N ASA-CR-172,172

NASA Contractor Report 172172

DBPSSP: A DATA BASE PROCESSOR SEMANTICS SPECIFICATION PACKAGE

Paul A. Fishwick

Kentron Technical Center
Hampton, Virginia 23666

Contract NAS1-16000
June 1983

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665

SUMMARY

A Semantics Specification Package (DBPSSP) for the Intel Data Base Processor (DBP) is defined. DBPSSP serves as a collection of cross-assembly tools that allow the analyst to assemble request blocks on the host computer for passage to the DBP. The assembly tools discussed in this report may be effectively used in conjunction with a DBP-compatible data communications protocol to form a query processor, precompiler, or file management system for the database processor. The source modules representing the components of DBPSSP are fully commented and included as an appendix to this report.

INTRODUCTION

DBPSSP (Data Base Processor Semantics Specification Package) is the second layer of the HILDA system. HILDA ("High Level Data Abstraction System") is a three layer data base management system supporting the data abstraction freatures of the Intel Data Base Processor (DBP). The purpose of HILDA is the establishment of a flexible method for efficiently communicating with the Intel Data Base Processor. Each layer within HILDA plays a specific role during this communication. These roles may be seen in figures 1 and 2. Figure 1 displays the method by which one may flexibly modify the syntax and semantics for the data base machine. Figure 2 shows the anatomy of a sample query made to the data base processor. The first layer within HILDA is SPP (Service Port Protocol) [1] and serves as the underlying data communications protocol allowing full access to the DBP data base management functionality. It is important to note that even though DBPSSP may be used with SPP, the assembly primitives and procedures within DBPSSP are independent of SPP. That is, another data communications protocol may be effectively used with DBPSSP if necessary. The purpose of this report is to document the design and implementation details associated with DBPSSP. A listing of the source modules is presented in appendix A.

AN OVERVIEW OF DBPSSP

DBPSSP is a collection of assembly tools used on the host computer to construct request modules that are to be sent to the Intel DBP. DBPSSP serves as a

N83-32352#

cross assembler in that Intel DBP "machine code" is assembled on the host computer and then directed to the data base machine for execution. Each request module sent to the DBP is of the form shown in figure 3. Every module contains an arbitrary number of commands. A command is always composed of exactly three primary sections:

1. Opcode Byte - the operation to be performed on the DBP (fetch, store, define database, etc.)
2. Parameters/Data - parameters and data which relate to the operation being performed.
3. Terminator Bytes - two bytes which represent the end of the current operation to be performed by the DBP.

This report outlines the capabilities and suggested usage for the DBPSSP component modules. DBPSSP should be thought of as a collection of procedures (or subroutines) that permit the software developer to easily construct data base requests to the Intel DBP. In this manner the analyst is free to develop a flexible front-end interpreter or compiler to the data base machine. Some highlights of DBPSSP are as follows:

1. Relative and Absolute Offsets - When assembling machine code for the DBP, it is necessary to "place" the code at the proper offset within the request module. In many cases one may build the request module sequentially from start to finish. This sequential mode of assembling is termed "relative" offsetting since the current assembled code is simply "tacked on" to the previously assembled code. One may choose, however, to assemble code at a specific offset within the command block. This random mode of assembling is termed "absolute" offsetting. The mode used by the software developer depends on the front-end driver accessing the assembly tools. A particular parsing method (for a query language, for instance) used for constructing a driver may dictate the use of one offset method over another.

2. Primitive and High-Order Procedures - DBPSSP is composed of a set of general primitive procedures and a set of high-order procedures which are based on the primitives. The high-order procedures are similar in appearance to assembler mnemonics for a given microprocessor: they have short names and contain few operands.

3. Macro Capability - Since DBPSSP is a set of procedures, it is straight-forward and useful to develop new "macros" (or parameterized procedures) which access the fundamental DBPSSP procedures.

THE COMPONENTS OF DBPSSP

DBPSSP is composed of a minimal set of general primitives and a set of higher order procedures. Each set is divided into "Control" modules and "Assembly" modules. The control modules effect the data communications options while the assembly modules are pure assembly directives pertaining specifically to the construction of the command blocks. The control modules are discussed in reference 1. Modules that are dependent on the specific data communications protocol used are denoted "(D)" next to the respective module name. Each module set is depicted below:

PRIMITIVES

INIT_COMM(D) - initialize DBP communications
DBP_SEND(D) - send a request module to the DBP
DBP_RECV(D) - receive a response module from the DBP
TRACE_START(D) - start tracing
TRACE_STOP(D) - stop tracing
PERFORM_START - start performance monitoring
PERFORM_STOP - stop performance monitoring (gather statistics)
DBP_BEGIN - start to assemble a command block
DBP_BITS_BEGIN - start bit masking operations
DBP_BITS - perform logical 'or'ing of bits
DBP_BITS_END - end bit masking operations
DBP_BYTES - assemble an ASCII string
DBP_INTEGER - assemble a 1, 2, or 4 byte integer

HIGH-ORDER PROCEDURES

INIT(D) - initialize DBP communications
SEND(D) - send the built request module to the DBP

RECV(D) - receive a response module from the DBP
TRON(D) - start trace
TROFF(D) - stop trace
PRON - start performance monitoring
PROFF - stop performance monitoring (gather statistics)
START - start encoding a command block
TERMINATE - Add 2 terminator bytes to the command block
BITSB - begin bit masking (relative offset)
BITS - logical 'or' on command block (relative offset)
BITSE - end bit masking (relative offset)
BITSB_A - same as 'BITSB' (absolute offset)
ASC - assemble an ASCII string (relative offset)
ASC_A - same as 'ASC' (absolute offset)
INT1 - assemble a 1-byte integer (relative offset)
INT1_A - same as 'INT1' (absolute offset)
INT2 - assemble as 2-byte integer (relative offset)
INT2_A - same as 'INT2' (absolute offset)
INT4 - assemble a 4-byte integer (relative offset)
INT4_A - same as 'INT4' (absolute offset)

Details on using the above procedures may be found in the source which is included as appendix A.

THE ASSEMBLY PROCESS

Figure 4 displays the assembly process occurring for a "REMARK <HOST> <HELLO>" DBP command. It is assumed in figure 4 that the user has developed a parsing method which will activate the semantic assembly primitives within DBPSSP (START, INT1, ASC, and SEND) when the REMARK command is encountered. A more substantial program is presented in appendix B that performs the conceptual DBP operations listed below (note that the keywords are designated using small letters):

1. Submit keys ADMIN
   Submit the ADMIN key to the session keyring.
2. Define database TESTING
   Define a new database called TESTING.

3. Keep database TESTING
   Make the database TESTING a permanent database.
4. Define file FILE1 DBPSYS
   Define a new file called FILE1 on the system disk (DBPSYS).
5. Define schema INT1 int*4 INT2 int*4 INT3 int*4 FILE1
   Define a schema containing exactly three 4-byte integers (INT1, INT2, and INT3) for the previously defined file FILE1.
6. Keep file FILE1
   Make the file FILE1 a permanent file.
7. List database TESTING
   Show the schema description for files within database TESTING.

