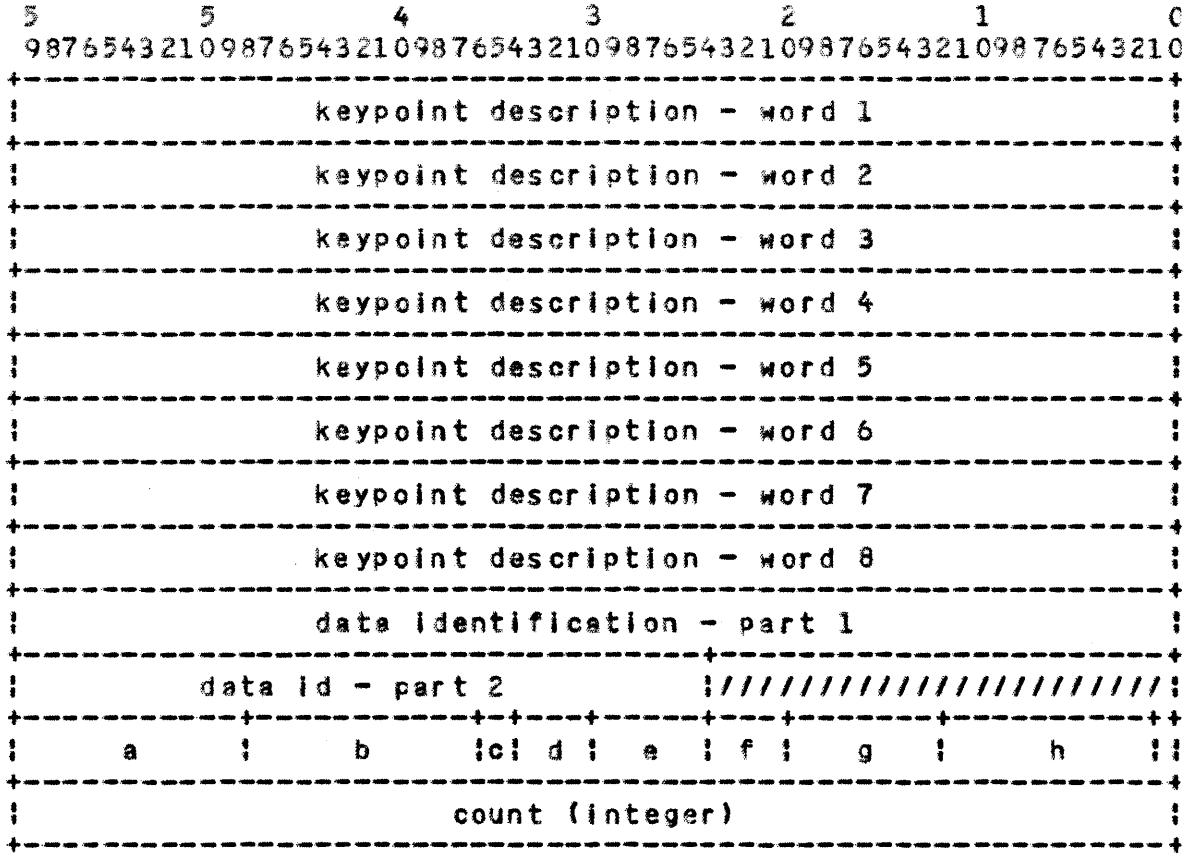
7.2.1 INPUT FILES

PARMFL: This file contains the input parameters for the control of the program.

KEYDESC: This file contains the keypoint common deck descriptions needed for the processing of the keypoints.

7.2.2 OUTPUT FILES

PARSFL: This file contains the parsed descriptions of the keypoints. The parsing produces a file of descriptions of the following format:



7.0 KEYPOINT DESCRIPTION PARSER (KPPAR)
7.2.2 OUTPUT FILES

Where:
- a - keypoint number,
- b - sub_id match value,
- c - data formatting,
- d - sub_id match length
- e - data length,
- f - keypoint class,
- g - special case flag,
- h - description index,

DEBGFL: This file contains some permanent debugging information. Also, lists input lines which do not conform to the language specification, together with an explanatory error or warning messages.

7.3 OPERATIONAL PROCEDURES

7.3.1 INSTALLATION

To install this program three files are needed: PMFPL, XULIB and PMFPLIB. To compile and save the binary of the parser program type:

```
SES.PMFINST KPPAR
```

7.3.2 KPPAR RUN PROCEDURES

An SES proc file is supplied in order to run the parser program. This procedure will enable the user to execute the program locally or submit a batch job. The following parameters are used within the procfile.

Any of the parameters defined in the SES module JOBPARM may be used. These include JOBTL, JOBFL, JOBCN, JOBPR, JOBUN, JOBPW, JOBFMLY, JOBCN, JOBPN, LOCAL, BATCHN, BATCH, DEFER, NODAYF, DAYFILE, AND DF. (See SES Procedure Writer's Guide, Appendix A).

PRF — Parsed keypoints common decks file. This file is the output file from KPPAR.
Example PRF=PARSFL Default PARSFL.

CDF — This specifies the name of the file containing the common decks. If no file is assigned, KPPAR will attempt to get the common decks using file KEYDESC in user number DEV1.
Example CDF=KEYD Default KEYDESC.

ID — If the the preprocessed keypoint data file is in another user's area this parameter must be specified.
Example ID=PM72 Default KEYPERF.

LIB — Name of file where the object code for XKPPAR is found. Ordinarily this parameter is obtained from the 'toollib' parameter in the user's PROFILE. This parameter would ordinarily be used for testing a new version of the preprocessor, or for use of a specially modified version of the preprocessor.
Example LIB=MYLIB Default XULIB.

7.0 KEYPOINT DESCRIPTION PARSER (KPPAR)
7.3.2 KPPAR RUN PROCEDURES

LIBUN User number where file specified by LIB (above) resides. Ordinarily this parameter is obtained from the 'tcolact' parameter in the user's PROFILE.
Example LIBUN=UN12 Default KEYPERF.

Some examples of how to invoke the KPPAR procedure follow:

SES.KPPAR	This call will parse file KEYDESC and the parsed output will be found into file PARSFL.
SES.KPPAR LOCAL	This call will run intercatively the job with the same default as in the previous example.

8.0 KEYPOINT PREPROCESSOR (KPPRE)

8.0 KEYPOINT PREPROCESSOR (KPPRE)

The keypoint pre-processor KPPRE is designed to take NOS/VE keypoints and the parsed keypoint common decks file as input and to produce an output keypoint file suitable for input to further data reduction stages. It will optionally output a formatted keypoint trace report for system debugging purposes. Another capability is to produce a printable keypoint descriptor file automatically generated from the input binary parsed file from the NOS/VE program library(s). This descriptor file may contain descriptions of all keypoints or only of those keypoints found on the keypoint data file. When keypoint pre-processing is selected, the compressed description is always the first record on the keypoint output file. This approach was chosen to minimize keypoint maintenance and coordination problems.

8.0 KEYPOINT PREPROCESSOR (KPPRE)
8.1 KPPRE GENERAL DESCRIPTION

8.1 KPPRE GENERAL DESCRIPTION

The KPPRE program will accept simulator, software, or hardware collected keypoints and the NOS/VE keypoint common decks file as input and, depending upon the input parameters, will generate any combination of:

- keypoint trace report
- keypoint description file
- pre-processed keypoint data file

It has the capability of outputting trace reports and pre-processed output files for all keypoints encountered in the data or only for those belonging to a specified task. Processing may also be turned on and off on given keypoints.

KPPRE was created by modifying and amalgamating three existing keypoint processing programs. In the process, a number of existing bugs were corrected and the user interface improved and made SCL like. This program was written in CYBIL-CC in order to simplify its eventual migration to CYBIL-II under NOS/VE. It currently runs under 170 state NOS versions 1 and 2.

8.2 CONTROL PARAMETERS

All control parameters may be specified by either their full name or by a one to three character abbreviation. In the following descriptions, the abbreviation is shown directly underneath the full parameter name.

FILETYPE = <SIM¦SOFT¦HARD> Default = HARD
FT

This parameter specifies whether the input keypoint file is from the C180 simulator or from software or PMF hardware collection under NOS/VE.

DESC = <YES¦NO¦Y¦N> Default = NO
D

This parameter specifies that a keypoint description file is to be output. This file will be used as input to further data reduction programs.

COMPDESC = <YES¦NO¦Y¦N> Default = NO
CDS

This parameter specifies that a keypoint description file, containing only the descriptions of the keypoints encountered in the input file, is to be generated. It is put down as the first record of the keypoint output file. Note that this toggle is forced on if pre-processing is requested since the data reduction will not work without the compressed description.

TRACE = <YES¦NO¦Y¦N> Default = NO
T

This parameter specifies if a trace report is to be generated.

PREPR = <YES¦NO¦Y¦N> Default = YES
P

This parameter specifies that a pre-processed keypoint file is to be generated. This file will be used as input to further data reduction programs. The compressed description toggle is forced on. The compressed description is the first record on the output file while the processed keypoints are the second.

8.0 KEYPOINT PREPROCESSOR (KPPRE)
8.2 CONTROL PARAMETERS

TASKID = <n> where n is a numeric task ID Default = -1
TK

This parameter is used to output all keypoints belonging to the specified task onto a file for further processing. Its base may be specified in parentheses after the number, example: 18(16) = 24(10). This file may then be used as input to the data reduction programs.

SKIPKPT = <n> where n is a numeric value Default = 0
SK <=99999999

This parameter is used to skip n keypoints at the beginning of the input file. (Note that this count is not exact as the number of keypoints skipped will be the next larger multiple of the number of keypoints per buffer.)

MAXKPT = <n> where n is a numeric value Default = 99999999
MK <=99999999

This parameter is used to specify the actual number of keypoints to be pre-processed (excluding any skipped by the SKIPKPT parameter).

START = <x> <y> where x is a class no. and Default = -1 -1
STR y a keypoint no.

This parameter specifies the keypoint to turn cyclically the pre-processor on. This parameter override the task selection.

STOP = <x> <y> where x is a class no. and Default = -1 -1
STP y a keypoint no.

This parameter specifies the keypoint to turn cyclically the pre-processor off. This parameter override the task selection.

Note: It is not advised to use start/stop toggling at the same time as task selection because the task entry keypoints will only be recognized if they fall within the intervals defined by the start and stop parameters.

8.0 KEYPOINT PREPROCESSOR (KPPRE)
8.3 FILE NAME PARAMETERS

8.3 FILE NAME PARAMETERS

DATAFILE = <file name> Default = DATAFL
DF
This parameter specifies the input file in which the collected keypoints reside.

PARSFILE = <file name> Default = PARSFL
PF
This parameter specifies the input file where the binary parsed descriptions reside. For more details see KPPAR section.

KPTFILE = <file name> Default = KPTDFL
KF
This parameter specifies the pre-processed keypoint output file for further processing by the data reduction programs.

TRACFILE = <file name> Default = TRACFL
TF
This parameter specifies the output file for the trace report.

TASKFILE = <file name> Default = TASKFL
TKF
This parameter specifies the name of the output file to which pre-processed keypoints for the task specified by the TASKID parameter are to be written.

KLUGFILE = <file name> Default = KLUGFL
KLF
This parameter specifies the name of the input file which the pre-processor checks to alter the keypoints it reads in to make them fit the descriptions the pre-processor has. It is a measure to provide a temporary fix for discrepancies between the keypoint common deck descriptions and the actual implementation.

Note that two file names are currently hard-coded. The input parameters are found on file "PARMFL" and debug output is written to file "DEBGFL".

8.4 DESCRIPTION OF FILES USED

8.4.1 INPUT FILES

PARMFL: This file contains the input parameters for the control of the execution of the program.

DATAFL: This file is a binary file composed of eight 12-bit parcels per keypoint (see Data Collection for more information).

PARSFL: This file contains the parsed keypoint descriptions. See KPPAR section for more details.

KLUGFL: This file is used to redefine keypoint descriptions and incompatible keypoint entries between the common decks and the actual code.

