PROCEEDINGS OF HP 3000

USER'S GROUP CONFERENCE

January 24-25, 1974

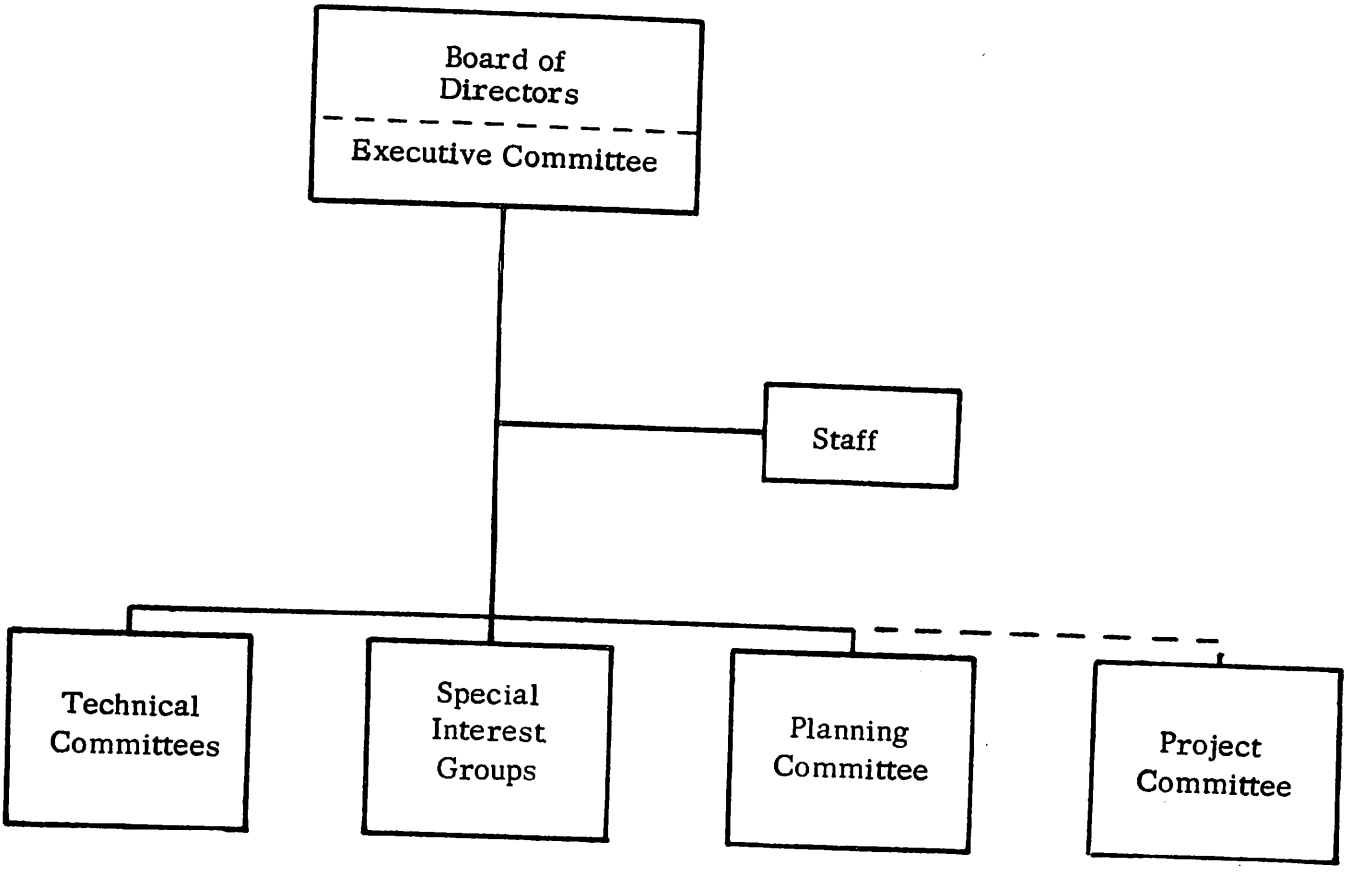
TABLE OF CONTENTS

INTRODUCTION

The following document represents information gathered at the HP 3000 User's Conference dated January 24 and 25, 1974 held at Rickey's Hyatt House, Palo Alto.

USER'S GROUP STRUCTURE

STRUCTURE OF THE USER'S GROUP



The specific structure, relationship, responsibilities, and definitions are described in the By-Laws.

The [Board of Directors] serves as the policy making body and financial overseer. The [Executive Committee] performs the management of the User's Group and represents the centralized organization. Its [Staff] performs the work necessary to maintain operations. Initially it will perform several functions such as operational planning and user's communique publication.

The [Technical Committees] are formed to monitor, comment on, and handle information for specific S/3000 areas. The initial technical committees are described below.

The ⌜Special Interest Groups⌝ provides a monitor for, comment on, and handling of information with regard to groups of users with a common interest such as education, business, or scientific.

The ⌜Planning Committee⌝ provides the detailed plans for meetings, operations, and direction of the User's Group.

The [Project Committee] provides for administration and execution of special User Group efforts aimed at arriving at a particular conclusion or solving a particular problem.

The committees may be formed as required and planned for by the planning committee.

The initial committees are:

Executive Committee

Directs organization
Administrates committees
Provides central interface with HP
Acts as planning committee
Publishes User's Group Communique
Specifies User's Group tasks
Generates User's Group position, status, or priority papers

Staff

Mailing of Communique
Maintenance of list of users
Establish document library

User Interface Technical Committee

Establish policy, standards, monitor function, and information control for the following areas of concern:

User Interface*--dialog messages and procedures
User documentation
System documentation
Software Library
Training
Marketing
Evaluate future S/3000 products

---

*Smooth the transition between the user's task and computer system technical requirements.

Subsystem Technical Committee

Establish policy, standards, monitor function, and information control for the following subsystems:

MPE
Languages: BASIC, FORTRAN, COBOL, SPL
EDITOR
STAR
SORT
Utility
SEGMENTER
Files

Committee Topics Not Considered at this Time

Data base and information retrieval
Plotting
Real-time control
Data collection
System testing
Communications
System performance

HP 3000 USER'S GROUP BYLAWS

SYSTEM 3000 USER'S GROUP BY-LAWS

ARTICLE I. PURPOSE

The System 3000 User's Group is an organization formed by individuals interested in the operation and enhancement of the Hewlett Packard System 3000. The principal areas of concern are:

1. Communication of information between users.
2. Development and exchange of System 3000 techniques.
3. Monitor and comment on design and development of S/3000 hardware, software, documentation, and procedures.
4. Forum for user-user and user-HP dialog.

ARTICLE II. MEMBERSHIP

1. Membership is open to any individual interested in the general purposes of the Group expressed in Article I.
2. There are two membership categories, full member representing a particular S/3000 site and associate member for any interested person.
3. Voting privileges provide each full member with one vote; Hewlett Packard personnel do not have voting privileges, except for one Hewlett Packard representative to the Board of Directors.
4. Membership fees are levied by the Board of Directors as required.
5. Membership is defined further in Article III, 8.

ARTICLE III. ORGANIZATIONAL STRUCTURE

The committees form the working entities of the User's Group. The basic structure of the User's Group consists of a Board of Directors and Executive Committee, Administrative Staff, Planning Committee, Technical Committees, Project Committee, and General Membership. All officers and committee chairmen are full members of the User's Group.

1. Board of Directors

   The Board of Directors provides the policies for direction of the User's Group and consists of the User's Group President, Vice President, Secretary, and Treasurer; Planning Committee Chairman; one Technical Committee Chairman; one Special Interest Group Committee Chairman;

Project Committee Chairman; Member at Large; and Hewlett Packard representative. The User's Group President shall be the Chairman of the Board. Each member has one vote. All members of the Board are officers of the User's Group.

Responsibilities:

a. Specify User's Group policy

b. Determine financial resources and allocate all expenditures of funds. Any single expenditure decision may be over-ridden by 2/3 majority of the voting membership.

c. Recommend membership and fees for approval by a majority of the voting membership.

The term of office of board members is defined by their representative office. The Member-at-Large is elected by the membership for a term of one year.

The Board of Directors shall meet at least once a year to conduct its business, not during the General Meeting.

2. Executive Committee

The Executive Committee provides the management for the User's Group and consists of four elected members and one member appointed by the President for a term of one year. The four elected committee members are:

President
Vice President
Secretary
Treasurer

Each elected committee member serves a term of two years and are elected at one time. The date of elections shall be specified by the Board of Directors.

Responsibilities of this committee are as follows:

a. Direct and maintain all operations of the User's Group.

b. Conduct all business of the User's Group.

c. Represent the User's Group and act as liaison to Hewlett Packard Corp.

d. Perform the function of any non-designated committee.

e. Designate committees to perform a particular function.

f. Maintain communication with all User's Group members.

g. Make arrangements for meetings.

h. Maintain and distribute a current membership list and enroll new members.

i. When necessary, set fees for meeting registration and distribution of printed information.

j. Carry out any other administrative function as designated by the User's Group.

k. The Executive Committee is responsible to the Board of Directors.

l. The President shall act as Chairman of this committee.

m. The Vice President shall serve in the absence of the President and perform other duties as directed by the President.

n. The Secretary shall maintain all non-financial records of the User's Group, make proper correspondence, and direct mailings.

o. The Treasurer shall maintain all financial records and administrate collection and dispersement of all monies.

p. The appointed member of the committee shall perform such functions as designated by the President.

The Executive Committee shall meet at least once a year to conduct its business, not during the General Meeting.

3. Staff

The Staff consists of a set of personnel chosen by the President† and confirmed by the Board of Directors. Their responsibilities are to perform clerical functions supporting the Board of Directors and Executive Committee. One Staff member shall be available during the normal working hours on every working day of the year. The staff members serve an indefinite term.

---

†Suggestions are made by Hewlett Packard since they are extensively involved in this activity.

4. Planning Committee

The Planning Committee provides detailed plans for meetings, User's Group operations, and direction of the User's Group. Specific areas of planning are designed by the Executive Committee. The Planning Committee is formed and dissolved as required by the Board of Directors.

5. Technical Committees

The Technical Committees are formed and dissolved as required by the Board of Directors to monitor, comment on, and handle information for specific S/3000 technical/operational/functional areas. Each Committee Chairman is appointed by the President for a term not to exceed two successive years. An administrative chairman over all technical committees may be appointed by the Executive Committee as required.

The Chairman may appoint, or optionally the members interested in the technical area may elect, as many members as required to administrate the committee's activity. Meetings and operational procedures are defined by each committee.

The responsibilities of each technical committee is to collect, disseminate, and act as a repository for information pertaining to their specific technical area. This committee is expected to

a) Keep the President and User's Group membership appraised of developments, techniques, documents, and other information in each technical area.

b) Maintain liaison with appropriate Hewlett Packard personnel.

c) Generate appropriate documentation for the specific technical area.

d) Promote System 3000 improvement for the particular area.

e) Handle user problems and complaints regarding the S/3000 technical area where appropriate.

6. Special Interest Committees

The special interest committees are formed and dissolved as required by the Board of Directors to monitor, comment on, and handle information for specific areas of user interest. Each committee chairman is appointed by the President for a term not to exceed two successive years. An administrative chairman over all special interest committees may be appointed by the Executive Committee as required.

The Chairman may appoint, or optionally the members interested in the technical area may elect, as many members as required to administrate the committee's activity. Meetings and operational procedures are defined by each committee.

The responsibilities of each special interest committee is to collect, disseminate, and act as a repository for information pertaining to the specific special interest. This committee is expected to:

a. Keep the President and Users concerned with each area of special interest appraised of developments, techniques, documentation, and other information for each area of special interest.

b. Maintain liaison with the appropriate Hewlett Packard personnel.

c. Generate appropriate documentation for the specific special interest.

d. Promote improvement in each area of special interest.

e. Handle user problems and complaints regarding the area of special interest where appropriate.

7. Project Committee

The Project Committee is formed and dissolved as required by the Board of Directors to administrate and execute special User's Group efforts aimed at arriving at a particular conclusion or solving a particular problem. A committee chairman is appointed by the President for a term not to exceed two years.

The Chairman may appoint specific members to head particular projects, meetings, operational procedures, and activities to be defined by the committee.

The responsibilities of this committee are to perform the designated project task and produce a final report.

8. General Membership

The General Membership shall consist of all full and associate members, including interested Hewlett Packard personnel. Membership may be requested at any time and renewed annually on a date specified by the Board of Directors. Any member may choose to support one or more technical or special interest committees.

Each member is encouraged to:

a. Contribute a nominal effort towards the support of the User's Group.

b. Contribute programs, techniques, documentation, and subsystem designs for the benefit of the group.

c. Promote the enhancement of the S/3000 and the User's Group.

d. Provide constructive criticism of the S/3000 and User's Group.

e. Contribute a limited manpower and computer time in support of the User's Group.

f. Join a committee effort.

A member of the User's Group acting in such a manner so as to extremely degrade the User's Group or System 3000 may be barred from User Group membership upon recommendation by the Executive Committee, approval by the Board of Directors, and two-thirds majority vote of the voting membership.

ARTICLE IV. MEETINGS

A general meeting of the User's Group membership shall be held annually. The purpose of this meeting will be to:

1. Conduct User's Group business; unless otherwise specified, parliamentary procedures are to be followed and Roberts Rules of Order shall prevail. Voting on items of official business† shall be conducted by mail.

2. Present the status and comments from each committee.

3. Entertain Hewlett Packard's remarks on the System 3000.

4. Engate in workshop sessions when necessary.

5. Provide for Technical, Special Interest, Planning, and Project Committee meetings as required.

---

†Official business shall consist of officer elections, levying of fees, ammendments to By-Laws, and other items as designated by the Board of Directors.

6. Entertain any special presentations.

7. Entertain other vendor products applicable to the System 3000.

A member of the Board of Directors shall act as Chairman of the meeting as designated by the President.

ARTICLE IV. AMENDMENTS

1. Amendments to these By-Laws must be submitted in writing to the Executive Committee at least 90 days before a regularly-scheduled User's Group meeting. The Board of Directors will transmit the proposed amendment to the membership with the notice for the next meeting.

2. A proposed amendment shall become a part of these By-Laws upon approval of two-thirds of the eligible voters in attendance at a meeting.

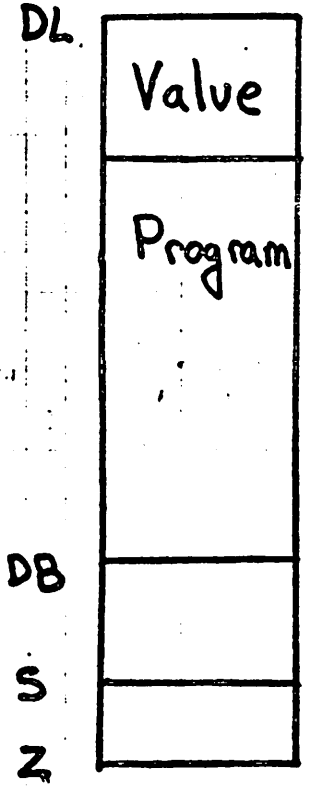
3. These By-Laws shall become effective upon approval of two-thirds of the membership present at the June 1974 meeting.

BASIC COMPILER - TERRY HAMM

BASIC Interpreter Runtime Environment

24 code segments ~37000 words

>RUN PROG



PROG Data Area

Program Area
- Symbol Table
- Statements
- Label Table

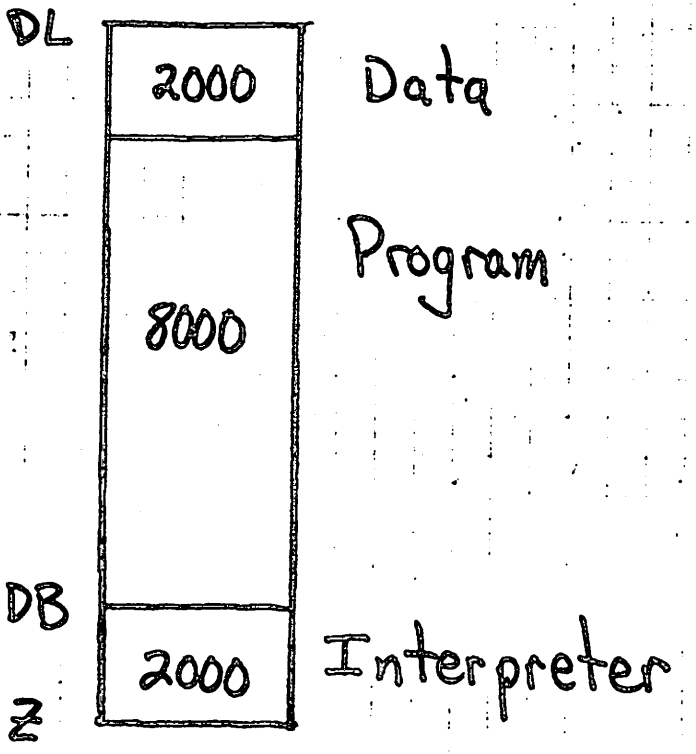
Data Area
Runtime Stack
} Interpreter

C-155

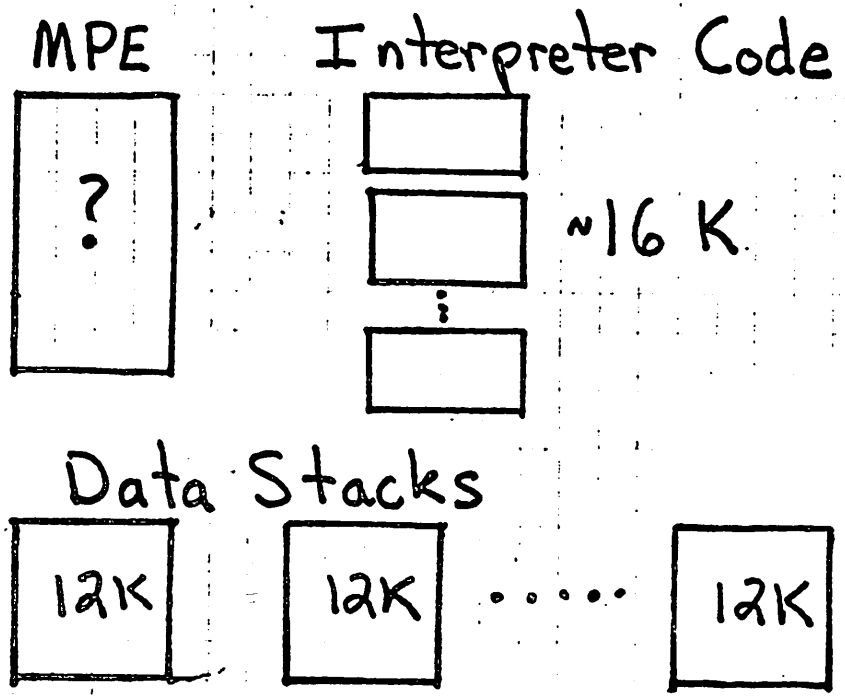
Data - 2000 words
Arrays, variables
file buffers

Program - 8000 words

Interpreter - 2000 words



Total
12K

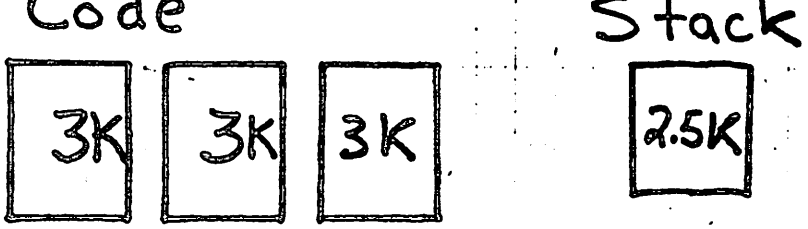


n	Memory
2	40 K
4	64 K
8	112 K
12	160 K
16	208 K

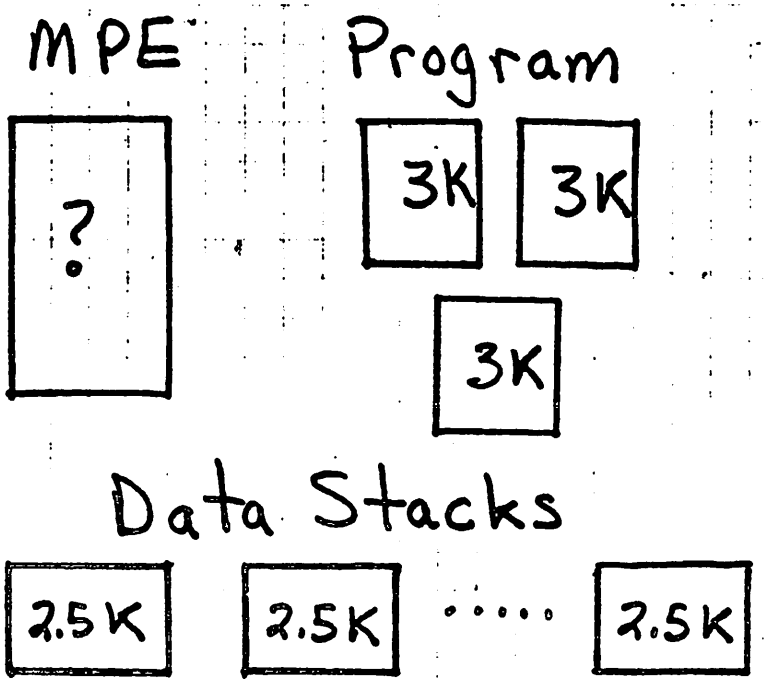
$12 \cdot n + 16$

FORTRAN

Program 9000 words
3 segments
Data 2500 words



Total 11.5K



n	Memory
2	14 K
4	19 K
8	29 K
12	39 K
16	49 K

$2.5 \cdot n + 9$

$\sim \frac{1}{4}$

HP 3000 BASIC COMPILER

BASIC Compiler

- Extension of Interpreter
- Compatible to Interpreter
- Runs as subsystem under MPE
- Commands control the compiling

Objectives

- Performance - improve BASIC program execution
  1. Sharable Code
  2. Smaller data stacks
  3. Executable code
- Compatibility
- Generality

User Interface

:BASICOMP [commandfile][,[uslfile][,listfile]]

	default file	use
command file	$STDIN	Input subsystem commands
uslfile	$OLDPASS	Output of RBM's
listfile	$STDLIST	List output

:BASICPREP [commandfile][,[progfile][,lstfile]]

:BASICGO [commandfile][,listfile]

Subsystem Commands

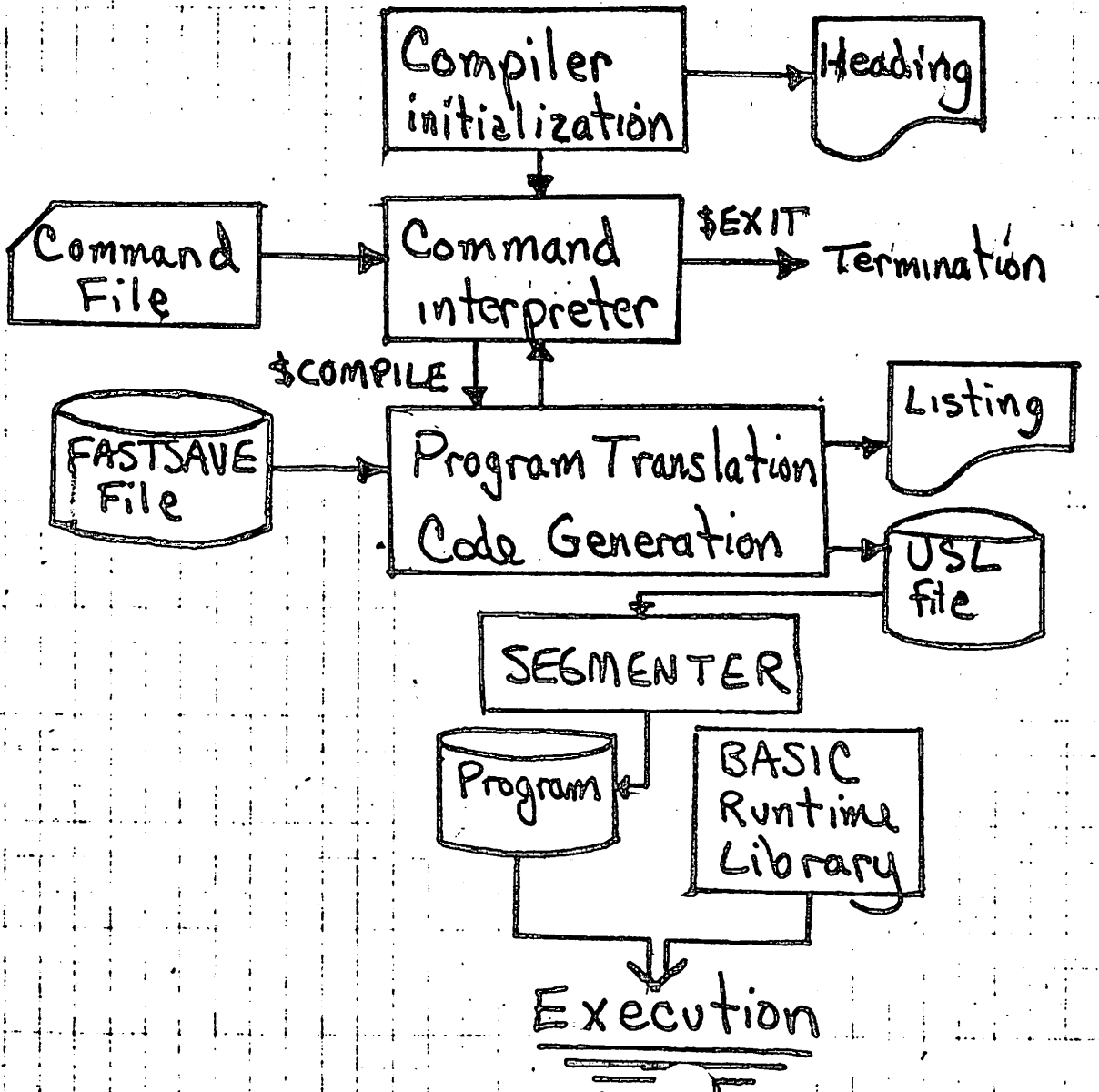
$CONTROL	parameter list
$ COMPILE	program name list
$ ENTRY	chain invoke list
$ TITLE	character string
$EXIT	

$CONTROL parameters

LIST	NOLIST
SOURCE	NOSOURCE
LABEL	NOLABEL
CODE	NOCODE
MAP	NOMAP
LINES =	number of lines
WARN	NOWARN
USLINIT	
SUBPROGRAM	START= name
SEGMENT = name	
INIT	

```
$CONTROL SOURCE,USLINIT,NOWARN
$CONTROL LABEL,LINES=50,START=C
$TITLE "PROGRAM C", <<MAIN>> "JAN 25"
$COMPILE  C(100,200)
$CONTROL  NOSOURCE,NOLABEL
$ COMPILE  A,B
$ ENTRY  A,B
$ ENTRY  SLPROG
$ EXIT
```

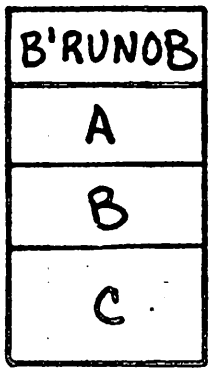
Compilation Process



13

```
$COMPILE  A, B, C
```

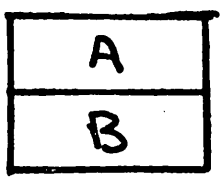
SEG'



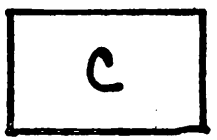

```
$ CONTROL  SEGMENT=SEG1
$ COMPILE  A, B
$ CONTROL  SEGMENT=SEG2
$ COMPILE  C
```

SEG'