For further information on the conceptual command syntax, see the DBP Reference Manual [2]. The program containing the above commands is presented in appendix B using two languages - FORTRAN 77 and Pascal. The FORTRAN program is coded using the primitives, while the Pascal program uses the high-order procedures. In general, the software designer will want to use the high-order procedures since the high-order procedures are more functionally precise with fewer parameters. Note the relative compactness of the Pascal program. The assembled modules and received DBP responses obtained after having sent the completed requests to the DBP are shown at the end of appendix B.

CONCLUDING REMARKS

The procedures within DBPSSP provide the developer with all of the necessary tools to construct an interactive query processor, compiler, or file management system for the Intel DBP. DBPSSP permits the definition of a complete semantics specification associated with any given command or language syntax that the developer may choose. For example, the third layer of HILDA in conjunction with a parser generator package specifies the syntax and semantics for an interpretive language "DBPQL" using DBPSSP.

APPENDIX A - DBPSSP Source

DBPSSP has been implemented using VAX VMS FORTRAN 77. The file "DBPSSP.FOR" contains the high-order procedures which will be used most often.

```
C=======================================================
C
C CONTENTS :
C
C   THIS FILE CONTAINS A SET OF ASSEMBLY TOOLS NECESSARY
C   TO EFFICIENTLY CONSTRUCT REQUEST MODULES FOR THE
C   DBP. EACH PROCEDURE ACTIVATES ONE OR MORE PRIMITIVE
C   ASSEMBLY PROCEDURES.
C
C DATE :
C
C   APRIL 20,1983
C
C=======================================================
C
      SUBROUTINE INIT
C
C INITIALIZE DBP COMMUNICATIONS VIA
C SPP( SERVICE PORT PROTOCOL )
C
      CALL INIT_COMM
      RETURN
      END
      SUBROUTINE START
C
C START ENCODING A REQUEST MODULE
C
      CALL DBP_BEGIN
      RETURN
      END
      SUBROUTINE BITSB
C
C PREPARE FOR INSERTING AN 'OR'ED VALUE WITHIN
C THE REQUEST MODULE
C
      CALL DBP_BITS_BEGIN(-1)
      RETURN
      END
      SUBROUTINE BITSB_A( OFFSET )
C
C *** ABSOLUTE OFFSET ***
C PREPARE FOR INSERTING AN 'OR'ED VALUE WITHIN
C THE REQUEST MODULE
C
      CALL DBP_BITS_BEGIN( OFFSET )
      RETURN
      END
      SUBROUTINE BITS( BYTE_VALUE )
C
C PERFORM AN 'OR' OPERATION OF 'BYTE_VALUE' ON THE
C CURRENT BYTE WITHIN THE REQUEST MODULE.
C
      BYTE BYTE_VALUE
      CALL DBP_BITS( BYTE_VALUE )
      RETURN
      END
      SUBROUTINE BITSE
C
C STOP THE 'OR'ING PROCESS FOR THE CURRENT BYTE BEING
C FORMED WITHIN THE REQUEST MODULE
C
```

```
      CALL DBP_BITS_END
      RETURN
      END
      SUBROUTINE ASC( STRING,LENGTH )
C
C INSERT AN ASCII STRING OF LENGTH 'LENGTH' WITHIN
C THE REQUEST MODULE.
C ALSO PLACE THE 'LENGTH' DIRECTLY IN FRONT OF THE
C ASCII BYTES
C
      CHARACTER*(*) STRING
      INTEGER*4 LENGTH
C
      CALL DBP_INTEGER( -1,LENGTH,1 )
      CALL DBP_BYTES( -1,STRING,LENGTH )
      RETURN
      END
      SUBROUTINE ASCX( STRING,LENGTH )
C
C INSERT AN ASCII STRING OF LENGTH 'LENGTH' WITHIN
C THE REQUEST MODULE.
C DO NOT PLACE THE 'LENGTH' WITHIN THE REQUEST MODULE
C BEING BUILT
C
      CHARACTER*(*) STRING
      INTEGER*4 LENGTH
C
      CALL DBP_BYTES( -1,STRING,LENGTH )
      RETURN
      END
      SUBROUTINE ASC_A( OFFSET,STRING,LENGTH )
C
C *** ABSOLUTE OFFSET ***
C
C INSERT AN ASCII STRING OF LENGTH 'LENGTH' WITHIN
C THE REQUEST MODULE.
C
      CHARACTER*(*) STRING
      INTEGER*4 LENGTH,OFFSET
C
      CALL DBP_INTEGER( OFFSET,LENGTH,1 )
      CALL DBP_BYTES( OFFSET+1,STRING,LENGTH )
      RETURN
      END
      SUBROUTINE ASCX_A( OFFSET,STRING,LENGTH )
C
C *** ABSOLUTE OFFSET ***
C
C INSERT AN ASCII STRING OF LENGTH 'LENGTH' WITHIN
C THE REQUEST MODULE.
C
      CHARACTER*(*) STRING
      INTEGER*4 LENGTH,OFFSET
C
      CALL DBP_BYTES( OFFSET+1,STRING,LENGTH )
      RETURN
      END
      SUBROUTINE INT1( VALUE )
C
C INSERT A ONE-BYTE INTEGER
C
```

```
      INTEGER*4 VALUE
      CALL DBP_INTEGER( -1,VALUE,1 )
      RETURN
      END
      SUBROUTINE INT1_A( OFFSET,VALUE )
C
C *** ABSOLUTE OFFSET ***
C
C INSERT A ONE-BYTE INTEGER
C
      INTEGER*4 OFFSET,VALUE
      CALL DBP_INTEGER( OFFSET,VALUE,1 )
      RETURN
      END
      SUBROUTINE INT2( VALUE )
C
C INSERT A TWO-BYTE INTEGER
C
      INTEGER*4 VALUE
      CALL DBP_INTEGER( -1,VALUE,2 )
      RETURN
      END
      SUBROUTINE INT2_A( OFFSET,VALUE )
C
C *** ABSOLUTE OFFSET ***
C
C INSERT A TWO-BYTE INTEGER
C
      INTEGER*4 OFFSET,VALUE
      CALL DBP_INTEGER( OFFSET,VALUE,2 )
      RETURN
      END
      SUBROUTINE INT4( VALUE )
C
C INSERT A FOUR-BYTE INTEGER
C
      INTEGER*4 VALUE
      CALL DBP_INTEGER( -1,VALUE,4 )
      RETURN
      END
      SUBROUTINE INT4_A( OFFSET,VALUE )
C
C *** ABSOLUTE OFFSET ***
C
C INSERT A FOUR-BYTE INTEGER
C
      INTEGER*4 OFFSET,VALUE
      CALL DBP_INTEGER( OFFSET,VALUE,4 )
      RETURN
      END
      SUBROUTINE TRON
C
C START THE DIAGNOSTIC TRACE UTILITY
C USE UNIT #9
C
      CALL TRACE_START(9)
      RETURN
      END
      SUBROUTINE TROFF
C
C STOP THE TRACE UTILITY
```

```
C
      CALL TRACE_STOP
      RETURN
      END
      SUBROUTINE PRON
C
C START THE PERFORMANCE MONITORING UTILITY
C
      CALL PERFORM_START
      RETURN
      END
      SUBROUTINE PROFF( CLOCK,CPU,BIO,DIO,PAGE )
C
C STOP THE PERFORMANCE MONITORING UTILITY AND
C RETRIEVE THE EXECUTION STATISTICS SINCE THE
C LAST ACTIVATION OF 'PRON'
C
      INTEGER*4 BIO,DIO,PAGE
      CALL PERFORM_STOP( CLOCK,CPU,BIO,DIO,PAGE )
      RETURN
      END
      SUBROUTINE TERMINATE
C