Language specification for the kludge file:

```
kludge_file     ::= [ <kludge> ...]
kludge          ::= <old_class> <spc> <old_kpn> <spc>
                    <kludge_action>
old_class       ::= <class_range>
class_range     ::= 0..0f(16)
old_kpn         ::= <kpn_range>
kpn_range       ::= 0..0fff(16)
kludge_action   ::= D <eol> !
                    C <spc> <new_class> <spc> <new_kpn> <eol> !
                    CD [ <spc> <match_value>] <eol>
                    <keypoint_descriptor>
new_class       ::= <class_range>
new_kpn         ::= <kpn_range>
match_value     ::= 0..0fff(16)
keypoint_descriptor ::= as described in IPNDOC appendix D.1.5
```

Example of a kludge file:

8.0 KEYPOINT PREPROCESSOR (KPPRE)
8.4.1 INPUT FILES

```
 2 4001 C 3 4001
 3 4001 C 2 4001
 8 761 d
 9 761 d
 2 1654 C 4 1654
 2 250 CD
 {E 'clp$old data length was 48' 'command' A20 }
 4 297 D
 3 1106 cd 9
 {X 4.9 'chg desc w/ mtch.val, add data field' 'segment' H16}
```

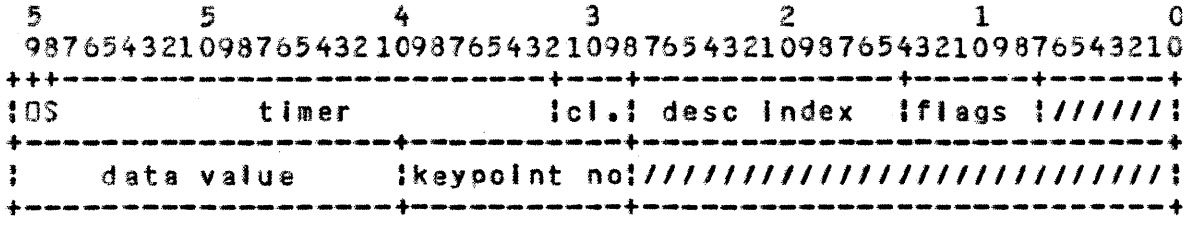
8.0 KEYPOINT PREPROCESSOR (KPPRE)
8.4.2 OUTPUT FILES

8.4.2 OUTPUT FILES

KPTOFL: This is an output file containing the legible compressed descriptions in the first record and the pre-processed keypoints in the second in the form of pairs of words.

The descriptions are parsed from the common decks into the following format: Index, Section Id, Class No., Kpn, Match Value, Match Length, Flag, Data Length, Data Format, Data Id, Indentation, Message.

The second record consists of binary pairs of words (for the hard keypoints) containing information as shown in the diagram below.



Where:
O - buffer overflow flag,
S - software lost data flag,
flags: MDCTjTm
M - 180 monitor mode (1 bit),
D - split data flag (1 bit),
C - 170 state (1 bit),
Tj - job trap handler (2 bits),
Tm - monitor trap handler (2 bits).

TRACFL: This file lists all keypoints encountered and all the data which they were carrying. This includes times, delta times, kpt numbers, classes, task numbers, modes, data, data format, section and message. Some of the information is drawn from the corresponding descriptor entry.

If a specific task has been requested then only the task with the requested task number will appear in the trace file.

The trace file also contains summaries of the keypoint descriptions, the undefined keypoints encountered, the number of keypoints encountered, the hardware and software overflows

8.0 KEYPOINT PREPROCESSOR (KPPRE)
8.4.2 OUTPUT FILES

and how many were deleted from the output file.

TASKFL: This file is in exactly the same format as KPTDFL except that only those keypoints found for the specified task are written onto the second record. It is intended to be used as input into further data reduction phases when only a single task is of interest.

DESCFL: This file contains the compressed keypoint descrictions for all the keypoints in the system or only the keypoints which are in the collection file (see parameters DSC or CDS).

DEBGFL: This file contains a listing of the parameters, and other debugging information. If the trace toggle is set there may be diagnostics for monitor and trap entry/exit out of sequence.

8.0 KEYPOINT PREPROCESSOR (KPPRE)
8.5 OUTPUT REPORTS

8.5 OUTPUT_REPORTS

8.5.1 KEYPOINT TRACE

Time	Delta T	DATA	Rep Err		Ident	Cl	Kptn	Trp Mtr	Task	Sct	Ty	Description
4926	87	400	H		MCR	2	4001	M	17	MT	E	ENTER 180 MONITOR MODE
4955	29	0	H			2	4002	C	17	MT	E	ENTER NOS 170
5525	570	40	H		MCR	3	4002	C	17	MT	X	EXIT NOS 170
5583	58	17	H		task_id	3	4001	J	17	MT	X	EXIT 180 MONITOR MODE
5606	23	C	H		fid.ord	2	171	J	17	BA	E	bap$get_next
5609	3	FFFF	H			0	4095	J	17			*** UNEXPECTED DATA KPT
5819	210	30	H	*	fba	2	152	J	17	BA	E	bap$block_manager
5942	123	30	H			4	152	J	17		D	******** UNDEFINED KEYP
5952	10	0	H			3	152	J	17	BA	X	bap$block_manager
6062	110	10	H			4	171	J	17		D	******** UNDEFINED KEYP
6148	86	0	I		status	3	171	J	17	BA	X	bap$get_next
6158	10	656F72	H			0	3954	J	17			*** UNEXPECTED DATA KPT
6161	3	10	H			0	16	J	17			*** UNEXPECTED DATA KPT
6165	4	30	H			0	48	J	17			*** UNEXPECTED DATA KPT
6192	27	0	I		status	3	157	J	17	BA	X	bap$control Status = 0 => no
6205	13	0	I		status	3	119	J	17	AM	X	amp$get_next_key
6303	98	C	H		fid.ord	2	112	J	17	AM	E	amp$fetch_access_information
6338	35	101	I		op	2	157	J	17	BA	E	bap$control
6368	30	C	H		fid.ord	2	165	J	17	BA	E	bap$fetch_access_info
6411	43	0	I		status	3	165	J	17	BA	X	bap$fetch_access_info
6420	9	0	I		status	3	157	J	17	BA	X	bap$control Status = 0 => no
6513	93	0	I		status	3	112	J	17	AM	X	amp$fetch_access_information
6731	218	0	H			3	283	J	17	CL	X	clp$get_data_line
6957	236	0	H			2	280	J	17	CL	E	clp$get_command_origin
7046	79	0	H			3	280	J	17	CL	X	clp$get_command_origin
7481	435	2	H		fid.ord	2	129	J	17	AM	E	amp$put_next
7557	86	2	H		fid.ord	2	148	J	17	AM	E	validate_caller_privilege
7606	39	0	I		status	3	148	J	17	AM	X	validate_caller_privilege
7638	32	0	H			2	1631	J	17	PM	E	pmp$get_task_id
7651	13	0	H			4	1665	J	17	PM	D	pmp$find_executing_task_xc
7663	12	0	I		status	3	1631	J	17	PM	X	pmp$get_task_id
7862	199	3	H		fid.ord	2	129	J	17	AM	E	amp$put_next
7898	36	134	I		op	2	157	J	17	BA	E	bap$control
7929	31	3	H		fid.ord	2	182	J	17	BA	E	bap$put_next: wsa, wsl
8060	131	1086	H	*	fba	2	152	J	17	BA	E	bap$block_manager
8185	125	1086	H			4	152	J	17		D	******** UNDEFINED KEYP
8195	10	0	H			3	152	J	17	BA	X	bap$block_manager
8301	106	14	H			4	182	J	17		D	******** UNDEFINED KEYP
8419	118	0	I		status	3	182	J	17	BA	X	bap$put_next: ba
8423	4	1086	H			0	182	J	17			*** UNEXPECTED DATA KPT

8.5.2 KEYPOINT SUMMARY

KEYPOINT SUMMARY

Count	Sect	Cl	Ty	Kpn	Message
4	AM	2	E	101	amp$access_method
4	AM	3	X	101	amp$access_method
4	AM	2	E	105	amp$close
4	AM	3	X	105	amp$close
1	AM	2	E	108	amp$copy_file
1	AM	3	X	108	amp$copy_file
1	AM	2	E	111	amp$fetch
1	AM	3	X	111	amp$fetch
28	AM	2	E	112	amp$fetch_access_information
28	AM	3	X	112	amp$fetch_access_information
26	AM	2	E	113	amp$fetch_fap_pointer
26	AM	3	X	113	amp$fetch_fap_pointer
5	AM	2	E	117	amp$get_file_attributes
5	AM	3	X	117	amp$get_file_attributes
28	AM	2	E	119	amp$get_next_key
28	AM	3	X	119	amp$get_next_key
4	AM	2	E	121	amp$get_partial
4	AM	3	X	121	amp$get_partial
4	AM	2	E	125	amp$open
4	AM	3	X	125	amp$open
2	AM	2	E	127	amp$put_direct
2	AM	3	X	127	amp$put_direct
73	AM	2	E	129	amp$put_next
73	AM	3	X	129	amp$put_next
6	AM	2	E	130	amp$put_partial
6	AM	3	X	130	amp$put_partial
6	AM	2	E	133	amp$return
6	AM	3	X	133	amp$return Status = 0 => normal
1	AM	2	E	146	amp$store_fap_pointer
1	AM	3	X	146	amp$store_fap_pointer
29	AM	2	E	148	validate_caller_privilege
29	AM	3	X	148	validate_caller_privilege
21	OS	2	E	1403	osp$set_status_abnormal
21	OS	3	X	1403	osp$set_status_abnormal
17	OS	2	E	1404	osp$append_status_parameter
17	OS	3	X	1404	osp$append_status_parameter
1	OS	2	E	1405	osp$append_status_integer
1	OS	3	X	1405	osp$append_status_integer
121	OS	4	D	1447	allocate
106	OS	4	D	1448	free
61	BA	2	E	152	bap$block_manager
61	BA	3	X	152	bap$block_manager
3	BA	2	E	155	bap$close

8.0 KEYPOINT PREPROCESSOR (KPPRE)
8.5.3 UNKNOWN KEYPOINT SUMMARY

8.5.3 UNKNOWN KEYPOINT SUMMARY

UNDEFINED KEYPOINT SUMMARY

Class	Keypoint	Count
15	0	2
2	1654	1
2	1132	3271
3	1132	3271
2	1133	3271
3	1133	3271

TOTAL KPT PROCESSED = 219118
DELETED KPT = 0
TOTAL KPT ON PROCESSED FILE = 219118
TOTAL UNDEFINED KEYPOINTS = 13087

8.6 OPERATIONAL_PROCEDURES

8.6.1 INSTALLATION

To install this program three files are needed: PMFPL, XULIB and PMFPLIB. To compile and save the binary of the pre-processor program type:

```
SES.PMFINST KPPRE
```

8.6.2 KPPRE RUN PROCEDURES

An SES proc file is supplied in order to run the pre-processor program. The following parameters are used within the procfile.

Any of the parameters defined in the SES module JOBPARM may be used. These include JOBTL, JOBFL, JOBCN, JOBPR, JOBUN, JOBPW, JOBFMLY, JOBCN, JOBPN, LOCAL, BATCHN, BATCH, DEFER, NODAYF, DAYFILE, AND DF. (See SES Procedure Writer's Guide, Appendix A)

TPF — If the raw input data is on tape, TPF is used to specify the VSN(s) of the tape(s) on which the data is found. If the VSN(s) have leading 0s the VSN must be enclosed in single quote marks. If the data is spread across more than one reel, the list of VSNs must be enclosed in parenthesis. Up to 10 reels of tape may be specified. Tapes must be labeled.
Example TPF=('000123',123456) Default none.

DTF — If the raw keypoint data is on mass storage, DTF specifies the file name in which the data is stored. The file may be local, direct access, or even indirect access although the last alternative is usually impractical.
Example DTF=DATAFL Default DATAFL.