SEG1



SEG2



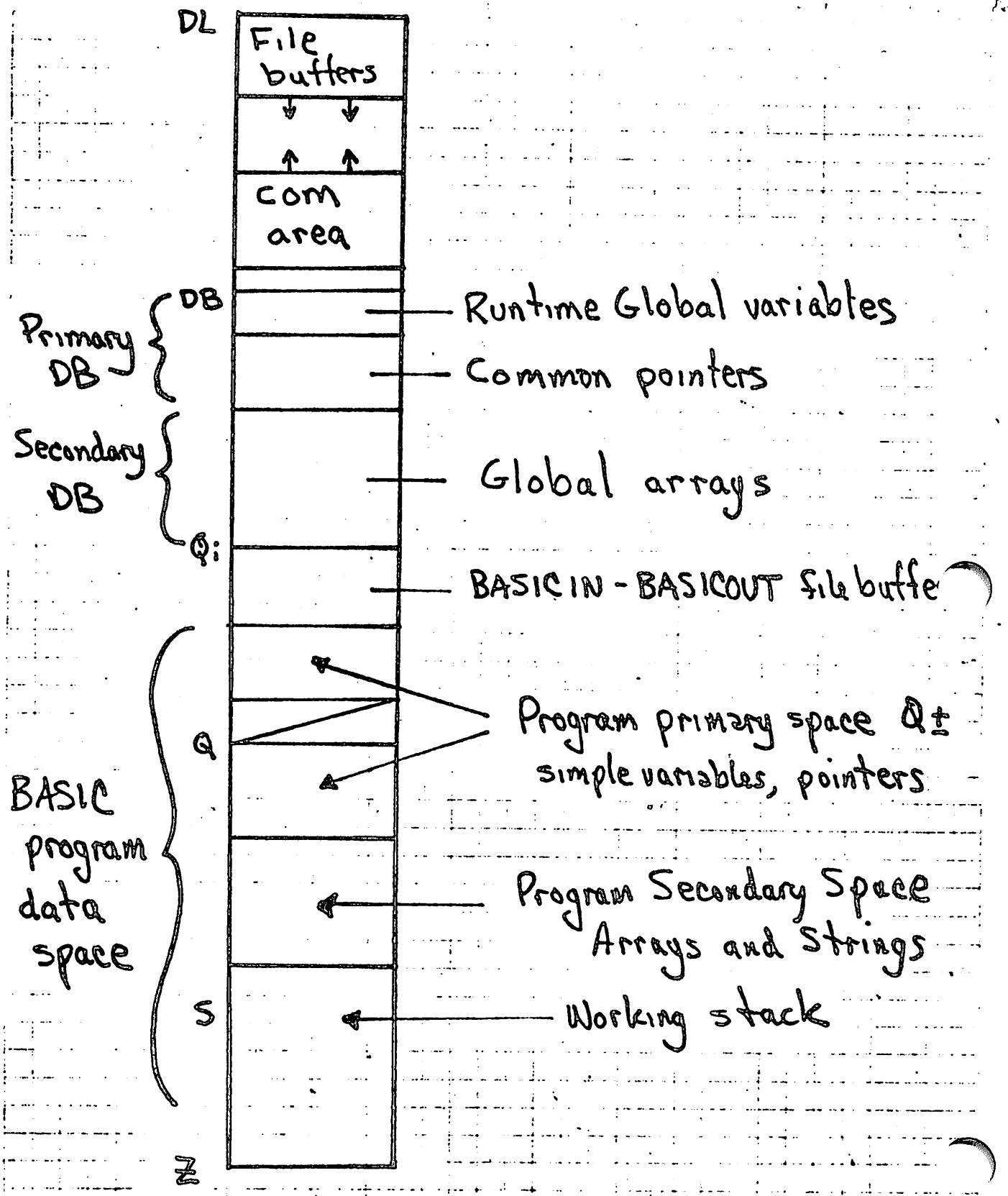
B'RUNOB

- Open files BASICIN - BASICOUT
- Allocate file buffers
- Enable arithmetic & library traps
- Initialize runtime global variables
- INVOKE to START program
- Handle termination upon return from START

BASIC runtime library

- COM area - file space management
- I/O utilities
  - Formatting
  - File manipulation
- String routines
- Matrix routines
- Built in functions

Runtime Stack



SYSTEM MANAGEMENT - Larry Birenbaum

SYSTEM MANAGEMENT

Introduction

"System Management" - SM, AM
- organization of accounts, groups, users
- accounting
- capability control
- file security

Coverage

Organization of accounts, groups, users

Evolution of structure

Attributes of structure
- 2 trees
- dual purposes of accounts, groups
- hierarchial

Benefits of structure
- partition of files
- define relationship between users and groups / files
- "parental limitations" - control hierarchy (SM, AM)

Additional attributes of structure
- logon control
- groups - logon & home
- accounting

Capability

- components
- Purpose
- Use
- (Maxpri and local attributes)

File Security

- "Security is a matrix"
- Concept
- Implementation
  - access / accessors
- Hierarchial protection
- Defaults
- Other forms
  - RELEASE
  - lockword

Comments on organizational structuring

- Basic structure?
- Distribution in structure
- Interpretation of groups
- Applications of groups

Some examples

BASIC STRUCTURE

ACCOUNTS

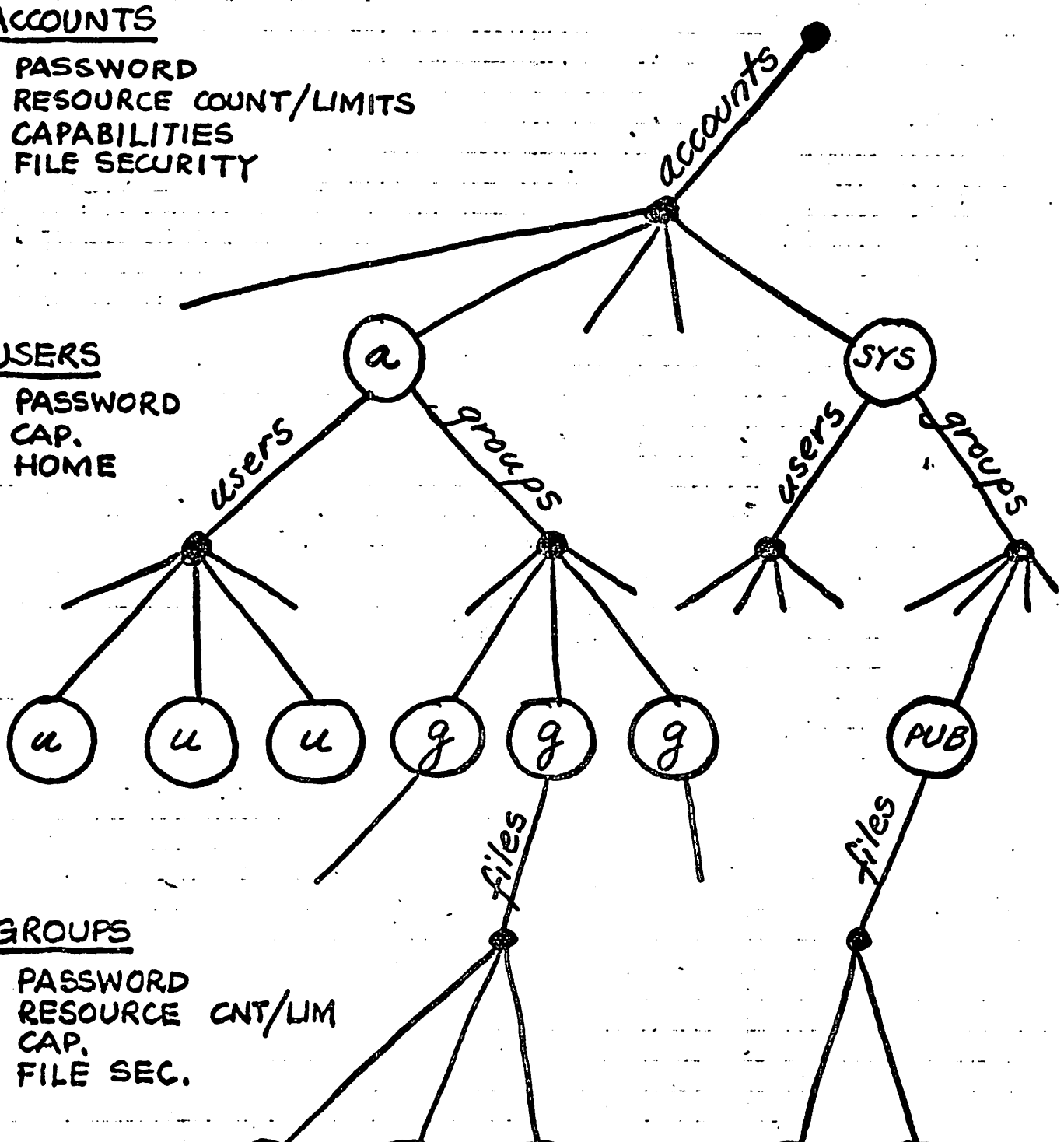
PASSWORD
RESOURCE COUNT/LIMITS
CAPABILITIES
FILE SECURITY

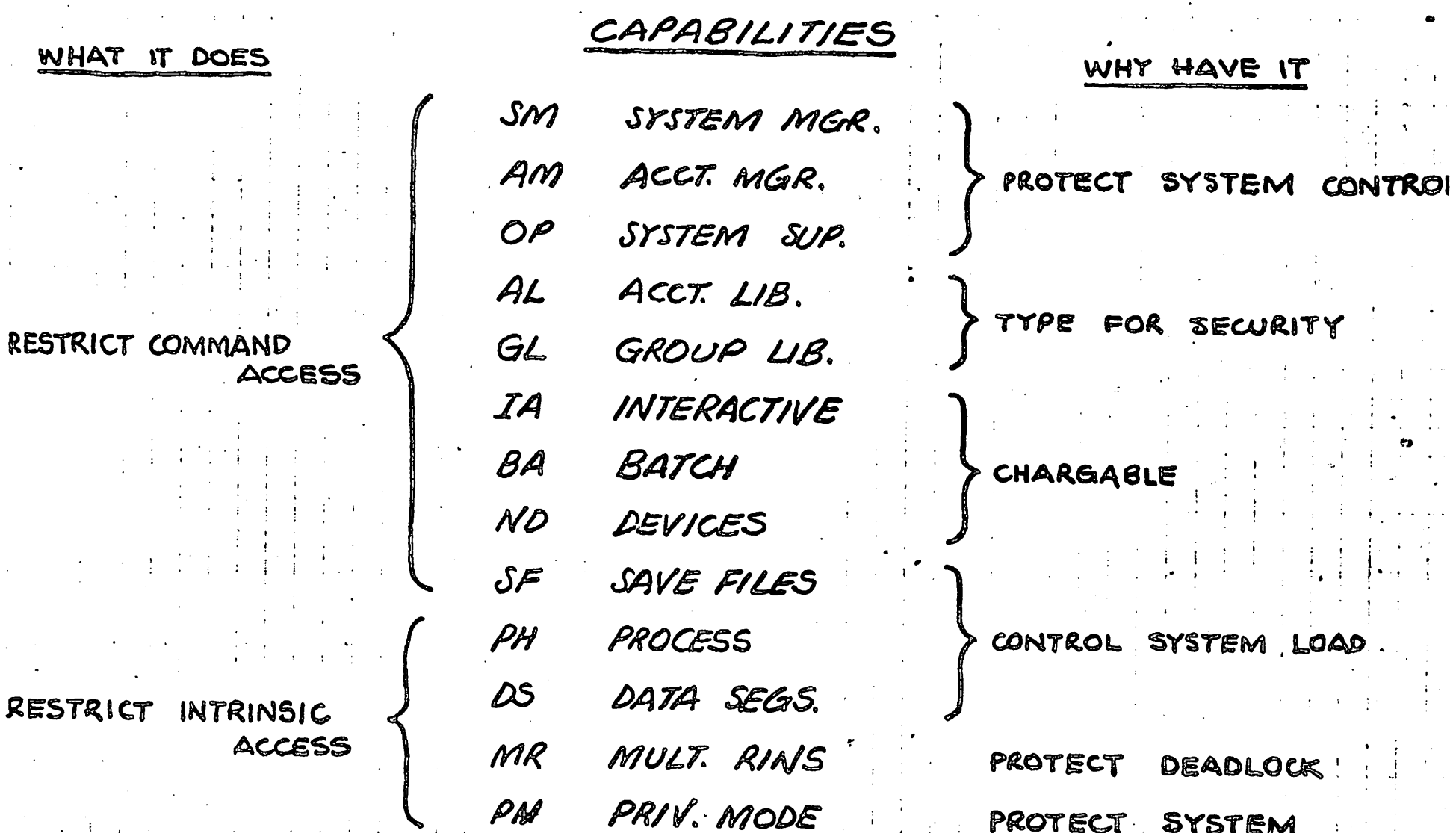
USERS

PASSWORD
CAP.
HOME

GROUPS

PASSWORD
RESOURCE CNT/LIM
CAP.
FILE SEC.

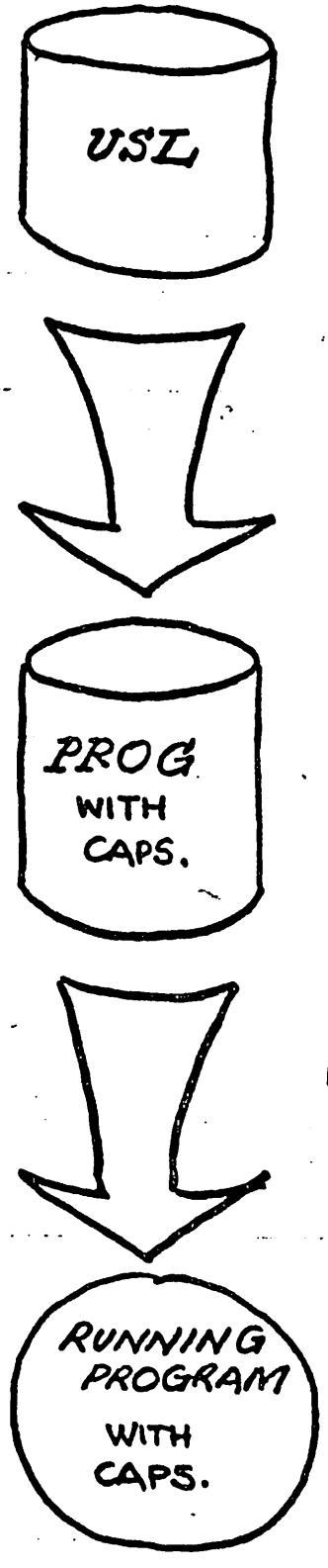



WHAT IT DOES
CAPABILITIES
WHY HAVE IT
RESTRICT COMMAND ACCESS
RESTRICT INTRINSIC ACCESS
SM SYSTEM MGR.
AM ACCT. MGR.
OP SYSTEM SUP.
AL ACCT. LIB.
GL GROUP LIB.
IA INTERACTIVE
BA BATCH
ND DEVICES
SF SAVE FILES
PH PROCESS
DS DATA SEGS.
MR MULT. RINS
PM PRIV. MODE
PROTECT SYSTEM CONTROL
TYPE FOR SECURITY
CHARGABLE
CONTROL SYSTEM LOAD
PROTECT DEADLOCK
PROTECT SYSTEM

PROGRAM CAPABILITIES

THE PROTECTION MECHANISM

AM, MR, DS, PH

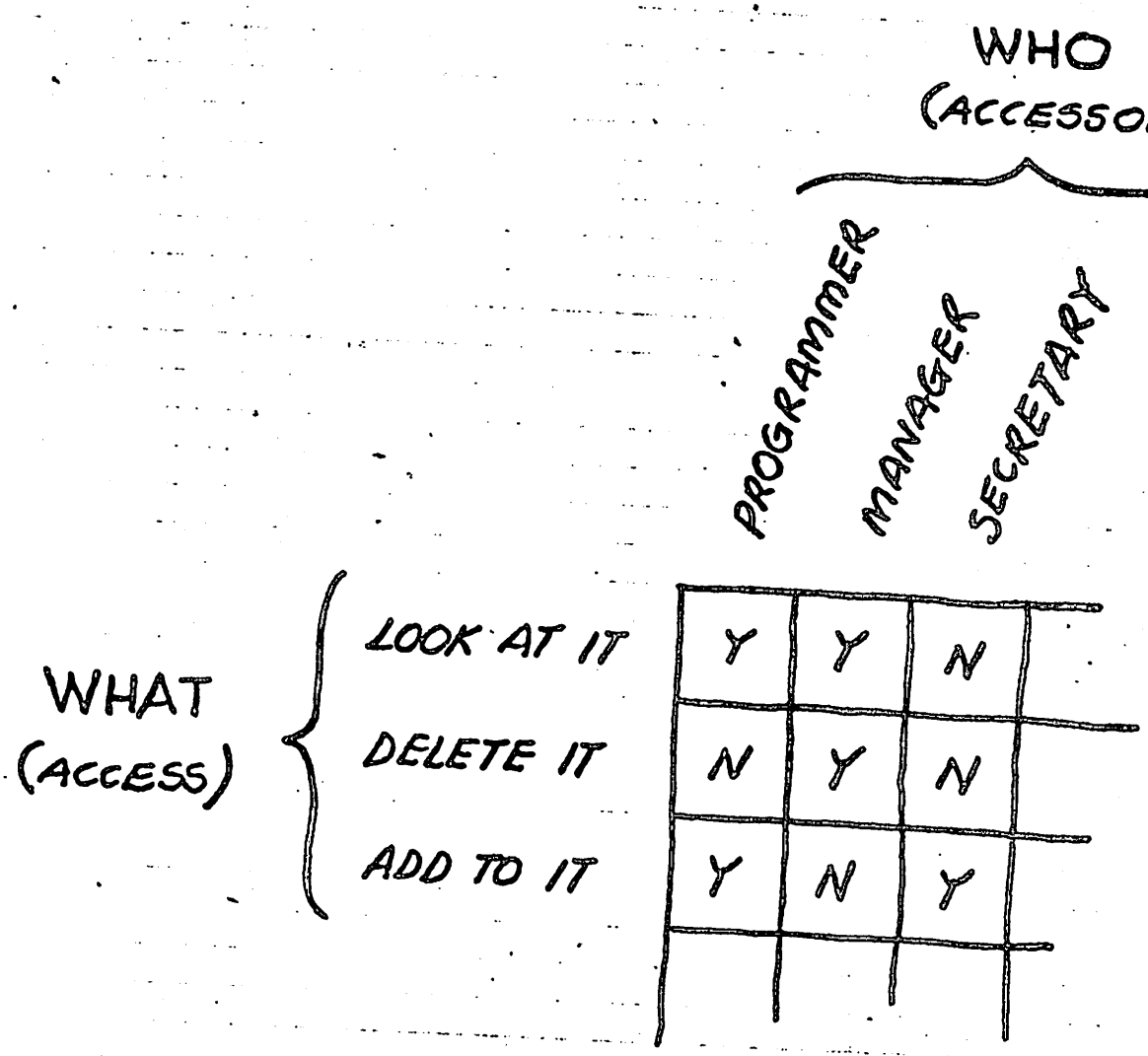


USER MUST HAVE CAPS.

PROGRAM FILE'S GROUP

MUST HAVE CAPS.

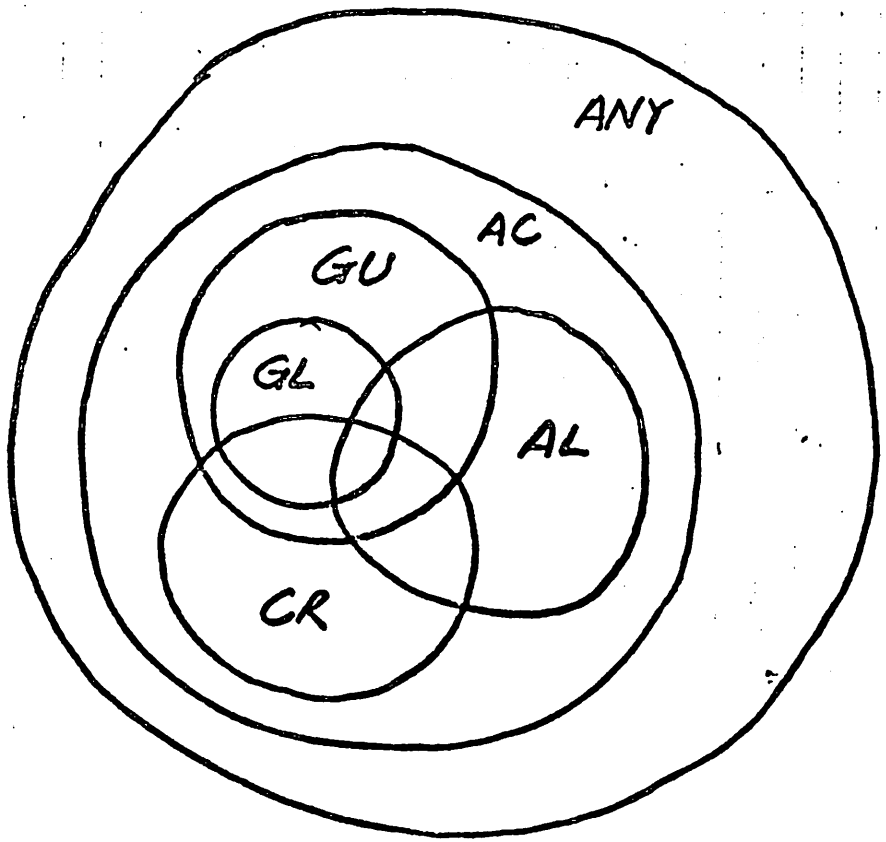
FILE SECURITY MATRIX



ACCESS

- R READ
- A APPEND
- W WRITE
- L LOCK
- X EXECUTE
- S SAVE

FOR A GIVEN FILE ACCESS



ACCESSOR

ANY	ANY USER
AC	ACCT MEMBER
AL	AC ∧ AL CAP
GU	HOME ∨ LOGON
GL	HOME ∧ GL
CR	CREATOR

SOME EXAMPLES

CRITERIA	SOLUTION
GROUP FOR "NEWS" FILE	GROUP SEC- R:ANY A:GU W,S:GL
READ- EVERYONE ADD TO FILE - REPORTER CREATE/DELETE FILES- EDITOR	REPORTER- HOME = GROUP EDITOR- HOME=GROUP; and GL
GROUP FOR PROPRIETARY PROGRAMS	GROUP SEC- X:ANY W,S:GU R:GL
RUN- SELECTED PEOPLE DELETE/CREATE FILES- LIBRARIAN MODIFY PROGRAMS- SYS. PROGRAMMER	[LOCKWORD ON FILES] LIBRARIAN- HOME=GROUP PROGRAMMER- HOME=GROUP; and GL
GROUP FOR DATA COLLECTION	GROUP SEC- A:ANY R:GU S:AL W:GL
ADD TO FILES- SALESMEN READ- DISTRICT MANAGERS DELETE & MODIFY- SALES MANAGER CREATE FILES- SYS. PROGRAMMER (special format)	DISTRICT MGR- HOME=GROUP SALES MGR- HOME=GROUP; and GL PROGRAMMER- AL
SIMPLE ACCOUNT	
UNLIMITED ACCESS TO ALL USERS N MALICIOUS USERS	give everyone AM

(MORE EXAMPLES)

GROUP FOR TIME ACCT'G ONLY; NO FILES
- give PASSWORD to authorized users.
- GROUP SEC- R:GU

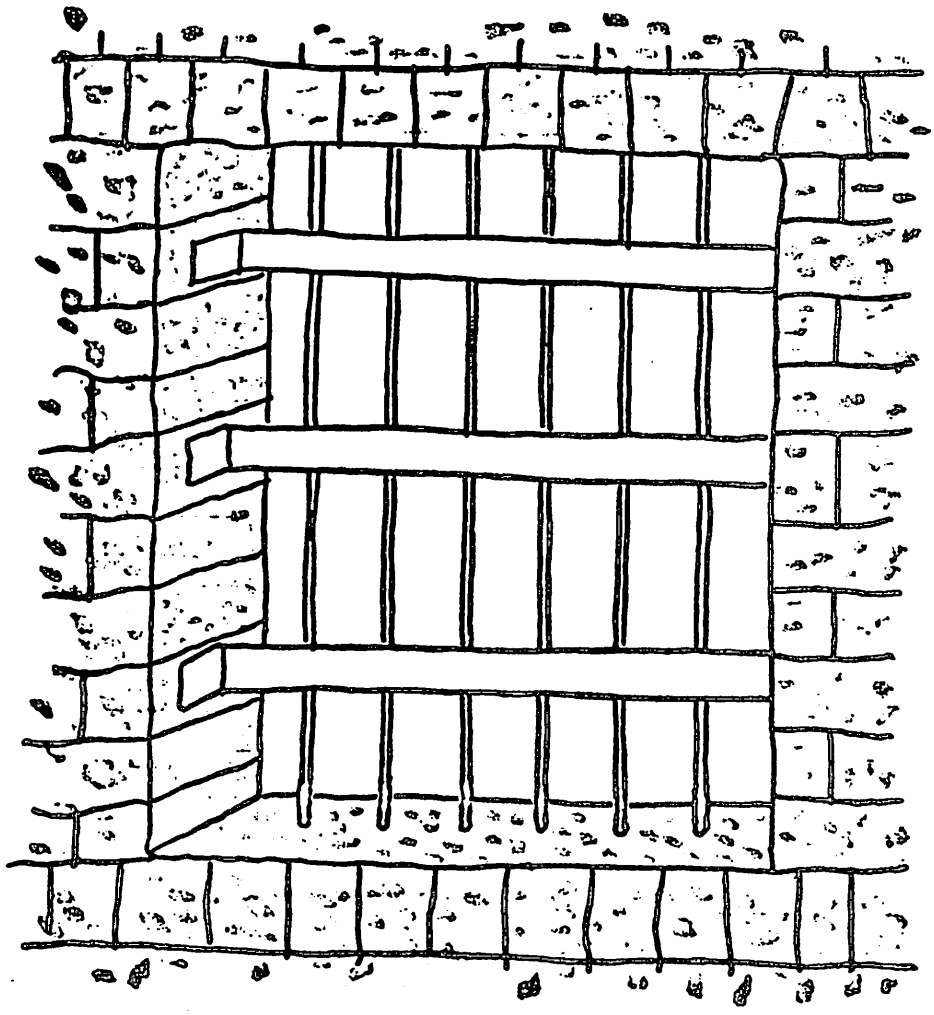
GROUP FOR FILES ONLY; NO TIME ACCT'G
- give PASSWORD to no one!
- GROUP SEC- as required.

ACCT. WITH NO FILES; SAVE ACCESS DISALLOWED- don't give account SF

USER WITH TWO PRIVATE GROUPS
- HOME = GROUP1
- logon with GROUP2 (PASSWORD)
- (default group security)

"SECURITY IS A MATRIX"

JAIL



UNSUPPORTED UTILITIES

THE UNSUPPORTED UTILITIES

This is a collection of programs that is ever growing and ever changing. We do not purport to support these, only offer words of experience and perhaps a little advice.

These programs were developed by various people and submitted for use by us all. I am breaking them into two sections; one called harmless and useful, and one called dangerous but useful. The latter group contains those utilities which either allow modification of code directly in the system or which perform their function outside of the normal channels of MPE.

Use these to the fullest; be cautious and if unsure, ask.

THE HARMLESS ONES

CLOSEDEV

This is a handy utility for the area of device hang-ups. CLOSEDEV will check the logical device number passed in the PARM part of the command. If the device is being acquired as a virtual device, but is not owned by a process, then the device is cleared and may be used again.

```
:RUN CLOSEDEV;PARM=
```

Supply the number of the device to be cleared. If you're way off base, you'll get:

"AIN'T NO SUCH DEVICE FELLA"

or

"DEVICE IS NOT A VIRTUAL FILE"

Hopefully, you'll finally get success,

"VIRTUAL FILE DELETED"

CROSSREF

This is a handy utility which takes a sequenced SPL readable source tape and produces a cross-reference. All identifiers in the program, global and local, are listed alphabetically along with the sequence numbers of every reference to that identifier. The first sequence number always refers to the actual declaration, not including FORWARD, ENTRY, or LABEL. These exceptions are included in the sequence numbers that follow. ENTRY and LABEL identifiers are assumed to be declared at the point where the label actually occurs. When DEFINE appears, a reference for all identifiers encountered within the define, and a reference also appears for the DEFINE identifier itself. The format of the print-out is as follows:

ID OF SUB OF PROC (VALUE PARAMETER REAL)

Where ID is the actual identifier name; SUB is the subroutine in which the ID was declared; and PROC is the procedure in which the SUB was declared. If ID was not declared in a subroutine then its "ID OF PROC." If ID was global, just "ID" will appear. The following resulted from mounting my source tape and typing

:RUN CROSSREF

```
                S.P.L. CROSS REFERENCE TABLE--- APR 10, 1973 VERSION

NUMBER OF CARD IMAGES=3364.  NUMBER OF SYMBOLS=250.  NUMBER OF REFERENCES=2650.

ADDAUXBUF OF GETREADY (SUBROUTINE)
        01440000  01690000  01708000

ANSWERED OF ERROR (LOGICAL)
        00191000  00302000  00303000  00319000  00333000  00336000

ARECLEN (INTEGER)
        00084000  02904000  02983000  03063000  03089000  03092000  03101000

ASCII (INTRINSIC)
        00159000  00402000  00407000  00412000  00417000  00458000  00462000

ASCIITOEBCDIC (PROCEDURE)
        02203000  02680000  03058000

ASCIITOEBCDIC (PROCEDURE)
        02153000  02678000  03055000

ATR OF ASCIITOEBCDIC (BYTE ARRAY)
        02208000  02228000
```

Obviously, this is only a sample, the listing went on and on.