C INSERT THE TERMINATOR BYTES INTO THE REQUEST STREAM
C
      CALL DBP_INTEGER( -1,'FF'X,1 )
      CALL DBP_INTEGER( -1,'00'X,1 )
      RETURN
      END
      SUBROUTINE SEND
C
C SEND THE BUILT REQUEST MODULE TO THE DBP
C
      CALL DBP_SEND
      RETURN
      END
      SUBROUTINE RECV( RESPONSE,NBYTES_RECV,MORE )
C
C RECEIVE THE MESSAGE FROM THE DBP. IF 'MORE' IS
C TRUE THEN WE SHOULD RE-ACTIVATE 'SEND'
C
      LOGICAL MORE
      BYTE RESPONSE(1024)
      INTEGER*4 NBYTES_RECV
C
      CALL DBP_RECV( RESPONSE,NBYTES_RECV,MORE )
      RETURN
      END
      SUBROUTINE PERFON
C
C TURN ON THE PERFORMANCE MONITORING
C
      LOGICAL PERF_DEBUG
      COMMON/ PERFMODE/ PERF_DEBUG
C
      PERF_DEBUG = .TRUE.
      RETURN
      END
      SUBROUTINE PERFOFF
C
C TURN OFF THE PERFORMANCE MONITORING
```

```
C
      LOGICAL PERF_DEBUG
      COMMON/ PERFMODE/ PERF_DEBUG
C
      PERF_DEBUG = .FALSE.
      RETURN
      END
      SUBROUTINE TRACEON
C
C TURN ON THE TRACE TO DISPLAY THE ENCODED SEND
C AND REQUEST MODULES BEING TRANSFERRED
C
      LOGICAL DEBUG
      COMMON/DEBUGMODE/ DEBUG
C
      DEBUG = .TRUE.
      RETURN
      END
      SUBROUTINE TRACEOFF
C
C TURN OFF THE TRACE TO DISPLAY ENCODED SEND
C AND REQUEST MODULES BEING TRANSFERRED
C
      LOGICAL DEBUG
      COMMON/DEBUGMODE/ DEBUG
C
      DEBUG = .FALSE.
      RETURN
      END
      SUBROUTINE DBP_BEGIN
C=======================================================
C
C PURPOSE :
C
C   START THE ENCODING PROCESS NECESSARY TO BUILD
C   A COMMAND BLOCK FOR PASSAGE TO THE DBP
C
C ARGUMENTS :
C
C   NONE
C
C DATE :
C
C   APRIL 2,1983
C
C=======================================================
C
      BYTE BUILT_MODULE( 1024 )
      INTEGER*4 CURRENT_OFFSET
      COMMON/OFFSETCOM/ BUILT_MODULE,CURRENT_OFFSET
C
C RESET THE CURRENT OFFSET COUNTER
C
      CURRENT_OFFSET = 0
      RETURN
      END
      SUBROUTINE DBP_INTEGER( OFFSET,VALUE,LENGTH )
C=======================================================
C
C PURPOSE :
C
C   TO INSERT A 1,2, OR 4 BYTE INTEGER INTO THE COMMAND BLOCK
C   BEING CONSTRUCTED.
C
C ARGUMENTS
C
C   OFFSET       - OFFSET FROM START OF THE COMMAND BLOCK
C                  BEING BUILT. STARTS AT ZERO.
C
C   VALUE        - VALUE TO BE INSERTED INTO THE COMMAND BLOCK
C
C   LENGTH       - NUMBER OF BYTES IN INTEGER 'VALUE'
C                  = 1,2, OR 4
C
C DATE :
C
C   APRIL 2,1983
C
C=======================================================
      BYTE BYTE_ARRAY(4)
      BYTE BUILT_MODULE( 1024 )
      INTEGER*4 OFFSET,VALUE,VALUE2,LENGTH,CURRENT_OFFSET
      INTEGER*4 POSITION
      EQUIVALENCE( VALUE2,BYTE_ARRAY(1) )
      COMMON/ OFFSETCOM/ BUILT_MODULE,CURRENT_OFFSET
C
      VALUE2 = VALUE
C
C UPDATE THE CURRENT POSITION WITHIN
C THE COMMAND BLOCK
C
      IF( OFFSET.EQ.-1 ) THEN
          POSITION = CURRENT_OFFSET
      ELSE
          POSITION = OFFSET
      ENDIF
      DO 100 I = 1,LENGTH
100   BUILT_MODULE( POSITION+I ) = BYTE_ARRAY(I)
C
C UPDATE THE OFFSET COUNTER
C
      CURRENT_OFFSET = POSITION + LENGTH
      RETURN
      END
      SUBROUTINE DBP_BYTES( OFFSET,STRING,LENGTH )
C=======================================================
C
C PURPOSE :
C
C   TO INSERT A CHARACTER STRING OF LENGTH 'LENGTH' INTO
C   THE COMMAND BLOCK BEING CONSTRUCTED
C
C ARGUMENTS
C
C   OFFSET        - OFFSET FROM START OF THE COMMAND BLOCK
C                   BEING BUILT. STARTS AT ZERO.
C
C   STRING        - CHARACTER STRING TO BE INSERTED INTO THE
C                   COMMAND BLOCK
C
C   LENGTH        - NUMBER OF BYTES IN CHARACTER STRING.
C
C DATE :
C
C   APRIL 2,1983
C
C=======================================================
      BYTE BUILT_MODULE( 1024 )
      INTEGER*4 OFFSET,LENGTH,CURRENT_OFFSET
      INTEGER*4 POSITION
      CHARACTER*(*) STRING
      COMMON/ OFFSETCOM/ BUILT_MODULE,CURRENT_OFFSET
C
C UPDATE THE CURRENT POSITION WITHIN
C THE COMMAND BLOCK
C
      IF( OFFSET.EQ.-1 ) THEN
          POSITION = CURRENT_OFFSET
      ELSE
          POSITION = OFFSET
      ENDIF
C
C INSERT THE STRING INTO THE MODULE BEING BUILT
C
      READ( STRING,100 ) ( BUILT_MODULE(I),I=POSITION+1,
     X                     POSITION+LENGTH )
100   FORMAT( <LENGTH>A1 )
C
C UPDATE THE OFFSET COUNTER
C
      CURRENT_OFFSET = POSITION + LENGTH
      RETURN
      END
      SUBROUTINE DBP_BITS( BYTE_VALUE )
C========================================================
C
C PURPOSE :
C
C   TO 'OR' THE GIVEN BYTE VALUE WITH THE BYTE
C   VALUE ALREADY PRESENT
C
C NOTE:
C
C   THE CURRENT_OFFSET COUNTER IS NOT INCREMENTED
C   THIS PERMITS MULTIPLE OR'S. WHEN LOGICAL 'OR'ING
C   IS DONE, USE ROUTINE 'DBP_BITS_END'
C
C
C ARGUMENTS
C
C
C   BYTE_VALUE   - BYTE VALUE TO 'OR'
C
C DATE :
C
C   APRIL 2,1983
C
C========================================================
      BYTE BUILT_MODULE( 1024 )
      INTEGER*4 CURRENT_OFFSET
      BYTE BYTE_VALUE
      COMMON/ OFFSETCOM/ BUILT_MODULE,CURRENT_OFFSET
C
C OR THE GIVEN BYTE WITH THE BYTE ALREADY THERE
C
      BUILT_MODULE(CURRENT_OFFSET+1 ) = BUILT_MODULE(CURRENT_OFFSET+1 ).OR.
     X         BYTE_VALUE
      RETURN
      END
      SUBROUTINE DBP_BITS_BEGIN( OFFSET )
C=====================================================
C
C PURPOSE :
C
C   TO INITIALIZE THE GIVEN BYTE WITHIN 'BUILT_MODULE'.
C   FUTURE 'OR'ING IS EXPECTED ON THE CURRENT BYTE,
C   SO THE CURRENT OFFSET COUNTER IS NOT
C   INCREMENTED.
C
C ARGUMENTS
C
C   OFFSET       - OFFSET FROM START OF THE COMMAND BLOCK
C                  BEING BUILT. STARTS AT ZERO.
C
C DATE :
C
C   APRIL 2,1983
C
C=====================================================
      BYTE BUILT_MODULE( 1024 )
      INTEGER*4 OFFSET,CURRENT_OFFSET
      INTEGER*4 POSITION