DSF — This parameter specifies the file where the keypoint descriptors will be stored.
Example DSF=DESCFL Default none, required parameter.

PRF — Parsed keypoints common decks file. This file is the output file from KPPAR. If the PRF file is not found, KPPRE runs KPPAR to create one.
Example PRF=PARSFL Default PARSFL.

8.0 KEYPOINT PREPROCESSOR (KPPRE)
8.6.2 KPPRE RUN PROCEDURES

KLF — This parameter allows specification of a file containing information about keypoints which should be ignored or erroneous (usually because they are improperly installed).
Example KLF=KLUGFL — Default KLUGFL.

KPF — This parameter specifies the file on which the preprocessed keypoints will be stored.
Example KPF=KPF1 — Default KPTDFL.

SVF — This parameter allows the VSN(s) of an output tape on which the preprocessed keypoints will be stored. Note that SVF specifies only the VSN(s), KPF is still required, although KPF is now the LFN of the tape. Up to 10 VSN(s) may be specified.
Example SVF=('000123','000456') — Default none.

TRF — This parameter specifies the file name for the trace file which stores diagnostics and data about the keypoints.
Example TRF=TRACFL — Default TRACFL.

TKF — This parameter specifies a preprocessed keypoint file for a selected task.
Example TKF=TASK1 — Default TASKFL.

SKP — Number of keypoints to skip at the beginning of the raw keypoint file.
Example SKP=1000 — Default 0.

MKP — Maximum number of keypoints to process from the raw keypoint data file.
Example MKP=2000 — Default 99999999

STR — Start class and keypoint. This parameter allows searching the raw keypoint data file to the point where a keypoint of a given class is found. The format is 'CLASS KEYPOINT'. Space between class and keypoint is required. See note on cycling in STP description.
Example STR='4 4000' — Default '-1 -1'.

STP — Stop class and keypoint. This parameter allows stopping the preprocessor after a given keypoint has been found. Note, however, that collection will resume if another keypoint of STR class and keypoint is found, so that this start stop process is cyclical.
Example STP='4 4001' — Default '-1 -1'.

TSK — This parameter allows selection of only those keypoints

corresponding to a given task to be preprocessed.
Example TSK=5 — Default -1, (all tasks).

PRE — This is a toggle to produce a preprocessed keypoint file.
Example PRE=N — Default Y.

CDS — This is a toggle to produce a compressed keypoint descriptor file.
Example CDS=N — Default Y.

RDS — This is a toggle to replace the description into the PL.
Example RDS=Y — Default N.

FLT — This parameter specifies the type of input data on the keypoints file. Allowable types are HARD, SOFT, SIM.
Example FLT=SOFT — Default HARD.

TRC — This parameter specifies whether trace output is to be generated.
Example TRC=Y — Default N.

DSC — This parameter specifies whether a full keypoint descriptor file is to be generated.
Example DSC=Y — Default N.

ID — This parameter allows specification of the UN of the workspace containing certain files. This allows a single copy of these files to exist, simplifying procedures when they are updated. The files accessed using the ID parameter are KLF, PRF and DTF.
Example ID=KEYPERF — Default user's own UN.

LIB — Name of file where the object code for XKPPRE is found. Ordinarily this parameter is obtained from the 'toollib' parameter in the user's PROFILE. This parameter would ordinarily be used for testing a new version of the preprocessor, or for use of a specially modified version of the preprocessor.
Example LIB=MYLIB — Default XULIB.

LIBUN — User number where file specified by LIB (above) resides. Ordinarily this parameter is obtained from the 'toolact' parameter in the user's PROFILE.
Example LIBUN=UN12 — Default KEYPERF.

8.0 KEYPOINT PREPROCESSOR (KPPRE)
8.6.2 KPPRE RUN PROCEDURES

Some examples of how to invoke the KPPRE procedure follow:

SES.KPPRE DSF=DESCF	This call will pre-process file DATAFL and output the pre-processed keypoints into default file KPTDFL.
SES.KPPRE DSF=D1 TRC=Y DSC=Y	This call will do the above and will also generate the trace output and print the full keypoint descriptor file.
SES.KPPRE TPF=ABC002 DSF=DSF1 .. PRE=N TSK=4	This call will process the keypoints collected on tape ABC002. It will not generate a full pre-processed keypoint file but will output a selective one for keypoints belonging to task number 4 (TASKFL).
SES.KPPRE DTF=KPTS DSF=D1 TRC=Y	This call will pre-process the keypoints on disk file KPTS, generating a full trace report and outputting the pre-processed keypoints on file KPTDFL.
SES.KPPRE DSF=D1 TRC=Y MKP=1000	This will produce a processed keypoint file called KPTDFL from the first 1000 keypoints of the input file DATAFL. It will also produce and print the trace file TRACFL containing the first 1000 keypoints.
SES.KPPRE DTF=OLDPAR DSF=D1	This will produce and print the file KPTDFL containing the list of the descriptions contained on the binary parsed file OLDPAR.

8.0 KEYPOINT PREPROCESSOR (KPPRE)
8.6.2 KPPRE RUN PROCEDURES

```
SES.KPPRE DSF=D1 MKP=1000 SKP=10000 ..
  TSK=24                          This will create and run a batch
                                  job to pre-process the 10000 to
                                  11000th keypoints (approx.), and
                                  print their descriptions. (The
                                  CDS is forced on anyway). The
                                  job will also create TASKFL
                                  which contains the pre-processed
                                  keypoints of task number 24
                                  decimal.
```

9.0 PRIMARY DATA REDUCTION (KPRED)

9.0 PRIMARY DATA REDUCTION (KPRED)

The PMF Primary Data Reduction program (KPRED) is designed to analyze keypoints collected during a single data collection and produce system level analysis reports. Later it will be expanded to allow the data reduced to be broken out by time period for use by the Extended Reporting Facility.

The data reduction program was written to provide an early tool to reduce keypoint data in order to give some statistics to support performance allocation and software development for the CYBER 180 and to form a base for long term reporting software.

KPRED currently runs in 170 state under NOS versions 1 and 2.

9.0 PRIMARY DATA REDUCTION (KPRED)
9.1 KPRED GENERAL DESCRIPTION

9.1 KPRED GENERAL DESCRIPTION

The keypoint data reduction program reduces keypoint data gathered from the NOS/VE operating system. Keypoints must be processed by the keypoint pre-processor previously described before they may be analyzed by KPRED.

The KPRED program is a tree-structured, modular program which reads the pre-processed keypoint file created by the keypoint pre-processor and summarizes system level statistics from this data.

Keypoint instructions are disseminated throughout the operating system, product set, and user code. Classes and Keypoint numbers are used to identify uniqueness of a keypoint. See the SIS/180 for further details of this structure.

9.2 PROGRAM DESCRIPTION

The program has been written with the aim of satisfying two kind of customers: system development and system performance. Thus, the program has been designed to have a common data gathering core with expansible data statistic layers.

9.2.1 DATA GATHERING

The common data gathering core is composed of a large multi-rooted tree where leaves represent pertinent information for identified keypoints.

In figure 2.1.1 the structure of the tree which keeps keypoint information is shown.

Dynamic allocation is used to allocate space only for different keypoint numbers (i.e. space is directly proportional to the number of different keypoints and not the number of collected keypoints).

The basic structure is composed as follows:

- A pointer (root) to each section: system, product and user.
- An array which contains the description of all the different keypoints.
- An array which contains per each sub-section (or per every block of 50 keypoint numbers); a pointer to the first and last allocated keypoint block for the sub-section (if any).
- A keypoint block contains uniqueness information to identify it; counters and vectors used for the histograms (dynamically allocated if requested); a pointer to the next block (if any); a pointer to the first and last data block (if any) which was gathered within the keypoint number.
- A data block contains the data itself, a count for the data, a pointer to the next block (if any), and a pointer to the first and last data keypoint block (if any) which would be the next following keypoint with no keypoint identification.
- A data keypoint block contains the data, a count and a

9.0 PRIMARY DATA REDUCTION (KPRED)
9.2.1 DATA GATHERING

pointer to the next block (if any).

- Task information is kept in a link list representation. Only existing tasks will use space.
- Unit (disk unit) information is kept in a link list form where only existing units will have space allocated.
- Tables for different summary information are dumped every specified time interval to the summary file.

9.0 PRIMARY DATA REDUCTION (KPRED)
9.2.1 DATA GATHERING

FIGURE 2.1.1 - Tree Structure of the Keypoint Information

9.2.2 DATA STATISTICS

Most of the statistics are computed by walking through the tree structure and collecting useful data to be used for computation. An independent special section has been defined in order to collect data to produce performance reports. This section is expansible according to needs since keypoint numbers are hard-coded.

The need to hard-code these keypoint numbers is attributed to the fact that these reports need recognition of events which involve sequence of keypoints (or a single keypoint) that have to be identified and trapped through one of these special processes in order to generate the reports.

9.2.3 STATISTIC OUTPUTS

Several forms of output are generated according to selection of input toggles (A, B, C, D, E, H, J, K, L, N, R, S, T and X).

SUMMARY

The summary statistics are generated exclusively by keeping accounting for each keypoint, they include:

- Elapsed time.
- Total number of keypoints.
- Keypoint frequency.
- Summary per each sub-section: number of keypoint and averages.
- Total number of keypoints by class.

This output will always be printed even with all outputs disactivated.

A==SYSTEM KEYPOINTS

This means only Keypoints of classes 0-5 are analyzed.

B==PRODUCT SET KEYPOINTS

Only keypoints of classes 6-10 are analyzed.

C==USER KEYPOINTS

Only keypoints of classes 11-14 are analyzed.

By default all keypoints are identified.

9.0 PRIMARY DATA REDUCTION (KPRED)
9.2.3 STATISTIC OUTPUTS

D=DATA STATISTICS

This information is gathered using the data field within the keypoint number and the data keypoint. The printout will be for each section and for each different keypoint if data exist. The output will contain:

- Keypoint name and number.
- Data content and data keypoint content.
- Counts and percentages.

Following is an example of the output:

```
PURGE BUFFER  4000   SVA   COUNT   %-COUNT

                     4F0     8      10.26   DATA-KPT COUNT   %-COUNT

                                                    8          1     12.50
                                                 1008          1     12.50
                                                 2008          1     12.50
                                                 3008          1     12.50

Kpt.name          Content          % of occurrences      % within the
                  data field       within the sub-       kpt. no.
                                   section

        Kpt. no.          Occurrences          Content of
                Name of                        data kpt.
              the data field                              Occurrences
```

9.0 PRIMARY DATA REDUCTION (KPRED)
9.2.3 STATISTIC OUTPUTS

E = ERRORS

Informative errors are printed to warn the user that something did not follow all rules governing keypoint sequences. Following is an example of an error message:

```
   3277001     .108.     Procedure entry/exit not in sequence.

 Kpt. time    error no.            error description
  stamp
```

The time stamp is printed because it has uniqueness among all other fields thus identifying the keypoint without ambiguity since no two time stamps are identical.

A summary table showing how many errors of which type is also printed at the end of the run.

H, L, N, X = HISTOGRAMS

For each identified keypoint three histograms are generated:

- Keypoint frequency.
- Time to next procedure call (or next occurrence of same keypoint).
- Time between entry/exit of procedures (where entry and exit kpt. are specified).

L - Histogram for Last Call (last occurrence).

N - Histogram for Next Keypoint (frequency).

X - Histogram for entry/exit of procedures.

Thus H includes all histograms. However L, N and X affect only the corresponding histograms.

A sample histogram output is shown in the Output Reports section.

9.0 PRIMARY DATA REDUCTION (KPRED)
9.2.3 STATISTIC OUTPUTS

J.K = TASK STATISTICS

For each task in the system the following data is produced.

- User time in microseconds and as a percentage of the total task time.
- Monitor time in microseconds and as a percentage of the total task time.
- Total time, percent against total run.
- Chronological order of appearance for all keypoints with counts and percentages;
- Total and percentages for the entire run;
- Segment and page faults information.