The output goes to PRF which defaults to line-printer.

DECOMP

Debugging high-level language code can be very tedious without a tool to break the code into machine executable symbols. The option, CODE, on your $CONTROL card will give an octal representation of the compiler generated code, but DECOMP gives a SPL mnemonic listing of your program with such information as all reference to external procedure names, the segment transfer table; and all entry points have their labels inserted in the proper location. The following is a DECOMP of the file FREE. Dialogue runs like this:

```
:RUN DECOMP

FILE NAME?
FREE
NO. OF SEGMENTS: 1
STARTING SEGMENT ? 0
THIS SEG HAS LENGTH OF %534
ENTER STARTING P-VALUE (PRECEDED BY %)%500

 END OF PROGRAM
```

Output for this program goes to the designator OUT which defaults to line printer.

This asked for a dump of Segment Ø beginning at location %500. Since the segment is %534, the results is as follows:

```
SEGMENT 0       PRIV=1       LENGTH =%000534

000500  004500    @                       DUP  ,-NOP
000501  051607    S                       STOR  Q- 007
000502  020063     3                      MVB  ,4  ,3
000503  031403    3                       EXIT %0003

000504  041605    C                       LOAD  Q- 005          <-- PROCEDURE ENTRY POINT
000505  037777    ?                       ANDI %0377
000506  004500    @                       DUP , NOP
000507  022001    $                       CMPI ,1
000510  141203    B                       BE    P+ 003
000511  031006    2                       PCAL DEBUG
000512  031007    2                       PCAL TERMINATE
000513  031402    3                       EXIT %0002
000514  106063    SEGMENT TRANSFER TABLE (PL-%017)
000515  101420    SEGMENT TRANSFER TABLE (PL-%016)
000516  101453    SEGMENT TRANSFER TABLE (PL-%015)
000517  106463    SEGMENT TRANSFER TABLE (PL-%014)
000520  107463    SEGMENT TRANSFER TABLE (PL-%013)
000521  104463    SEGMENT TRANSFER TABLE (PL-%012)
000522  101056    SEGMENT TRANSFER TABLE (PL-%011)
000523  103063    SEGMENT TRANSFER TABLE (PL-%010)
000524  100456    SEGMENT TRANSFER TABLE (PL-%007)
000525  100470    SEGMENT TRANSFER TABLE (PL-%006)
000526  100426    SEGMENT TRANSFER TABLE (PL-%005)
000527  000504    SEGMENT TRANSFER TABLE (PL-%004)
000530  000444    SEGMENT TRANSFER TABLE (PL-%003)
000531  000110    SEGMENT TRANSFER TABLE (PL-%002)
000532  000000    SEGMENT TRANSFER TABLE (PL-%001)
000533  040017
```

DISKDUMP

It is often necessary to see how a program is stored on the disk. This is especially true when investigating such items as USL headers or the like. Entries on the disk can be dumped by this utility in an octal format. All the user must provide is the file name and the range of records to be dumped to the output designator OUT, which defaults to the line-printer.

```
:RUN DISKDUMP

FILE NAME? FREE
RANGE? 0,5

 END OF PROGRAM
```

The above example of dialogue caused five records to be dumped. The following is record 3 of the file FREE. This third record begins the segment 0 of the file as can be compared with the output from the utility PATCH. The first records of this dump were omitted.

```
FILE FREE.HPUNSUP.SUPPORT  RECORD %000003

  0  024410  013302  027410  041000  021013  000600  031012  000600  ).../.B."...2...
 10  041001  040050  021001  035013  040046  031005  051010  141203  B.@("..:.@&2.R...
 20  031006  031007  041005  000606  031012  041002  021011  021320  2.2.B...2.B."."..
 30  031012  000600  041005  025024  031010  004300  021015  166006  2...B.*.2..."...
 40  000600  173006  040021  020121  011502  031011  004546  020220  ....@. Q.B2..F .
 50  040014  020122  011413  004002  041003  021007  000600  031012  @.-R....B."...2.
 60  140436  000414  016000  006440  006440  003221  141204  004500  ....... . .....@
 70  022007  141303  000200  140417  031013  051007  141562  021001  $........2.R..R".
100  021377  131007  012602  140427  041007  031002  140464  031011  "........B.2..42.
110  034026  035004  000700  041604  021001  000606  041004  000600  8.:...C."...B...
120  021200  041401  041402  031015  031004  000600  043004  004500  ".C.C.2.2...F..@
130  051403  021012  173006  170005  010201  140013  000000  000024  S."..............
140  021440  047506  020105  047124  051111  042523  020075  020040  # OF ENTRIES =
150  021017  020042  031014  051404  000600  021401  047004  021012  ". "2.S...#.N.".
160  041404  022417  004300  177006  170003  010201  140011  020040  C.%.............
170  020124  040502  046105  020123  044532  042440  036440  021020    TABLE SIZE = ".
```

DPAN

This utility produces a formatted analysis of HP3000 system dump tape produced via the front panel of the machine. The dump itself must be acquired properly or the desired snap-shot of memory will be trash. But assuming the dump is correct, DPAN will analyze and list the following information:

1. ID page
2. Register pages
3. Low fixed core
4. System global area
5. DRT
6. CST
7. DST
8. PCB
9. Scheduling queue
10. Linked memory
11. Octal or hex main memory dump

Operation proceeds in the following steps:

1. Mount the SYSTEM DUMP TAPE.
2. Ready the MT driver and select the desired unit number.
3. Enter any necessary FILE commands.
   a) :FILE DPANMAST; DEV = TAPn
      to assign the SYSTEM DUMP TAPE to tape unit #n.
   b) :FILE DPANLIST; DEV = TAPn
      to assign tape unit #n for all list output instead of line printer.
4. Set SWITCH REGISTER for desired options (see below).
5. Enter :RUN DPAN and wait.

The SWITCH REGISTER options are given below:

BIT #	UP SIGNIFICANCE
Ø	Suppress formatting of tables; octal only will be output.
1	Suppress all table analysis and printing; outputs: a) ID PAGE b) REGISTER PAGE c) LOW FIXED CORE d) Octal dump of all memory
2	Causes restart at end of current execution
3	Call DEBUG at all points present
4	Gets main memory dump in hex
5-9	Leave down
10	Terminate main memory dump immediately
11-15	Leave down

FINFO

There are times when during the course of debugging it is helpful to get file information whether it's about system files such as $SDTIN or about files which perhaps FORTRAN, has opened for you. You may think it was opened variable; in reality it was opened fixed. To get this information is quite simple; just run FINFO and supply a qualified file name. An example of a qualified and unqualified file follows:

```
:RUN FINFO

FILE NAME:FREE

+-F-I-L-E---I-N-F-O-R-M-A-T-I-O-N---D-I-S-P-L-A-Y+
[  FILE NAME IS FREE.HPUNSUP.SUPPORT               ]
[  FOPTIONS: SYS,B,*FORMAL*,F,N,SL,DEQ             ]
[  AOPTIONS:   INPUT,SREC,NOLOCK,DEF,BUFFER        ]
[  DEVICE TYPE: 0   LU: 2    DRT: 7    UNIT: 0     ]
[  RECORD SIZE: 128    BLOCK SIZE: 128   (WORDS)   ]
[  EXTENT SIZE: 12     MAX EXTENTS: 1              ]
[  RECPTR: 0           RECLIMIT: 11                ]
[  LOGCOUNT: 0            PHYSCOUNT: 0             ]
[  EOF AT: 8           LABEL ADDR: %00200007350    ]
[  FILE CODE: 1029   ID IS FIELD     ULABELS: 0    ]
[  PHYSICAL STATUS: 1111000000000000               ]
[  ERROR NUMBER: 0     TLOG: 0                     ]
[  BLOCK NUMBER: 0              FACTOR: 1          ]
+-------------------------------------------------+
FILE NAME:FORTRAN

+-F-I-L-E---I-N-F-O-R-M-A-T-I-O-N---D-I-S-P-L-A-Y+
[  ERROR NUMBER: 52    TLOG: 0                     ]
[  BLOCK NUMBER: 0              FACTOR: 0          ]
+-------------------------------------------------+
FILE NAME:

 END OF PROGRAM
```

Output defaults to $STDLIST.

FREE

This program supplies information about the free space available on the system's disk. This could be valuable in assessing disk utilization. The program will prompt the user for the logical number of the disk. The following example shows two disks logical unit 1 and 2.

```
 RUN FREE

 DISC FREE SPACE LISTER

LOGICAL DEVICE #? 1
# OF ENTRIES = 3    TABLE SIZE = 8 SECTORS
MIN = %34    MAX = %37777
SECTOR (%)    LENGTH (%)
15532         3
16412         7
23014         14764

LOGICAL DEVICE #? 2
# OF ENTRIES = 14    TABLE SIZE = 16 SECTORS
MIN = %44    MAX = %547277
SECTOR (%)    LENGTH (%)
17335         10
32142         11
32164         5
37230         10
41521         16
43302         7
44022         25
55221         27
111737        5
112467        2
114552        7
115132        43
116354        52
122172        425106

LOGICAL DEVICE #?

 END OF PROGRAM
```

The output file is LIST which defaults to $STDLIST. A word of caution: Be sure to furnish the correct logical unit number. The results otherwise could be catastrophic.

LISTCRET

For all the millions of times, you've received a tape that could not be restored because the creator was unknown, there is LISTCRET. This utility works for STORE and SYSDUMP tapes. Output goes to $STDLIST unless you set up a file equation setting FTN06 to the proper device. There is also a provision for multi-reel tapes. The example came from system console in session mode.

```
:RUN LISTCRET

 IS THE TAPE A CONTINUATION REEL (Y/N) ?
N

?IO/16:49/#S1/24/LDEV# FOR FTN01 ON TAPE (NUM)=REPLY 24,7

STORE TAPE LIST PROGRAM**********************

FILE NAME                                CREATOR
---------                                -------

INSTALL .HPUTIL    .SUPPORT              FIELD
INSTRUCT.HPUTIL    .SUPPORT              FIELD
J00J212A.HPUTIL    .SUPPORT              FIELD
M00M212A.HPUTIL    .SUPPORT              MGR
P00P212A.HPUTIL    .SUPPORT              FIELD
U00U212A.HPUTIL    .SUPPORT              MGR
 END OF PROGRAM
```

LISTDIRC

This little utility prints out directory information in either an expanded or security form. The user specifies what level he wants from file to user. The expanded print-out has the precaution of allowing system manager unlimited access to all entries, account manager unlimited access to all entries in his account and anyone else can access only his own entries, the results is as follows:

```
:RUN LISTDIRC

ABREVIATED SECURITY PRINT? (Y/N) => Y
FILE  NAME?  =>FREE
GROUP NAME?  =>HPUNSUP
ACCT  NAME?  =>SUPPORT
FILE NAME: FREE.HPUNSUP.SUPPORT.

SECURITY:      READ: AC
(ACCOUNT)    APPEND: AC
              WRITE: AC
               LOCK: AC
            EXECUTE: AC
           SAV FILE: AC

SECURITY:      READ: GU
(GROUP)      APPEND: GU
              WRITE: GU
               LOCK: GU
            EXECUTE: GU
           SAV FILE: GU

SECURITY:      READ: ANY
(FILE)       APPEND: ANY
              WRITE: ANY
               LOCK: ANY
            EXECUTE: ANY
FILE SECURITY OFF
FILE ACCESS FOR FIELD.SUPPORT: READ,WRITE,APPEND,LOCK,EXECUTE

FILE  NAME?  =>
TERMINATE? (Y/N) => Y

 END OF PROGRAM
```

Output goes to the file OP which defaults to $STDLIST.

LISTEQS

This program is used to list all temporary files and file equations that are present in the current job session. This information is needed most urgently at around 2:30 a.m. when your code begins to blur and you can't remember if you set your output file to LP or CARD.

To use the program, type:

```
:RUN LISTEQ
```

The following example of output shows that this session had no temporary files and one file equation; i.e., :FILE OUT;DEV=LP.

```
END OF PROGRAM
```

Output goes to LIST which defaults to the line printer.

LISTLOG

With the logging features of MPE this utility provides a dump of the log file. The formal designator of the input or log file is LOGFILE. The formal designator for the output file is LOGLIST. A parameter is provided to suppress certain record types. By setting the bit for the appropriate type to 1, its output is suppressed. Record types and bit positions are:

Logging Error	0
System Up	1
Job Initiation	2
Job Termination	3
Process Termination	4
File Close	5
System Shutdown	6

An example is:

To list Job Initiation and Job Termination records on Tape

```
:FILE LOGFILE = LOGØØ34
:FILE LOGLIST; DEV = TATE
:RUN LISTLOG; PARM = %163
```

The following example produced much output. I enclose a sample for your enjoyment.

```
:FILE LOGLIST;DEV=LP
:FILE LOGFILE=LOGØØ77
:RUN LISTLOG.HPUNSUP.SUPPORT


 END OF PROGRAM
```

```
   TIME     TYPE  JOB#                        DATE: FRI, JAN, 18,1974     LOGFILE: LOG0077

                                FILE NAME          * DISP * DOM *  SECTORS   * DEV T/# *  RECORDS  *   BLOCKS   *
11:5 :1 :8  FILE  SYS     LOG0077 .PUB      .SYS        1      0      128        0  /2      0            0

                          UPD# * FIX# * CORE * CST * DST * PCB * IOQ * TRL * ICS * JOBS:MAX / RUNNING *
11:5 :2 :3   UP   SYS      00     04     64     157   137   57    64    48    511      20       21

                           USER    *  ACCOUNT *   JOB    *  LOGON G  * LDEV IN  OUT  * LINE LIM  * CPU LIMIT * INP * OUTP *
11:13:53:9  JOB   S 1     FIELD      SUPPORT              HPUNSUP          23  23      0            0          8     8

                                FILE NAME          * DISP * DOM *  SECTORS   * DEV T/# *  RECORDS  *   BLOCKS   *
11:14:15:3  FILE  S 1     $STDLIST.         .          0      1      0          16 /23     5            5

                                FILE NAME          * DISP * DOM *  SECTORS   * DEV T/# *  RECORDS  *   BLOCKS   *
11:14:25:2  FILE  S 1     VANP      .HPUNSUP .SUPPORT    4      1      330        0  /2      0            0

                                FILE NAME          * DISP * DOM *  SECTORS   * DEV T/# *  RECORDS  *   BLOCKS   *
11:15:36:1  FILE  S 1     T         .         .          0      1      0          24 /8      45           45

                                FILE NAME          * DISP * DOM *  SECTORS   * DEV T/# *  RECORDS  *   BLOCKS   *
11:15:36:7  FILE  S 1     CANDIDAT.HPUNSUP .SUPPORT    0      0      18         0  /2      2            3

                                FILE NAME          * DISP * DOM *  SECTORS   * DEV T/# *  RECORDS  *   BLOCKS   *
11:15:36:9  FILE  S 1     ERROR     .HPUNSUP .SUPPORT    0      0      8          0  /2      0            0

                                FILE NAME          * DISP * DOM *  SECTORS   * DEV T/# *  RECORDS  *   BLOCKS   *
11:15:35:1  FILE  S 1     GOOD      .HPUNSUP .SUPPORT    0      0      14         0  /2      1            2

                                FILE NAME          * DISP * DOM *  SECTORS   * DEV T/# *  RECORDS  *   BLOCKS   *
11:15:35:2  FILE  S 1     SYSLIST   .         .          0      1      0          16 /23     2            2

                                FILE NAME          * DISP * DOM *  SECTORS   * DEV T/# *  RECORDS  *   BLOCKS   *
11:15:42:5  FILE  S 1     LOAD      .PUB      .SYS        0      1      201        0  /1      2            1

                          PROG SEG * SL SEG * MAX STACK * MAX DS * VIRT ST *
11:15:46:6  PROC  S 1       1         0        3840       16       62

                                FILE NAME          * DISP * DOM *  SECTORS   * DEV T/# *  RECORDS  *   BLOCKS   *
11:15:46:7  FILE  S 1     SL        .PUB      .SYS        0      1      1792       0  /1      61           60

                                FILE NAME          * DISP * DOM *  SECTORS   * DEV T/# *  RECORDS  *   BLOCKS   *
11:15:46:8  FILE  S 1     EDITOR    .PUB      .SYS        0      1      233        0  /2      33           1

                                FILE NAME          * DISP * DOM *  SECTORS   * DEV T/# *  RECORDS  *   BLOCKS   *
11:15:47:5  FILE  S 1     EDITOH    .PUB      .SYS        0      1      233        0  /2      2            1

                                FILE NAME          * DISP * DOM *  SECTORS   * DEV T/# *  RECORDS  *   BLOCKS   *
11:16:24:7  FILE  S 1     VANP      .HPUNSUP .SUPPORT    0      1      322        0  /2      960          320

                                FILE NAME          * DISP * DOM *  SECTORS   * DEV T/# *  RECORDS  *   BLOCKS   *
11:17:7 :0  FILE  S 1     EDTLIST   .         .          0      0      0          32 /5      13           13

                                FILE NAME          * DISP * DOM *  SECTORS   * DEV T/# *  RECORDS  *   BLOCKS   *
11:17:30:1  FILE  S 1     EDTLIST   .         .          0      0      0          32 /5      18           18

                                FILE NAME          * DISP * DOM *  SECTORS   * DEV T/# *  RECORDS  *   BLOCKS   *
11:10:54:7  FILE  S 1     EDTLIST   .         .          0      0      0          32 /5      18           18
```

LISTLST

This utility gives the analyst loader segment table information. Depending upon which bit or combination of bit is set, the resulting printout is

1 - Loaded programs
2 - Directory
3 - Actual CST map

or any combination. The first part is important for finding out how may users are sharing the code for the loaded programs. The second part furnishes such information as the number of CSTs required for the loaded program. The third part is a list of all CSTs available to the system. The "A" means allocated; "C" means core resident; and "X" means system code. The code types are

SSL - System Segment Library
PSL - Public Segment Library
GSL - Group Segment Library
PROG - Program file

Output is on file FTNOG which defaults to $STDLIST. The command is :RUN LISTLST;PARM= where PARMs bit patterns are as follows:

(15:1) = 1 - Prints loaded programs
(14:1) = 1 - Prints Directory
(13-1) = 1 - Prints CST Table

The following example shows PARM = 7 which turns all three bits on.

```
:RUN LISTLST;PARM=7

LOADER SEGMENT TABLE        DATE: THU, JAN 24, 1974,  4:02 PM

********LOADED PROGRAMS

LOAD     .PUB      .SYS       , # USERS= 1
LISTLST  .HPUNSUP  .SUPPORT   , # USERS= 1
COBOL    .PUB      .SYS       , # USERS= 1

********DIRECTORY

 1) ENTRY TYPE: SL FILE
    SEGMENTS COPIED TO SYSTEM DISC: NO
    FILE NAME: SL       .PUB       .SYS
    # OF CST'S REFERENCED:      68

 2) ENTRY TYPE: PROGRAM FILE
    SEGMENTS COPIED TO SYSTEM DISC: NO
    LOAD MODE: NORMAL
    FILE NAME: LOAD     .PUB       .SYS
    # OF CST'S REFERENCED:       1
```

********ACTUAL CST MAP

ACTUAL CST #	LOGICAL CST #	FLAGS	CODE TYPE	REF	SOURCE FILE		
020	001	X	SSL	3	SL	.PUB	.SYS
021	002	X	SSL	3	SL	.PUB	.SYS
022	017	X	SSL	3	SL	.PUB	.SYS
023	023	X	SSL	3	SL	.PUB	.SYS
024	025	X	SSL	3	SL	.PUB	.SYS
025	031	X	SSL	3	SL	.PUB	.SYS
026	037	X	SSL	3	SL	.PUB	.SYS
027	041	X	SSL	3	SL	.PUB	.SYS

SLLIST

In order to list out the system SL, this utility is provided. It simply lists all the segments and corresponding segment numbers. There are many applications for this utility, not the least of which is to check whether or not a segment made it into the SL after a shakey SYSDUMP. The command is all that is needed. Output goes to the designator OUT which defaults to the line-printer.

```
 RUN SLLIST

SL FILE SL.PUB.SYS

                    41 DATASEG          102 JOBTABLE
  1 FILESYS2        42 S'LIB'01         103 RINS
  2 FILESYS1        43 CHECKER
  3 TRACE1'         44 UTILITY          105 CLIB'01
  4 TRACE0'         45 SEGUTIL          106 LOADER1
  5 COBLIB15        46 MMDISKR          107 PROCMAIL
  6 COBLIB14        47 FILESYS7         110 SDMCOMM
  7 COBLIB10        50 MERGSEG2
 10 COBLIB09        51 DEBUG
 11 COBLIB05        52 SYSDEBUG
 12 COBLIB03        53 SYSDSPLY

                                        117 CISUBS
                                        120 CIORGMAN
 17 MORGUE          60 FILESYS6A        121 CIINIT
                    61 FILESYS6         122 CIFILEM
                    62 FILESYS5         123 CIFILEB
                    63 FILESYS4         124 CIERR
 23 ABORTRAP                            125 CILISTF
                    65 IOPM             126 CIMISC
 25 DISKSPC         66 TTYINT           127 STORE
 26 COBLIB02        67 IOUTILTY         130 RESTORE
                    70 CLIB'09          131 CXSTOREST
                    71 CLIB'08          132 MMCORER
 31 MESSAGE         72 CLIB'07
 32 COBLIB01        73 CLIB'06          134 CROUTINE
                    74 CLIB'05
 34 S'LIB'04        75 CLIB'04          136 PCREATE
                    76 CLIB'03          137 PINT
 36 S'LIB'03        77 DIRC             140 SORTSEG2
 37 FILESYS3       100 ALLOCATE         141 FIRMWARESIM
 40 S'LIB'02       101 CLIB'02

END OF PROGRAM
```

TAPECOPY

One of the hardest working utilities is TAPECOPY. This program copies and verifies up to six copies of a STORE, Cold Load, or SPL readable tape. From a terminal a user may quickly copy a tape, verify that it is good and terminate or make multi-copies.

```
:RUN TAPECOPY

HP 3000 TAPE COPY AND VERIFY PROGRAM
VERSION 2.0
ENTER FORMAT(STORE/COLD LOAD/SINGLE FILE)
```

Entering the chosen format, the console will request a tape unit for the master and then a unit for the copy. A message then appears, if all is well (good tape, proper density, etc.), that says the copy is in progress. At the end the tape rewinds and the verification process begins. Upon completion, a message is issued whether the copy was good or bad; and if it was good, do you wish to copy another?

THE "BE SURE YOU KNOW WHAT YOU'RE DOING" ONES

DISKEDIT

This is another utility which allows you to modify the code on disk directly, according to absolute sector addresses. Parameters are free format.

Commands:

>LIST ldn

Specifies the device to which >DUMP listing is to go

Defaults:

no parameter - job list device

Initially: job list device.

>DISC ldn

Specifies the disc.

Initially: 1

>MODIFY sectornum, relwdaddr [,numwds]

<sectornum>	-	the absolute sector address (decimal or octal)
<relwdaddr>	-	the word address of the first word to be modified, relative to word Ø of <sectornum>. This may span sectors.
<numwds>	-	the number of words to modify. This may span sectors. Default: 1.

The command returns with the sector number being modified (octal), and for each word to modified:

aaa: dddddd, %

<aaa> is the sector relative word displacement (octal)

<dddddd> is the contents of this word (octal)

The user then responds with

<newval> - new octal replacement, or

* - leave alone or

/ - abort MODIFY operation

The message, "WRITTEN" is printed when each sector is physically written. Before this, the sector has not been altered.

> EXIT

Just for a visual example the following provides a dialogue. The prompt is given and the user must enter the desired command.

```
 RUN DISKEDIT

DISC EDIT/DUMP
<DISC 1
<MODIFY 5,24
SECTOR %00000000005
030:  041414,%*
WRITTEN
<EXIT

 END OF PROGRAM
:
```

GIVEFILE

With MPE Version B, the user is not allowed to rename files across accounts boundaries. This utility gives the user the ability to move certain files from one account to another. Once the file has been moved, the entry disappears from the account giving up the file. The user must be running GIVEFILE from the account to which he wishesto bring the file. Otherwise, he will get:

```
DIRECINSERTFILE ERROR
```

If the file is not qualified properly or does not exist, the following appears:

```
FILE NOT FOUND
```

Certain files are protected as is FORTRAN in the following example. Note J00J222A in the SUPPORT account, HP32222 group, is being moved to group HPUNSUP and renamed to JOB.

```
:RUN GIVEFILE

OLD FILENAME? FORTRAN.MANAGER.SYS
FILE NOT FOUND
OLD FILENAME? FORTRAN.PUB.SYS
UNABLE TO OPEN FILE EXCLUSIVELY
OLD FILENAME? M00M102A.HP32102.SUPPORT
FILE NOT FOUND
OLD FILENAME? J00J222A.HP32222.SUPPORT
CREATOR = FIELD    ? YES
NEW FILENAME? JOB
OLD FILENAME?

 END OF PROGRAM
```

MEMMAP

This program produces a snapshot of memory on specified second intervals. The matrix produced by the memory scan is recursive and when studied as a whole, it can show such things as

1. the dynamic use of memory and the lack of fragmentation.

2. the system utilizes core whenever the user processes drop.

3. the clearing of memory for large data stacks can be noted by running the program at more than one terminal in different synchronisms and with different stack sizes.

Output defaults to $STDLIST with the PRINT instrinsic. The following is an example of MEMMAP run with a 15 second interval which must be specified by the user.

```
:RUN MEMMAP

INTERVAL IN SECS 15

HP-3000 CORE MAPPER   FRI, JUN 29, 1973,  8:46 AM

KWORDS +                 1                   2                   3
0       NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
4       NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
8       NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
12      NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
16      NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
20      UUUUUUUUUUUUUUUUSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSU
24      SSSSSSDUUUUUUUUUUUUUUUU.SSSSSSSSSSSSSSSSSSSSSDDDDDDDDDDDDDSSSSS
28      DDDDDD.SSSSSSSSSSSSSSSSSSSSSSS.....SSSSSSSSSSSSS
32      SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS..SSSSSSSSSSSSSSSDDDDDD
36      DSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS.SSSSSSSSSSSSSSS
40      SSSSSSSSSSSSSSSSSSSSSSSSSSUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU..
44      .....EEEEEEEEEEEEEEEEEEEEDDDDDDDDDDDDDDDDDDDDDDDDDSSSSSS....
48      ............SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS.......SSSSS
52      SSSSSSSSSSSSSS.............DDDDDDDDDDDDDSSSSSSSSSSSSSSSSSSSSS
56      SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS.......SSSSSSSSSSSSS
60      SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS.........

LEGEND:-  S - SYSTEM CODE          #22     N - NON LINKED MEMORY
          U - USER CODE            #3      . - FREE MEMORY
          D - ANY DATA STACK       #7
          E - ANY EXTRA DATA SEG   #1      64 WORD RESOLUTION
```

```
HP-3000 CORE MAPPER   FRI, JUN 29, 1973,  8:46 AM

KWORDS +              1                   2                   3
0       NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
4       NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
8       NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
12      NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
16      NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
20      UUUUUUUUUUUUUUSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
24      SSSSSSS.....SSSSSSSSSSSSSSSSSSSSSDDDDDDDDDSSSSSSSSS
28      SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
32      SSSSS.UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUSSSSSSSSSSSSSSSS.....
36      .SSSSSSSSSSSSSSSSSSSSSSSS.SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
40      SSSSSSSSSSSSSSSSSSSSSSSSS.....DDDDDDDDDUUUUUUUUUUUUUUUUDDDDDDD
44      DDDDDUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUSSSSSSSSSSSSSSSSSSSSSS
48      SSDDDDDDDDDDDDDDDDDDDDDDDDUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUSSSS
52      SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
56      SSSSSSSSSSSSSSSSSSS.DDDDDDDDDDDDDDDDDDDSSSSSSSSSSSSSSSSSSSSS
60      SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS.........

LEGEND:-  S - SYSTEM CODE          #22    N - NON LINKED MEMORY
          U - USER CODE            #5     . - FREE MEMORY
          D - ANY DATA STACK       #5
          E - ANY EXTRA DATA SEG   #0     64 WORD RESOLUTION

HP-3000 CORE MAPPER   FRI, JUN 29, 1973,  8:46 AM

KWORDS +              1                   2                   3
0       NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
4       NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
8       NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
12      NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
16      NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
20      UUUUUUSSSSSSSSSSSSSSSSSSSSSSSSSS.....DDDDDDDDDDDDSSSSS
24      SSSSSSSSSSSSSSSSSSSSSSSSS.SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
28      SSSSSUUUUUUUUUUUUUUUUDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD.....
32      .....SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS..
36      ..........SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
40      SSSSSSSSS.....SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
44      SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS.....
48      .UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU.....DDDDDDDDDDDDDDDDDDDDD
52      DDDSSSSSSSDDDDDDDDDSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS.....SSSSSSSSSSS
56      SSSSSSSSSSSSSSSSSS.....DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDU
60      UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUDDDDDSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS

LEGEND:-  S - SYSTEM CODE          #16    N - NON LINKED MEMORY
          U - USER CODE            #4     . - FREE MEMORY
          D - ANY DATA STACK       #7
          E - ANY EXTRA DATA SEG   #0     64 WORD RESOLUTION
```

PATCH

For the analyst who thinks he has found the solution to a bug and wishes to try a patch to a lame program file, we have PATCH. This utility allows both displaying and modifying of the contents of program files on disk. The following is an example of a dump of the file FREE.

```
:RUN PATCH
  FILE=?FREE
?D,0,0,1
  024410
?D,0,0,5
  024410
  013302
  027410
  041000
  021013
?

 END OF PROGRAM
```

The prompt is "?". The user may then give one of the following directives,

D,segment number, address, number of locations.
M,segment number, address, number of locations.

inorder to dump or modify a code segment. If changes are to be made to the global area use:

M
D , D, address, number of locations.