      COMMON/ OFFSETCOM/ BUILT_MODULE,CURRENT_OFFSET
C
C UPDATE THE CURRENT POSITION WITHIN
C THE COMMAND BLOCK
C
      IF( OFFSET.EQ.-1 ) THEN
          POSITION = CURRENT_OFFSET
      ELSE
          POSITION = OFFSET
      ENDIF
C
      BUILT_MODULE( POSITION+1 ) = 0
      RETURN
      END
      SUBROUTINE DBP_BITS_END
C=======================================================
C
C PURPOSE :
C
C   SIGNIFIES THAT THE 'OR'ING PROCESS ON THE CURRENT
C   MODULE BYTE IS DONE. TIME TO CONTINUE CONSTRUCTION
C   OF THE REST OF THE MODULE. INCREMENT THE CURRENT
C   OFFSET COUNTER.
C
C ARGUMENTS
C
C   NONE
C
C DATE :
C
C   APRIL 2,1983
C
C=======================================================
      BYTE BUILT_MODULE( 1024 )
      INTEGER*4 OFFSET,CURRENT_OFFSET
      COMMON/ OFFSETCOM/ BUILT_MODULE,CURRENT_OFFSET
C
      CURRENT_OFFSET = CURRENT_OFFSET + 1
      RETURN
      END
      SUBROUTINE DBP_SEND
C=======================================================
C
C PURPOSE :
C
C   SEND THE COMMAND BLOCK TO THE DBP.
C
C NOTE :
C
C   THIS ROUTINE CALLS THE 'SPP' PACKAGE
C   ( SERVICE PORT PROTOCOL )
C   TO PERMIT HOST-DBP COMMUNICATION
C
C ARGUMENTS
C
C   NONE
C
C DATE :
C
C   APRIL 2,1983
C
C=======================================================
      BYTE BUILT_MODULE( 1024 ),PRBYTES( 1024 )
      INTEGER*4 CURRENT_OFFSET,TOTAL_BYTES,UNIT
      LOGICAL*4 MORE_TO_COME,DEBUG,PERF_DEBUG
      COMMON/ OFFSETCOM/ BUILT_MODULE,CURRENT_OFFSET
      COMMON/ DEBUGMODE/ DEBUG
      COMMON/ PERFMODE/ PERF_DEBUG
      DATA PERF_DEBUG/.FALSE./,DEBUG/.FALSE./,UNIT/6/
C
C 1. SEND THE BUILT COMMAND BLOCK TO THE DBP
C 2. LOOP TO RECEIVE ALL DBP RESPONSES
C
C
C OUTPUT THE REQUEST BLOCK IF IN DEBUG MODE
C
      IF( DEBUG ) THEN
C
C SET UP ASCII BYTES
C
      DO 50 I = 1,CURRENT_OFFSET+1
      IF( (BUILT_MODULE(I).LT.'20'X).OR.
     X    (BUILT_MODULE(I).GT.'7E'X)) THEN
         PRBYTES(I) = '2E'X
      ELSE
         PRBYTES(I) = BUILT_MODULE(I)
      ENDIF
50    CONTINUE
      WRITE( UNIT,100 ) CURRENT_OFFSET
100   FORMAT(' == DBP REQUEST =='/' # of bytes is ',I5,
     X       /,' Byte Stream :'/ )
      MULTIPLE16 = (CURRENT_OFFSET/16)*16
      LEFTOVER   = CURRENT_OFFSET - MULTIPLE16
      IF( MULTIPLE16.GT.0 ) THEN
        DO 200 I = 1,MULTIPLE16,16
        WRITE( UNIT,150 ) (BUILT_MODULE(I1),I1=I,I+15),
     X   (PRBYTES(I2),I2=I,I+15)
150     FORMAT(16(1X,Z2.2),2X,16A1)
200     CONTINUE
      ENDIF
      IF( LEFTOVER.GT.0 ) THEN
```

```
          WRITE( UNIT,210 ) (BUILT_MODULE(I1),I1=MULTIPLE16+1,
     X    MULTIPLE16+LEFTOVER),(PRBYTES(I2),I2=MULTIPLE16+1,
     X    MULTIPLE16+LEFTOVER)
210       FORMAT(<LEFTOVER>(1X,Z2.2),<16-LEFTOVER>(3X),2X,
     X           <LEFTOVER>A1 )
        ENDIF
        WRITE( UNIT,250 )
250     FORMAT(/)
        ENDIF
        IF( PERF_DEBUG ) CALL PRON
        CALL SEND_REQUEST( BUILT_MODULE,CURRENT_OFFSET,1,1,1 )
C
C FINISHED WITH THIS COMMAND
C
9999    RETURN
        END
```

```
      SUBROUTINE DBP_RECV( RESPONSE,TOTAL_BYTES,MORE )
C=====================================================
C
C PURPOSE :
C
C   RECEIVE THE RESPONSE FROM THE DBP
C
C NOTE :
C
C   THIS ROUTINE CALLS THE 'SPP' PACKAGE
C   ( SERVICE PORT PROTOCOL )
C   TO PERMIT HOST-DBP COMMUNICATION
C
C ARGUMENTS
C
C   RESPONSE    - THE BYTE RESPONSE FROM THE DBP
C   MORE        - = .TRUE. IF THERE IS MORE TO COME
C                    FROM THE DBP
C
C               - = .FALSE. IF ALL THE DATA FROM THE
C                    DBP HAS BEEN RECEIVED
C
C
C DATE :
C
C   APRIL 20,1983
C
C=====================================================
      BYTE RESPONSE( 1024 ),PRBYTES( 1024 )
      INTEGER*4 DIO,BIO,PAGE
      INTEGER*4 CURRENT_OFFSET,TOTAL_BYTES,UNIT
      LOGICAL*4 MORE,DEBUG,PERF_DEBUG
      COMMON/DEBUGMODE/ DEBUG
      COMMON/PERFMODE/ PERF_DEBUG
      DATA PERF_DEBUG/.FALSE./,DEBUG/.FALSE./,UNIT/6/
C
      CALL RECV_RESPONSE( RESPONSE,TOTAL_BYTES,1,MORE )
      IF( PERF_DEBUG ) THEN
        CALL PROFF( CLOCK,CPU,BIO,DIO,PAGE )
        WRITE(6,10) CLOCK,CPU,BIO,DIO,PAGE
10     FORMAT(/' Clock ',F12.5/,' CPU ',F12.5/,
     X   ' Buffered I/O count ',I6/,' Direct I/O count ',I6,/,
     X   ' Page Fault count ',I6 ,//)
      ENDIF
      IF( TOTAL_BYTES.GT.1024 ) THEN
        WRITE( 6,25 ) TOTAL_BYTES
25      FORMAT(' Error, DBP says that it has ',
     X  I7,' bytes to send back.',
     X  /' This exceeds the limit of 1024.' )
      GO TO 9999
      ENDIF
C
C OUTPUT THE RESPONSE IF IN DEBUG MODE
C
      IF( DEBUG ) THEN
C
C SET UP ASCII BYTES
C
      DO 500 I = 1,TOTAL_BYTES
      IF( (RESPONSE(I).LT.'20'X).OR.
     X    (RESPONSE(I).GT.'7E'X)) THEN
```

```
             PRBYTES(I) = '2E'X
          ELSE
             PRBYTES(I) = RESPONSE(I)
          ENDIF
500       CONTINUE
          WRITE( UNIT,600 ) TOTAL_BYTES
600       FORMAT(' == DBP RESPONSE =='/' # of bytes is ',I5,
     X         /,' Byte Stream :'/ )
          MULTIPLE16 = (TOTAL_BYTES/16)*16
          LEFTOVER   = TOTAL_BYTES - MULTIPLE16
          IF( MULTIPLE16.GT.0 ) THEN
            DO 700 I = 1,TOTAL_BYTES,16
            WRITE( UNIT,650 ) (RESPONSE(I1),I1=I,I+15),
     X       (PRBYTES(I2),I2=I,I+15)
650         FORMAT(16(1X,Z2.2),2X,16A1)
700         CONTINUE
          ENDIF
          IF( LEFTOVER.GT.0 ) THEN
            WRITE( UNIT,675 ) (RESPONSE(I1),I1=MULTIPLE16+1,
     X      MULTIPLE16+LEFTOVER),(PRBYTES(I2),I2=MULTIPLE16+1,
     X      MULTIPLE16+LEFTOVER)
675         FORMAT(<LEFTOVER>(1X,Z2.2),<16-LEFTOVER>(3X),2X,
     X             <LEFTOVER>A1 )
          ENDIF
          WRITE( UNIT,750 )
750       FORMAT(/)
          ENDIF
C
C FINISHED WITH THIS COMMAND
C
9999      RETURN
          END
```