The K toggle is used to print a keypoint summary by task.

The NOS/VE monitor exit keypoint (number 4001) is used to keep task accounting.

Following is an example of the output:

J Toggle

Task no.	User Time	% UM	MTR Time	% MM	TOT. Time	% Time
	21753	43.44	21028	32.86	42781	37.50
	Chronological order if selected (K toggle)					
	Segment and Page faults information					
TOTAL	50073	43.90	63997	56.10	114070	100.00
DIFFER	50073	0.00	68205	6.17	114070	0.00

9.0 PRIMARY DATA REDUCTION (KPRED)
9.2.3 STATISTIC OUTPUTS

K_Ioggle

	COUNT	KEYPOINT % TASK	% TOTAL
pem$start_stop_Kpt	1	0.47	0.17
pmK$execute	2	0.94	0.33
allocate space in heap	9	4.25	1.50
:			
:			
TOTAL	601		99.83
DIFFER.	1		0.17

The line named "DIFFER" is a checkpoint sum which is able to verify if the accounting was kept accurately. The "1" showed under the keypoint count is caused by a faulty keypoint which gave an error condition, thus it was not recorded.

9.0 PRIMARY DATA REDUCTION (KPRED)
9.2.3 STATISTIC OUTPUTS

R - Reports

This output should give the performance evaluation report. However, since most of the keypoints are not available to produce such a full report, only a small subset is generated.

- CPU Usage (per each CPU):

  - MM time with percentage,
  - UM time with percentage,
  - Total CPU time,
  - HW/SW comparison times.

Keypoint #4001 is used to keep the CPU time accounting.

- Memory Manager

  - Total page faults/sec.
  - Page faults/sec. with I/O.
  - Page faults/sec. without I/O, broken down into:
    - new pages,
    - reclaimed,
    - others.

Keypoint #1106 is used to keep accounting for the page faults.

- Peripheral Device Usage

  - no. of I/O requests,
  - avg. byte transfer
  - type of request (input, output),
  - avg. I/O queue length.

No keypoints are available at the time of this update for I/O statistics.

- Summary Histograms

  - overall keypoint frequency,
  - overall time to next procedure call,
  - overall time between entry/exit of procedures.

9.0 PRIMARY DATA REDUCTION (KPRED)
9.2.3 STATISTIC OUTPUTS

S = Summary by sub-section

For each sub-section within each section (system, product, user), a summary report is produced:

- Keypoint name and keypoint number
- Count, percentage, average within the sub-section.
- Average frequency, next call, and entry/exit.
- Total by section, average, and maximums.

Following is an example of this output.

```
LG                                  USAGE BY FUNCTION

Kpt-name        Kpt   Count %Count  %Count Avg.Time Max  ...  Avg.Time Max
                No.           Module  Total Next Call     ...  Ent/Ext

LGk$LOG_ASCII   850       4  100.00   0.66  47165.00 91856    2012.50 2110
                            4           0.66  47165.00 91856    2012.50 2110

Sub-section
Name
```

I = Trace Dump

For every keypoint, one line of output is produced using the following format:

```
2348  12   288  H   PFTI    4    1222  M   MM  3  D  Make page table entry


time     data  format    class#  kpt#      subsect.     Keypoint description
(usec)  field  (H=hex.)                  MM=memory manager
    delta          data                CPU Mode  task no.
    time        description      M=monitor     Class Type (D=debug)
```

9.3 DESCRIPTION OF FILES USED

9.3.1 INPUT FILES

INPUT: This file contains the program input parameters (toggles).

FKPDS: This file contains the description of all the keypoints in the data file. It is the first record of the data file KPTDFL which is an output file from the pre-processor program KPPRE.

FKPDT: This file contains the pre-processed keypoints. It is the second record of the output file KPTDFL generated by the pre-procesor program KPPRE.

FKPDD: This file contains information for data statistics recording. This reduces the runtime CM usage considerably when used correctly. The file format is:

Class Keypoint D (for delete)
this mean the data will not be recorded for that specific keypoint.

9.3.2 OUTPUT FILES

OUTPUT: This file contains all the selected outputs: trace, histograms, summary reports, data statistics, etc.

FDEBG: This file contains a small amount of debugging information which keeps tracks of the logic flow of the program.

FKPSM: This file contains summary information per selected time interval for the I/O, System, Memory Mgr., Task Mgr., etc. This file will be used as input by the summary reports generator.

9.0 PRIMARY DATA REDUCTION (KPRED)
9.4 OUTPUT REPORTS

9.4 OUTPUT REPORTS

9.4.1 KEYPOINT TRACE

OV	TIME MICROSEC	DELTA TIME	DATA	R	DATA NAME	CLSS	KPT	M D	SC TN	TASK	T	DESCRIPTION
	0	0	0			15	0	U		0	P	** UNDEFINED KEYPOINT **
	346	346	0	H		2	256	U	CL	0	E	CLP$GET_VALUE
	668	322	0	H		3	256	U	CL	0	X	CLP$GET_VALUE
	692	24	0	H		2	289	U	CL	0	E	CLP$POP_BLOCK_STACK
	1351	659	0	H		3	289	U	CL	0	X	CLP$POP_BLOCK_STACK
	1415	64	400	H		2	4004	U	MT	0	E	ENTER JOB TRAP HANDLER
	1584	169	0	H		2	1617	U	PM	0	E	PMP$ESTABLISH_CONDITION_HANDLER
	1677	93	0	I	STATUS	3	1617	U	PM	0	X	PMP$ESTABLISH_CONDITION_HANDLER
	1894	217	0	H		2	1647	U	PM	0	E	PMP$TASK_DEBUG_MODE_ON
	1900	6	0	I	DBG_MODE	3	1647	U	PM	0	X	PMP$TASK_DEBUG_MODE_ON
	1955	55	0	H		2	1617	U	PM	0	E	PMP$ESTABLISH_CONDITION_HANDLER
	2040	85	0	I	STATUS	3	1617	U	PM	0	X	PMP$ESTABLISH_CONDITION_HANDLER
	2093	53	0	H		2	1617	U	PM	0	E	PMP$ESTABLISH_CONDITION_HANDLER
	2181	88	0	I	STATUS	3	1617	U	PM	0	X	PMP$ESTABLISH_CONDITION_HANDLER
	2373	192	0	H		2	1617	U	PM	0	E	PMP$ESTABLISH_CONDITION_HANDLER
	2457	84	0	I	STATUS	3	1617	U	PM	0	X	PMP$ESTABLISH_CONDITION_HANDLER
	2581	124	0	H		2	1617	U	PM	0	E	PMP$ESTABLISH_CONDITION_HANDLER
	2705	124	0	I	STATUS	3	1617	U	PM	0	X	PMP$ESTABLISH_CONDITION_HANDLER
	2812	107	0	H		2	1617	U	PM	0	E	PMP$ESTABLISH_CONDITION_HANDLER
	2902	90	0	I	STATUS	3	1617	U	PM	0	X	PMP$ESTABLISH_CONDITION_HANDLER
	3057	155	80	H	MCR	3	4004	U	MT	0	X	EXIT JOB TRAP HANDLER
	3079	22	0	H		3	250	U	CL	0	X	CLP$PROCESS_COMMAND
												3079 .117. NEW BLOCK WITH EXIT
	3260	181	0	H		2	283	U	CL	0	E	CLP$GET_DATA_LINE
	3424	164	10	H	FID.ORD	2	119	U	AM	0	E	AMP$GET_NEXT_KEY
	3498	74	10	H	FID.ORD	2	199	U	BA	0	E	BAP$SYS_BLK_VARIABLE_REC_FAP
	3523	25	10	H	FID.ORD	2	171	U	BA	0	E	BAP$GET_NEXT
	3810	287	0	I	STATUS	3	171	U	BA	0	X	BAP$GET_NEXT
	3821	11	0	I	STATUS	3	199	U	BA	0	X	BAP$SYS_BLK_VARIABLE_REC_FAP
	3833	12	0	I	STATUS	3	119	U	AM	0	X	AMP$GET_NEXT_KEY
	3925	92	10	H	FID.ORD	2	112	U	AM	0	E	AMP$FETCH_ACCESS_INFORMATION
	3963	38	10	H	FID.ORD	2	199	U	BA	0	E	BAP$SYS_BLK_VARIABLE_REC_FAP
	3988	25	10	H	FID.ORD	2	101	U	AM	0	E	AMP$ACCESS_METHOD
	4034	46	101	I	OP	2	157	U	BA	0	E	BAP$CONTROL
	4066	32	10	H	FID.ORD	2	165	U	BA	0	E	BAP$FETCH_ACCESS_INFO
	4113	47	0	I	STATUS	3	165	U	BA	0	X	BAP$FETCH_ACCESS_INFO
	4123	10	0	I	STATUS	3	157	U	BA	0	X	BAP$CONTROL STATUS = 0 => NORMAL
	4132	9	0	I	STATUS	3	101	U	AM	0	X	AMP$ACCESS_METHOD
	4141	9	0	I	STATUS	3	199	U	BA	0	X	BAP$SYS_BLK_VARIABLE_REC_FAP
	4153	12	0	I	STATUS	3	112	U	AM	0	X	AMP$FETCH_ACCESS_INFORMATION
	4379	226	0	H		3	283	U	CL	0	X	CLP$GET_DATA_LINE

9.0 PRIMARY DATA REDUCTION (KPRED)
9.4.2 ERROR SUMMARY

9.4.2 ERROR SUMMARY

ERR-NO	ERROR MESSAGE	COUNT
100	KEYPOINT MATCH NOT FOUND	26
104	CPU STATE OUT OF SEQUENCE	4
113	PROCEDURE EXIT NOT IN SEQUENCE	9

9.0 PRIMARY DATA REDUCTION (KPRED)
9.4.3 KEYPOINT SUMMARY

9.4.3 KEYPOINT SUMMARY

PMF RUN

CONTROL-TOGGLES CLNT RUN-NO. 99 START-TIME 0 STOP-TIME 54212533 ELAPSED-TIME

TOTAL-KPT 13534 KPT-FREQUENCY 4005.65 USEC HW-BUFF-OVFL 1 SW-BUFF-OVFL

CLASS	COUNT	%-TOT
1	6	0.04
2	6632	49.00
3	6646	49.11
4	1	0.01
8	132	0.98
9	114	0.84
15	3	0.02
	13534	100.00

SECTN-NAME	COUNT	PERCENT
AM (100)	572	4.23
BA (150)	931	6.88
BA (200)	2	0.01
CL (250)	2007	14.83
DM (400)	1062	7.85
DM (450)	120	0.89
LG (850)	8	0.06
MM (1100)	514	3.80
MM (1150)	174	1.29
OS (1400)	292	2.16
PM (1600)	846	6.25
PM (1650)	5	0.04
ST (1850)	8	0.06
TM (1900)	508	3.75
AV (2100)	2	0.01
IO (2200)	284	2.10
RM (2250)	34	0.25
MT (4000)	5893	43.54
FC (250)	10	0.07
CC (750)	236	1.74
	13508	99.81

9.0 PRIMARY DATA REDUCTION (KPRED)
9.4.4 SYSTEM SUMMARY

9.4.4 SYSTEM SUMMARY

MT CPU USAGE

CPU 0

CPU-TIME	54212533	100.00%
IDLE	0	0.00%
TOTAL	54212533	
MTR-MODE	50707911	93.54%
USR-MODE	2615508	4.82%
170-MODE	889114	1.64%

	OCCURRENCES	FREQUENCY	AVG.DURATION
MONITOR MODE	1289	42057.82	39338.95
USER MODE	1290	42025.22	2027.53
170 MODE	1275	42519.63	697.34
MONITOR MODE TRAPS	357	151855.83	497.38
USER MODE TRAPS	24	2258855.54	76766.42

9.0 PRIMARY DATA REDUCTION (KPRED)
9.4.5 PAGING SUMMARY

9.4.5 PAGING SUMMARY

MM

	P A G E F A U L T S					
	O/S		USER		TOTAL	
	PER SEC.	COUNT	PER SEC.	COUNT	PER SEC.	COU
PAGE FAULTS	3.87	210	0.00	0	3.87	210
PAGE FAULTS (I/O)	1.75	95	0.00	0	1.75	
PAGE FAULTS (NO I/O)	2.12	115	0.00	0	2.12	
RECLAIMED	0.46	25	0.00	0	0.46	
NEW	1.55	84	0.00	0	1.55	
PT FULL	0.11	6	0.00	0	0.11	

9.0 PRIMARY DATA REDUCTION (KPRED)
9.4.6 TASK MANAGER SUMMARY

9.4.6 TASK MANAGER SUMMARY

TM TASK MANAGER

CPU 0

TASK SWITCHES	1290
TASK SWITCHES TO SAME TASK	1287
TASK SWITCHES FREQUENCY	42025.22

9.0 PRIMARY DATA REDUCTION (KPRED)
9.4.7 COMMON I/O SUMMARY

9.4.7 COMMON I/O SUMMARY

ID PERIPHERAL DEVICE USAGE

To be supplied.