As shown above, the contents of the specified addresses are displayed. If you are modifying, the change will be echoed to insure the correctness of the change.

DEBUG AND TRACE - ALAN HEWER, DOUG JEUNG

DEBUG

- DEBUG IS A CALLABLE INTRINSIC WHICH ALLOWS NON-PRIV USERS TO DYNAMICALLY CHECKOUT A PROGRAM WHEN RUNNING IN A SESSION.

- IT IS ORIENTED TOWARD USERS WHO ARE KNOWLEDGEABLE ABOUT THE CODE AND DATA STRUCTURES WHICH CONSTITUTE THE EXECUTING PROGRAM.

- SIMPLE INTERACTIVE COMMANDS ALLOW USERS TO
  - DISPLAY / MODIFY REGISTER CONTENTS
  - DISPLAY / MODIFY DATA IN THE STACK
  - SET / RESET CODE BREAKPOINTS

- PROTECTION IS GUARANTEED.

INVOCATION

DEBUG IS ENTERED BY THE FOLLOWING MEANS

- DIRECT EXTERNAL CALL TO THE INTRINSIC DEBUG WITH EXTERNAL DECLARATION :

  PROCEDURE DEBUG ; OPTION EXTERNAL ;

- VIA AN EXISTING CODE BREAKPOINT.

NOTE THAT IT IS NECESSARY TO COMPILE ONLY ONE DIRECT CALL TO DEBUG WITHIN THE PROGRAM. CONTROL MAY BE TRANSFERRED TO DEBUG AT ALL OTHER LOCATIONS WITHIN THE PROGRAM UTILISING THE BREAKPOINT FACILITY.

ENTRY

A WELCOME MESSAGE IS OUTPUT OF THE FORM:

{DEBUG / BREAK} . <offset>

DEBUG - INDICATES A DIRECT CALL TO DEBUG

BREAK - INDICATES A BREAKPOINT ENTRY

 - LOGICAL PROGRAM SEGMENT

<offset> - LOCATION OF THE INSTRUCTION, IN ABOVE SEGMENT, TO BE EXECUTED UPON RETURN. IF 'BREAK', THE INSTRUCTION BREAKPOINTED IS NOT EXECUTED PRIOR TO ENTRY TO DEBUG.

PROMPT CHARACTER "?"

COMMAND – SINGLE LETTER COMMAND MNEMONIC
– PARAMETERS
– BLANKS IGNORED
– TERMINATED WITH CARRIAGE RETURN

ERROR MESSAGES

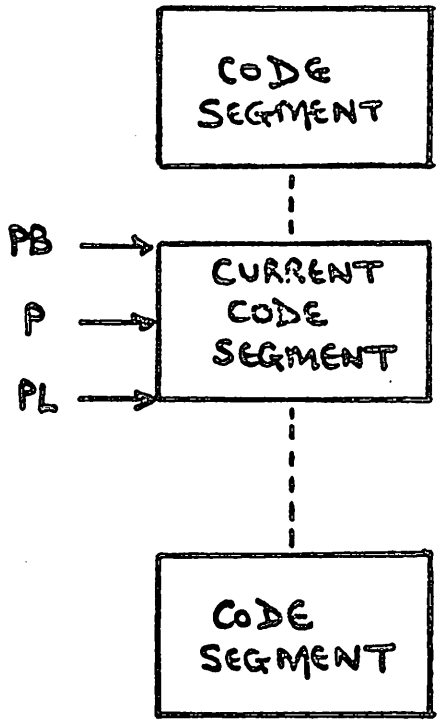
"NONO n" – ILLEGAL SYNTAX OR INALID INFORMATION

"XTRA n" – TOO MUCH INPUT ON A MODIFY MEMORY

"FULL n" – BREAKPOINT TABLE FULL

n = CHAR POSITION AT WHICH ERROR DETECTED

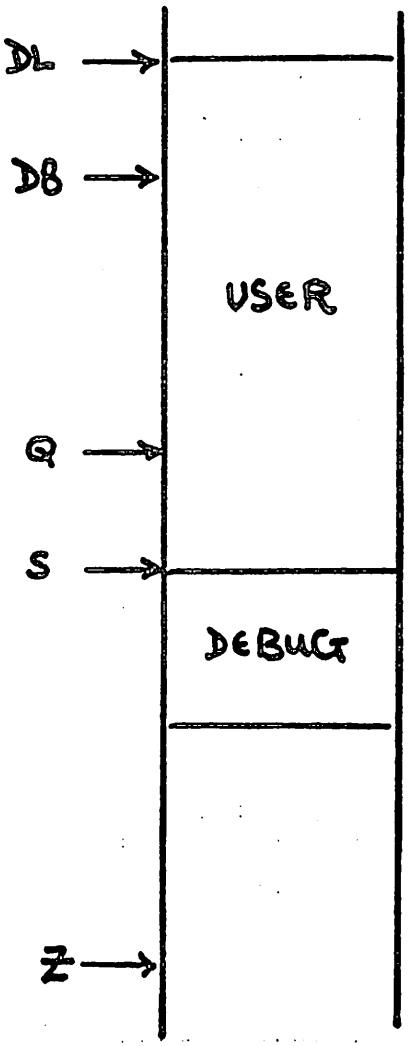
PROGRAM



STATUS

X

STACK



COMMAND SYNTAX

number	::=	octaldigit	{ OCTAL ONLY
expr	::=	[+|-] number [{+|-} number]	
offset	::=	expr	{ (P-PB) ADDRESS
segment	::=	expr	
location	::=	[segment .] offset	{ DEFAULT SEGMENT IS CURRENT
reg	::=	Q | S | X | ST | P | Z | DL	
count	::=	expr	
stackadr	::=	[Q|S|DL] expr [I [expr]]	{ I = INDIRECTION DEFAULT BASE = DB
datalist	::=	[expr] [, [expr]] ..... [.]	{ . = LIST TERMINATOR
filename	::=	anydigit	

COMMANDS

D	[(filename)] [reg]	DISPLAY REGISTERS
M	reg = expr [, reg = expr]	MODIFY REGISTERS
D	[(filename)] stackadr [, count]	DISPLAY MEMORY (STACK)
M	stackadr [, count] [= datalist]	MODIFY MEMORY (")
T		TRACE STACK MARKERS
B	location [, location]	SET BREAKPOINTS
C	[location [, location] ...]	CLEAR BREAKPOINTS
R	[location]	RESUME

DISPLAY REGISTERS D [(filename)] [reg].....

D	DISPLAY ALL REGISTERS
D S X Z	DISPLAY S, X, Z REGISTERS
D (L) ST	DISPLAY STATUS REG ON FILE L

IF A FILE IS SPECIFIED, DEV=LP IS TAKEN. THIS CAN BE OVERRIDDEN BY A :FILE COMMAND

MODIFY REGISTERS M reg = expr [, reg = expr].....

M X = 4	SET X TO 4
M Q = 50, S = 160	SET Q TO 50, S TO 160
M P = 3.66	(SPECIAL CASE) SET P TO LOCATION 66 OF LOGICAL SEGMENT 3.

NOTE THAT DL <= Ø <= Q <= S <= Z.

ONLY BITS 2 THRU 7 OF STATUS MAY BE CHANGED.

DISPLAY MEMORY

D [(filename)] stackadr [, count]

D 25 — DISPLAY LOCATION DB+25

D Q+6I, 5 — DISPLAY 5 WORDS POINTED TO BY LOCATION Q+6

D (LP) Q+6I,5 — SAME AS ABOVE, BUT ON FILE LP

MODIFY MEMORY

M stackadr [, count] [= datalist]

M S-2 = 555 — CONTENTS OF LOCATION S-2 SET TO 555

M DL+5I, 20 — MODIFY 20 WORDS OF LOCATIONS POINTED TO BY DL+5. ASSUME THIS ADDRESS IS DB+24

THEN: DEBUG OUTPUTS THE ADDRESS PLUS CONTENTS, THE USER INSERTS CONSECUTIVE NEW CONTENTS

+000024 = 4, 1, 2, , , 5

+000031 = ø, , , 10 .

A BLANK LEAVES UNCHANGED AS IN LOCATIONS 26, 27, 31, 32 ABOVE. UP TO 20 WORDS WERE SPECIFIED, BUT A PERIOD "." ACTS AS A TERMINATOR

TRACE STACK T

T PRINTS A TRACE OF USER STACK MARKERS SHOWING LOGICAL SEGMENT # AND P LOCATION.

7.50 (CURRENT LOCATION)

4.65 (LOC FROM WHICH 7.50 WAS CALLED)

0.10 (LOC " " 4.65 " ")

0.24 (LOC " " 0.10 " " . THIS IS THE LAST STACK MARKER FOUND AND SHOULD BE IN OUTER BLOCK OF PROGRAM)

SET BREAKPOINT

B location [, location] .....

B 24 — SET BP AT LOC 24 OF CURRENT SEGMENT

B 3.10, 16 — SET BP's AT LOC 10 OF SEG 3, AND 16 IN CURRENT SEGMENT.

CLEAR BREAKPOINT

C [location [, location] ....]

C 24, 3.10 — CLEAR BREAKPOINTS ORIGINALLY SET IN LOC 24 OF CURRENT SEG, 10 IN SEG 3.

C — CLEAR ALL BREAKPOINTS

RESUME

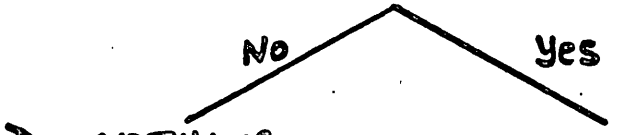
R [location]

R — RETURN TO USER PROGRAM AT EITHER INSTRUCTION FOLLOWING DIRECT CALL TO DEBUG OR INSTRUCTION WHICH GENERATED BREAK.

R 5.33 — SET BREAKPOINT AT 5.33 AND THEN RETURN.

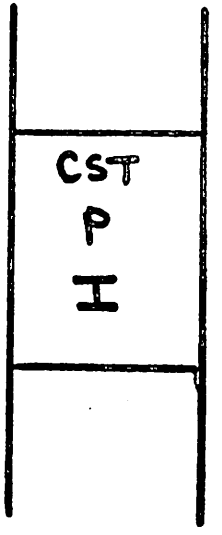
BREAKPOINT MECHANISM

SET BP AT LOCATION LCST. P. ASSUME VALID.
CONVERTS LCST (logical) TO CST (physical).
ENTER CST AND P IN BREAKPOINT TABLE
IS SEGMENT PRESENT IN MEMORY?

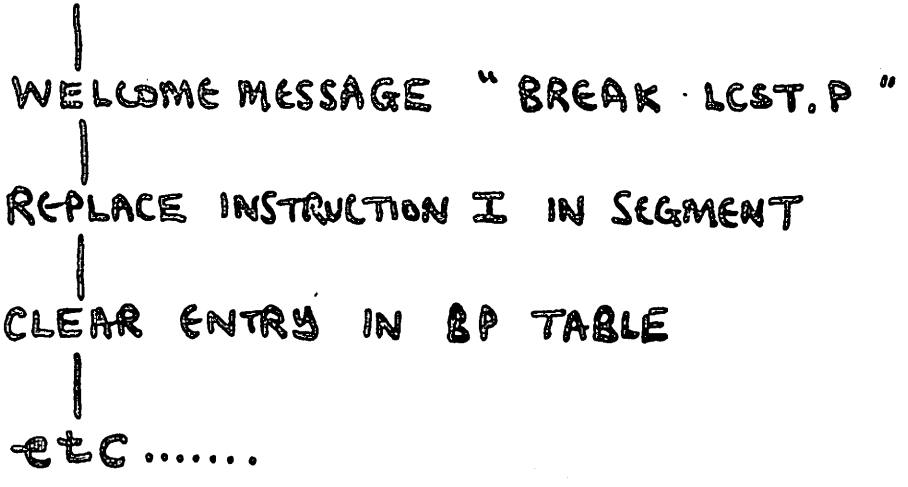


WHENEVER A SEGMENT IS BROUGHT INTO MEMORY
IT IS CHECKED FOR A POTENTIAL BREAKPOINT.
IF SO, THE INSTRUCTION I IS STORED IN BP TABLE
AND REPLACED WITH %30000.
THUS, WHENEVER A SEGMENT IS PRESENT IN MEMORY,
IT CONTAINS ALL BREAKPOINTS WHICH ARE OUTSTANDING.

BREAKPOINT TABLE



DURING EXECUTION OF THE SEGMENT. HITTING THE
%30000 PROVOKES AN ILLEGAL INSTRUCTION TRAP
(INTERNAL INTERRUPT 15).
CHECK BP TABLE FOR VALID BREAKPOINT.
IF NOT... ABORT USER
IF YES ... CALL DEBUG.



TRACE

- SYMBOL ORIENTED DEBUGGING TOOL
  - AVAILABLE WITH FORTRAN AND SPL
  - REQUIRES COMPILE TIME COMMANDS TO SET UP TABLE OF IDENTIFIERS TO BE TRACED
- DOES NOT REQUIRE USER TO BE KNOWLEDGEABLE ABOUT CODE AND DATA STRUCTURES
- SIMPLE INTERACTIVE COMMANDS ALLOW USERS TO
  - DISPLAY / MODIFY DATA
  - LIST SELECTIVELY THE CONTENTS OF VARIABLES WHICH HAVE BEEN MODIFIED
  - RETURN CONTROL TO USER AT SELECTED POINTS

USER ACTION AT COMPILE TIME

- USER MUST PRECEDE SOURCE DATA TO COMPILER WITH

  $ TRACE <program unit> ; <identifier list>

  <program unit> IS NAME OF MAIN PROGRAM OR SUBROUTINE (FORTRAN), PROCEDURE (SPL)

  <identifier list> NAMES OF

  VARIABLES

  ARRAYS

  LABELS

  ROUTINES

- ACTION BY COMPILER AS A RESULT OF ABOVE
  - BUILD (COMPILE TIME) A TABLE WHICH RESIDES BEHIND THE USER'S SECONDARY DB AT RUN TIME
  - GENERATE CALLS TO LIBRARY ROUTINE TRACE1'

CALLING SEQUENCES

CONDITION CODE					
PUST OF WHICH THIS ENTRY IS A PART					
I'	X'	TS	M	S	LOCATION OF ENTRY IN PUST

I' = 1 DIRECT

X' = 1 NOT INDEXED

TS = 11 ENTERING A TRACED ROUTINE

00 CALLING A TRACED ROUTINE

01 RETURNING FROM A TRACED ROUTINE

10 EXITING FROM A TRACED ROUTINE

M = 1 USE TS (ROUTINE)

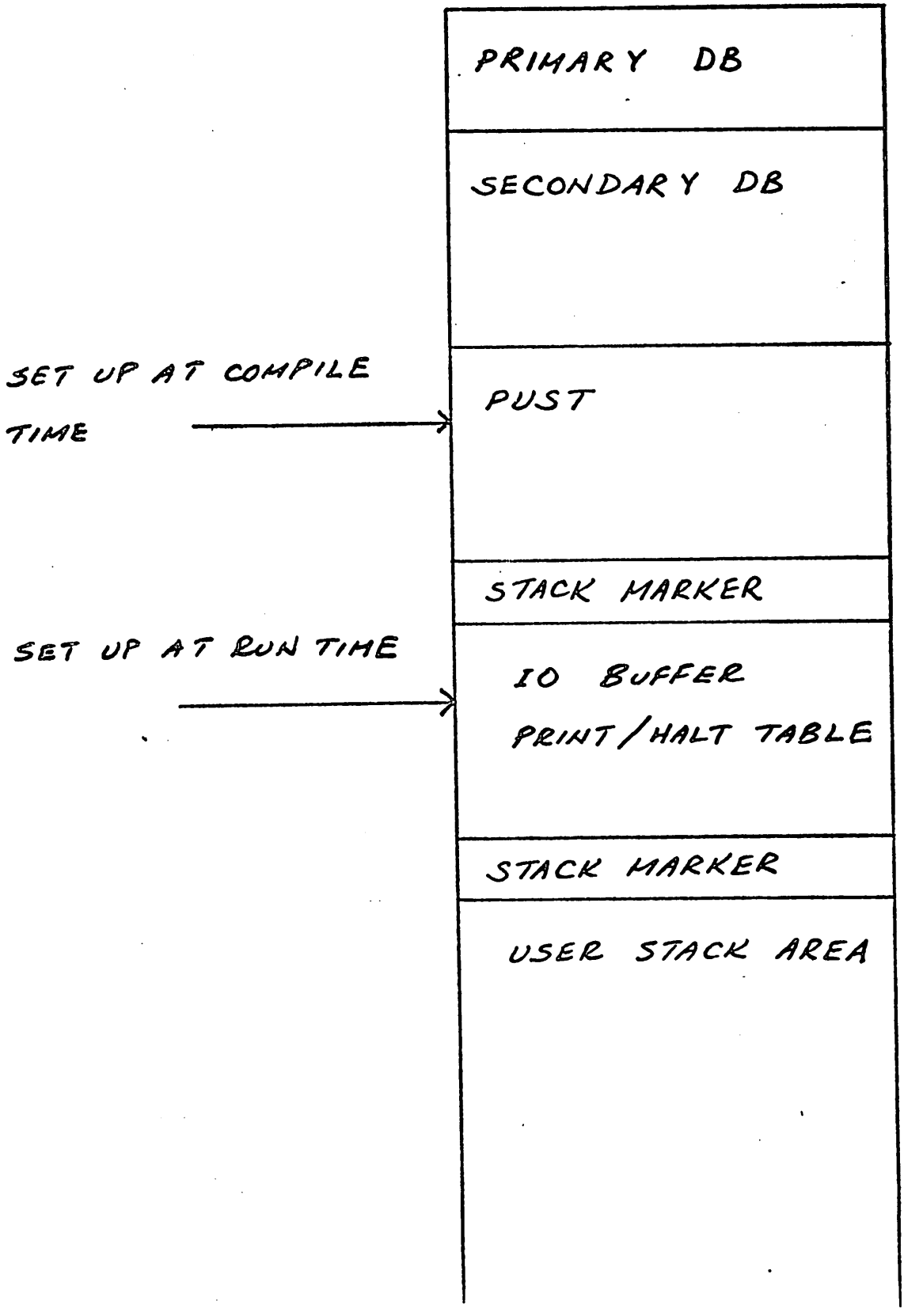
0 VARIABLE, ARRAY OR LABEL

S = 1 SPL

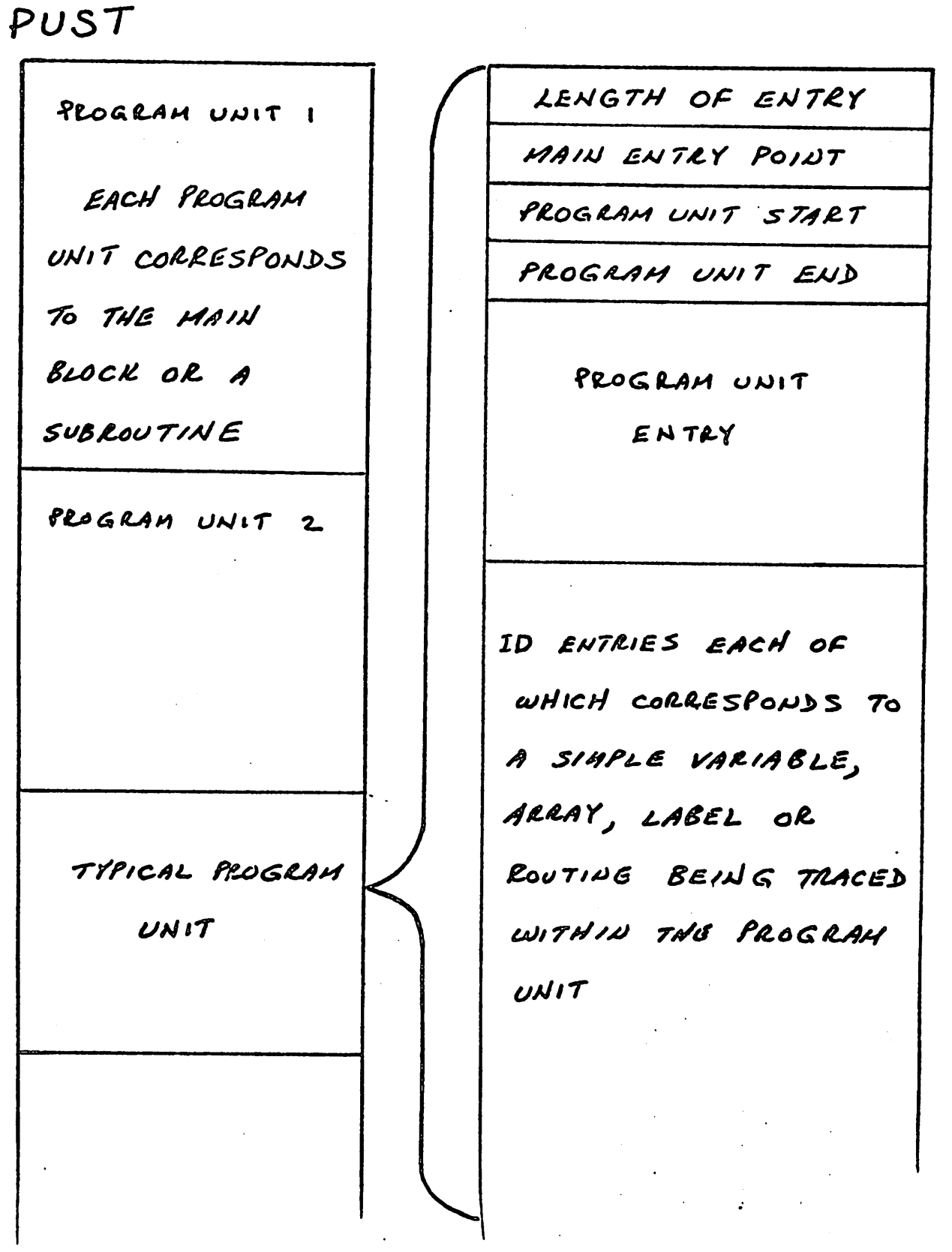
0 FORTRAN

(4)

USER DATA AREA



PUST TABLE ENTRY



FORMAT OF ID ENTRIES

CLASS		TYPE	ENTRY LENGTH
# OF CHARACTERS		CHAR 1	
CHAR 2		CHAR 3	
CHAR 4		CHAR 5	
CHAR 6		ETC	

CLASS	=	00001	SIMPLE VARIABLE
		00010	ARRAY
		10000	LABEL
		01000	SUBROUTINE
		01100	FUNCTION ROUTINE
TYPE	=	0000	LOGICAL
		0001	INTEGER
		0010	REAL
		0011	DOUBLE
		0100	STRING
		0101	LONG
		0110	COMPLEX

FORMAT OF ID ENTRIES (CONT.)

LABEL :

SIMPLE VAR :

ARRAY :

	MODE	I	X	DISPLACEMENT

MODE = 000 DB +
010 Q +
011 Q −
101 S −

ROUTINE :

etc.	P_3
P_2	P_1
#PARAMETERS	#WORDS IN STACK

P_i | L | M | TYPE |

L = 00 USE M AND T
01 SPL LABEL PASSED AS PARAMETER
FORTRAN POINTER TO ARRAY

M = 00 VALUE
01 REFERENCE

USER ACTION AT RUN TIME

- COMMANDS WHICH MAY BE USED
  - PRINT
  - HALT
  - GO
  - SET
  - DROP

- IDENTIFIERS WHICH MAY BE TRACED
  - SIMPLE VARIABLE
  - ARRAY
  - LABEL
  - ROUTINE

GO COMMAND

PROMPT CHARACTER "*"

USER TYPES IN

GO — EXECUTION CONTINUES

GO <label name>

EXECUTION CONTINUES FROM LABEL LOCATION WITHIN PROGRAM UNIT

PRINT COMMAND (HALT)

USER TYPES IN

PRINT <program unit>
<identifier> [<conditions>]
<identifier> [<conditions>]
- - -
- - - -
BLANK RECORD

PROMPT CHARACTER IS AGAIN EMITTED BY TRACE UNTIL A "GO" COMMAND IS ENTERED

<program unit> NAME OF MAIN PROGRAM OR ROUTINE
<identifier> VARIABLE, ARRAY, LABEL OR ROUTINE

PRINT RESULTS IN IDENTIFIER AND VALUE IF ANY TO BE PRINTED

HALT SAME AS PRINT AND IN ADDITION CONTROL RETURNED TO USER

- CONDITIONS C1, C2, C3 AND C4
  - NOT REQUIRED
  - IF PRESENT MUST BE IN ORDER
  - C1 SUBSCRIPT FOR ARRAYS

    ARR (* =)

    ↑ INTEGER CONSTANT OR INTEGER SIMPLE VARIABLE

  - C2 VALUE

    I >

    ↑ CONSTANT OR VARIABLE OF SAME TYPE AS "I".