APPENDIX B - DBPSSP Examples

FORTRAN and Pascal program examples are given to aid the reader in evaluating the utility of DBPSSP. A brief trace of the requests and responses is also included.

```
      PROGRAM TESTFOR
C
C A FORTRAN EXAMPLE USING THE DBPSSP PRIMITIVE
C ROUTINES
C
      BYTE RESPONSE( 1024 )
      INTEGER*4 BYTES_RECV
      LOGICAL MORE
C
      CALL TRACE_START( 9 )
      CALL TRACEON
      CALL INIT_COMM
C
C SUBMIT KEYS 'ADMIN'
C
      CALL DBP_BEGIN
      CALL DBP_INTEGER( -1,'07'X,1 )
      CALL DBP_INTEGER( -1,5,1 )
      CALL DBP_BYTES( -1,'ADMIN',5 )
      CALL DBP_INTEGER( -1,'FF'X,1 )
      CALL DBP_INTEGER( -1,'00'X,1 )
      CALL DBP_SEND
      CALL DBP_RECV( RESPONSE,BYTES_RECV,MORE )
C
C DEFINE DATABASE CALLED 'TESTING'
C
      CALL DBP_BEGIN
      CALL DBP_INTEGER( -1,'60'X,1 )
      CALL DBP_INTEGER( -1,7,1 )
      CALL DBP_BYTES( -1,'TESTING',7 )
      CALL DBP_INTEGER( -1,'FF'X,1 )
      CALL DBP_INTEGER( -1,'00'X,1 )
      CALL DBP_SEND
      CALL DBP_RECV( RESPONSE,BYTES_RECV,MORE )
C
C KEEP DATABASE 'TESTING'
C
      CALL DBP_BEGIN
      CALL DBP_INTEGER( -1,'64'X,1 )
      CALL DBP_INTEGER( -1,7,1 )
      CALL DBP_BYTES( -1,'TESTING',7 )
      CALL DBP_INTEGER( -1,7,1 )
      CALL DBP_BYTES( -1,'TESTING',7 )
      CALL DBP_INTEGER( -1,'FF'X,1 )
      CALL DBP_INTEGER( -1,'00'X,1 )
      CALL DBP_SEND
      CALL DBP_RECV( RESPONSE,BYTES_RECV,MORE )
C
C DEFINE FILE CALLED 'FILE1'
C
      CALL DBP_BEGIN
      CALL DBP_INTEGER( -1,'40'X,1 )
      CALL DBP_INTEGER( -1,5,1 )
      CALL DBP_BYTES( -1,'FILE1',5 )
      CALL DBP_INTEGER( -1,1,1 )
      CALL DBP_BITS_BEGIN( -1 )
      CALL DBP_BITS( '1000'X )
      CALL DBP_BITS_END
      CALL DBP_INTEGER( -1,6,1 )
      CALL DBP_BYTES( -1,'DBPSYS',6 )
      CALL DBP_INTEGER( -1,2,1 )
      CALL DBP_INTEGER( -1,10,2 )
      CALL DBP_INTEGER( -1,2,1 )
      CALL DBP_INTEGER( -1,0,2 )
      CALL DBP_INTEGER( -1,'FF'X,1 )
      CALL DBP_INTEGER( -1,'00'X,1 )
      CALL DBP_SEND
      CALL DBP_RECV( RESPONSE,BYTES_RECV,MORE )
C
C DEFINE SCHEMA ON PERMANENT FILE 'FILE1'
C
      CALL DBP_BEGIN
      CALL DBP_INTEGER( -1,'49'X,1 )
      CALL DBP_INTEGER( -1,5,1 )
      CALL DBP_BYTES( -1,'FILE1',5 )
      CALL DBP_INTEGER( -1,1,1 )
      CALL DBP_BITS_BEGIN( -1 )
      CALL DBP_BITS( '0000'X )
      CALL DBP_BITS( '0000'X )
      CALL DBP_BITS_END
C
C SCHEMA SPECIFICATION - SET UP AS
C
C INT1 INTEGER*4
C INT2 INTEGER*4
C INT3 INTEGER*4
C
      CALL DBP_INTEGER( -1,0,1 )
      CALL DBP_INTEGER( -1,2,1 )
      CALL DBP_INTEGER( -1,20,2 )
      CALL DBP_INTEGER( -1,2,1 )
      CALL DBP_INTEGER( -1,20,2 )
C
      CALL DBP_INTEGER( -1,4,1 )
      CALL DBP_BYTES( -1,'INT1',4 )
      CALL DBP_INTEGER( -1,1,1 )
      CALL DBP_BITS_BEGIN( -1 )
      CALL DBP_BITS( '0001'X )
      CALL DBP_BITS_END
      CALL DBP_INTEGER( -1,1,1 )
      CALL DBP_INTEGER( -1,4,1 )
C
      CALL DBP_INTEGER( -1,4,1 )
      CALL DBP_BYTES( -1,'INT2',4 )
      CALL DBP_INTEGER( -1,1,1 )
      CALL DBP_BITS_BEGIN( -1 )
      CALL DBP_BITS( '0001'X )
      CALL DBP_BITS_END
      CALL DBP_INTEGER( -1,1,1 )
      CALL DBP_INTEGER( -1,4,1 )
C
      CALL DBP_INTEGER( -1,4,1 )
      CALL DBP_BYTES( -1,'INT3',4 )
      CALL DBP_INTEGER( -1,1,1 )
      CALL DBP_BITS_BEGIN( -1 )
      CALL DBP_BITS( '0001'X )
      CALL DBP_BITS_END
      CALL DBP_INTEGER( -1,1,1 )
      CALL DBP_INTEGER( -1,4,1 )
C DONE
      CALL DBP_INTEGER( -1,'FF'X,1 )
      CALL DBP_INTEGER( -1,'00'X,1 )
      CALL DBP_SEND
      CALL DBP_RECV( RESPONSE,BYTES_RECV,MORE )
C
C KEEP FILE
C
      CALL DBP_BEGIN
      CALL DBP_INTEGER( -1,'41'X,1 )
      CALL DBP_INTEGER( -1,5,1 )
      CALL DBP_BYTES( -1,'FILE1',5 )
      CALL DBP_INTEGER( -1,5,1 )
      CALL DBP_BYTES( -1,'FILE1',5 )
      CALL DBP_INTEGER( -1,7,1 )
      CALL DBP_BYTES( -1,'TESTING',7 )
      CALL DBP_INTEGER( -1,'FF'X,1 )
      CALL DBP_INTEGER( -1,'00'X,1 )
      CALL DBP_SEND
      CALL DBP_RECV( RESPONSE,BYTES_RECV,MORE )
C
C LIST DATABASE 'TESTING'
C
      CALL DBP_BEGIN
      CALL DBP_INTEGER( -1,'90'X,1 )
      CALL DBP_INTEGER( -1,7,1 )
      CALL DBP_BYTES( -1,'TESTING',7 )
      CALL DBP_INTEGER( -1,1,1 )
      CALL DBP_INTEGER( -1,'F0'X,1 )
      CALL DBP_INTEGER( -1,'FF'X,1 )
      CALL DBP_INTEGER( -1,'00'X,1 )
      CALL DBP_SEND
      CALL DBP_RECV( RESPONSE,BYTES_RECV,MORE )
C
      CALL TRACE_STOP
      CALL TRACEOFF
      CALL EXIT
      END
```

```
PROGRAM TESTPAS( INPUT,OUTPUT );