9.0 PRIMARY DATA REDUCTION (KPRED)
9.4.8 HISTOGRAM EXAMPLE

9.4.8 HISTOGRAM EXAMPLE

TIME BETWEEN ENTRY AND EXITS (MICROSEC)

	%X-GRP	Y-STEP %Y	%X	37	0 370 740 1110 1480 1850
0		0	0	0	I
1		0	0	1	I
2		0	0	3	I
3		0	0	4	I*
4		0	0	5	I*
5		0	0	6	I*
6		0	0	8	I*
7	0.00	1	0	9	I***
8		0	0	10	I*
9		3	0	13	I*****
10		2	0	25	I****
11		2	0	38	I****
12		1	0	50	I***
13		1	0	63	I**
14		1	0	75	I**
15	0.01	2	0	88	I*****
16		2	0	100	I***
17		27	0	125	I**********
18		12	0	250	I**********
19		4	0	375	I********
20		6	0	500	I**********
21		2	0	625	I*****
22		2	0	750	I*****
23	0.46	3	0	875	I******
24		5	0	1000	I*********
25		11	0	1250	I**********
26		3	0	2500	I*******
27		2	0	3750	I****
28		1	0	5000	I**
29		0	0	6250	I*
30		0	0	7500	I*
31	1.27	0	0	8750	I*
32		0	0	10000	I*
33		1	0	12500	I**
34		0	0	25000	I*
35		0	0	37500	I*
36		1	1	50000	I**
37		0	0	62500	I*
38		0	0	75000	I*
39	3.59	0	1	87500	I*
:				:	
58		0	0	25000000	I
59		0	38	37500000	I*

TOTAL Y-AXIS
MAX PEAK Y-AXIS

TOTAL X-AXIS 2
MAX PEAK X-AXIS

VARIANCE 9.30296294
STAND.DEVIATION

SKEWNESS 4.47837218
REL.SKEWNESS

MEDIAN BIN

MINIMUM
MAXIMUM

AVERAGE

9.0 PRIMARY DATA REDUCTION (KPRED)
9.4.9 TASK STATISTICS EXAMPLE

9.4.9 TASK STATISTICS EXAMPLE

TASK-NO	USER-TIME	MODE	MTR-TIME	MODE	TOT-TIME	%-TOT	COUNT	%-TASK	%-TOT
73	81475	2.32	1381875	2.73	1463350	2.70			
	3 4001 EXIT 180 MONITOR MODE						77	14.18	0.
	2 129 AMP$PUT_NEXT						8	1.47	0.
	2 148 VALIDATE_CALLER_PRIVILEGE						4	0.74	0.
	2 1631 PMP$GET_TASK_ID						4	0.74	0.
	2 178 BAP$NULL_DEVICE						4	0.74	0.
	2 1617 PMP$ESTABLISH_CONDITION_HANDLER						52	9.58	0.
	2 263 CLP$SCAN_TOKEN						74	13.63	0.
	2 250 CLP$PROCESS_COMMAND						3	0.55	0.
	2 293 CLP$FIND_PATH_DESC_VIA_LFN						12	2.21	0.
	2 284 CLP$CONVERT_STRING_TO_FILE						4	0.74	0.
	2 4002 ENTER NOS 170						46	8.47	0.
	2 1918 SWITCH TASK						16	2.95	0.
	2 160 BAP$FETCH_ART_TABLE_POINTER						12	2.21	0.
	2 265 CLP$CONVERT_STRING_TO_NAME						18	3.31	0.
	2 117 AMP$GET_FILE_ATTRIBUTES						4	0.74	0.
	2 167 BAP$GET_FILE_ATTR						5	0.92	0.
	2 4003 ENTER MONITOR TRAP HANDLER						14	2.58	0.
	2 2201 IOP$PROCESS_IO_COMPLETIONS						6	1.10	0.
	2 444 DMP$TRANSFER_UNIT_WRITTEN						6	1.10	0.
	2 497 DMP$FETCH_EOI						2	0.37	0.
	2 288 CLP$PUSH_BLOCK_STACK						4	0.74	0.
	2 1600 PMP$EXECUTE_TASK						2	0.37	0.
	: :								
	2 251 CLP$SCAN_PARAMETER_LIST						2	0.37	0.
							543		4.

SEGMENT NUMBER	AVAIL QUEUE	AV.MOD QUEUE	VALID IN-PT	NO MEMORY	PG.WITH IO-ACTV	PAGE ON-DISK	PT-IS FULL	PG.IO DRV.REJ	NEW PAGE	BEYON FTL.LI
3	0	0	0	0	0	0	0	0	1	
F	0	1	0	0	0	0	0	0	0	
11	0	1	0	0	0	0	0	0	0	
5	0	5	0	0	0	0	0	0	0	
10	0	1	0	0	0	0	0	0	0	
28	0	4	0	0	0	0	0	0	0	
2B	0	1	0	0	0	0	0	0	0	
3B	0	0	0	0	0	1	0	0	0	
B	1	0	0	0	0	0	0	0	0	
3C	0	1	0	0	0	0	0	0	0	
1C	0	0	0	0	0	1	0	0	0	
	1	14	0	0	0	2	0	0	1	

9.0 PRIMARY DATA REDUCTION (KPRED)
9.4.10 DATA STATISTICS EXAMPLE

9.4.10 DATA STATISTICS EXAMPLE

PAGE TABLE IS FULL*	1 1101 PTI		COUNT	%-COUNT
		843	1	16.67
		845	1	16.67
		EBB	1	16.67
		FB7	1	16.67
		FAB	1	16.67
		F47	1	16.67
			6	100.00

PAGE TABLE FULL PROCESSOR	2 1146 PTI		COUNT	%-COUNT
		824	1	16.67
		826	1	16.67
		E9C	1	16.67
		F98	1	16.67
		F8C	1	16.67
		F28	1	16.67
			6	100.00

PAGE TABLE FULL PROCESSOR	3 1146 PTI		COUNT	%-COUNT
		1	5	83.33
		2	1	16.67
			6	100.00

DELETE SEGMENT BY SEGMENT NUMBER	2 1105		COUNT	%-COUNT
		1	2	100.00

DELETE SEGMENT BY SEGMENT NUMBER	3 1105		COUNT	%-COUNT
		1	2	100.00

FREE PAGES	2 1104		COUNT	%-COUNT
		C 12	2	100.00

9.0 PRIMARY DATA REDUCTION (KPRED)
9.4.11 SECTION SUMMARY

9.4.11 SECTION SUMMARY

MM USAGE BY FUNCTIONS

KPT-NAME	KPT-NO	COUNT	%-COUNT MODULE	%-COUNT TOTAL	AVG-TIME NEXT-CALL MILLISEC	MAX	AVG-TIME NEXT-KPT MICROSEC	MAX	AVG-TIM ENT/EX MICROSE
PAGE FAULT PROCESSOR	1106	420	81.71	3.10	29459.04	54042	326.56	4555	2645.7
SET/GET SEGMENT LENGTH MONITOR REQUEST	1142	32	6.23	0.24	19590.69	51958	146.69	550	485.5
PAGE TABLE IS FULL*	1101	6	1.17	0.04	8232.83	18405	248.83	377	0.0
PERIODIC CALL	1136	16	3.11	0.12	32370.00	49396	55.19	100	5310.2
PAGE TABLE FULL PROCESSOR	1146	12	2.33	0.09	33234.83	49396	984.17	2603	6898.1
DELETE SEGMENT BY SEGMENT NUMBER	1105	24	4.67	0.18	51649.58	53258	393.87	2686	2048.6
FREE PAGES	1104	4	0.78	0.03	51953.00	51959	157.25	313	305.0
		514	3.80		29735.83	54042	323.19	4555	2647.4

9.0 PRIMARY DATA REDUCTION (KPRED)
9.4.12 INTERVAL SUMMARY FILE

9.4.12 INTERVAL SUMMARY FILE

```
    10000
4000   9812 0  253      9128      252       510     259       175   ++++K+9== 09958RSB:IH
```

```
1900  19502 0  240     240
4000  29238 0  257      8946     257       600     250       191      0    0      32       16
1100  29238 0   19         0       0         0       0         0        7    1      0        11        0
1900  29238 0  257     257
4000  39212 0  241      9196     241       624     218       154      0    0      36       13
1100  39212 0   26         2       1         0       0         0        8    0      0        15        0
1900  39212 0  241     241
4000  48674 0  122      9155     122       191     138        94      0    0      80       44
1100  48674 0   50         3       2         0       0         0       37    2      0         6        0
1900  48674 0  122     122
```

9.5 OPERATIONAL_PROCEDURES

9.5.1 INSTALLATION

To install this program three files are needed: PMFPL, XULIB and PMFPLIB. To compile and save the binary of the data reduction program type:

```
SES.PMFINST KPRED
```

9.5.2 DATA REDUCTION OPTIONS

To run the data reduction program:

```
XKPRED,input,output,fdebg,fkpds,fkpdf,fkpsm,fkpdd.
```

input: TGL RNO SKP SKC MER MDT INT SCT

where:

TGL: program control toggles (ABCDEHJKLNRSTX),

RNO: run number,

MKP: maximum number of keypoints to process,

SKP: number of keypoints to skip at the beginning of the keypoint file,

SKC: start keypoint class,

MDT: maximum number of data lines to record,

MER: maximum number of errors to printed,

INT: summaries interval in millisec.,

SCT: only keypoints with this section number will be printed in the trace.

The input must follow these rules and/or restrictions:

- all the input parameters are optional. However, if the ith

parameter is specified, all his predecessors must also be present,

- any number of blanks can preceed or follow each of the tokens of the input, at least one blank has to appear in between each token,

- if the input file is empty the following defaults are assumed:

  . all outputs are printed (TGL = {}),
  . the run number is equal to zero (RNO = 0),
  . all keypoints are processed (MKP = -1),
  . no keypoints are skipped at the beginning of the data file (SKP = 0),
  . the start keypoint class is 15 (SKC = 15),
  . up to 1000 errors will be printed (MER = 1000),
  . up to 100 data fields will be recorded (MDT = 100),
  . no summary will be processed (INT = -1),
  . the trace will print all keypoints belonging to any section (SCT = 9999).

- TGL (optional), the program control toggles consist of a string of characters (A through Z), the string may be empty,

- RNO (optional), the run number is any positive integer of six digits or less.

- MKP (optional), the maximum number of keypoints to process is an integer which follows these rules:

  . 0 = no keypoints will be processed, only the PMF counters statistics will be printed (if PMF counters are present in the data file),
  . n = only up to n keypoints will be processed,
  . -n = process all keypoints,

- SKP (optional), the number of keypoints to skip is an integer following these rules:

  . 0 = do not skip keypoints,
  . n = skip up to n keypoints at the beginning of the data file,
  . -n = skip all keypoints, thus printing only the PMF

counters statistics (if PMF counters are present in the data file).

- SKC (optional), the start keypoint class is an integer between 0 and 15.

  . 0 = start with any class
  . n = start reducing with the first kpt. of class n.