    ARR (*:24) <=

  - C3 BOUNDS

    I L1 - L2 MUST BE LABELS IN PROGRAM UNIT

  - C4 COUNT

    I @5 EVERY FIFTH OCCURRENCE

SET COMMAND

TRACE OUTPUT IS UNDERLINED IN FOLLOWING EXAMPLE

* SET PROG1 cr	cr = CARRIAGE RETURN
BOB = 31 cr	PRINTS VALUE
ARR (42) = 22 / 33 cr	PRINTS VALUE WHICH MAY BE CHANGED
ARR (42), 3 = 33 14 5	BLOCK OUTPUT
DB + 5 = 31	REGISTER REFERENCE ALLOWED

DROP COMMAND

DROP <program unit>
<identifier>
<identifier>
- - - -
- - - -
BLANK RECORD

SELECTIVELY REMOVES ENTRIES FROM PRINT/HALT TABLE

DROP <program unit>
$ALL

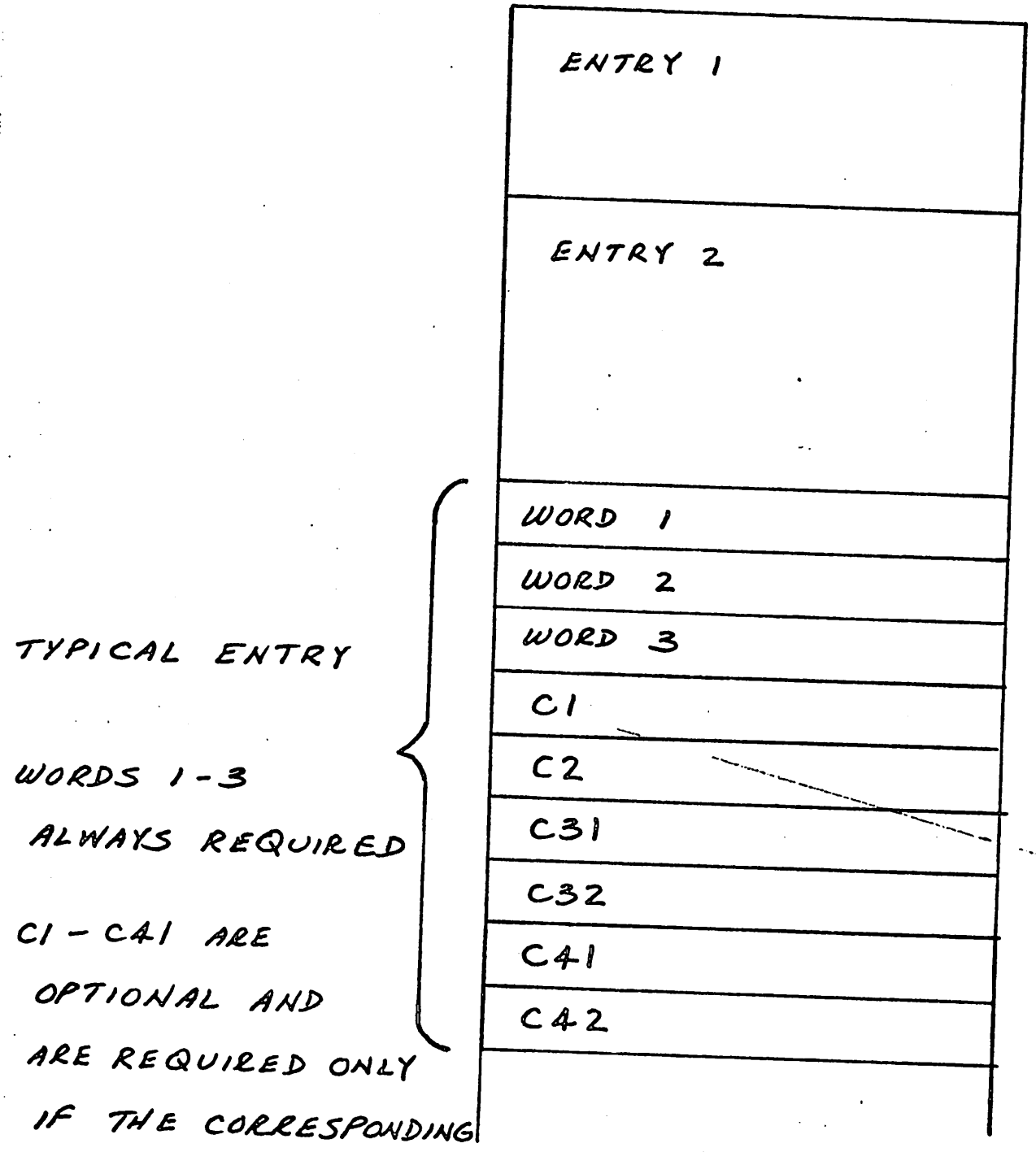
REMOVES ALL ENTRIES BELONGING TO PROGRAM UNIT

DROPALL

REMOVES ALL ENTRIES

(14)

PRINT / HALT TABLE



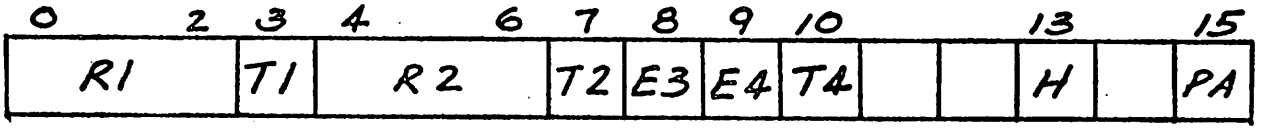
(4)

PRINT / HALT TABLE ENTRY

WORD 1 LENGTH OF ENTRY

WORD 2 POINTER TO BASE OF PUST

WORD 3



CONDITION 1 R1 = 000 NO CONDITION 1
001 <
010 =
011 <=
100 >
101 <>
110 >=

T1 = 1 C1 IS ADDRESS
0 C1 IS A CONSTANT

CONDITION 2 R2, T2 CORRESPOND TO R1, T1

CONDITION 3 E3 = 1 CONDITION 3 EXISTS

C31 LOWER BOUND
C32 UPPER BOUND

CONDITION 4 E4 = 1 CONDITION 4 EXISTS

T4 = 1 C42 IS ADDRESS
0 C42 IS A CONSTANT

C41 COUNTER

H = 1 HALT ENTRY OTHERWISE PRINT ENTRY

PA = 1 FOR PROCEDURES ONLY, PRINTS PARAMETERS

CONTROL Y

- RETURNS CONTROL TO USER EVEN IF NO "HALT" HAS BEEN ENTERED IN TABLE.
  - IF EXECUTING IN A TRACED PROGRAM UNIT THEN A MESSAGE INFORMING USER OF THE PROGRAM UNIT IS EMITTED, FOLLOWED BY THE PROMPT CHARACTER "*"
  - IF NOT IN A TRACED PROGRAM UNIT THEN CONTROL IS RETURNED TO USER AS SOON AS A TRACED PROGRAM UNIT IS ENTERED.

HP 3000

USER CONTRIBUTED DOCUMENTATION

ANDERSON COLLEGE

PROGRAM LOGANA sourcefile=LOGASRC

Function: This program summarizes a range of log files into one new file compatible with program LOGRPT. The log files analyzed are purged by this program.

Program documentation: For detail information on files, messages, and errors, see source listing lines 3/56.

Run instructions:

1. This program may be run either as job or session.

2. User must have system manager capability.

3. :RUN LOGANA.PUB.SYS

4. FILE NOT FOUND
   On beginning execution the program purges SORTLOG. This message appears if SORTLOG does not exist. No response is required.

5. ENTER BEGINNING LOG NBR? XXXX
   Enter 4 digit number of first log file in range. Example: if LOGØ123 is the first in the range, enter Ø123. (CR not required.)

6. ENTER ENDING LOG NBR? XXXX
   Enter 4 digit number of last log file in range.
   NOTE: The last log number must be greater than the first and all log files in the range must exist. If a log file within the range is missing, a dummy may be created with the command :BUILD LOGXXXX;REC=-5Ø8,1,V,BINARY;DISC=1,1,1.

7. The log files within the range are now summarized into SORTLOG and purged. A maximum of 20 jobs may be open at one time in the log files. Job records are not carried over between log files. The summary records in SORTLOG are now arranged in chronological order and contain the following:

   A. Time stamp-on
   B. Job type and number
   C. User name
   D. Job name
   E. Group name
   F. CPU time (seconds)
   G. Connect time (minutes)
   H. Records processed (all files except printer (LDEV=6) and terminals (LDEV>1Ø)).
   I. Lines printed (records processed LDEV=6).
   J. Time stamp-off

8. END OF PROGRAM

9. To prepare the file SORTLOG for use by program LOGRPT, SORT must be run as follows:

```
:FILE INPUT=SORTLOG
:FILE OUTPUT=SORTLOG
:RUN SORT;STACK=1ØØØØ
>KEY AT 16 IS 8 LONG
>KEY AT 32 IS 8 LONG
>KEY AT  8 IS 8 LONG
>KEY AT  Ø IS 6 LONG
>:EOD
END OF PROGRAM
```

Anderson College
Computing Center

Jan. 1974

PROGRAM LOGRPT sourcefile=LOGRSRC

Function: This program produces a report of usage for all jobs, users, groups, and accounts. Input is from the file SORTLOG as prepared by programs LOGANA and SORT plus a rate file RATES. The output is a printed report which shows the following information for each job:

A. Job name
B. Job number
C. Date-on
D. Time-on
E. Time-off
G. Connect time and connect time charge
H. CPU time and CPU time charge
I. Records processed and record processing charge
J. Lines printed and printer charge
K. Total job charge.

The output is ordered by user, group, and account and totals are given at each level plus grand totals.

Program Documentation: For detail information on files (including file RATES), messages, and errors, see source listing lines 3/55, 79/95. Note: This program requires that procedure DOLLAROUT be in SL.PUB.SYS as an intrinsic.

Run instructions:

1. This program may be run either as job or session.
2. User must have system manager capability.
3. :RUN LOGRPT
   The report is now printed on the line printer.
4. END OF PROGRAM

The file SORTLOG remains intact and LOGRPT may be rerun.

Anderson College
Computing Center

Jan. 1974

Program FOOTBALL

Basic program file = FOOTBALL
Basic data file = DATA1
Basic data file = DATA2

Function: This program simulates a regulation football game using two terminals. Each team's plays (offensive and defensive) are called by a "quarterback" at a separate terminal. The files DATA1 and DATA2 contain probabilities for each kind of play (DATA1) and defensive skew factor (DATA2).

Run Instructions:

1. The I/O table (MPE) must be modified to allow terminals to accept data (JAID).

2. Determine the LDEV # for the "visitors'" terminal. This may be done by logging on and running WHO.PUB.SYS.

3. Enter a :DATA command* on the "visitors'" terminal.

4. Log on* the "home" terminal and determine its LDEV #.

5. Enter these file commands on the "home" terminal:
   :FILE TEAM1;DEV=("home" terminal LDEV #)
   :FILE TEAM2;DEV=("visitors'"terminal LDEV #)

6. Run the program as follows:
   :BASIC
   >RUN FOOTBALL
   .
   .
   .

7. The "visitors'" terminal will be released when

   a. FOOTBALL ends, or
   b. The home terminal hangs up, or
   c. The visitors' terminal enters :EOD.

8. All rules, options, etc are explained as the program runs.

* The DATA command in Step 3 must use exactly the same job identification as the user in Step 4.

Example

Home Terminal

```
:HELLO PRO.GAMES
:FILE TEAM1;DEV=14
:FILE TEAM2;DEV=15
:BASIC
>RUN FOOTBALL
      .
      .
      .
```

Visitors' Terminal

```
:DATA PRO.GAMES

      .
      .
      .
```

Anderson College
Computing Center
Jan. 1974

PROGRAM WHO sourcefile = WHOSRC

Function: This program, when executed, prints a formatted dump of the information available from the intrinsic WHO. The output is to $STDLIST; a sample is shown below.

```
:RUN WHO.PUB.SYS

YOUR CURRENT ATTRIBUTES ARE AS FOLLOWS:
USER = FIELD
HOME GRP = PUB
LOGON GRP = USERS
ACCT = SUPPORT
INTERACTIVE, DUPLICATIVE, SESSION

USER ATTRIBUTES: GL, AL, AM,
FILE ACCESS:  SF, ND
CAPABILITY CLASS: IA, BA
LOGICAL DEV # = 13


END OF PROGRAM
```

Anderson College
Computing Center

Jan. 1974

PROCEDURE `DOLLARIN(word,string,length);`

```
VALUE length;
INTEGER length;
DOUBLE word;
BYTE ARRAY string;                          sourcefile=SLPUBSRC
```

Function: This procedure converts an ASCII string containing a dollar amount into a double word representing the corresponding number of cents. Dollar signs ($), commas (,), blanks ( ), and plus signs (+) in the ASCII string are ignored. A minus sign (-) or the letters (CR) anywhere in the string will cause the returned value to be negative. Any other non-numeric characters except the decimal point (.) will cause an error condition.

word	The double word into which the converted value is placed.
string	The byte array containing the ASCII string to be converted.
length	The length of the byte array (cannot be negative).

The DOLLARIN command can result in the following condition codes:

CCE	Successful conversion.
CCG	Illegal character found in string.
CCL	(This condition code is not returned.)

Anderson College
Computing Center

Jan. 1974

PROCEDURE DOLLAROUT(word,string,length,type);

```
VALUE word,length,type;
DOUBLE word;
INTEGER length,type;
BYTE ARRAY string;                          sourcefile=SLPUBSRC
```

Function: This procedure converts a double word into an ASCII string with the two low order digits to the right of the decimal point. A dollar sign ($) and/or commas are optionally included in the string. The double word is assumed to represent an integer number of cents, not dollars.

word	The double word to be converted to ASCII code. Its value is interpreted as an integer number of cents, and may be positive or negative.
string	The byte array into which the converted value is placed. The result is right justified.
length	The length of the byte array. (Note that 4≤length ≤132.)
type	An integer identifying the kind of conversion desired (see HP 3000 Compiler Library: INEXT'):

type<Ø	-2 PFw.2 format,example:	1234.56
type=Ø	-2 PNw.2 format,example:	1,234.56
type>Ø	-2 PMw.2 format,example:	$1,234.56

The DOLLAROUT command can result in the following condition codes:

CCE	Successful conversion.
CCG	No conversion; length>132.
CCL	No conversion; length not long enough to contain the results.

Anderson College
Computing Center

Jan. 1974

PROCEDURE DUMPSTACK; sourcefile = SLPUBSRC

Function: This procedure, when called, prints the following information: the contents of DL, DB, Q, S, Z, Index, Status and P followed by a dump of the stack from DB+Ø to S. The values printed correspond to the values just before the call to DUMPSTACK is made. All values are in octal and are clearly labeled. Output is to file "LIST" which defaults to device "LP" with characteristics ASCII, CCTL, WRITE ONLY. The output file must have a record width of 66 words or greater.

With this procedure the condition code remains unchanged. Any errors encountered will cause the program to print "*ERROR" followed by termination.

Anderson College
Computing Center

Jan. 1974

PROCEDURE DUMPREG; sourcefile = SLPUBSRC

Function: This procedure prints the current contents of the stack registers (DB, DL, Z, STAT, X, Q, S) on $STDLIST when called. The values printed correspond to the values just before the call to DUMPREG is made. Note that DL, Z, Q, and S are DB relative and that DL is normally negative in two's complement form. This is useful in debugging programs where stack overflow or underflow occurs.

With this procedure the condition code remains unchanged.

Sample output:

```
DB = %156250
DL = %177732
 Z = %00160S
ST = %060151
 X = %000007
 Q = %000273
 S = %000314
```

Anderson College
Computing Center

Jan. 1974

PROCEDURE READCARD (buffer);

ARRAY buffer;

sourcefile = SLPUBSRC

Function: When called, this procedure causes a card to be read from the card reader (logical device #5), and stores the data in column-binary form. Each card column is stored in the corresponding word of buffer with the bit in row 12 stored in bit 11 and the bit from row 9 stored in bit Ø.* READCARD bypasses the MPE file system. Therefore the following should be noted:

1. A :DATA card is not required to precede the data.
2. Cards with a colon in column 1 are not intercepted or otherwise recognized by MPE. Therefore these cards may be read as data.
3. No check is made for other users on the card reader. Care must be used not to "steal" another user's data.
4. An :EOF card is not needed or required. The user must determine programatically when the last card has been read.

buffer — The logical array into which the column binary data is stored. It must be at least 80 words in length.

The READCARD command can result in the following condition codes:

CCE — The card read was successful.

CCL — An error occurred during the read operation or insufficient stack space exists for the operation.

CCG — (This condition is not returned.)

This procedure must be compiled by a user with privileged mode capability.

* See 30106-90001 Maintenance Manual: 30106A Card Reader Subsystem for details.

Anderson College
Computing Center
Jan. 1974

PROCEDURE HOLTH (source,dest,cnt,code);

```
ARRAY source;
INTEGER cnt;
LOGICAL code;
BYTE ARRAY dest;                                sourcefile = SLPUBSRC
```

Function: This procedure converts data from column binary form to ASCII or IBM 1620 BCD form. It is normally used with procedure READCARD to process cards containing part ASCII and part binary data.

source — The word array containing the column binary data to be converted.

dest — The byte array into which the converted output is stored.

cnt — An integer whose value indicates the number of words to be converted.

code — A logical word indicating the type of conversion:

TRUE = IBM 162Ø BCD,
FALSE = ASCII.

With this procedure the condition code remains unchanged.

Anderson College
Computing Center
Jan. 1974

Running Compilers as programs instead of subsystems

Occasionally it is useful to have more than one version of a compiler available at one time. This may be done by storing one version and running it as a program.

Example: a new version of SPL is received but the old version is kept until the new one is checked out. The following commands save the old version and put up the new one:

```
:HELLO MANAGER.SYS
:RENAME SPL,SPLOLD
:FILE TAPE;DEV=TAPE
:RESTORE *TAPE;SPL;SHOW     (new SPL is up as
                              subsystem)
:BYE
```

The new version may now be invoked with the :SPL command in the usual manner. To call the old version it is first necessary to put in file equations using the formal designators of the compiler. Example:

```
:FILE SPLTEXT=MYPGMSR
:FILE SPLUSL=$NEWPASS
:FILE SPLLIST;DEV=LP
:FILE SPLMAST=$NULL
:FILE SPLNEW=$NULL
```

To run the compiler, give a RUN command with parameter=31. Example:

```
:RUN SPLOLD.PUB.SYS;PARM=31
```

With appropriate file names, this procedure should work for other compilers as well.

Anderson College
Computing Center

ANDERSON COLLEGE COMPUTING CENTER

Report on Multiprogramming Efficiency and File System Overhead Under MPE B.1.S4

November 8, 1973

Introduction

The accounting system of MPE version B does not count CPU time used by the system on behalf of a user (such as I/O handling) as part of the user's CPU time. This experiment was conducted to determine:

1. If the unaccounted for CPU time could be correlated with the number of records processed;
2. What the average overhead is per record processed;
3. What additional overhead is incurred by multiprogramming.

Conclusions

1. Unaccounted for CPU time correlates very highly with the number of records processed.
2. The average overhead per record processed is approximately 65 msec.
3. The additional overhead incurred by multiprogramming is about two percent when code can be shared.

Description

A single job was selected which made heavy use of the file system, was short enough to be tested in a reasonable time, and which could be run simultaneously by several users. The job was to compile and prepare an SPL program of 402 lines of source code (LOGRPT). Paper tapes were prepared with the following commands:

```
:HELLO CR.CC
:SPLPREP LOGRSRC
:BYE
```

The source file (LOGRSRC) included a NOLIST command in the second line.

Initially this job was run, from paper tape, with no other users on the system. Following this session, the same job was run from three

terminals with all sessions starting within a 20 second period. In each case the session was run entirely from paper tape, eliminating human response-time as a factor. The following table shows the results obtained. All times are in seconds; sessions 2-4 were run at the same time.

Job	Connect time	CPU time	Records processed
#S1	107.9	27	1344
#S2	329.6	28	1267
#S3	297.5	27	1232
#S4	283.8	25	1155

Discussion

1. The formula CON=CPU+T*REC was applied to the above data where:

   CON is total elapsed time for one or more sessions,
   CPU is total CPU time for one or more sessions,
   T is a factor to be assigned as overhead time per record,
   REC is the number of disc records processed by one or more sessions.

   In the case of #S1 the factor T was found to be 60.1 msec. per record while in the combined run of #S2,3,4 it was found to be 67.6 msec. The difference is probably due to the additional system overhead involved in running three simultaneous jobs.

   The close agreement between these figures would indicate that a value of 60-70 msec. is very consistent when averaged over a medium sized job.

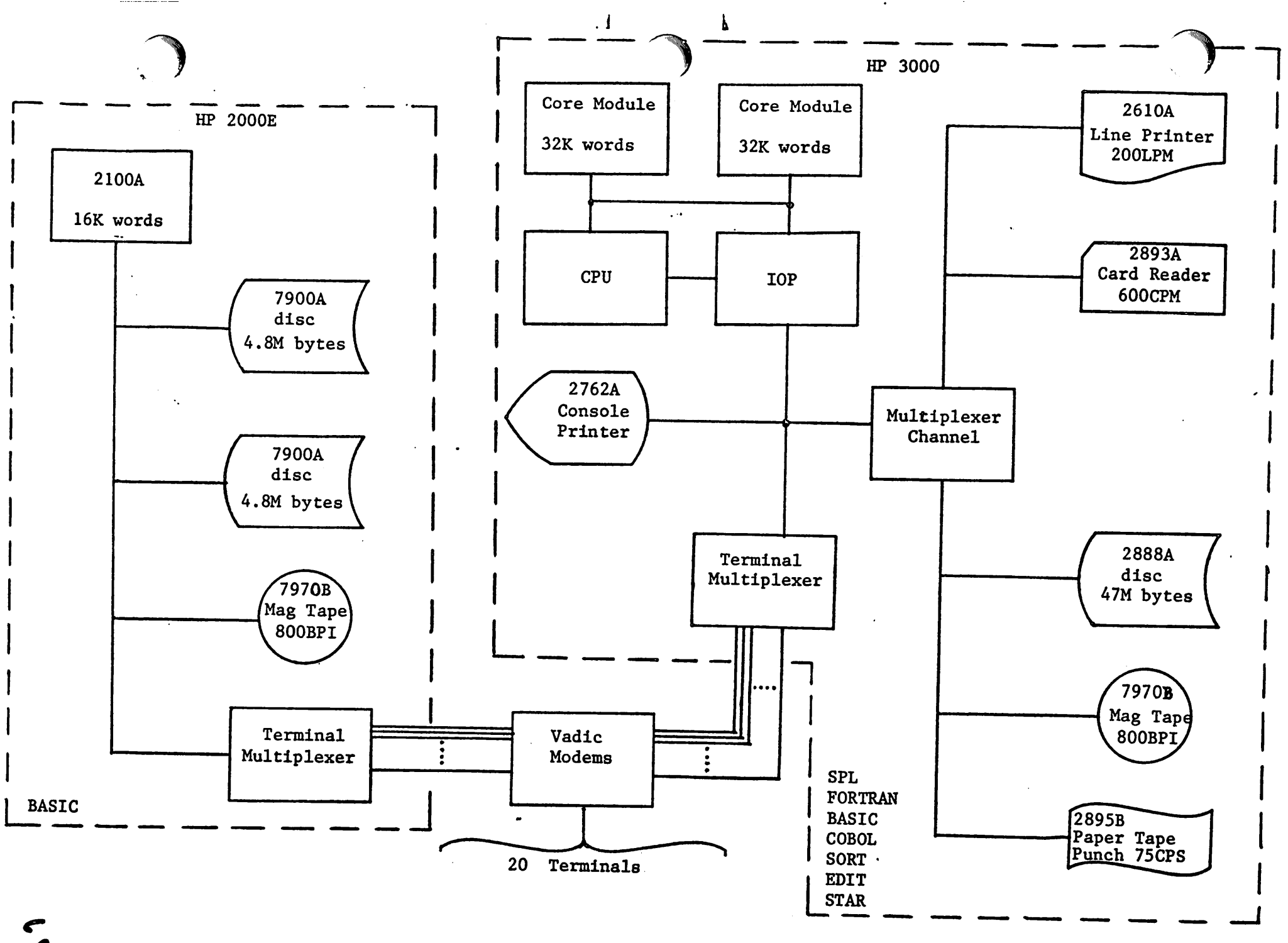
2. The total elapsed time for #S2,3,4 was 329.6 seconds. This compares closely with 323.7 seconds which is triple the time for #S1. The difference is less than 2%. Since sessions #S2,3,4 were run very nearly in synchronism, they would be expected to be either I/O bound or compute bound simultaneously, thus negating any gain in efficiency due to multiprogramming. In actual experience the sessions seemed to leap-frog past each other and #S2 which started first was the last to finish. This is probably due to the nature of code sharing as noted below.

3. The following table shows the files closed by each session with the number of records processed.

	File	Records	Session #S1	#S2	#S3	#S4
1.	LOAD.PUB.SYS	2	X	X		
2.	SL.PUB.SYS	47	X	X		
3.	SPL.PUB.SYS	63	X	X		
4.	SPL.PUB.SYS	2	X	X	X	X
5.	$NEWPASS	384	X	X	X	X
6.	TEMPLIST	Ø	X	X	X	X
7.	TEMPCODE	282	X	X	X	X
8.	SPLINTR.PUB.SYS	20	X	X	X	X
9.	LOGRSRC	422	X	X	X	X
10.	SPLLIST	8*	X	X	X	X
11.	LOAD.PUB.SYS	2	X		X	
12.	SL.PUB.SYS	52	X		X	
13.	SEGPROC.PUB.SYS	23	X		X	
14.	SEGPROC.PUB.SYS	2	X	X	X	X
15.	$NEWPASS	37	X	X	X	X
16.	$OLDPASS	6	X	X	X	X

* $STDLIST, not counted in totals.

It can be seen that #S2 bore the cost of allocating the compiler (files 1-3) while #S3 bore the cost of allocating the segmenter (files 11-13). Thus in each case the other two sessions were given a "free ride" because they could share the allocated code. The efficiency thus gained appears to nearly (but not entirely) compensate for the added overhead involved in multiprogramming.


HP 2000E
2100A
16K words
7900A
disc
4.8M bytes
7900A
disc
4.8M bytes
7970B
Mag Tape
800BPI
Terminal
Multiplexer
BASIC
HP 3000
Core Module
32K words
Core Module
32K words
CPU
IOP
2762A
Console
Printer
Terminal
Multiplexer
Vadic
Modems
20 Terminals
Multiplexer
Channel
2610A
Line Printer
200LPM
2893A
Card Reader
600CPM
2888A
disc
47M bytes
7970B
Mag Tape
800BPI
2895B
Paper Tape
Punch 75CPS
SPL
FORTRAN
BASIC
COBOL
SORT
EDIT
STAR

HOLLOMAN AIR FORCE BASE

CDC 6600 TO HP 3000 FORTRAN CONVERSION GUIDE

ROBERT G. TANTZEN

JANUARY 1974

CENTRAL INERTIAL GUIDANCE TEST FACILITY
6585TH TEST GROUP
HOLLOMAN AIR FORCE BASE, NEW MEXICO

CDC 6600 TO HP 3000 FORTRAN CONVERSION GUIDE

This paper contains the information necessary to convert Fortran programs from the CDC 6600 to the HP 3000. A separate paper will describe how most of the necessary changes can be made on the HP 3000 using the TEXT EDITOR program.

CONTENTS

A. Program Size

The total memory required to execute a program consists of four parts:

1. The program code, including library routines.

2. System routines for control of job execution.

3. Data area.

4. Buffer area for input and output.

On the 6600 almost all of this is contained in the user's field length. On the HP 3000, programs are divided into segments, only one of which needs to be in core memory at any one time. Library and system routines also can be shared by many users. To design an efficient program the user has only to consider his program and his own subroutines, and his data and buffer space.

A large program should be divided into code segments, each segment being one or more subprograms. Although the segment size is probably not very critical, a desirable size to aim for is 4000 to 8000 words, roughly about 600 lines of Fortran source code.

B. Data Storage Size

Both the data area (data stack) and the data buffers occupy space in core memory when a job executes. The data stack cannot be broken into segments, as is possible with programs. Therefore, it is very important to keep its size as small as possible.

The absolute machine limit for a data stack is 32000 words. The limit for any one program should be considerably smaller to allow multi-programming. We recommend to stay under 8000 words (4000 REAL values). The input/output buffer size depends on the size of the logical records to be transmitted. Buffering can be avoided altogether; this is done through control cards, so it is not a factor when writing the program.

The limits given above are initial recommendations. Actual experience with the HP 3000 will indicate any necessary changes.

C. Language Differences

Only those differences relevant to conversion of CDC 6600 programs are considered here.

1. Coding Procedures

a. Original card decks may have been punched on IBM 026 or 029 keypunch. If 026, then the original characters + = ( ) appear as $ # % < and must be changed to their original.

b. Comment Cards: If the character in column 1 is $ or *, change it to C.

c. Multiple statements per line, separated by $, are not allowed in HP. Each statement must start on a new card or line.

2. Constants (See also Paragraph F)

a. Octal CDC: $\pm$On$_1$....n$_i$ i = 6-20

$\pm$n$_1$....n$_i$B i = 1-20

HP: $\pm$%n$_1$....n$_i$ i = 1-6

b. Logical CDC: .TRUE. or .T.; .FALSE. or .F.

HP: .TRUE. .FALSE.

c. Hollerith

Hollerith constants, of form nH..... are allowed in DATA and FORMAT statements only. In assignment statements character strings enclosed in quotes must be used, ".....". If in CDC the variable is type integer or logical, make it type CHARACTER*10 or less than 10 depending on how many characters are actually used.

In HP variables of other types can be assigned to character bit patterns as follows:

L = %"AB"L (logical, 1 word)

I = %"AB" (integer, 1 word)

71

R = %"ABCD"R (real, 2 words)

D = %"ABCDEF"D (double, 3 words)

d. Character string in FORMAT statement:

CDC *.....*

HP "....." or '.....'

e. Integer

In CDC an integer is often used to hold character information, typically read in under A10 format. In HP the variable should be made type CHARACTER*10; or an integer array, dimension 5, to be read into with 5A2 format.