(* TEST OF DBPSSP *)

TYPE LIMIT = ARRAY[ 1..1024 ] OF CHAR;
VAR RESPONSE: LIMIT;
    TOTAL_BYTES: INTEGER;
    MORE: BOOLEAN;
    I : INTEGER;



(* DBPSSP SUPPORT PROCEDURES - EXTERNAL *)

PROCEDURE INIT; FORTRAN;
PROCEDURE START; FORTRAN;
PROCEDURE TRON; FORTRAN;
PROCEDURE TROFF; FORTRAN;
PROCEDURE BITSB; FORTRAN;
PROCEDURE BITS( BYTEVALUE:INTEGER ); FORTRAN;
PROCEDURE BITSE; FORTRAN;
PROCEDURE ASC( %STDESCR STRING:PACKED ARRAY[INTEGER]
               OF CHAR; LENGTH:INTEGER ); FORTRAN ;
PROCEDURE INT1( INTEGER_VALUE:INTEGER ); FORTRAN;
PROCEDURE INT2( INTEGER_VALUE:INTEGER ); FORTRAN;
PROCEDURE INT4( INTEGER_VALUE:INTEGER ); FORTRAN;
PROCEDURE TERMINATE; FORTRAN;
PROCEDURE SEND; FORTRAN;
PROCEDURE RECV( VAR RESPONSE:LIMIT;
                VAR TOTAL_BYTES:INTEGER;
                VAR MORE:BOOLEAN ); FORTRAN;
PROCEDURE TRACEON; FORTRAN;
PROCEDURE TRACEOFF; FORTRAN;



BEGIN
   TRACEON;
   TRON;
   INIT;

(* SUBMIT KEYS 'ADMIN' *)

   START;
   INT1(7);
   ASC('ADMIN',5);
   TERMINATE;
   SEND;
   RECV( RESPONSE,TOTAL_BYTES,MORE );

(* DEFINE DATABASE CALLED 'TESTING' *)

   START;
   INT1(96);
   ASC('TESTING',7);
   TERMINATE;
   SEND;
   RECV( RESPONSE,TOTAL_BYTES,MORE );

(* KEEP DATABASE 'TESTING' *)
```

```
   START;
   INT1(100);
   ASC('TESTING',7);
   ASC('TESTING',7);
   TERMINATE;
   SEND;
   RECV( RESPONSE,TOTAL_BYTES,MORE );

(* DEFINE FILE CALLED 'FILE1' *)

   START;
   INT1(64);
   ASC('FILE1',5);
   INT1(1);
   BITSB; BITS(8); BITSE;
   ASC('DBPSYS',6);
   INT1(2);
   INT2(10);
   INT1(2);
   INT2(0);
   TERMINATE;
   SEND;
   RECV( RESPONSE,TOTAL_BYTES,MORE );

(* DEFINE SCHEMA ON PERMANENT FILE 'FILE' *)

   START;
   INT1(73);
   ASC('FILE1',5);
   INT1(1);
   BITSB; BITS(0); BITSE;

(* SCHEMA SPECIFICATION - SET UP AS

   INT1 INTEGER*4
   INT2 INTEGER*4
   INT3 INTEGER*4

*)

   INT1(0);
   INT1(2);
   INT2(20);
   INT1(2);
   INT2(20);

   ASC('INT1',4);
   INT1(1);
   BITSB; BITS(1); BITSE;
   INT1(1);
   INT1(4);

   ASC('INT2',4);
   INT1(1);
   BITSB; BITS(1); BITSE;
   INT1(1);
   INT1(4);

   ASC('INT3',4);
   INT1(1);
   BITSB; BITS(1); BITSE;
   INT1(1);
   INT1(4);

   TERMINATE;
   SEND;
   RECV( RESPONSE,TOTAL_BYTES,MORE );

(* KEEP FILE *)

   START;
   INT1(65);
   ASC('FILE1',5);
   ASC('FILE1',5);
   ASC('TESTING',7);
   TERMINATE;
   SEND;
   RECV( RESPONSE,TOTAL_BYTES,MORE );

(* LIST DATABASE 'TESTING' *)

   START;
   INT1(144);
   ASC('TESTING',7);
   INT1(1);
   INT1(240);
   TERMINATE;
   SEND;
   RECV( RESPONSE,TOTAL_BYTES,MORE );

   TROFF
END.
```

```
$ @TESTFOR
  _TTB0: allocated
== DBP REQUEST ==
# of bytes is     9
Byte Stream :

07 05 41 44 4D 49 4E FF 00                            ..ADMIN..

== DBP RESPONSE ==
# of bytes is     0
Byte Stream :



== DBP REQUEST ==
# of bytes is    11
Byte Stream :

60 07 54 45 53 54 49 4E 47 FF 00                      `.TESTING..

== DBP RESPONSE ==
# of bytes is     0
Byte Stream :



== DBP REQUEST ==
# of bytes is    19
Byte Stream :

64 07 54 45 53 54 49 4E 47 07 54 45 53 54 49 4E  d.TESTING.TESTIN
47 FF 00                                         G..

== DBP RESPONSE ==
# of bytes is     0
Byte Stream :



== DBP REQUEST ==
# of bytes is    24
Byte Stream :

40 05 46 49 4C 45 31 01 00 06 44 42 50 53 59 53  @.FILE1...DBPSYS
02 0A 00 02 00 00 FF 00                          ........

== DBP RESPONSE ==
# of bytes is     0
Byte Stream :



== DBP REQUEST ==
# of bytes is    45
Byte Stream :

49 05 46 49 4C 45 31 01 00 00 02 14 00 02 14 00  I.FILE1.........
04 49 4E 54 31 01 01 01 04 04 49 4E 54 32 01 01  .INT1.....INT2..
01 04 04 49 4E 54 33 01 01 01 04 FF 00           ...INT3......
```

```
== DBP RESPONSE ==
# of bytes is     0
Byte Stream :



== DBP REQUEST ==
# of bytes is    23
Byte Stream :

41 05 46 49 4C 45 31 05 46 49 4C 45 31 07 54 45  A.FILE1.FILE1.TE
53 54 49 4E 47 FF 00                             STING..


== DBP RESPONSE ==
# of bytes is     0
Byte Stream :



== DBP REQUEST ==
# of bytes is    13
Byte Stream :

90 07 54 45 53 54 49 4E 47 01 F0 FF 00           ..TESTING....


== DBP RESPONSE ==
# of bytes is    55
Byte Stream :

F8 02 90 F0 01 00 01 01 06 00 03 00 00 00 00 07  ................
54 45 53 54 49 4E 47 01 03 06 00 03 03 00 05 00  TESTING.........
05 46 49 4C 45 31 01 00 06 00 03 03 00 05 00 05  .FILE1..........
46 49 4C 45 31 FF 00 B7 F5 2E 00 B7 F5 2E 00 00  FILE1..
46 49 4C 45 31 FF 00                             FILE1..


$
```