- MER (optional), this parameter allows the user to limit the number of error messages to be printed. The error summary talbe is always active, regardless of the value of MER or the toggle 'E':

  . 0 = has the same effect as toggle 'E', no error messages.
  . n = prints up to n error messages,
  . -n = all error messages are printed.

- MDT (optional), this parameter limits the number of data to be recorded. This reduces the runtime memory usage.

  . 0 = do not save any data, thus the same effect as toggle 'D',
  . n = save up to n data items for each keypoint,
  . -n = save all data.

- INT (optional), this parameter allow the user to save on a file (FKPSM) summary information at specified interval (millisec.). This file will be used by the summary reports generator.

  . n = dump summaries every n milliseconds,
  . -n = do not dump summaries.

- SCT (optional), the section number is an integer (with up to four digits) which may be equal to any of the section numbers defined in the keypoint description file.

  . If no trace is selected, this parameter is ignored.
  . If specified with a value of 9999 all sections will be printed.

9.0 PRIMARY DATA REDUCTION (KPRED)
9.5.3 PROGRAM CONTROL PARAMETERS

9.5.3 PROGRAM CONTROL PARAMETERS

The data reduction program has been provided with several program control toggles. These toggles will deactivate parts of the output if selected.

By default all different outputs are printed; if any one toggle is set, it deselects the corresponding output. The following is a brief description of these toggles:

A,B,C: deselect processing of system, product set, or user keypoints respectively.

D: deselects the data output. This consists of statistics for the data values (if any) gathered by keypoint number.

E: deselects non fatal error messages.

H: deselects all histograms output (thus it implies L, N and X also).

J: deselects task statistics.

K: deselects task keypoint description.

L: deselects (Last) histograms for frequency of calls for a specific keypoint.

N: deselects (Next) histogram for frequency of a specific keypoint.

R: deselects the report information which includes: CPU usage, page fault statistics, peripherals device usage and summary histograms for each histogram type.

S: deselects summary output. This is a summary of every section of the operating system.

T: deselects the trace dump for every keypoint.

X: deselects (entry/exit) histograms for average entry/exit times of selected procedures.

If all toggles are selected, only one page of output will be generated. This consists of a very concise summary including: elapsed time, total number of keypoints, overall frequency, run number, keypoint counts for every section, etc.

9.5.4 KPRED RUN PROCEDURES

An SES procedure is provided to run KPRED. The following describes the proc in details.

Any of the parameters defined in the SES module JOBPARM may be used. These include JOBTL, JOBFL, JOBCN, JOBPR, JOBUN, JOBPW, JOBFMLY, JOBCN, JOBPN, LOCAL, BATCHN, BATCH, DEFER, NODAYF, DAYFILE, AND DF. (See SES Procedure Writer's Guide, Appendix A)

TPF — If the preprocessed data is on tape, TPF is used to specify the VSN(s) of the tape(s) on which the data is found. If the VSN(s) have leading 0s the VSN must be enclosed in single quote marks. If the data is spread across more than one reel, the list of VSNs must be enclosed in parenthesis. Up to 10 reels of tape may be specified. Tapes must be labeled.
Example TPF=('000123',123456) Default none.

DTF — If the preprocessed data is on mass storage, DTF specifies the file name in which the data is stored. If DTF is not found, KPRED runs KPPRE to create one. The file may be local, direct access, or even indirect access although the last alternative is usually impractical.
Example DTF=DATAFL Default DATAFL.

DSF — This parameter specifies the file where the keypoint descriptors are found. This is an output from KPPRE.
Example DSF=DESCFL Default none, required parameter.

DDF — This parameter allows specification of an input file for reducing the processing during data statistics analysis. The file format is:
Class Keypoint D (for delete)
this means the data field will not be recorded for that keypoint.
Example DDF=DELDT Default DATDEF.

RDF — This parameter specifies the lfn of the KPRED output file.
Example RDF=OUT Default OKPRED.

DGF — This parameter specifies the lfn of the KPRED debug file.
Example DGF=DEBUG Default DBGF.

9.0 PRIMARY DATA REDUCTION (KPRED)
9.5.4 KPRED RUN PROCEDURES

SMF — This parameter allows specification of the KPRED summary file.
Example SMF=SUMMARY — Default SMYF.

TGL — This parameter allows setting of various program toggles. The permitted toggles are ABCDEJKLNRSTX.
Example TGL=BCX — Default CLNT.

TIT — This parameter allows specifying a title of up to 40 characters.
Example TIT='PMF RUN FOR FMU' — Default PMF RUN.

RNO — This parameter allows specifying a run number (mex of 6 digits).
Example RNO=1 — Default 99.

SKC — This parameter allows specifying the starting keypoint class for processing.
Example SKC=13 — Default 0.

MER — This parameter specifies the number of errors to print. If the number of errors exceeds this number processing continues but the remaining errors are not printed.
Example MER=500 — Default 1000

MDT — This parameter specifies the maximum number of data values to save per keypoint.
Example MDT=25 — Default 100.

INT — This parameter allows the summary data to be dumped at some specified interval (in milliseconds). However, the follow on program for formatting the output has not been written yet. Therefore, do not use this parameter.
Example INT=100 — Default -1, (ignore interval).

SCT — This parameter allows specification of which section number to print. This parameter works in concordance with the program toggle T (trace). This would be used if only one section of keypoints (ie., 1500-1550) is of interest (if the number is not the start of a section, modulo 50 division will be done such that for instance 1525 would print the keypoints 1500-1549).
Example SCT=1500 — Default 9999, (all sections).

9.0 PRIMARY DATA REDUCTION (KPRED)
9.5.4 KPRED RUN PROCEDURES

SKP — Number of keypoints to skip at the beginning of the raw keypoint file (DTF or TPF).
Example SKP=1000 — Default 0.

MKP — Maximum number of keypoints to process from DTF.
Example MKP=2000 — Default 99999999

ID — This parameter allows specification of the UN of the workspace containing certain files. This allows a single copy of these files to exist, simplifying procedures when they are updated. The files accessed using the ID parameter are DDF, DTF and DSF.
Example ID=KEYPERF — Default user's own UN.

LIB — Name of file where the object code for XKPRED is found. Ordinarily this parameter is obtained from the 'toollib' parameter in the user's PROFILE. This parameter would ordinarily be used for testing a new version of the preprocessor, or for use of a specially modified version of the preprocessor.
Example LIB=MYLIB — Default XULIB.

LIBUN — User number where file specified by LIB (above) resides. Ordinarily this parameter is obtained from the 'toolact' parameter in the user's PROFILE.
Example LIBUN=UN12 — Default KEYPERF.

Some examples of how to invoke the KPPRE procedure follow:

```
SES.KPRED DSF=DSF1 DTF=KPTDFL
```
This call will reduce all the keypoints on file KPTDFL and the output will be found on file DKPRED. The toggles setting are: no user kpt., no trace and entry/exit histogram only.

```
SES.KPRED TPF=ABC001 DSF=DSF1 ..
  RND=131282 MKP=5000
```
This call reduces the keypoints which are on tape ABC001, will have a run number such as the date 13 Dec 1982, and will reduce only the fitst 5000 keypoints.

10.0 KEYPOINT CALL ANALYSIS TOOL

The Call Analysis program is a method of obtaining statistical data about program dynamics from the keypoints collected using the Performance Monitoring Facility (PMF). The data is used to reconstruct the calls made by the various procedures which have had keypoints inserted at their entry and exit points. The analysis will only work if the programs being analyzed are written in a structured manner such as is obtained from languages like CYBIL or PASCAL. It will also work for any other programs which use the discipline of performing in a block type of structure. Branching around entrys or exits with keypoints will substantially reduce the accuracy of the analysis. The Call Analysis program requires that at a minimum all task switching by the operating system must be keypointed.

10.0 KEYPOINT CALL ANALYSIS TOOL
10.1 DESCRIPTION OF FILES USED

10.1 DESCRIPTION OF FILES USED

There are five files associated with a run of CALLAN. Three of these are input files, the other two are output files.

PARMFL is the file containing the parameters. In the current version of CALLAN there is only one parameter which is recognized. This parameter allows the run to be terminated without running until an end of file is encountered on the keypoint file. The format of this parameter is:

RECORDS = nnnnnn

"RECORDS" should be left justified. The "=" must have a space on either side. nnnnnn is a decimal number indicating how many keypoints to process. CALLAN does not begin counting records until after the first entrance to a task is encountered. Thus, the keypoint file will read from 1 to several hundred keypoints more than the amount indicated by this parameter. CALLAN will stop reading the keypoint file when an end of file is encountered or the count is exceeded, whichever comes first. The running time goes up very rapidly as the size of the tree increases. The first keypoints on the file are processed in about 1 msec per record on an S2 whereas after 225000 records have been read, the average time is around 35 msecs per keypoint. The file on which these measurements were taken contained some irregularities which may account for this tremendous increase in running time. The algorithm used should not be that sensitive to file size, and a proper keypoint file may run much faster. This will be determined when more keypoint files become available. The memory requirements also increase with file size. At the beginning the analyzer will run in about 60K. By the end of the file used in the measurements, with 333000 keypoints, the memory requirements increased to 274K. For files with the irregularities in the file used, this memory increase is not surprising. In a proper file the memory requirements should be considerably less.

DESCFL is the file containing the keypoint descriptors. It is the first logical record of the output file of the keypoint pre-processor. Three categories of keypoints are recognized, the operating system keypoints from classes 2 and 3, the product set keypoints from classes 8 and 9, and the user keypoints from classes 12 and 13. Within each of these categories, if a descriptor appears more than once, the descriptor for the last entry keypoint on the file is used. Thus, if there is a descriptor for a keypoint in both entry and exit classes, the 2, 8, or 12 descriptors will be used. The same keypoint number may be used in each of the 3 categories. That is, there may be three

10.0 KEYPOINT CALL ANALYSIS TOOL
10.1 DESCRIPTION OF FILES USED

keypoint 750s, for instance, one in the operating system, one in the product set, and one in the user keypoints.

DATAFL contains the keypoints which are to be analyzed. It is obtained from the second logical record of the output file from the keypoint pre-processor. Only keypoints of classes 2,3,8,9,12 and 13 are used by CALLAN. Any other keypoints are ignored. All keypoints, up to the point of entry to the first task on the keypoint file are also ignored.

REPORT is the file containing the summarized data for the keypoint analysis. The first line indicates the amount of time that was monitored. The statistical data is broken down into reports for each task found on the keypoint file. The first line of the task is a report on the task itself. It is currently all 0s but will eventually be used to hold task related information. Following this is the report on the keypoints in ascending order.

TAPE1 is a file containing supplementary information about the keypoint file. It is primarily a debugging file, but it may also be used to deduce information about the keypoint file. This file is written during the analysis phase, so that if the program aborts, the data on this file can lead to an approximate point where the difficulty lies.

10.2 OUTPUT REPORTS

The call analyzer separates the keypoints such that statistics are kept for each task. Each task is reported by the number found in the 4001 keypoint. This keypoint gives the task id of the task being dispatched on exit from NOS/VE monitor.

The data presented in the report is the summary of procedure characteristics in a dynamic environment. During the analysis phase of the Call Analyzer, each keypoint is constructed in its rightful place in the hierarchy of a tree, with all keypoints having a common parent being linked together in one branch. Parent in the sense used here is the term for the situation where there has been an entry keypoint executed, but the subsequent exit has not yet been executed. If an exit keypoint were to be lost, the keypoint would remain as an ancestor of all the rest of the keypoints in the task. A separate tree is maintained for each task on the keypoint file. The same keypoint may occur at many levels of the tree. It may appear as a child of itself. This would be the case of recursive calls. It also occurs in some cases where the execution is forced by an interrupt or trap mechanism. It could also occur as the result of a lost keypoint. In any case, the date for all occurrences of a given keypoint within a task are gathered into one set of summary statistics. The data regarding each of the keypoint's immediate children are also gathered. In the case of a child, only the statistics which relate ot the particular parent are reported. For instance, if keypoint 750 is called 10 times by 3 different parents, the times and number of times called as a child is reported in the context of its relationship with its immediate parent. But when the statistics of 750 as a parent are reported, all instances of its usage are reported.