3. Variables

A variable is considered logically .FALSE. if

CDC: whole word is zero
HP: least significant bit is zero

4. Expressions

a. Relational expressions

A op B, where op can be .EQ. .NE. .GT. .GE. .LT. .LE.

CDC: A and B can be of different type.
HP: Only integer, real, double precision can be mixed, but not complex or character. For the latter two, only .EQ. and .NE. may be used.

b. Logical Expressions

A op B op C .... where op can be .NOT. .AND. .OR. .XOR.
CDC: Does not have .XOR.
The others can be abbreviated to .N. .A. .O.
A and B can be type logical or integer
HP: No abbreviated notation
A and B must be type logical. For an integer variable I this can be achieved by using BOOL(I) intead of I.

c. Masking expressions (bit-by-bit logical operation)

A op B, where op can be .NOT. .AND. .OR. .XOR.
CDC: Does not have .XOR.
A and B any type except logical. Short forms .N. .A. .O. allowed
HP: A and B must be type logical. No short forms allowed.

If operands are type integer, use BOOL(I) to make them type logical. If more than 16 bits are involved "EQUIVALENCING" a LOGICAL array to the variable and performing the masking on a word by word basis. Sometimes using "partial word designators" may come in handy.

5. Replacement (Assignment) Statement

a. Mixed mode.

A = E (A is a variable, E is an expression). Normally, if A and E are of different type, an implicit type conversion occurs across the equal sign. Exceptions are:

CDC: If A is integer, E is logical, no conversion occurs.
HP: Only types INTEGER, REAL, DOUBLE PRECISION, COMPLEX may be mixed.

For type LOGICAL and CHARACTER, A and E must agree in type.

b. Multiple Assignment Statement

CDC: A = B = C..... = E
HP: Not allowed.

6. Type Declaration, Storage Allocation

a. CDC: DOUBLE PRECISION or just DOUBLE. The word TYPE may may precede any type name.
HP: DOUBLE PRECISION only. The word TYPE not allowed.

b. COMMON

CDC: Identifier can be numeric
HP: Identifier must start with a letter.

c. DATA

CDC: DATA can preset any variables and arrays except blank COMMON.
HP: DATA cannot preset any COMMON. Must use BLOCKDATA subprogram to do this.

CDC: Implied DO-loop notation in variable list is permitted, e.g. (A(J), J = 10, 20).
HP: Not permitted. The alternate form DATA ($V = C_1,....C_M$) must be changed to the normal form with slashes, DATA $V/C_1....C_M/$

7. Control Statements

a. Assigned GO TO

GO TO i ($M_1,.....M_M$)

CDC has the comma after the i optional.
HP requires the comma.

b. Computed GO

GO TO (M_1....M_M),I

CDC has the comma before the i optional.
HP requires the comma.

c. Two-way logical IF

IF (EXPR)M_1,M_2

HP does not have this. In CDC this can be used in two ways:

(1) EXPR is arithmetic (contains no relational operators like .GT.). Then EXPR is tested for zero (M_1) or non-zero (M_2). In this case use IF(EXPR)M_1, M_2, M_1.

(2) EXPR is logical, i.e., contains relational operators, or the name of a logical variable. Then use IF(INT(EXPR))M_1, M_2, M_1.

d. PAUSE and STOP

The optional number after a PAUSE or STOP is considered octal by CDC, decimal by HP. No change necessary.

e. RETURN

CDC: RETURN in a main program causes exit to the operating system.

HP: Undefined or not permitted ?

f. EXIT

CDC: CALL EXIT returns control to monitor.
HP: Use STOP.

8. Programs, Subprograms.

a. The CDC PROGRAM card contains all filenames used. This must be deleted.

b. CDC may have FORTRAN VI, FORTRAN IV, etc in front of PROGRAM, SUBROUTINE, or FUNCTION. Remove that. (FORTRAN VI forces pre-testing of DO-loops and allows END to act as RETURN in subroutines.)

c. ENTRY statement in subroutines.

CDC provides alternate entry points to subroutines.

HP does not. Add one parameter to subroutine and use it there to jump to the desired point.

9. Segmentation, Overlays.

CDC: Used to save memory space.
HP: Due to the different design architecture of the 3000 segmentation is performed differently. Remove all OVERLAY and CALL OVERLAY cards.

10. Input, Output

a. Standard Input, Standard List, Standard Punch.

CDC:		HP:	
	READ fm, List		READ (5,fm) List
	PRINT fm, List		WRITE (6,fm) List
	PUNCH fm, List		WRITE (4,fm) List

b. NAMELIST

HP does not have this feature. Replace with regular formatted READ or WRITE.

c. BUFFER IN, BUFFER OUT, IF UNIT, LENGTH.

HP does not have this, replace with

READ (i,ERR=M_1, END = M_2) List
WRITE (i,ERR = M)

Keep in mind that BUFFER transfers data directly to or from core, while READ and WRITE normally use buffers. Buffing can be avoided by specifying NOBUF in FILE command. When reading a tape record with parity errors it is mandatory that the program can examine the record. How this can be achieved has not yet been resolved.

CDC allows to BUFFER IN a logical record of unknown length, then use the LENGTH function to find the number of words actually read. To accomplish this in HP Fortran, we may have to call FREAD directly from Fortran or via an SPL procedure.

d. End-of-file check after READ.

CDC has IF(EOF,i)
IF(ENDFILEi)
HP does this with
READ (i, END=M)

e. Special forms of READ and WRITE

CDC:		HP:	
	READ INPUT TAPE i, fm, List		READ (i,fm) List
	WRITE OUTPUT TAPE i, fm, List		WRITE (i,fm) List
	READ TAPE i, List		READ (i) List
	WRITE TAPE i, List		WRITE (i) List

f. ENDFILE i

CDC: Writes EOF
HP: Writes EOF and closes the file. In order to write multi-file tapes, it must be possible to write an EOF without invoking file closure. (Under investigation.)

g. ENCODE, DECODE

CDC: DECODE (n,fm,CA) list [CA = integer array]
HP: READ (CA,fm) list [CA = character variable, size n]
Same for ENCODE, WRITE.

D. Library Function.

1. HP does not have ASIN(X) and ACOS(X). Both can be expressed in terms of ATAN. If used frequently the two functions can be written and placed in the library.

2. Randon Numbers

CDC has RANF(R) as a library function.

HP has RAND as an external subroutine. User should check if the numbers generated meet his criteria.

3. Shift Function

CDC has LS(A,i) to shift one word left or right.

HP has K=K(b,n), partial-word designator, which can be used for right shift. Some re-programming may be necessary because the difference in word length enters into the picture.

4. Byte Handling

CDC has the INBY and ENBY functions to manipulate bytes of arbitrary length.

Possibly an SPL procedure should be written, or reprogramming using partial word designators solves the problem.

E. Library Subroutines

1. Time and Date

CDC: DATE gives day, month, year
CLOCK gives hours, min, sec

HP: CHRONOS gives all of above, but in a different format. Probably must write a Fortran callable subroutine which in turn calls the CHRONOS intrinsic (no parameters).

CDC: SECOND gives CP time used
TIMTGØ gives CP time remaining
HP: PROCTIME gives CP time used. Remaining time not obtainable. Probably must write subroutine since PROCTIME has no parameters and different format for time returned.

2. File Handling

The following CDC Fortran subroutines seem not to have an HP equivalent:

PARTOUT moves a file to the appropriate output queue (print, punch, microfilm) and rewinds that local file to accept further information.

SKFILE skips files forward or backward on multi-files.

RETURNS releases a tape file and the tape drive.

UNLOADS releases a tape file, not the tape drive.

The last two routines allow the return of files before the end of a program.

IDENT reads or writes ID-records on data files. This subroutine will be programmed and placed on the system library.

3. Messages

CDC routines REMARK, DISPLA send a message to the operator or the dayfile.

HP has PRINTOP for message to operator. The dayfile information is mixed with other putput on $STDLIST.

F. Effects of Different Word Size

Precision and Range of Numbers

	CDC	HP
Type REAL, significant decimals	14.5	6.9
smallest absolute value	10^{-294}	10^{-77}
largest absolute value	10^{322}	10^{77}
Type DOUBLE, significant decimals	29	11.7
smallest absolute value	10^{-294}	10^{-77}
largest absolute value	10^{322}	10^{77}
Type integer, largest value	$2^{48}-1$	$2^{15}-1$
Integer subscript, largest value	$2^{15}-1$	$2^{15}-1$

Points to Consider:

1. Check if any integer variables or constants may be greater than $2^{15}-1 = 32767_{10}$.

2. Check if any real variables should be made double precision.

3. Check if any real or double precision variables could exceed 10^{77}.

4. Check if CDC program has any double precision variables or arrays. If so, analyze the problem to see if it can be done at all on the HP 3000.

5. Check if integer variables or arrays are used to store character information, 10 characters per word. If so, may have to use type CHARACTER.

HUGHES AIRCRAFT

- PAPER TAPE TO FILE SUBSYSTEM

- THIS PROGRAM
  - READS BYTE DATA FROM THE PAPER TAPE READER TO ANY FILE
  - WILL DELETE OR REPLACE ANY CHARACTER INPUT
  - WILL PROCESS FIXED OR VARIABLE LENGTH RECORDS

- FILES: PTAPEINS SOURCE FILE
  PTAPEIN PROGRAM FILE

- R. Lanphar

Hughes Aircraft Company
P.O. Box 3310 600/B255
Fullerton, California 92634
(714) 871-3232 Ext. 3975

DESCRIPTION

This S.P.L. Interactive Program is useful in getting paper tape programs and data into the system. Readable tapes include those with fixed length records or variable length records which contain an identifiable terminal character (example: line feed, 12_8). If desired the program can strip off the parity bit. User specified characters are replaced by another character or deleted from the record (example: delete CR , line feed, and rubout characters). Compatible record formats include:

Paper-Tape Record Format	File Record Format
Fixed	Fixed
Variable	Fixed
Variable	Variable

Multiple rolls of paper tape can be processed into the same file. An option is available to rerun the program with the same parameters.

The dialog is easy to follow with extensive error checking and error messages.

LIMITS

(A) All record sizes must be ≤256 bytes

(B) Maximum number of terminal characters is 10

(C) Maximum number of search and replacement character pairs is 20

(D) Disc file must not exist prior to execution of the program

(E) File names can be fully qualified with up to a maximum of 27 characters in length.

All of the above limitations are recoverable.

USER INTERFACE

The session mode is required to interface with the program. The program is executed by

```
:RUN PTAPE IN.USERS.SUPPORT
```

Example dialog is attached.

```
:RUN PTAPEIN.USERS.SUPPORT

PAPER TAPE TO FILE SUBSYSTEM

TERMINATE ALL RESPONSES OR SEQUENCE OF RESPONSES
WITH A CARRIAGE RETURN.

FILE NAME =? VF1
ARE RECORDS ON PAPER TAPE FIXED(F) OR VARIABLE(V)? V

ARE RECORDS ON FILE TO BE FIXED(F) OR VARIABLE(V)? F

FILE RECORD SIZE(BYTES)  =?80

DO YOU WANT PARITY BIT MASKED OUT(Y,N) ? Y

TERMINAL CHARACTER IN OCTAL (%XXX) =?%12
TERMINAL CHARACTER IN OCTAL (%XXX) =?

TERMINAL CHARACTERS ARE NOT DELETED
UNLESS SPECIFIED BELOW.

DO YOU WANT ANY CHARACTERS REPLACED
OR DELETED FROM THE INPUT (Y,N)? Y

TO REPLACE CHARACTERS FOLLOW THE QUESTION
MARK PROMPT WITH THE SEARCH AND REPLACEMENT
CHARACTERS INPUT RESPECTIVELY AS TWO OCTAL
NUMBERS SEPARATED BY A COMMA.

TO DELETE CHARACTERS SET THE REPLACEMENT
CHARACTER TO %0.

FORMAT: ?%XXX,%XXX
        ?%12,0
        ?%15,0
        ?%177,0
        ?
ANOTHER ROLL (Y,N)?Y
ANOTHER ROLL (Y,N)?Y
ANOTHER ROLL (Y,N)?N
NUMBER OF RECORDS PROCESSED = 30

DO YOU WISH TO RUN THE PROGRAM AGAIN
USING THE SAME PARAMETERS (Y,N) ?Y

FILE NAME = ?VF2
ANOTHER ROLL (Y,N)?Y
ANOTHER ROLL (Y,N)?N
NUMBER OF RECORDS PROCESSED = 20

DO YOU WISH TO RUN THE PROGRAM AGAIN
USING THE SAME PARAMETERS (Y,N) ?N

END OF PROGRAM
:
```

COMPILE FROM EDITOR

These routines from Hewlett Packard may be invoked from the EDITOR using the PROCEDURE edit command. Compile, load, and use example is attached.

Files: SPLCOMPS, source
FTNCOMPS, source

Doug Mecham

Hughes Aircraft Company
P. O. Box 3310 601/H219
Fullerton, California 92634

```
:JOB     MANAGER.SYS. STAFF
 PRI= CS:  INPRI= 8 :  TIME= 2
 JOB NUMBER = #J17
 WED. JAN 23. 1974. 10:18 AM
 HP32000B.00.S4

:PURGE FTNCOMPU
:BUILD  FTNCOMPU:CODE=USL
:SPL FTNCOMPS.FTNCOMPU
```

```
00000 0    $CONTROL SUBPROGRAM,SEGMENT=COMPFTN,ADP
00000 0         BEGIN
00000 1         INTRINSIC COMMAND,CREATE,ACTIVATE ;
00000 1    <<
00000 1       TO INVOKE THE PROCEDURE WITHIN THE EDITOR....USE
00000 1                    /KEEP FTNTEXT
00000 1                    /PROCEDURE COMPILEF,X,Y
00000 1            WHERE X=G(GROUP LIB,) OR P(PUBLIC LIB,) OR S(SYSTEM LIB
00000 1            AND Y=ANYTHING.
00000 1         THE ABOVE COMMANDS WILL CHAIN IN FTN.  AFTER COMPILATION ET
00000 1         CONTINUE EDITTING OR USE USL FILE IN $OLDPASS.
00000 1                 CONTRIBUTED BY FRITZ JOERN, GENEVA
00000 1                 MODIFIED BY CHENG T. CHENG, CUPERTINO-TRAINING.
00000 1         >>
00000 1            LOGICAL PROCEDURE COMPILEF(STRING,SIZE,NUMBER,SPACE) ;
00000 1            BYTE ARRAY STRING,NUMBER ;
00000 1            INTEGER SIZE ;
00000 1            ARRAY SPACE ;
00000 1                 BEGIN
00000 2                 BYTE ARRAY COMM(0:21) ;
                      0 +001
00000 2                 INTEGER PIN ;
                      0 +002
00000 2                 LOGICAL ERR ;
                      0 +003
00000 2                 IF NUMBER <> "   1" THEN GO TO AROUND ;
00016 2                 MOVE COMM:= "FILE FTNLIST;DEV=LP" ;
00036 2                 COMM(19) := %15 ;
00041 2                 COMMAND(COMM,ERR,PIN) ;
00045 2                 GO TO TEXT ;
00046 2    AROUND:      IF NUMBER <> "   2" THEN GO TO SKIP ;
00057 2                 MOVE COMM:= "FILE FTNLIST=$STDLIST" ;
00101 2                 COMM(21) := %15 ;
00104 2                 COMMAND(COMM,ERR,PIN) ;
00110 2    TEXT:        MOVE COMM := "FILE FTNTEXT,OLD" ;
00126 2                 COMM(16) := %15 ;
00131 2                 COMMAND(COMM,ERR,PIN) ;
00135 2    SKIP:        MOVE COMM := "FORTRAN.PUB.SYS " ;
00153 2                 COMM(16) := %15 ;
00156 2                 CREATE(COMM,,PIN,3,1) ;
00166 2                 ACTIVATE(PIN,%2) ;
00172 2                 COMPILEF := TRUE ;
00174 2                 IF NUMBER = "   3" THEN GO TO PASS ;
00206 2                 MOVE COMM := "RESET FTNTEXT" ;
00224 2                 COMM(13) := %15 ;
00227 2                 COMMAND(COMM,ERR,PIN) ;
00233 2                 MOVE COMM := "RESET FTNLIST" ;
00250 2                 COMM(13) := %15 ;
00253 2                 COMMAND(COMM,ERR,PIN) ;
00257 2                 MOVE COMM := "PURGE FTNTEXT" ;
00274 2                 COMM(13) := %15 ;
00277 2                 COMMAND(COMM,ERR,PIN) ;
00303 2    PASS:        RETURN ;
00304 2                 END;
00000 1         END.
PRIMARY DB STORAGE=%000:    SECONDARY DB STORAGE=%00000
NO. ERRORS=000:             NO. WARNINGS=000
```

```
PAGE 0002    HEWLETT-PACKARD

 PROCESSOR TIME=0:00:03:     ELAPSED TIME=0:00:33


 END OF COMPILE
:SEGMENTER

SEGMENTER SUBSYSTEM (2.0)
USL FTNCOMPII
LISTUSL

USL FILE FTNCOMPII.STAFF.SYS

COMPETN
   COMPILEE          305  P  A C N R

FILE SIZE          377600
DIR. USED              41          INFO USED          333
DIR. GARB.              0          INFO GARB.           0
DIR. AVAIL.         37337          INFO AVAIL.     337445

SL SL.PUB.SYS
ADDSL COMPETN.PMAP
COMPETN          47
   NAME              STT  CODE ENTRY SEG
   COMPILEE            1     0     0
   CREATE              2                ?
   ACTIVATE            3                ?
   COMMAND             4                ?
   SEGMENT LENGTH          314

EXIT

 END OF SUBSYSTEM
:EOD
IGNORED
:EDITOR
```

```
:HP32201A.02.1 EDIT/3000  WED, JAN 23, 1974, 10:21 AM
/TEXT TESTFTN
/LIST ALL
   1              PROGRAM TEST
   2              WRITE(6,*)"HERE WE ARE USING THE FORTRAN COMPILER"
   3              STOP
   4              END
/KEEP FTNTEXT
/PROCEDURE COMPILEF.S.2


PAGE 0001    HEWLETT-PACKARD 32102A.00.5   FORTRAN/3000      WED, JAN 23, 1974, 10:21

 0001000         PROGRAM TEST
 0002000         WRITE(6,*)"HERE WE ARE USING THE FORTRAN COMPILER"
 0003000         STOP
 0004000         END


***  NO ERRORS,  NO WARNINGS:   PROGRAM UNIT COMPILED  ****
COMPILATION TIME    0.436 SECONDS


/EXIT


 END OF SUBSYSTEM
:PURGE SPLCOMPU
:BUILD SPLCOMPU;CODE=USL
:SPL SPLCOMPS.SPLCOMPU
```

```
00000 0    $CONTROL SUBPROGRAM,SEGMENT=COMPSPL,ADR
00000 0         BEGIN
00000 1         INTRINSIC COMMAND,CREATE,ACTIVATE ;
00000 1    <<
00000 1       TO INVOKE THE PROCEDURE WITHIN THE EDITOR....USE
00000 1                    /KEEP SPLTEXT
00000 1                    /PROCEDURE COMPILES,X,Y
00000 1           WHERE X=G(GROUP LIB.) OR P(PUBLIC LIB.) OR S(SYSTEM LIB)
00000 1           AND Y=ANYTHING.
00000 1         THE ABOVE COMMANDS WILL CHAIN IN SPL.  AFTER COMPILATION EITHER
00000 1         CONTINUE EDITING OR USE $OLDPASS.
00000 1                CONTRIBUTED BY FRITZ JOERN, GENEVA
00000 1                MODIFIED BY CHENG T. CHENG, CUPERTINO-TRAINING.
00000 1         >>
00000 1            LOGICAL PROCEDURE COMPILES(STRING,SIZE,NUMBER,SPACE) ;
00000 1            BYTE ARRAY STRING,NUMBER ;
00000 1            INTEGER SIZE ;
00000 1            ARRAY SPACE ;
00000 1                 BEGIN
00000 2                 BYTE ARRAY COMM(0:21) ;
                      0 +001
00000 2                 INTEGER PIN ;
                      0 +002
00000 2                 LOGICAL ERR ;
                      0 +003
00000 2                 IF NUMBER <> "   1" THEN GO TO AROUND ;
00016 2                 MOVE COMM:= "FILE SPLLIST;DEV=LP" ;
00036 2                 COMM(19) := %15 ;
00041 2                 COMMAND(COMM,ERR,PIN) ;
00045 2                 GO TO TEXT ;
00046 2    AROUND:      IF NUMBER <> "   2" THEN GO TO SKIP ;
00057 2                 MOVE COMM:= "FILE SPLLIST=$STDLIST" ;
00101 2                 COMM(21) := %15 ;
00104 2                 COMMAND(COMM,ERR,PIN) ;
00110 2    TEXT:        MOVE COMM := "FILE SPLTEXT,OLD" ;
00126 2                 COMM(16) := %15 ;
00131 2                 COMMAND(COMM,ERR,PIN) ;
00135 2    SKIP:        MOVE COMM := "SPL,PUB,SYS " ;
00151 2                 COMM(12) := %15 ;
00154 2                 CREATE(COMM,,PIN,3,1) ;
00164 2                 ACTIVATE(PIN,%2) ;
00170 2                 COMPILES := TRUE ;
00172 2                 IF NUMBER = "   3" THEN GO TO PASS ;
00204 2                 MOVE COMM := "RESET SPLTEXT" ;
00222 2                 COMM(13) := %15 ;
00225 2                 COMMAND(COMM,ERR,PIN) ;
00231 2                 MOVE COMM := "RESET SPLLIST" ;
00246 2                 COMM(13) := %15 ;
00251 2                 COMMAND(COMM,ERR,PIN) ;
00255 2                 MOVE COMM := "PURGE SPLTEXT" ;
00272 2                 COMM(13) := %15 ;
00275 2                 COMMAND(COMM,ERR,PIN) ;
00301 2    PASS:        RETURN ;
00302 2                 END;
00000 1         END.
PRIMARY DB STORAGE=%000:    SECONDARY DB STORAGE=%00000
NO. ERRORS=000:             NO. WARNINGS=000

PROCESSOR TIME=0:00:03:     ELAPSED TIME=1:00:36
```

```
 END OF COMPILE
:SEGMENTER

SEGMENTER SUBSYSTEM (2.0)
USL SPLCOMPU
LISTUSL

USL FILE SPLCOMPU.STAFF.SYS

COMPSPL
   COMPILES          303  P  A C N P

FILE SIZE          377600
DIR. USED              41          INFO USED          331
DIR. GARB.              0          INFO GARB.           0
DIR. AVAIL.         37337          INFO AVAIL.     337447

SL SL.PUB.SYS
ADDSL COMPSPL.PMAP
COMPSPL          35
   NAME            STT  CODE ENTRY SEG
   COMPILES          1     0     0
   CREATE            2                 ?
   ACTIVATE          3                 ?
   COMMAND           4                 ?
   SEGMENT LENGTH          310

EXIT

 END OF SUBSYSTEM
:EOD
IGNORED
:PURGE *OLDPASS
:EDITOR

HP32201A.02.1 EDIT/3000   (C) HEWLETT-PACKARD CO. JAN 23, 1974, 10:24 AM
/TEXT TESTSPL
/LIST ALL
   1      <<     A SMALL SPL PROGRAM     >>
   2      BEGIN
   3      BYTE ARRAY OUTPUT(0:79):=80(" ");
   4      INTRINSIC PRINT;
   5      PROCEDURE DATELINE(BUF);
   6           BYTE ARRAY BUF;
   7           OPTION EXTERNAL;
   8      BEGIN
   9           DATELINE(OUTPUT);
  10           PRINT(OUTPUT,-80,%40);
  11      END;
  12      END.
/KEEP SPLTEXT
/PROCEDURE COMPILES.S.?
```

```
PAGE 0001   HEWLETT-PACKARD 32100A.04.2  SPL/3000  WED, JAN 23, 1974, 10:24 AM

          00000 0    <<     A SMALL SPL PROGRAM     >>
          00000 0    BEGIN
          00000 1    BYTE ARRAY OUTPUT(0:79):=80(" "):
          00003 1    INTRINSIC PRINT;
          00003 1    PROCEDURE DATELINE(BUF);
          00000 1         BYTE ARRAY BUF;
          00000 1         OPTION EXTERNAL;
          00000 1    BEGIN
          00000 2         DATELINE(OUTPUT);
          00002 2         PRINT(OUTPUT,-80,%40);
                                  ^
***** WARNING *****  ARITHMETIC RIGHT SHIFT EMITTED

          00007 2    END;
          00007 1    END.
PRIMARY DB STORAGE=%001:    SECONDARY DB STORAGE=%00050
NO. ERRORS=000:             NO. WARNINGS=001
PROCESSOR TIME=0:00:01:     ELAPSED TIME=0:00:12

/EXIT

END OF SUBSYSTEM
:PREPRUN $OLDPASS

END OF PREPARE

WED, JAN 23, 1974, 10:25 AM

END OF PROGRAM
:EOJ

CPU (SEC) = 66
ELAPSED (MIN) = 7
WED, JAN 23, 1974, 10:25 AM
END OF JOB
```

- HP 2100-HP 3000 DATA COLLECTION LINK

- This subsystem provides one technique for transferring data from a HP 2100 to a HP 3000 via the asynchronous multiplexer.

- Files: ANACOMS, NUMRECS, NACOM, TANACOMS

- Doug Mecham

Hughes Aircraft Company
P.O. Box 3310 601/H219
Fullerton, California 92634
(714) 871-3232 Ext. 3077

Description

There are two routines that comprise this subsystem, a FORTRAN Program in the HP 3000 and a routine in the HP 2100. The approach is to have the 2100 routine send :DATA DATA.SYS to the 3000; the latter program then begins to read data until an end of data is received, :EOD. The current subsystem transfers a file name, FORTRAN format, and as many data records as required.

The filename is limited to eight characters; the format is fixed. E14.8; and the maximum number of values per record is 99.

User Interface:

1. The 2100 routine is initiated by using the CALL SEND 1 (filename, number-of-values-per-record) entry point.

2. The HP 3000 program is initiated, :RUN ANACOM.

3. Data is sent from the 2100 by using the CALL SEND 5 (location-of-data) entry point. This is repeated as many times as required.

4. To terminate, the CALL SEND 9 entry point is used; the 3000 program then terminates.

At the HP 3000 standard list device the file name, format and number of records transmitted is printed.

Notes:

The HP 3000 program uses an SPL subroutine to obtain the number of records transmitted.

The filename is entered at the 2100 console using repeated use of the A2 format into an integer array.

To dump the collected data the data file is equivalenced to FTN08, e.g.

```
:FILE FTN08=MYDATA,OLD
```

Files:

ANACOMS	Source of HP 3000 FORTRAN program to collect data.
NUMRECS	Source of HP 3000 SPL procedure to determine number of records transferred.
NACOM	Source of HP 2100 assembly routine to initiate, send data, and terminate communication link.
TANACOMS	Source of HP 3000 FORTRAN program to dump file of collected data.

- N! and 2^N

- These programs calculate N factorial and powers of 2. DATA value NN governs largest factorial number or power of 2.

- Files: NFACT, POWER2

- Ulysses Okawa

  Hughes Aircraft Company
  P.O. Box 3310 600/D131
  Fullerton, California 92634
  (714) 871-3232 Ext. 2335

- HP 7202 PLOTTER FORTRAN SUBROUTINE

- This program formats two arrays and outputs for the HP 7202 plotter. Program output is to file FTN01.

- File: PLOT 7202

- Howard Harry

Hughes Aircraft Company
P.O. Box 3310 600/C135
Fullerton, California 92634
(714) 871-3232 Ext. 3687

- MATH SUBROUTINES

- Files and description

TAYLOR	Computes Taylor distribution for given sidelobe level and n bar.
MAXMIN	Finds maximum and minimum values of an array.
GAUSS	Generates a random number array from Gaussian distribution.
BESSELM0	Compute modified Bessel functions of order 0.
BESSELF0	Compute Bessel functions of order 0.
BESSELF1	Compute Bessel functions of order 1.
RANDG	Returns a random number of a Gaussian distribution.

- Howard Harry

Hughes Aircraft Company
P.O. Box 3310 600/C135
Fullerton, California 92634
(714) 871-3232 Ext. 3687

- READ ANY DECK OF CARDS AND OUTPUT TO A FILE

- This BASIC program provides the capability to read any deck of cards from the card reader even if colon is in column one.