REFERENCES

1. Fishwick, P. A.: SPP: A Data Base Processor Data Communications Protocol. NASA CR-172144, May 1983.

2. DBP DBMS Reference Manual. Intel Corporation, Austin, TX, Revision 001, Order No. 222100-001, August 1982.

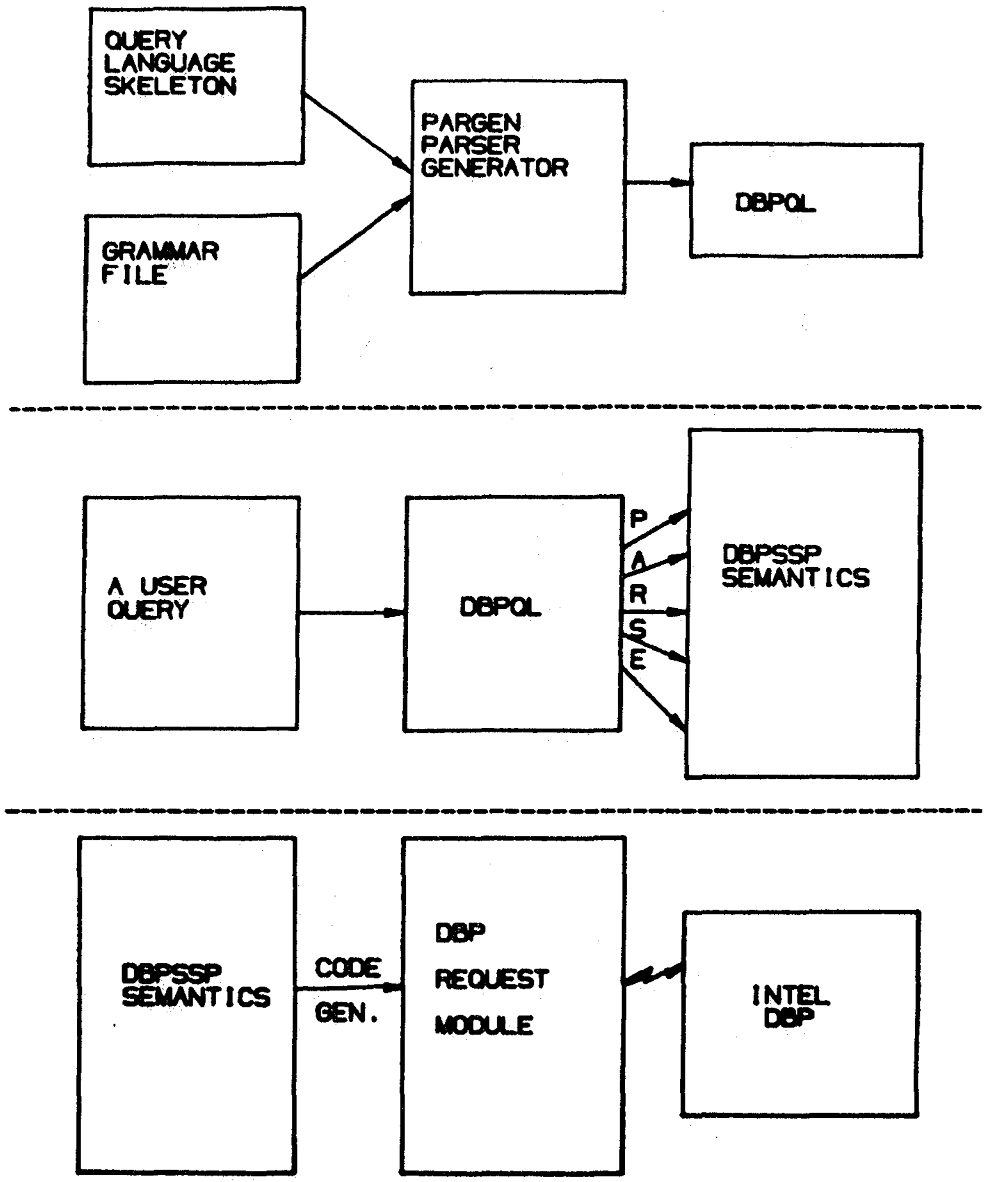

Figure 1 - HILDA: A general flow chart

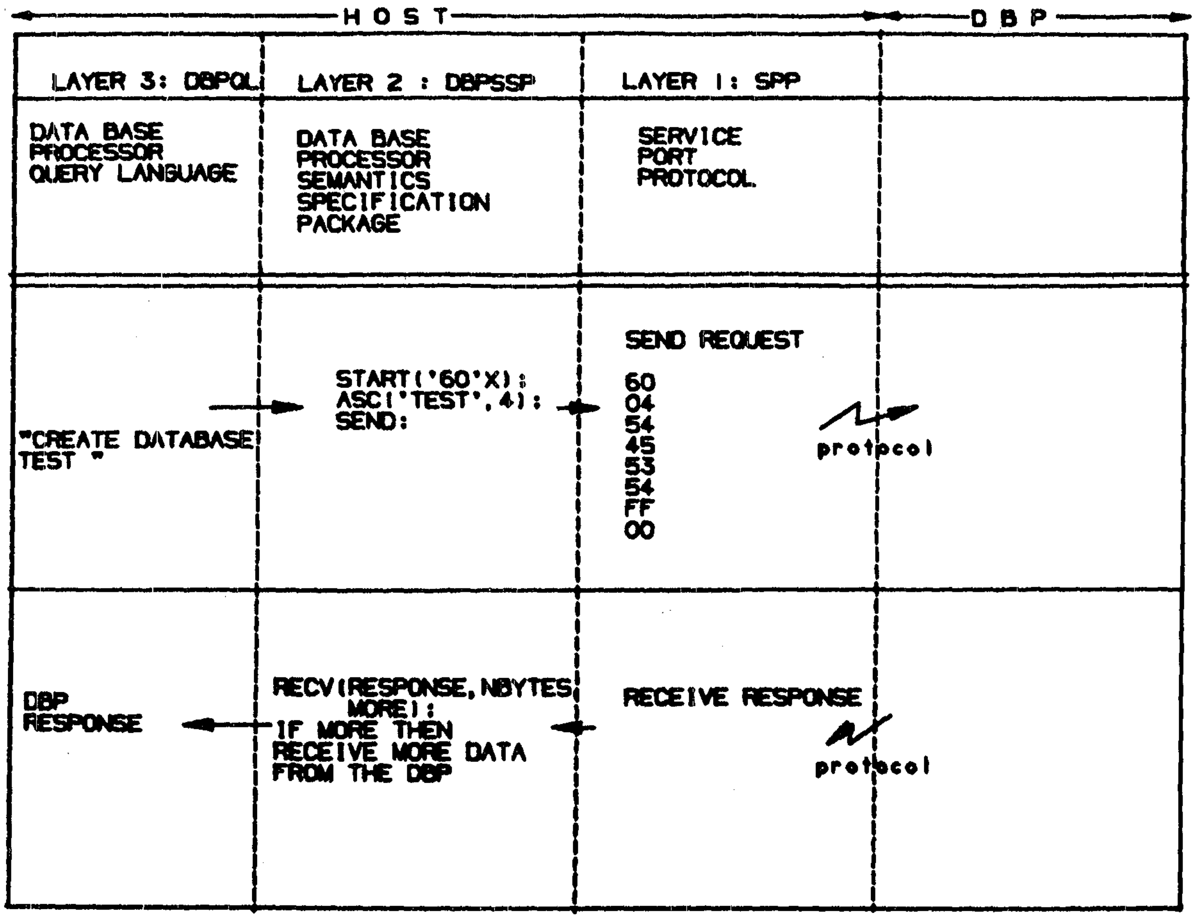

Figure 2 - HILDA: A sample query

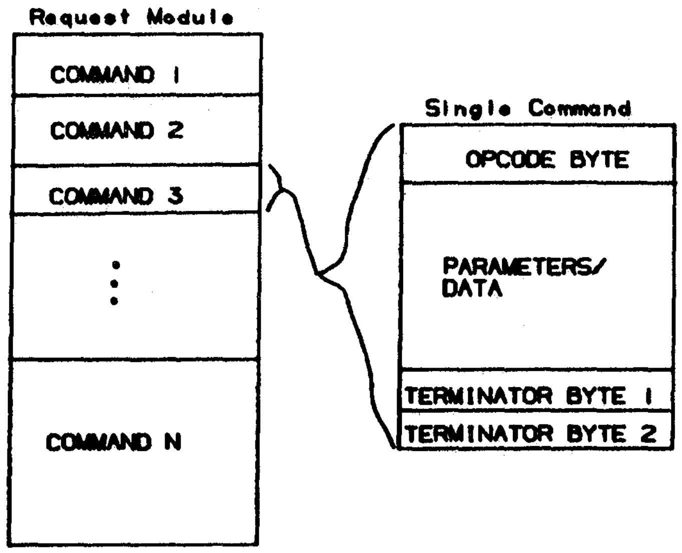

Figure 3 - Request Module Form

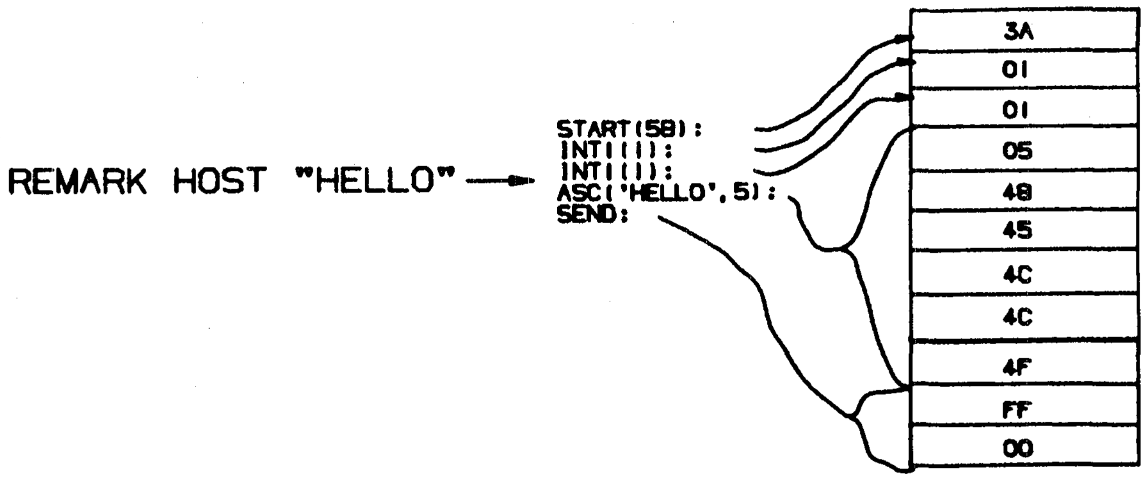

Figure 4 - A sample assembly for "REMARK"

1. Report No. NASA CR-172172	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle DBPSSP: A DATA BASE PROCESSOR SEMANTICS SPECIFICATION PACKAGE		5. Report Date June 1983
		6. Performing Organization Code
7. Author(s) Paul A. Fishwick		8. Performing Organization Report No.
9. Performing Organization Name and Address Kentron Technical Center Hampton, VA 23666		10. Work Unit No.
		11. Contract or Grant No. NAS1-16000
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, DC 20546		13. Type of Report and Period Covered Contractor Report
		14. Sponsoring Agency Code

15. Supplementary Notes

Langley Technical Monitor: Timothy R. Rau
Final Report

16. Abstract

A Semantics Specification Package (DBPSSP) for the Intel Data Base Processor (DBP) is defined. DBPSSP serves as a collection of cross-assembly tools that allow the analyst to assemble request blocks on the host computer for passage to the DBP. The assembly tools discussed in this report may be effectively used in conjunction with a DBP-compatible data communications protocol to form a query processor, precompiler, or file management system for the database processor. The source modules representing the components of DBPSSP are fully commented and included as an appendix to this report.

17. Key Words (Suggested by Author(s)) Semantics specification; Data base machine; Data base management	18. Distribution Statement Unclassified - Unlimited; Subject Category 61

19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages 36	22. Price A03

N-305

End of Document