It will be instructive to describe the method by which the data is analyzed. To reconstruct the environment in which the procedures existed when the keypoints were generated, a stack is maintained for each task. When an entry keypoint is encountered, the keypoint is entered on the stack. To allow the same keypoint number to be used for operating system, product set, and user keypoints, the keypoint is concatenated with 1, 2 or 3 respectively. When an exit keypoint is encountered it should be the same as the bottom keypoint on the stack for properly structured procedures. At the time of entry on the stack necessary data is stored in a procedure record. When the exit is encountered, the CP time and real time are determined for the procedure is calculated and stored in the procedure record. If an improper exit is eccountered, due to improper keypoint

discipline or because of lost keypoints, the stack is searched backward to find a matching entry somewhere in the stack. If an entry is found, the keypoints between the current position and the match are discarded. This explains why the two counts on the report are sometimes different. "Count" is incremented each time an entry keypoint is entered on the stack. "Valid count" is only incremented when a proper exit is found. The difference between these two is the number of keypoints that were discarded plus the number of keypoints that were outstanding at the end of the run. The keypoints outstanding at the end of the run may be found near the end of TAPE1 under the heading 'summary of stack index'.

As each keypoint is entered, it is inserted into a linked list of procedure records. For each procedure record, the linkage is such that the parent and all children of the procedure may be determined. The procedure records are maintained such that if the procedure is called by more than 1 parent then a new procedure record is allocated. In this way statistics may be gathered for each procedure in its role as a child of different parents. A page fault keypoint has the page fault charged to its parent. The time statistics are accumulated only where keypoints have a valid exit. However, all page faults are recorded, so that a page fault will show up for a procedure call even if it is subsequently discarded.

10.3 OPERATIONAL PROCEDURES

10.3.1 INSTALLATION

To install this program three files are needed: PMFPL, XULIB and PMFPLIB. To compile and save the binaries of the call analysis program type:

```
SES.PMFINST CALLAN COMBINE PRINT
```

10.3.2 SES PROCEDURE

An SES procedure is provided in order to run the call analysis program. Following is the description of all the proc parameters.

Any of the parameters defined in the SES module JOBPARM may be used. These include JOBTL, JOBFL, JOBCN, JOBPR, JOBUN, JOBPW, JOBFMLY, JOBCN, JOBPN, LOCAL, BATCHN, BATCH, DEFER, NODAYF, DAYFILE, AND DF. (See SES Procedure Writer's Guide, Appendix A)

TPF	Tape number or numbers containing preprocessed keypoints. If the VSN has leading zeros the VSN must be enclosed in string quotes ('). If the file is a multi-reel tape, then the list of tapes must be enclosed in parenthesis. Example TPF=('000123',543210)	Default none, one of TPF and DTF must be specified.
DTF	Disk file containing the preprocessed keypoints. Example DTF=DTF1	Default none one of TPF and DTF must be specified.
DSF	File on which the keypoint descriptors reside. Example DSF=DSF1	Default none, required parameter.
REPORT	Permanent file name for the report file. Example REPORT=REP1	Default REPORT.
TAPE1	Permanent file name for the debugging file. Example TAPE1=DEB1	Default TAPE1.

10.0 KEYPOINT CALL ANALYSIS TOOL
10.3.2 SES PROCEDURE

FIRST — Name of permanent file where intermediate sort data is stored.
Example FIRST=INTERM Default FIRST.

ID — If the the preprocessed keypoint data file is in another user's area this parameter must be specified.
Example ID=PM72 Default KEYPERF.

COMMENT — A title for each page on the report may be specified using this parameter. The comment may be up to 50 characters long. The string is enclosed in single quotes (').

PRINT — Dispose a copy of the report to the printer.

NOEXIT — No EXIT card inserted in the job stream.

RECORDS — If you wish to process less than the entire file specify the number of records you wish to process as RECORDS=nnnnnn.

TOTAL — Process entire file without regard to task boundaries.

NEWSORT — This keyword allows you to obtain a report sorted in another order without running the analysis phase of the call analysis (the time consuming part). Use it, coding the same parameters as you used for the original run.

STR — This parameter allows skipping the preprocessed input file until a given class and keypoint is found.
Example STR='12 4004'.

STP — This parameter allows termination of the call analysis run when a given class and keypoint are found on the preprocessed keypoint file.
Example STP='13 4004'.

The following are the allowable parameters for the SORTKEY list. They must be enclosed within parenthesis.

SORTKEY — This is the keyword for the various sort options. The SORTKEY options appear below.

KEYPNT — Report is in increasing values of keypoints and class (All keypoints for a given class come first).
Example SORTKEY=(KEYPNT).

10.0 KEYPOINT CALL ANALYSIS TOOL
10.3.2 SES PROCEDURE

FUNCT — Report is in increasing values of the 2 letter section name. (The 2 letters in the third column of the report)

COUNT — Report is in increasing values of the count of entries to the keypoint

VCOUNT — Report is sorted in increasing value of matched exit count for the keypoints

CPTIME — Report is sorted on increasing values of CP time in the procedure.

AVCP — Report is sorted on increasing values of average CP time (CPTIME/MATCHED COUNT)

RLTIME — Report is sorted in increasing values of Real time

AVRT — Report is sorted in increasing order of average real time in the procedure

CCPT — Report is sorted in order of increasing values of Child CP time

PAGES — Report is sorted in order of increasing number of page faults.

CPAGES — Report is sorted in order of increasing number of Child page faults.

DESCR — Report is sorted with descriptions in alphanumeric order.

CHKYPT — The children of each parent are sorted keypoint order.

CHCPTM — The children of each parent are sorted in order of increasing CP time.

Any of the SORTKEY parameters may be sorted in decreasing order by prefixing the appropriate parameter with the letter D. For instance, to sort such that the highest CP time is printed first, use SORTKEY=DCPTIME.

The order in which the SORTKEY parameters appear in the list determines which key is most significant. eg. SORTKEY=(DCPTIME,DRLTIM,KEYPNT,DCHCPT) will sort first on CPTIME (highest first), then on real time(descending), then on keypoint(ascending) and finally on child CP time(descending). It

10.0 KEYPOINT CALL ANALYSIS TOOL
10.3.2 SES PROCEDURE

is suggested that when sorting in orders other than keypnt, at least one of the minor keys should be KEYPNT, in order to have all the data concerning a keypoint to appear in the same part of the report.

If you want to sort on any other field that appears on the report all that is required is to define a field and a key in this proc. To find the column number where the field appears, attach the file determined by the FIRST parameter in a previous run. This file may be inspected with an editor, keeping in mind that the line length is about 300 ascii characters. Determine the column in which the desired sort parameter lies, and modify the NOTE lines defining fields and keys for SORP accordingly.

If you inspect this proc you will see that the file named by the FIRST parameter is saved as a direct access file. The first program, CALLAN uses the bulk of the processing time, so different sort options may be obtained very cheaply by starting the program with the COMBINE program. This file is used by the NEWSORT option to allow a report sorted in some other order. Any number of reports may be generated by running KPTRUN with the NEWSORT parameter and using the proper copy of the FIRST file.

10.3.3 RUNNING CALL ANALYSIS

The PROC 'KPTRUN' uses the file 'CALLAN', produced by the 'INSTALL' PROC. KPTRUN has 3 parameters. The first parameter is the name of the direct access file containing the output from the keypoint pre-processor. The default name is KPTDFL but any PFN may be used by specifying parameter 1. This file is assumed to contain 2 logical records. The first record is the keypoint descriptors. The second record is the keypoint data file.

The second parameter is the number of keypoints to be processed. The default is 50000. If 'CALLAN' is run without using this PROC, there is an internal limit of 300000 records. CALLAN reads the value from a file named PARMFL. Currently, 'RECORDS =' is the only parameter recognized on PARMFL. Please note the '=' sign must be surrounded by spaces. The main consideration in selecting the number of records to process is the running time. While at this point we have timings only from one run, these show that to process the first 10000 records takes a little less than 25 seconds, the last 10000 records on a 225000 record file take almost 400 seconds. This file had a number of irregularities which may be the cause of this large increase in running time, but still the running time is expected to increase as the

complexity of the call paths increases. The memory requirements also increase dramatically as the number of records increases. To run the 225000 record file which was used for debugging, 274K octal was required. Much of this was probably due to the irregularities in the file. Still, runs of 100000 records or so are probably the most cost-effective.

The third parameter is the name of the permanent file where the supplementary information concerning the run is stored. Most of this information is for debugging, but it is also useful to learn about the general structure of the file. This file may get to be over 1000 prus and is generally inspected at the terminal, as there are many lines of output (17000 or more on a large keypoint file).

11.0 KEYPOINT QUERY FILE ANALYZER (KPQRY)

11.0 KEYPOINT QUERY FILE ANALYZER (KPQRY)

The keypoint query file analyzer (KPQRY) is a tool designed to assist in investigating the contents of large keypoint files. This is useful for debugging and for determining the characteristics of keypoint collection data. It has proved to be useful in discovering unexpected and unusual sequences of keypoints and in tracing abnormalities in the data file. It is designed to handle any type of keypoint file but is currently restricted to hardware collected keypoints and to the output from the keypoint pre-processor.

This program is not a complete or formal implementation of an investigative facility but rather was developed as a debugging tool.

11.0 KEYPOINT QUERY FILE ANALYZER (KPQRY)
11.1 KPQRY GENERAL DESCRIPTION

11.1 KPQRY GENERAL DESCRIPTION

The KPQRY program reads the keypoint file and extracts pertinent information according to the input-query parameters given. The first line of input specifies the type of file being analyzed. Addition parameter lines are given in the form of a small query language. No attempt has been made to make this language SCL-like.

The implementation of the program has been done in such a way as to make the addition of other types of qeuries easy without disturbing the main logic of the program and without extensive changes to other areas of the program. All that is required is to add the new mnemonic to a CASE statement and then to add the condition processor in the body of the statement.

KPQRY reads and analyzes all query directives and then evaluates these directives for each keypoint entry. The query mnemonic determines which field(s) of the keypoint entry are to be processed. The grammar specification is right-recursive allowing the <explist> of the query to be repeated as many times as can fit into an input line.

One or more spaces must separate each token of the query line. Recursion is not allowed for binary queries such as class and keypoint number.

11.0 KEYPOINT QUERY FILE ANALYZER (KPQRY)
11.1.1 QUERY LANGUAGE GRAMMAR DEFINITION

11.1.1 QUERY LANGUAGE GRAMMAR DEFINITION

<query> = <mne> ! <mne> <list>

<list> = <list> <explist> ! <unop> NAME ! <unop> NUM

<explist> = <binop> <list> ! null

<unop> = - ! null

<binop> = + ! *

<mne> = CLSS (class number)
KPN (keypoint number)
DATA (data content)
CLKP (keypoint class and number)
CLDT (keypoint class and data content)
ENEX (entry/exit matching)
BOVF (buffer overflow)
CNT (count occurences)
TINT (time interval between dumps)
TIM (time interval)

NAME = job name, command, etc.

NUM = class, keypoint number, task id, etc.

- = not

+ = or

* = and

11.0 KEYPOINT QUERY FILE ANALYZER (KPQRY)
11.1.2 QUERY EXAMPLES

11.1.2 QUERY EXAMPLES

```
HARD
CLSS -3 * -4 * -5
```

This request will print all occurences of classes other than 3, 4, and 5 from a hardware collection keypoint file.

```
PRE
CLKP 2 * 1106
```

This request will print all examples of the keypoint with class = 2 and keypoint id = 1106 from a pre-processed keypoint file.

```
SIM
BOVF
```

This request would output all occurences of buffer overflow from a hardware collected file.

```
PRE
TIM 1234 * 5678
```

This request would print all keypoints in the time interval from 1234 to 5678 microseconds in a pre-processed keypoint file.