- Files: CARDIN

- Roger Berchtold

Hughes Aircraft Company
P.O. Box 3310 600/C135
Fullerton, California 92634
(714) 871-3232 Ext. 3513

- Description:

This BASIC program reads a deck of cards from the card reader as a data file. The card deck to be stored is put into the card reader backwards between a :DATA USERS,SUPPORT card an an :EOD. The last card is read first; note the cards are right side up.

- Listing:

```
CARDIN
  10 REM THIS PROGRAM ACCEPTS CARDS FROM THE CARD READER AND LIST THE
  20 REM   CARD CONTENTS ON THE OUTPUT DEVICE.
  30 REM THE CARDS MUST BE PLACED IN THE CARD READER IN REVERSE ORDER:
  40 REM   CARDS MUST "NOT" BE INVERTED.
  50 REM FILE "CARDSTOR" MUST EXIST.
  55 REM THE OUTPUT FILE IS DESIGNATED BY THE FILENAME "OUTPUT".
  56 REM       AND SHOULD BE SPECIFIED IN A FILE EQUATION BEFORE RUNNING
  60 DIM A$[80],B$[80]
  70 B$[1:80]=" "
  80 N=1
  90 ON END #1 THEN 180
 100 FILES CARDIN,CARDSTOR,OUTPUT
 110 LINPUT #1:A$
 120 FOR X=1 TO 80 STEP 1
 130   B$[(81-X):1]=A$[X:1]
 140 NEXT X
 150 PRINT #2,N;B$
 160 N=N+1
 170 GOTO 90
 180 FOR R=(N-1) TO 1 STEP -1
 190   LINPUT #2,R:B$
 200   PRINT #3:B$
 210 NEXT R
 220 PRINT "END"
```

INTEGER ARRAY SORT

This FORTRAN program sorts a multi-dimensioned interger array.

File: ISORT

Steve Fabian

Hughes Aircraft Company
P.O. Box 3310 600/E248
Fullerton, California 92634
(714) 871-3232 Ext. 3427

ITEL LEASING CORPORATION

USER PROCEDURES

- ANSWER (*,*) — FORTRAN SUBROUTINE

  Inputs a "YES" or "NO" answer from standard input device. Returns to first label specified if answer is YES; returns to second label if answer is NO. Requests YES or NO answer if answer isn't one of the two.

  Useful in programs which require a conversational structure.

- ANSWER' (YESLABEL, NOLABEL) — SPL PROCEDURE

  LABEL YESLABEL,NOLABEL

  Same as ANSWER, but for use with SPL procedures.

- ASCIITOREAL (CHAR, LENGTH) — FORTRAN FUNCTION (REAL)

  CHAR: character string to be converted
  LENGTH: number of characters in CHAR

  Converts ASCII character string to equivalent decimal floating point number.

  Used by SPL procedures to convert ASCII input of a real number to its decimal equivalent.

Procedure head for SPL program:

```
Real Procedure ASCIITOREAL (CHAR,LENGTH);
BYTE ARRAY CHAR:
INTEGER LENGTH:
OPTION EXTERNAL:
```

For further information, contact

Madeline A. Lombaerde
Itel Leasing Corporation
1 Embarcadero Center
San Francisco, CA 94111
(415) 983-0488

```
1       SUBROUTINE ANSWER(*,*)
2       CHARACTER*3 ANS,YES,NO,BLANK
3       YES="YES"
4       NO="NO "
5       1 ACCEPT ANS
6       IF(ANS.EQ.YES) RETURN 1
7       IF(ANS.EQ.NO) RETURN 2
8       DISPLAY "TYPE YES OR NO"
9       GO TO 1
10      END
```

TEXT FILE : ANSWER1

```
1     BEGIN
2     PROCEDURE ANSWER'(YESLABEL,NOLABEL);
3     LABEL YESLABEL,NOLABEL;
4     BEGIN
5        BYTE ARRAY ANSWER(0:3);
6        ARRAY ANS(*)=ANSWER,YES(0:0),NO(0:0),MESG(0:6);
7        INTEGER LEN;
8        INTRINSIC READ,PRINT;
9     <<                                                >>
10    <<                  STATEMENTS                    >>
11    <<                                                >>
12       ANS:="  ";
13       YES:="YE";
14       NO:="NO";
15       MOVE MESG:="TYPE YES OR NO";
16       READ'ANS: LEN:=READ(ANSWER,-4);
17       IF ANS=YES THEN GO TO YESLABEL
18                  ELSE IF ANS=NO OR LEN=0 THEN GO TO NOLABEL;
19       PRINT(MESG,7,0);
20       GO READ'ANS;
21    END;
22    END.
```

TEXT FILE: ANSWER2

```
1       FUNCTION ASCIITOREAL(CHAR,LENGTH)
2       CHARACTER*(LENGTH) CHAR
3       ASCIITOREAL=RNUM(CHAR)
4       RETURN
5       END
```

TEXT FILE : ASCREAL

McMASTER UNIVERSITY

TABLE OF CONTENTS

I. OPERATION

McMASTER UNIVERSITY
COMPUTATION SERVICES UNIT
HEALTH SCIENCES CENTRE
HAMILTON, ONTARIO, CANADA
L8S 4J9

TITLE ACUMLOG

FUNCTION To analyze HPS/3000 Log Files and append a summary record into a Master File for each JOB and SESSION

AUTHOR P. Balnys

LANGUAGE FORTRAN

DESCRIPTION

The program prompts the operator for the name of the LOG File to be analyzed and the name of the Master File. A Master File will be automatically built if required. The information for each JOB and SESSION on the Log File is summarized and a record written to the Master File as well as a report written to the Line Printer. A monetary charge for each JOB and SESSION is indicated as well as identification information and resource utilization.

SPECIAL CONSIDERATIONS

1) Program requires the Line Printer.
2) Program uses two temporary disc files.
3) Subroutines DATE and COSTING are utilized.

McMASTER UNIVERSITY
COMPUTATION SERVICES UNIT
HEALTH SCIENCES CENTRE
HAMILTON, ONTARIO, CANADA
L8S 4J9

TITLE MTHLOG

FUNCTION To produce a monthly report from the Master Log File.

AUTHOR P. Balnys

LANGUAGE FORTRAN

DESCRIPTION

This program is run after a SORT of the Master LOG File has been done. The Master Log File must be organized by ACCOUNT, USER, JOB. A report is produced containing detailed records for each JOB and SESSION as well as summary information for each JOB and USER and ACCOUNT.

SPECIAL CONSIDERATIONS

1) Program requires the Line Printer.
2) Master Log File must first be sorted by ACCOUNT name, USER name and JOB name.
3) Subroutine DATE is utilized.

McMASTER UNIVERSITY
COMPUTATION SERVICES UNIT
HEALTH SCIENCES CENTRE
HAMILTON, ONTARIO, CANADA
L8S 4J9

TITLE SUMLOG

FUNCTION To produce a summary report of the sorted Master Log File for chargibg purposes.

AUTHOR P. Balnys

LANGUAGE FORTRAN

DESCRIPTION

This program produces a report much like the MTHLOG program but no detail records for each JOB and SESSION. All Master records with the same JOB name, USER name and ACCOUNT name are summed together to produce a charging report. A summary record is also produced for each USER within an account and for each ACCOUNT.

SPECIAL CONSIDERATIONS

1) The Master Log File must be sorted.
2) The program requires the LP.

McMASTER UNIVERSITY
COMPUTATION SERVICES UNIT
HEALTH SCIENCES CENTRE
HAMILTON, ONTARIO, CANADA
L8S 4J9

TITLE CARD

FUNCTION To build a file corresponding to a USER ACCOUNT and copy card image data into this file

AUTHOR P. Balnys

LANGUAGE FORTRAN

DESCRIPTION

This program is used as an intermediate spooling mechanism to input data to a spooling input file.

The specified file is first purged, if it exists, and then built so that an empty file will first exist into which the data is entered.

SPECIAL CONSIDERATIONS

A special account, namely CR must be provided and contain a group corresponding to all other accounts on the system. The USER name becomes the file name

e.g. SMITH. GENERAL. CR

McMASTER UNIVERSITY
COMPUTATION SERVICES UNIT
HEALTH SCIENCES CENTRE
HAMILTON, ONTARIO, CANADA
L8S 4J9

TITLE PRINTER

FUNCTION To empty the specified disc file to the Line Printer

AUTHOR P. Balnys

LANGUAGE FORTRAN

DESCRIPTION

This program is used as an intermediate spooling mechanism. A specified file corresponding to a USER ACCOUNT is dumped to the Line Printer and the file is Purged and rebuilt

SPECIAL CONSIDERATIONS

Batch job

AN INTERIM SOLUTION FOR SPOOLING

I. OBJECTIVES:

1. To enable the time-share users to read cards from the Card Reader and output results to the Line Printer.
2. The procedures established should be simple for the operator and user to follow.
3. When spooling in MPE is available next April, the effect on the user should be minimal.

II. METHODS:

The method is to assign to each user of the system, who has the need for spooling, two files for their card reader input and printer output. By choosing meaningful names for the files, Groups and Accounts, they can be referred to easily. Also, since all printer files are within one account, the operator can unspool them with relatively simple utility programme.

III. PROGRAMME UNITS:

1. Programme Name: CARD
   Function : To create a card image file
   RUN Instructions:

```
:JOB MGR.CR
:RUN CARD.PUB.LIB
 USER.ACCT
    data cards
:EOD
:EOJ
```

2. Programme Name: PRINTER
   Function : To list an ASCII file to Line Printer

RUN Instructions:

```
:JOB MGR.LP
:RUN PRINTER.PUB.LIB
 USER.ACCT
:EOD
:EOJ
```

IV. LIMITATIONS:

1. There is a maximum of 2000 cards for card input and 3000 lines for printer output. This can, however, be changed by modifying the BUILD commands when the spool files are created by MGR.CR/MGR.LP initially and in the PRINTER program.

2. Neither the card or printer spool file can be accessed simultaneously by more than one running programme by the same user. That is, the user must wait until his files have been unspooled before running any programme that will require them.

V. OPERATING INSTRUCTIONS:

Create two new accounts CR AND LP with MGR as the only user in each account. The capabilities to be used are: IA, BA, ND, SF.

MGR.CR/MGR.LP:

Create for each user who needs spooling, the spool files with the commands:-

```
:BUILD USER.ACCT.CR;REC= -80,3,F,ASCII;DISC=2000,16
:BUILD USER.ACCT.LP;REC=-130,1,F,ASCII;DISC=3000,16;
                                                    CCTL
```

CONSOLE OPERATOR:

1. Run CARD as soon as the user's deck is submitted.

2. Run PRINTER when requested by the user.

USER.ACCT:

A. Card Input:

1. Submit the deck to run CARD and wait until the job is done.
2. Use the FILE command to equate the formal file designator for the card input file to *USER.ACCT.CR*

Examples:

i. FORTRAN Compile:-

```
:FILE CARD = USER.ACCT.CR,OLD
:FORTRAN * CARD , uslfile, listfile
```

ii. FORTRAN RUN:-

```
:FILE FTN02=  USER.ACCT.CR,OLD
:RUN programme
```

iii. EDITOR Run:-

```
:FILE CARD = USER.ACCT.CR,OLD
:EDITOR
/TEXT * CARD, UNN
```

B. Printer Output:

1. Use the FILE command to equate the formal file designator for the printer file to *USER.ACCT.LP*

Examples:

i. FORTRAN Compile:-

```
:FILE  PRINTER = USER.ACCT.LP,OLD
:FORTRAN sourcefile, uslfile, * PRINTER
```

ii. FORTRAN Run:-

```
: FILE FTN03 = USER.ACCT.LP,OLD
:RUN programme
```

2. Use:TELLOP to inform the operator of printer output.

II. UTILITY

McMASTER UNIVERSITY
COMPUTATION SERVICES UNIT
HEALTH SCIENCES CENTRE
HAMILTON, ONTARIO, CANADA
L8S 4J9

TITLE Subroutine BUSYTEST (NAME, LU, IAC, IFLAG)

FUNCTION To test whether a specified file is currently open (in use) and if so, return a flag indicating this.

AUTHOR P. Balnys

LANGUAGE FORTRAN

DESCRIPTION

Input Parameters:

Name - Actual file name with group and account if required, contained in this character variable of length 26.
LU - Fortran logical unit number to be used by the calling program when referencing this file.
IAC = 0 indicates READ/WRITE access is desired
= 1 indicates APPEND access is desired

Output Parameters:

IFLAG = 0 means the file has been opened for this program use
= 1 means the file is busy
= -1 means a fatal error from opening the file has occurred

SPECIAL CONSIDERATIONS

RUNNING CDC-6400 DECKS ON HP/3000

The following steps will enable you to run a CDC Fortran deck on to the HP/3000.

1. Remove all CDC control cards,

   e.g. Job card, FTN, LGO, EOR, EOF,...etc.

2. Replace

   ```
   PROGRAM TST(INPUT,OUTPUT,....)   card
   ```
   by
   ```
   $CONTROL FILE=5,FILE=6,LABEL,MAP card
   ```

   Any other files used in the programs should be declared similarly.

3. If you have any end-of-file test in the program

   e.g. IF (EOF(5)...,...)

   then it should be removed and the end-of-file test should be included in the appropriate READ statement,

   ```
   e.g.     READ (5,100,EOF=n)...,...
        100 FORMAT (.........)
   ```

   When an EOF occurs on logical unit #5, this statement will transfer control to label n.

4. The range of integer values on HP/3000 is limited to the range between -32768 and +32767. If you feel that an integer variable in your program may take a value beyond this range, you should declare it as a REAL variable (-and change the FORMAT statements, if necessary).

5. The deck should not contain calls to library subroutines that do not exist in the HP/3000 compiler library. You should consult the manual on HP/3000 Compiler Library.

6. If your program reads alphanumeric strings under FORMAT (...,A10,...), you will need to change the corresponding variables(s) to type CHARACTER *10.

7. If you have any doubts or questions please consult the programming assistant in Room 2D9.

...P.2

McMASTER UNIVERSITY
COMPUTATION SERVICES UNIT
HEALTH SCIENCES CENTRE
HAMILTON, ONTARIO, CANADA
L8S 4J9

TITLE CDCCONV

FUNCTION To convert a CDC-6400 FORTRAN program for use on HP/3000

AUTHOR K. Ahmed

LANGUAGE Text Editor command language

DESCRIPTION

CDCCONV is a USE- file to be used by HP/3000 Text Editor. It replaces the 026 punch characters by the corresponding 029 punch characters. It also replaces all the asterisks (*) by apostrophes (') in all the FORMAT statements.

SPECIAL CONSIDERATIONS

The CDCCONV use file does most of the work of conversion; however, some additional minor conversions have to be done manually.

CDCCONV uses a compiled subroutine CORRECT which must be stored in the Project SL.

EXAMPLE

```
/LIST ALL
                PROGRAM TST %INPUT,OUTPUT,TAPE5=INPUT,TAPE6=OUTPUT<
   2      C
   3      C  EXAMPLE OF A CDC PROGRAM CONVERTED TO H.P. USING THE EDITOR.
   4      C
   5              READ%5,12< N
   6      10      FORMAT%I4<
   7              DO 20 I=1,N
   8              READ%5,11< X
   9      11      FORMAT%F10.5<
  10              S=10.*SIN%X<
  11              C=10.*COS%X<
  12              CALL PLOTPT%X,S,4<
  13              CALL PLOTPT%X,C,5<
  14      20      CONTINUE
  15              CALL OUTPLT
  16              WRITE%6,30<
  17      30      FORMAT%1X,*PLOT OF SINE AND COSINE FUNCTIONS*,/,
  18             1 1X,*THE Y-AXIS IS SCALED UP BY A FACTOR OF 10.*<
  19              STOP
  20              END
/
/USE CDCCONV.PUB.LIB

*21*STRING NOT FOUND BEFORE LIMIT
AT DEPTH 3
    TRING NOT FOUND BEFORE LIMIT
A.  EPTH 3
/
/LIST ALL
   1              PROGRAM TST
   2      C
   3      C  EXAMPLE OF A CDC PROGRAM CONVERTED TO H.P. USING THE EDITOR.
   4      C
   5              READ(5,12) N
   6      10      FORMAT(I4)
   7              DO 20 I=1,N
   8              READ(5,11) X
   9      11      FORMAT(F10.5)
  10              S=10.*SIN(X)
  11              C=10.*COS(X)
  12              CALL PLOTPT(X,S,4)
  13              CALL PLOTPT(X,C,5)
  14      20      CONTINUE
  15              CALL OUTPLT
  16              WRITE(6,30)
  17      30      FORMAT(1X,'PLOT OF SINE AND COSINE FUNCTIONS',/,
  18             1 1X,'THE Y-AXIS IS SCALED UP BY A FACTOR OF 10.')
  19              STOP
  20              END
/
    EX4
```

Ignore these messages.

```
:JOB username.acctname
:EDITOR
/ADD
$CONTROL FILE=5,FILE=6,LABEL,MAP
     :
     :
     :
  (CDC source cards)
     :
     :
///
/USE CDCCONV.PUB.LIB
/LIST ALL
/KEEP filename
/END
:EOD
:FORTRAN filename
:PREPRUN $OLDPASS;PMAP
     :
  (Data cards)
     :
:EOD
:SAVE $OLDPASS, Progname
:EOJ
```

These commands convert the CDC punch (Ø26) to the HP punch (Ø29) and store the source as a permanent file under filename specified. This is required only on the first run.

(if you want to save the compiled program for future runs).

In case you do not want to save the source file, you should purge it at the end of the run,

```
     :
     :
:PURGE filename
:EOJ
```

otherwise it will occupy unnecessary file space.

If you have saved the compiled program (ref to MPE-JCL) (by :SAVE $OLDPASS, Programme), then in the future runs, the deck set-up should contain

```
:JOB username.acctname
:RUN Progname

     :
     :
  (Data cards)
     :
     :
:EOD
:EOJ
```

Please purge all unnecessary source and program files.

GENERAL I/O SUBROUTINES FOR HP S/3000

BY: Paul Balnys

I N D E X

McMASTER UNIVERSITY
COMPUTATION SERVICES UNIT
HEALTH SCIENCES CENTRE
HAMILTON, ONTARIO, CANADA
L8S 4J9

TITLE GENRLIO - RL file

FUNCTION Group of subroutines designed to handle free field input and output

AUTHOR P. Balnys

LANGUAGE FORTRAN and SPL

DESCRIPTION

The subroutines perform the input, conversion and output of data. Error recovery is left to the calling program. The correction of input can be performed interactively by the use of special instructions similar to those used by the EDITOR subsystem. Provision has been made for entering more than one response at a time as well as continuing over more than one line.

SPECIAL CONSIDERATIONS

Incorporate these subroutines into your program by:

PREP $ OLDPASS, $ NEWPASS; RL = GENRLIO

II. BASIC OBJECTIVES

1. To provide programmers using the HP S/3000 with the capability to handle free field input and output with minimum programming discomfort.

2. To facilitate standardization among programmers which will inhibit diversified language and machine dependence.

3. To enhance the standard FORTRAN format capabilities to include double integer representation. (i.e. integers out of the range of $\pm$32, 768.)

4. To save memory by ensuring that these subroutines are sharable (re-enterent) by different users simultaneously.

5. To allow the programmer to specify file characteristics; such as, input/output file numbers, length of input/output records, etc.

6. To provide an edit capability for user correction of invalid input.

7. To provide error recovery in control of the calling programme.

8. To perform conversion of numeric ASCII characters and special ASCII characters; such as, +,-,.,E,etc. into a specified internal representation and vice versa.

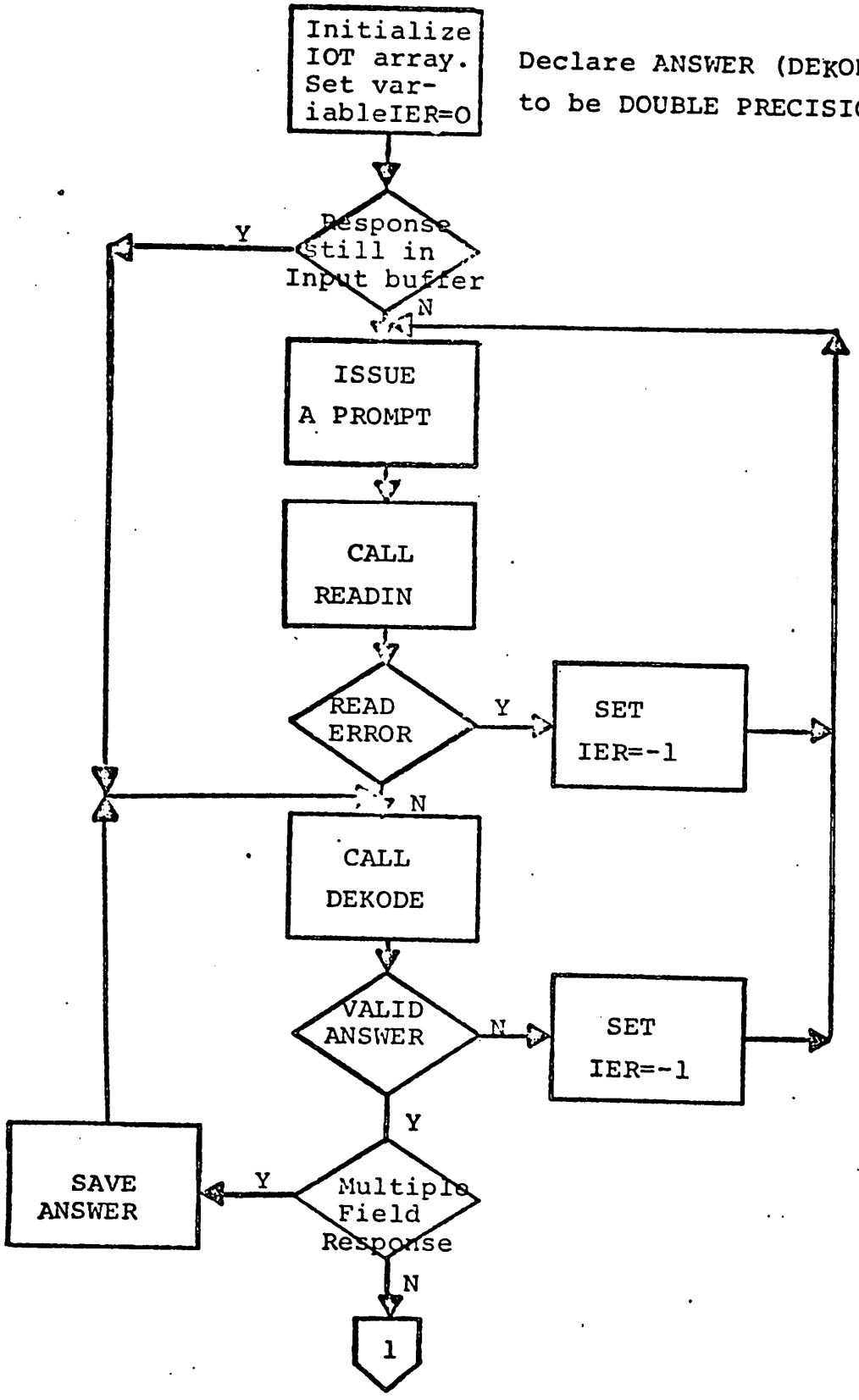
I. INTRODUCTION

The General I/O Subroutines were designed for and implimented on the HP S/3000.

There are machine-dependent characteristics of these subroutines, such as, the use of CHARACTER declarations in FORTRAN and the use of SPL (Systems Programming Language for HP S/3000).

Conversion to another system would be extensive because of the use of SPL and procedures intrinsic to HP S/3000 Operating System. The time required to do such a conversion is certainly dependent upon the installation, but would require somewhere in the neighborhood of three to four man-weeks.

V. PROGRAMME ORGANIZATION



III. INPUT REQUIREMENTS

1. To allow user to enter more than one response during one input operation.

2. To provide a continuation facility to allow a user to enter input over one record size.

3. To establish a meaning for several key symbols; such as, END, EXIT, etc.

4. To provide the facility for interpreting the symbols YES, NO, PROMPT LONG and PROMPT SHORT.

5. To provide interactive edit capabilities for invalid input corrections.

IV. OUTPUT REQUIREMENTS

1. To provide capabilities in moving characters from one buffer to another; from and to specified locations.

2. To facilitate character repetition when building an output buffer.

3. To provide spacing control when writing a buffer to a specified file.

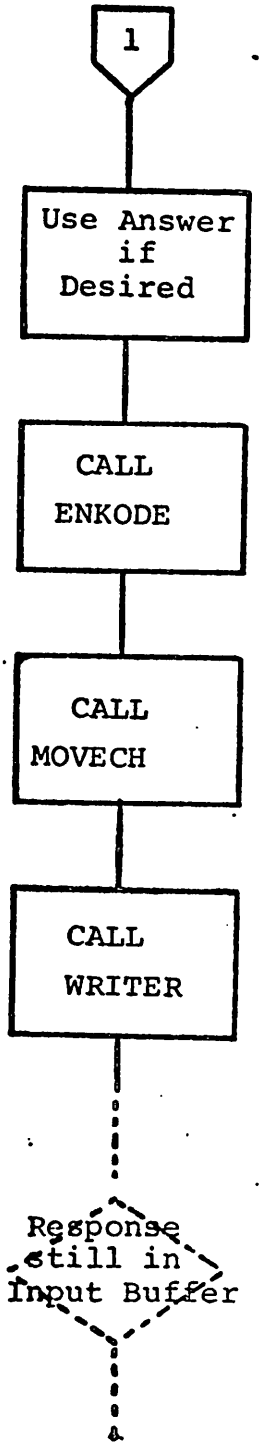
VI. LIMITATIONS AND ASSUMPTIONS

1. When correcting an invalid response, the user may use a maximum of 10 correction instructions, separated by blanks, at one time.

2. Double integer capabilities are not present in FORTRAN, but a set of ASCII characters can be converted to an internal double integer and back to ASCII again, using DEKODE and ENKODE respectively. The value returned from DEKODE must not be used in FORTRAN but can be passed directly to ENKODE.

3. Input records and output records are always assumed to be of character type.

4. Declare ANSWER (DEKODE parameter) to be DOUBLE PRECISION.



Repeat as Above.

PROGRAMME NAME: READIN

FUNCTION: To read a single or multiple line of ASCII characters from a specified input file into an input buffer.

To interpret and execute the field correction instructions, namely: MOD, I, D, R, "CR". (See Note:)

FORTRAN CALL: CALL READIN (IOT, INBUFF, IER)

ARGUMENT DESCRIPTION:

INPUT..IOT..... I/O table 12 elements long containing the following information:

IOT(1)... Input file number.

IOT(2)... Output file number.

IOT(3)... Input record length. (Indicates number of characters)

IOT(4)... Output record length. (Indicates number of characters)

IOT(5)... Input buffer size, i.e., the dimensions of INBUFF, a character array.

IOT(6)... Count of the number of remaining responses.

IOT(7)... Pointer to the next field in the input buffer.

IOT(8)... Number of characters in the input buffer.

IOT(9)... =0 to not interpret "YES" and "NO" as special responses.
=1 to map "YES" into 1 and "NO" into 2.

IOT(10).. =0 to indicate that long prompts are to be issued.
=1 to indicate that short prompts are to be issued.

IOT(11).. =0 means blanks are delimeters and commas are treated as alpha characters.
=1 means commas are delimeters and blanks are treated as alpha characters except preceeding blanks in a field are ignored.

IOT(12).. =0 means have processed an entire response.
>0 means have not processed an entire response. There is at least one field yet to be decoded.

OUTPUT..INBUFF..... The input buffer of ASCII characters

VII. SUBROUTINE DESCRIPTION

PROGRAMME NAME: DEKODE

FUNCTION: To decode the current field in the input buffer, INBUFF, into a specified internal (binary) representation. The conversion is performed on integer and real numbers only. If the string to be decoded is not integer or real, a flag is returned indicating an alphanumeric response has been detected (See Note 1:) except as follows:

1. If "YES" or "NO" is detected and IOT(9) indicates that interpreting is to be done, then 1 or 2 is returned respectively.
2. If "PROMPT LONG" or "PROMPT SHORT" is detected, then IOT(10) is set to 0 or 1 respectively.
3. Comments are enclosed by <<.....>> and are ignored.

FORTRAN CALL: CALL DEKODE (IOT, INBUFF, ANSWER, FTYP)

ARGUMENT DESCRIPTION:

INPUT..IOT..... I/O table described in READIN.

INBUFF.... The input buffer of ASCII characters to be decoded.

FTYP..... Indicates the type of internal representation desired:
=0 signifies alphanumeric type.
=1 signifies single word integer.
=2 signifies double word integer.
=3 signifies floating point number.
=4 signifies double precision floating point number.