11.2 OUTPUT REPORTS

HARD
TIM 1234 * 5000

OVFL	TIME	CLASS	KPN	DATA(16)	DATA(10)
	1351	3	289	0	0
	1415	2	4004	400	1024
	1584	2	1617	0	0
	1677	3	1617	0	0
	1894	2	1647	0	0
	1900	3	1647	0	0
	1955	2	1617	0	0
	2040	3	1617	0	0
	2093	2	1617	0	0
	2181	3	1617	0	0
	2373	2	1617	0	0
	2457	3	1617	0	0
	2581	2	1617	0	0
	2705	3	1617	0	0
	2812	2	1617	0	0
	2902	3	1617	0	0
	3057	3	4004	80	128
	3079	3	250	0	0
	3260	2	283	0	0
	3424	2	119	10	16
	3498	2	199	10	16
	3523	2	171	10	16
	3810	3	171	0	0
	3821	3	199	0	0
	3833	3	119	0	0
	3925	2	112	10	16
	3963	2	199	10	16
	3988	2	101	10	16
	4034	2	157	65	101
	4066	2	165	10	16
	4113	3	165	0	0
	4123	3	157	0	0
	4132	3	101	0	0
	4141	3	199	0	0
	4153	3	112	0	0
	4379	3	283	0	0
	4509	2	850	0	0
	4655	2	1622	0	0
	4884	2	276	0	0
	4963	3	276	0	0
	4985	2	276	0	0

OCCURRENCES 41

11.3 OPERATIONAL PROCEDURES

11.3.1 INSTALLATION

To install this program three files are needed: PMFPL, XULIB and PMFPLIB. To compile and save the binary of the query program type:

```
SES.PMFINST KPQRY
```

11.3.2 HOW TO RUN KPQRY

The KPQRY program may be run by the following control card call:

```
XKPQRY,input,output,binfile.
```

where: input - contains the query directives
output - contains the output report
binfile - contains the binary keypoint file.

11.3.3 SES PROCEDURE

An SES procedure called KPQRY is available to run the query processor either locally or as a batch job. The parameters for this proc are given below.

Any of the parameters defined in the SES module JOBPARM may be used. These include JOBTL, JOBFL, JOBCN, JOBPR, JOBUN, JOBPW, JOBFMLY, JOBCN, JOBPN, LOCAL, BATCHN, BATCH, DEFER, NODAYF, DAYFILE, AND DF. (See SES Procedure Writer's Guide, Appendix A)

DTF — If the raw data or preprocessed keypoints is on mass storage, DTF specifies the file name in which the data is stored. The file may be local, direct access, or even indirect access although the last alternative is usually impractical.
Example DTF=DATAFL Default DATAFL.

11.0 KEYPOINT QUERY FILE ANALYZER (KPQRY)
11.3.3 SES PROCEDURE

TPF — Tape number or numbers containing the raw data or preprocessed keypoints. If the VSN has leading zeros the VSN must be enclosed in string quotes ('). If the file is a multi-reel tape, then the list of tapes must be enclosed in parenthesis.
Example TPF=('000123',543210) Default none.

IF — This specifies the input parameter file where the KPQRY directives are found.
Example IF=PARM Default none, (interactive input).

OF — This specifies the file on which the output will be written.
Example OF=OUT Default none, (interactive output).

LIB — Name of file where the object code for XKPQRY is found. Ordinarily this parameter is obtained from the 'toolib' parameter in the user's PROFILE. This parameter would ordinarily be used for testing a new version of the preprocessor, or for use of a specially modified version of the preprocessor.
Example LIB=MYLIB Default XULIB.

LIBUN — User number where file specified by LIB (above) resides. Ordinarily this parameter is obtained from the 'toolact' parameter in the user's PROFILE.
Example LIBUN=UN12 Default KEYPERF.

Some examples of how to invoke the KPPRE procedure follow:

SES.KPQRY DTF=KPTS — This will enable the user to run the query with interactive input/output. However, only one query per run is permitted.

12.0 PMF COUNTER (REG. 22) STATISTICS (KPCNT)

12.0 PMF COUNTER (REG. 22) STATISTICS (KPCNT)

The PMF counters report has been removed from the data reduction program KPRED for modularity of processing and to alleviate the data reduction program from this burden.

12.0 PMF COUNTER (REG. 22) STATISTICS (KPCNT)
12.1 KPCNT GENERAL DESCRIPTION

12.1 KPCNT GENERAL DESCRIPTION

The program reads in the Reg. 22 from the data collected file and generate statistics according to machine type and selected counters.

12.0 PMF COUNTER (REG. 22) STATISTICS (KPCNT)
12.2 STATISTICS OUTPUT

12.2 STATISTICS OUTPUT

P - PMF Counter Statistics

Statistics according to the PMF reg. 22 set-up are available. Unless the counter options are explicitly changed in the data collection, they include:

- elapsed time
- number of instructions executed
- cache hit ratio
- CPU Monitor Mode
- MIPS rate
- average time in Monitor Mode,
- number of instructions per return statement.

These statistics are derived exclusively by the content of PMF Register 22.

12.0 PMF COUNTER (REG. 22) STATISTICS (KPCNT)
12.2.1 OUTPUT REPORT

12.2.1 OUTPUT REPORT

S2 SVL PMF RUN, K3099

PERFORMANCE MONITORING FACILITY -- STATUS, CONTROL AND COUNTERS

PMF-OP-IN-PROCESS 0 STOP-DETECTED-KP-CLASS 1 STOP-DETECTED-COUNTER-OVERFLOW 0 KP-TIMER-CARRY-OUT 0

PMF-KP-REQUEST 1 START-ON-KP-CLASS 1 STOP-ON-KP-CLASS 1

START-KP-CLASS 15 STOP-KP-CLASS 15

STOP-ON-COUNTER-OVERFLOW-DESIGNATOR

PROCESSOR-INSTRUCTION-ARGUMENT 004 PROCESSOR-INSTRUCTION-MASK 377

SELECT-A0-AND-B0 0 A0-INPUT-SELECTOR 1 B0-INPUT-SELECTOR 13
SELECT-A1-AND-B1 0 A1-INPUT-SELECTOR 5 B1-INPUT-SELECTOR 12
SELECT-A2-AND-B2 0 A2-INPUT-SELECTOR 7 B2-INPUT-SELECTOR 3
SELECT-A3-AND-B3 1 A3-INPUT-SELECTOR 16 B3-INPUT-SELECTOR 20

COUNTER-A0	47115539	CM READ(OPERND WITH CACHE HIT)	B0	93766828	INSTRUCTION COMPLETE
COUNTER-A1	3285315	CM READ(OPRND WITH CACHE MISS)	B1	677608	SELECTED INSTRUCTION COMPL
COUNTER-A2	1531	PAGE TABLE SEARCH WITHOUT FIND	B2	230756	CM READ FOR SEGMENT TABLE
COUNTER-A3	11123474	CY180 MONITOR MODE	B3	54212534	ONE MICROSECOND

INSTR. EXECUTED = 93766928

CACHE HIT RATIO = 93.481%

CPU MONITOR MODE = 20.518%

MIPS RATE = 1.730

NO.INSTR./RETURN = 138.359

12.0 PMF COUNTER (REG. 22) STATISTICS (KPCNT)
12.3 OPERATIONAL PROCEDURES

12.3 OPERATIONAL PROCEDURES

12.3.1 INSTALLATION

To install this program three files are needed: PMFPL, XULIB and PMFPLIB. To compile and save the binary of the pmf counters program type:

```
SES.PMFINST KPCNT
```

12.3.2 HOW TO RUN KPCNT

To run the KPCNT program:

```
XKPCNT,output,cntfl.
```

where: output - contains the statistics printout
cntfl - contains the last record of the data collection file.

12.3.3 SES PROCEDURE

A SES procedure called KPCNT is available to run this program. It will automatically extract the counter record from the data collection file. The procfile will enable the user to run the program locally or submit a batch job. The following parameters are available.

Any of the parameters defined in the SES module JOBPARM may be used. These include JOBTL, JOBFL, JOBCN, JOBPR, JOBUN, JOBPW, JOBFMLY, JOBCN, JOBPN, LOCAL, BATCHN, BATCH, DEFER, NODAYF, DAYFILE, AND DF. (See SES Procedure Writer's Guide, Appendix A)

DTF — If the raw keypoint data is on mass storage, DTF specifies the file name in which the data is stored. The file may be local, direct access, or even indirect access although the last alternative is usually impractical.
Example DTF=DATAFL — Default DATAFL.

12.0 PMF COUNTER (REG. 22) STATISTICS (KPCNT)
12.3.3 SES PROCEDURE

TPF — Tape number or numbers containing preprocessed keypoints. If the VSN has leading zeros the VSN must be enclosed in string quotes ('). If the file is a multi-reel tape, then the list of tapes must be enclosed in parenthesis.
Example TPF=('000123',543210) — Default none, one of TPF and DTF must be specified.

CNF — This file will contain the counter statistics.
Example CNF=OKPCNT — Default OKPCNT.

TIT — This parameter allows specifying a title of up to 40 characters.
Example TIT='PMF RUN FOR FMU' — Default PMF RUN.

LIB — Name of file where the object code for XKPCNT is found. Ordinarily this parameter is obtained from the 'toollib' parameter in the user's PROFILE. This parameter would ordinarily be used for testing a new version of the preprocessor, or for use of a specially modified version of the preprocessor.
Example LIB=MYLIB — Default XULIB.

LIBUN — User number where file specified by LIB (above) resides. Ordinarily this parameter is obtained from the 'toolact' parameter in the user's PROFILE.
Example LIBUN=UN12 — Default KEYPERF.

Following are some examples of how to invoke KPCNT:

SES.KPCNT DTF=DATAFL	This call will get file DATAFL, generate the statistics and dispose the output to the printer.
SES.KPCNT TPF=ABC001	This call will submit a job to request tape ABC001, generate the statistics and dispose the output to the printer.

13.0 STATISTICS FACILITY

13.0 STATISTICS FACILITY

To be supplied.

14.0 REFERENCES

14.0 REFERENCES

1. CYBER 180 Mainframe Model-Independent General Design Specification, Revision T
DCS document number ARH1700
October 15, 1981

1. CYBER 180 System Interface Standard, Revision H
DCS document number S2196
August 27, 1982

1. NOS/VE Cycle 7 Helpful Hints
NOS/VE Integration and Evaluation
September 23, 1982

A1.0 APPENDIX A -- SVL USAGE

A1.0 APPENDIX A -- SVL USAGE

The master account for PRF utilities at Sunnyvale is KEYPERF. SES procedures described in this documentation may be run by typing SES,KEYPERF.utility name or by putting KEYPERF into the search path in your PROFILE. CCL procs may be obtained by ACQUIRE,PROCFIL/UN=KEYPERF and then running these procs as described.

In order to run PMF data collection on SN101 and SN102, PMF must be invoked from the console under DIS prior to running the job to be monitored. The sequence to do this is as follows:

```
X.DIS.
SUN(KEYPERF)
GET(PMF)
PMF(TY=COLLECT,TP=vsn)
DROP.
```

Data collection is actually initiated by the first class 15 keypoint encountered by the PMF hardware and stopped by the second one. If, for some reason, the collection does not stop, it may be terminated by

CFO,Jsn.STOP

or as a last resort by

DROP,jsn.

B1.0 APPENDIX B -- COLLECTION RUN

This example shows how to start and stop a keypoint collection run:

```
SES.PMF TY=COLLECT                  from 170 side, to set up PMF
                                    Reg.22,
actk,h,s,mm=(2,3,8,9,15) ..         from 180 side, to enable
  jm=(2,3,8,9,15)                   keypoint collection,
emik                                from 180 side, to issue
                                    kpt. class 15 to start
                                    collection,
exet ...                            from 180 side, start benchmark,
:
emik                                from 180 side, when benchmark
                                    terminated, to stop collection,
deak,s                              from 180 side, to deactivate the
                                    keypoint collection.
```

Table of Contents