OUTPUT..ANSWER.. Word or words containing the decoded number or pointer to first character of alphanumeric field in INBUFF.

Must declare ANSWER as DOUBLE PRECISION and equivalence it to a single integer and a real

read in with unnecessary (2 or more consecutive) blanks suppressed.

IER.. =0 if valid read.
=1 if End of Data encountered.
=-1 if physical end of file reached.
=-2 if buffer size exceeded.

NOTE:

For an interactive user, special editing instructions may be used to correct invalid responses.

Upon detecting an invalid response, the calling programme may issue an errror message, the prompt at which the invalid response occurred and call READIN.

At this time, the user may enter the MOD instruction which means modify. Upon detecting the MOD instruction, the incorrect response is displayed, so that, the user may indicate the character(s) to be changed through three other instructions, namely:

Ichar	-	Insert character(s) after this point.
Dchar	-	Delete character at this point.
Rchar	-	Replace character(s) at this point.

More than one of the above three instructions may be entered on the same line or on different lines to correct more than one part of the invalid response.

To leave the modify mode, the user has only to enter a carriage return, whereupon, control returns to the calling programme ready to enter the decoding routine.

The modify instructions do not alter any remaining responses already in the input buffer.

After an error has occurred and the MOD instruction is not used, then, the response entered will simply replace what was ever in the input buffer and decoding will resume.

After an error has occurred and the MOD instruction is used but the correction instructions are not, then, the response entered will replace whatever was in the input buffer and decoding continues.

Can have a maximum of 10 corrections at one time. If you have I's or R's then one blank is inserted.

PROGRAMME NAME: ENKODE

FUNCTION: To convert an internal (binary) number to ASCII characters according to a specified format.

FORTRAN CALL: CALL ENKODE (VALUE, TYPE, KIND, WIDTH, DIGITS, STRING).

ARGUMENT DESCRIPTION:

INPUT..VALUE.... The variable containing the number to be converted.

TYPE..... The type of internal representation.
=1 for integer.
=2 for double integer.
=3 for real.
=4 for double precision real.

KIND..... The kind of conversion desired.
=0 signifies default, i.e., if single word integer use I6 format, if double word integer use I12 format, if floating point use Gw.d format.
=1 signifies Iw.
=2 signifies Fw.d.
=3 signifies Ew.d.
=4 signifies Dw.d.
=5 signifies Nw.d.
=6 signifies Mw.d.

WIDTH.... Width of entire ASCII string including all special characters.

DIGITS... Number of fractional digits desired.

OUTPUT.STRING... Output array containing the ASCII characters

NOTE:

If an error occurs, such as WIDTH too small for result specified by KIND, if DIGITS <0 or WIDTH ≤ 0, then, the output STRING will contain #'s.

OUTPUT..FTYP..... Type of conversion performed if any.

<0 indicates the number of character within the alphanumeric field encountered.

=0, for Blank field found. i.e. No characters in response.

=1 for single word integer.

=2 for double word integer.

=3 for floating point number.

=4 for double precision floating point number.

=11 signifies "HELP" detected.

=12 signifies "END" detected.

=13 signifies "EXIT" detected.

=14 signifies "PAUSE" detected.

NOTE 1:

If an alphanumeric response (not real or integer) has been detected then ANSWER is a single word integer which points to the first character of this response in INBUFF and the absolute value of FTYP specifies the number of characters in this response. Because of this, one must equivalence ANSWER to an integer variable and use the integer variable when FTYP<0.

NOTE 2:

A response may contain one or more fields delimited by blanks or commas. Each field is decoded one at a time, returning the decoded result and type specified. When the end of a response (last field decoded) is reached, IOT(12) is set to 0.

PROGRAMME NAME: WRITER

FUNCTION: To write the ASCII characters in the array OUTLIN from a specified starting and ending position.

FORTRAN CALL: CALL WRITER (IOT, OUTLIN, IBEG, IEND, CARR).

ARGUMENT DESCRIPTION:

INPUT..IOT.....	The I/O table described in READIN.
OUTLIN....	The character array containing the characters to be written to the specified file.
IBEG.....	Indicates the starting position in OUTLIN to begin writing.
IEND.....	Indicates the ending position in OUTLIN for writing.
CARR.....	Carriage control indicator. (See carriage control codes).

SPECIAL NOTE:

If IEND is negative, the character array OUTLIN is scanned up to the last non-blank character and written to the output file starting from OUTLIN(1).

At any time, if there are more characters in OUTLIN, the output record size will hold, an automatic skip to the next line is performed and the remaining characters are written to the output file with single spacing.

CARRIAGE CONTROL CODES

INTEGER CODE	MEANING
3	- triple spacing
2	- double spacing
1	- single spacing
0	- no spacing
-1	- skip 1 page
-2	- skip 2 pages
-3	- skip 3 pages

PROGRAMME NAME: MOVECH

FUNCTION: To move a string of ASCII characters from array STRING to array OUTLIN from and to a specified location. Also, to provide for character repetition in OUTLIN.

FORTRAN CALL: CALL MOVECH (STRING, IBEG, NCHAR, OUTLIN).

ARGUMENT DESCRIPTION:

INPUT..STRING.... Character array containing the ASCII characters to be moved. First character in location 1.

IBEG..... >0 signifies transfer from STRING(1) to OUTLIN (IBEG) going from left to right.
<0 signifies transfer from STRING (NCHAR) to OUTLIN (IBEG) going from right to left.

NCHAR..... Character counter indicator.
≥0 signifies the number of characters in STRING starting at location 1.
<0 means repeat the character found in STRING(1) the absolute value of NCHAR times.

OUTPUT..OUTLIN... Character array receiving the string of characters.

APPENDIX B:

KEY SYMBOLS AND MEANING

SYMBOL		MEANING
$	-	Continuation line character (last character on the line).
&	-	Delimiter between responses in a multiple response.
" " or ","	-	Delimiters between fields in a response. Two consecutive commas signify a null or unknown field in a response. May use one or the other as delimiters, not both. (See IOT(11) description)
END	-	To signify the end of the current minor level of inquiry.
EXIT	-	To indicate that control is to return to the previous major level of inquiry.
PAUSE	-	To indicate a break during interaction. The calling programme is to issue messages as to what information to retain and then release the system. The system can be re-entered at this same point at a later time.
HELP	-	Signifies to the calling programme that an information block is to be accessed and information extracted to aid the user during this interaction.
PROMPT LONG	-	Sets IOT(10) to 0.
PROMPT SHORT	-	Sets IOT(10) to 1.

APPENDIX A:

FIELD CORRECTION INSTRUCTIONS

INSTRUCTION		MEANING
MOD	-	To indicate to the I/O Routines that correction instructions are to follow so the last response is re-written to the interactive device.
Icccc...c	-	To insert one or more contiguous characters at the location after the I. If a blank is to be inserted, only one at a time can be inserted.
D	-	To delete the character directly above the D.
R	-	To replace the character(s) directly above the R.
	-	If a blank is to be the character, only one blank at a time can be done.
"CR"	-	To leave the MODify mode and return to the calling programme.
$	-	To skip the number of characters in the buffer specified by IOT,3).

NOTE:
Allowed a maximum of 10 correction instructions at one time.

PLOTTING ROUTINES

THE PLOTTING ROUTINES PLOTPT AND OUTPLT, AVAILABLE ON CDC/6400 HAVE BEEN MODIFIED FOR USE ON HP/3000.

THESE ROUTINES ENABLE A ROUGH GRAPH TO BE PLOTTED BETWEEN PAIRS OF (X,Y) VALUES. EACH PLOT OCCUPIES ONE LINE-PRINTER PAGE (132 CHARACTERS WIDE) A LINE AT THE BOTTOM IS AVAILABLE FOR A HEADING. UP TO 700 POINTS CAN BE PLOTTED ON EACH PLOT. THE SCALE IS CHOSEN AUTOMATICALLY TO BE REASONABLY "NICE", BUT THE USER CAN SPECIFY HIS OWN SCALE BY MEANS OF A CALL TO THE SCALE SUBROUTINE.

A DESCRIPTION OF EACH OF THE ROUTINES IN THE PLOT PACKAGE FOLLOWS:

A) THE FIRST STEP IS TO INITIALIZE THE PLOTTING MATRIX BY THE CALL

CALL INITPLT(N)

WHERE N IS THE FORTRAN LOGICAL DEVICE NUMBER ON WHICH THE PLOT IS TO APPEAR.

B) TO PLACE THE COORDINATES (X,Y) OF A POINT IN THE CURRENT PLOT TOGETHER WITH CODE-NUMBER OF THE CHARACTER TO BE PLOTTED, WE EXECUTE

CALL PLOTPT(X,Y, CODE-NUMBER)

X AND Y MUST BE *real* QUANTITIES (OR EXPRESSIONS) GIVING THE COORDINATES AND THE "CODE-NUMBER" IS AN INTEGER THAT CORRESPONDS TO CHARACTER AS TABULATED ON THE NEXT PAGE.

McMASTER UNIVERSITY
COMPUTATION SERVICES UNIT
HEALTH SCIENCES CENTRE
HAMILTON, ONTARIO, CANADA
L8S 4J9

TITLE PLOT ROUTINES

FUNCTION To plot a graph between two variables on line-printer.

AUTHOR Converted to HP/3000 by K. Ahmed

LANGUAGE FORTRAN

DESCRIPTION

The package consists of the following subroutines:

INITPLT (n) - initializes plot matrix; n= logical device no. for output file

PLOTPT(x,y,m) - stores the point (x,y) in the plotting matrix; m is an integer code for the plot-character to be used.

OUTPLT - when all the points have been stored, a call to OUTPLT prints out the graph on the specified file (logical device #n,

SCALE (xmin, xmax, ymin, ymax), specifies the scale of plot [optional]

SPECIAL CONSIDERATIONS

Upto 700 points can be plotted on the graph. If SCALE is not specified, a suitable scale is automatically chosen.

A Simple Example (The resulting Plot appears on the next page)

```
:JOB username.accountname
:FORTRAN
$CONTROL FILE=5,FILE=6,LABEL,MAP
      PROGRAM TST
      CALL INITPLT (6)
      DO 1 I=1,50
      X=I-1
      Y=SIN(0.1*X)
      CALL PLOTPT(X,Y,4)
      CALL PLOTPT(X,COS(0.1*X),46)
1     CONTINUE
      CALL OUTPLT
      WRITE (6,10)
10    FORMAT (1X,'SIN/COS')
      STOP
      END
:EOD
:PREPRUN $OLDPASS,$NEWPASS;RL=PLOTRL.PUB.LIB
:EOJ
```

C) IF WE WISH TO CHOOSE THE SCALES OF THE PLOT, WE EXECUTE (*optional*)

CALL SCALE (XMIN, XMAX, YMIN, YMAX)

POINTS (X,Y) WILL NOW BE PLOTTED ONLY IF

XMIN $\leqslant$ X $\leqslant$ XMAX, YMIN $\leqslant$ Y $<$ YMAX

D) WHEN ALL THE POINTS HAVE BEEN PLACED WE CAN CAUSE THE PLOT TO BE PRINTED BY EXECUTING

CALL OUTPLT

AT THE END OF THIS CALL THE PLOTTING MATRIX IS AUTOMATICALLY RE-INITIALIZED AND THE SCALE IS RESET TO THE DEFAULT OPTION.

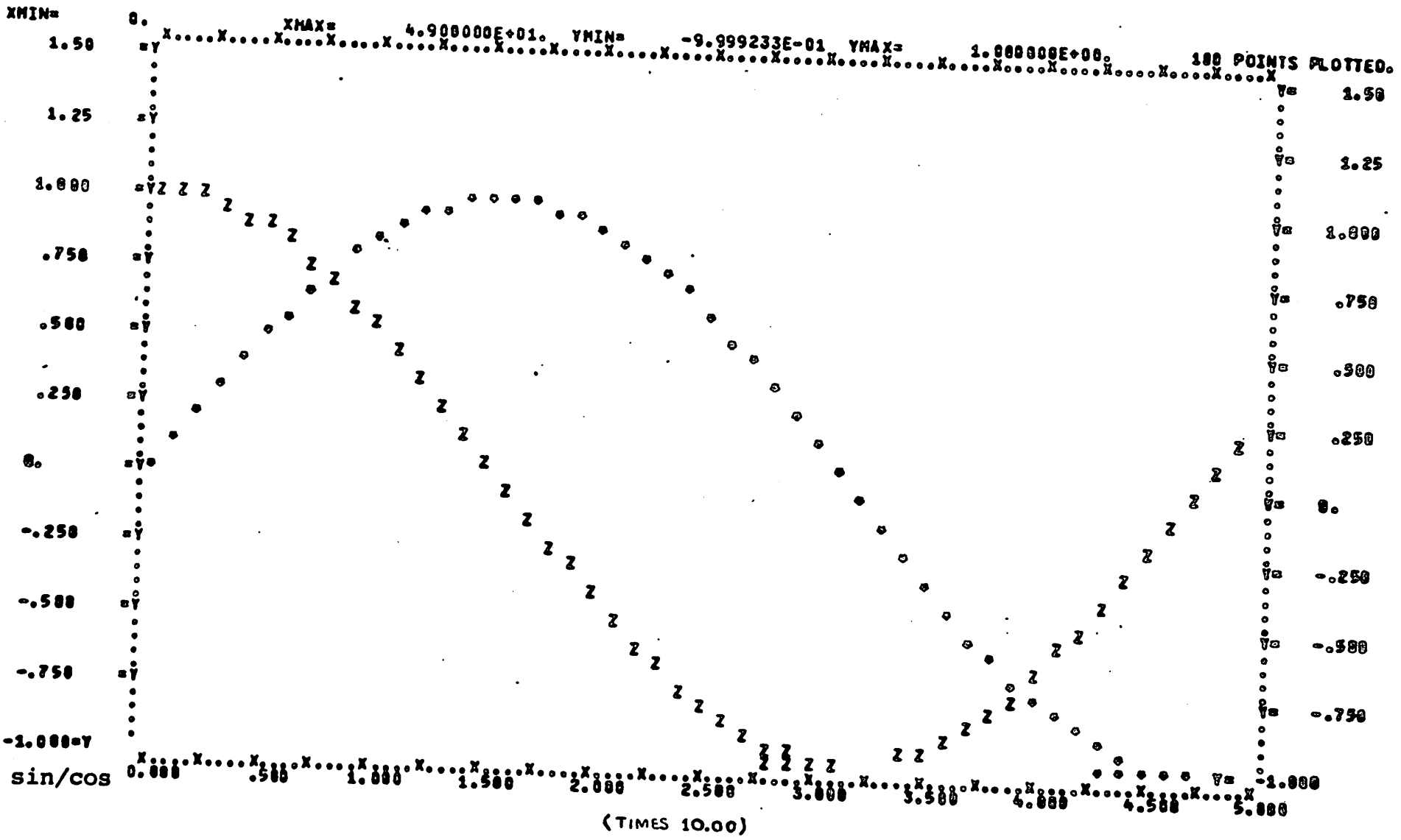
E) IN ORDER TO USE THESE SUBROUTINES, THE USER MUST SPECIFY THE PLOT LIBRARY IN THE PREP (OR PREPRUN) COMMAND, E.G.

:PREP $OLDPASS, $NEWPASS;RL=PLOTRL.PUB.LIB

Table of Character Values

Character	Value	Character	Value	Character	Value	Character	Value
(blank)	0	0	10	A	21	N	34
=	1	1	11	B	22	Ø	35
+	2	2	12	C	23	P	36
-	3	3	13	D	24	Q	37
*	4	4	14	E	25	R	38
/	5	5	15	F	26	S	39
(	6	6	16	G	27	T	40
)	7	7	17	H	28	U	41
,	8	8	18	I	29	V	42
.	9	9	19	J	30	W	43
		$	20	K	31	X	44
				L	32	Y	45
				M	33	Z	46

III. STATISTICAL


XMIN= 0.
XMAX= 4.900000E+01.
YMIN= -9.999233E-01
YMAX= 1.000000E+00.
100 POINTS PLOTTED.
1.50
1.25
1.000
.750
.500
.250
0.
-.250
-.500
-.750
-1.000
sin/cos
0.000
.500
1.000
1.500
2.000
2.500
3.000
3.500
4.000
4.500
5.000
(TIMES 10.00)

McMASTER UNIVERSITY
COMPUTATION SERVICES UNIT
HEALTH SCIENCES CENTRE
HAMILTON, ONTARIO, CANADA
L8S 4J9

TITLE CCSS (Conversational Computer Statistical System)

FUNCTION Provides basic descriptive statistical procedures with subsetting capability.

AUTHOR R. Kronmal

LANGUAGE FORTRAN

DESCRIPTION

This interactive system consists of a set of programs for managing and analysing data sets with miltiple record types. It requires no computer programming experience to use.

The system provides options for:

1. Code sheet entry
2. Clean data file
3. Put data into file
4. Tabling
5. Listing
6. Scattergram plot
7. Summary
8. Cumulative density plot
9. Bar graph

SPECIAL CONSIDERATIONS

The system requires the interim spooling mechanisms as explained in CARD and PRINTER.

GROWTH/CCSS

McMASTER UNIVERSITY
COMPUTATION SERVICES UNIT
HEALTH SCIENCES CENTRE
HAMILTON, ONTARIO, CANADA
L8S 4J9

TITLE CHISQ

FUNCTION Analyses 2-way contingency tables

AUTHOR C. Goldsmith/P. Balnys

LANGUAGE FORTRAN

DESCRIPTION

This is an interactive program providing the following options for analysing 2-way contingency tables with up to 20 rows and up to 20 columns.

1) Data input
2) Printout of table
3) Expected values, chi-square contributions and percentages
4) Chi-square
5) Fisher-Irwin Exact Test (for 2 x 2 tables)
6) Tests of Symmetry and Agreement (for square tables)
7) Comparison of Columns
8) Correlation Analyses
9) Ability to analyse another table

SPECIAL CONSIDERATIONS

This program requires the generalized I/O routines (GENRLIO)

McMASTER UNIVERSITY
COMPUTATION SERVICES UNIT
HEALTH SCIENCES CENTRE
HAMILTON, ONTARIO, CANADA
L8S 4J9

TITLE TRAND

FUNCTION generates uniform random numbers between 0.0 and 1.0

AUTHOR

LANGUAGE SPL

DESCRIPTION

TRAND is a subroutine with calling sequence

CALL TRAND (Y,R).

Y is an output parameter containing the random number

R is the seed for the random number generator

R is set for the first call to TRAND and returns as the seed for the next call to TRAND.

SPECIAL CONSIDERATIONS

McMASTER UNIVERSITY
COMPUTATION SERVICES UNIT
HEALTH SCIENCES CENTRE
HAMILTON, ONTARIO, CANADA
L8S 4J9

TITLE SAMPLE

FUNCTION Selects random sample groups from a given population

AUTHOR J.W. Bush

LANGUAGE FORTRAN

DESCRIPTION

This program is run interactively. The user enters the size of the population from which the sampling to be done, and the size of the samples required. He then enters a number between 0 and 1 for the random number generator (TRAND). The selected samples are printed out in sorted order.

SPECIAL CONSIDERATIONS

required subroutines: RSMPL
TRAND
SHUFL
SORT

limitations: the combined total of the samples must be less than 5000.

IV. EDUCATION

McMASTER UNIVERSITY
COMPUTATION SERVICES UNIT
HEALTH SCIENCES CENTRE
HAMILTON, ONTARIO, CANADA
L8S 4J9

TITLE MACMAN

FUNCTION Simulation of Blood circulation in Human Body

AUTHOR C.J. Dickinson

LANGUAGE FORTRAN

DESCRIPTION

"MACMAN" is a synthetic person who has a heart inside a chest, systemic arteries and arterioles, a capillary bed, and veins collecting blood from the capillary bed and returning it to the heart. "MACMAN" thus has a complete systemic circulatory system, and when the heart is working it will circulate blood. To speed up computation, the heart is treated as a single chamber filling the right atrium and pumping blood out into the aorta. The pulmonary circulation is regarded as simply a parallel path, and not (as in life) in series with the systemic circulation. However, this makes the model unrealistic only when one side of the heart is able to pump much less than the other (e.g. because of valve disease). "MACMAN" cannot therefore simulate the effects of valve lesions but it can simulate most types of generalized heart disease. "MACMAN" also possesses synthetic baro-receptors similar in operation to these which are normally situated at the bifurcation of the common carotid artery and at the aortic arch. These act in such a way as to stabilize blood pressure.

SPECIAL CONSIDERATIONS

McMASTER UNIVERSITY
COMPUTATION SERVICES UNIT
HEALTH SCIENCES CENTRE
HAMILTON, ONTARIO, CANADA
L8S 4J9

TITLE MACPUF

FUNCTION Simulation of Respiratory System

AUTHOR C.J. Dickinson

LANGUAGE FORTRAN

DESCRIPTION

"MACPUF" is a computer model of the lungs and airways, the pulmonary and systemic circulation, the tissues and the brain which is designed to study gas transport between lungs and tissues, the control of ventilation, hydrogen in regulation, and complex interactions between them.

SPECIAL CONSIDERATIONS

McMASTER UNIVERSITY
COMPUTATION SERVICES UNIT
HEALTH SCIENCES CENTRE
HAMILTON, ONTARIO, CANADA
L8S 4J9

TITLE MACPEE

FUNCTION Simulation of Kidney and Body Fluids

AUTHOR C. J. Dickinson

LANGUAGE FORTRAN

DESCRIPTION

"MACPEE" is a digital computer model of the systemic circulation and kidneys designed to study the circulation and the kidneys together. The heart and systemic circulation are virtually identical with those in "MACMAN" and initial values for systemic arterial and venous resistances, cardiac contractility, blood volume and blood viscosity, and baroreceptor stabilizing function are in general the same as in "MACMAN"; but "MACPEE" operates over periods of days whereas "MACMAN" operates over minutes. "MACPEE" is a very much more complex model since it incorporates not only the whole of "MACMAN" but also has:-

1) kidneys, excreting water, urea, sodium and potassium

2) a gut absorbing water, protein and electrolytes

3) thirst, governing water control in accordance with the body's needs

4) an endocrine system with individual control of the secretion rates of vasopressin (antidiuretic hormone), aldosterone, and angiotensin.

SPECIAL CONSIDERATIONS

McMASTER UNIVERSITY
COMPUTATION SERVICES UNIT
HEALTH SCIENCES CENTRE
HAMILTON, ONTARIO, CANADA
L8S 4J9

TITLE CPR (Cardio-pulmonary Resuscitation)

FUNCTION A question-answer dialogue to take the student through steps of Cardio-pulmonary Resuscitation.

AUTHOR Dr. E. Hoffer (Mass. Gen. Hosp.) adapted to H.P./3000 by K. Ahmed.

LANGUAGE FORTRAN

DESCRIPTION

A Fortran driver program is used to take the student through a text file (consisting of card-image records) through multiple choice question-answer sequences. Usually the questions are repeated until the correct answer is typed; in some cases explanation and instructions are provided.

The program also refers the student to a booklet containing Electro-cardiogram traces.

SPECIAL CONSIDERATIONS

The driver program is fairly general and could be used on other text files of similar nature.

McMASTER UNIVERSITY
COMPUTATION SERVICES UNIT
HEALTH SCIENCES CENTRE
HAMILTON, ONTARIO, CANADA
L8S 4J9

TITLE THYROID

FUNCTION Refer to description

AUTHOR G.D. Anderson

LANGUAGE fortran

DESCRIPTION

The THYROID Programme estimates the probability that a patient has underactive, normal, or overactive thyroid function based on symptoms observed on the patient. The probabilities calculated by this programme are based on data from 879 patients with thyroid disease.

SPECIAL CONSIDERATIONS

McMASTER UNIVERSITY
COMPUTATION SERVICES UNIT
HEALTH SCIENCES CENTRE
HAMILTON, ONTARIO, CANADA
L8S 4J9

TITLE BONETUMR

FUNCTION Refer to description

AUTHOR G. D. Anderson

LANGUAGE FORTRAN

DESCRIPTION

The BONE TUMOR Programme is an example of the use of elementary probability and an expression called Bayes Theorem to estimate the relative probabilities for several types of Bone Tumor given specific radiological observations on a patient with a Bone Tumor present. The probability estimates made by this Programme are based on information from a large number of patients with primary Bone Tumors which is stored within the Programme.

SPECIAL CONSIDERATIONS

McMASTER UNIVERSITY
COMPUTATION SERVICES UNIT
HEALTH SCIENCES CENTRE
HAMILTON, ONTARIO, CANADA
L8S 4J9

TITLE BPFILM

FUNCTION Refer to description

AUTHOR C. Goldsmith

LANGUAGE FORTRAN

DESCRIPTION

This program will help you to analyse your own "OBSERVER" variation in the clinical measurement of Blood Pressure. The pre-requisite for running this program consists of going through Film #MP 21 entitled "Blood Pressure Readings" and writing down your Blood Pressure measurements for each of the 14 patients.

SPECIAL CONSIDERATIONS

V. USER TRAINING

FORTRAN

McMASTER UNIVERSITY MEDICAL SCHOOL

HAMILTON, ONTARIO

F O R T R A N

Introduction

The FORTRAN/3000 Language is based upon the ANSI Standard FORTRAN. In addition, FORTRAN/3000 has many extensions which expand the capabilities and increase the power of the language.

Example of a FORTRAN Program

Following is an example of a FORTRAN Source Program, illustrating a few of the extensions to the language. These are indicated by bracketted numbers across from a particular line and are explained below the example.

EXAMPLE:

```
1)      $CONTROL LABEL, MAP, FILE=4
            PROGRAM TEST
            DIMENSION ID(10)
2)          CHARACTER  X( 5)
            I=4
3)       5  READ (I,10,END=30)ID,X
            WRITE(6,20)ID(1), ID,X
        10  FORMAT(10I5,5A1)
4)          WRITE(8@ID(1),10)ID,X
5)      20  FORMAT(1X,"RECORD NUMBER",
            I6,/,1X,10I6,5(1X,A1))
            IF(ID(1).LT.10)GO TO 5
        30  STOP
            END
```

Special Features of FORTRAN/3000

1) The $CONTROL Command indicates list and compilation options.

2) Arrays and variables of character type may be declared. Each element in the array occupies one-half of a 16-bit word.

3) In the read statement there is an action label reference i.e. upon detection of an end-of-file control will branch to the statement labelled with the number 30.

4) In this write statement, there is reference to a direct-access file. The value of the arithmetic expression following the "@" character, indicates the record number.

FORTRAN

5) In addition to the "nH" representation for outputting Hollerith characters, the characters may simply be enclosed in quotes.

*The following pages describe further, the special features of HP S/3000 FORTRAN. For more detailed descriptions, please refer to the pages in the FORTRAN Reference Manual as indicated on the right hand side in brackets.

ANSI Standard FORTRAN.

HP S/3000 FORTRAN conforms to the American National Standard Institute's Standard for FORTRAN.

Non-Standard Features. (FORTRAN B-1, B-4)

There are some restrictions and extensions on S/3000 FORTRAN compared to ANSI FORTRAN, of which the most important are:

1) Character type data may be used.

2) An array may have up to 255 dimensions.

3) Direct-access files may be referenced.

CDC 6400 vs HP S/3000.

There are a few differences between HP S/3000 FORTRAN and CDC 6400 FORTRAN, such as:

1) The character *, in CDC FORTRAN, is used in format statements to signify Hollerith fields but in HP S/3000 FORTRAN, the " must be used.
2) In CDC FORTRAN, file specifications are indicated on the Program Card and compiler options on the FTN Card, whereas, in HP FORTRAN, the file specifications and compiler options are indicated on the $CONTROL Card before the Source Program.
3) HP FORTRAN allows the Program Card to be optional, i.e., it may or may not be included.
4) The word sizes differ between the two Computers. CDC 6400 uses a 60 bit word, while HP S/3000 uses a 16 bit word. The smaller word size restricts the accuracy and the absolute value which may be stored in a word.

Word Size

NOTE: The largest and smallest values a simple integer may have in HP S/3000 are respectively, 32767 and -32768.

Accuracy (FORTRAN 2-17)

$CONTROL Card (FORTRAN 11-2, 11-4)

There is a FORTRAN statement which precedes each main or subprogram unit and defaults if omitted. It begins with the characters $CONTROL starting in column one. Following the letters $CONTROL are various key words separated by commas, indicating list and compilation options of which the

most useful are:

1) LABEL - this causes the compiler to emit a list of labels and their respective addresses after each compiled program unit. This is useful in tracing back execution errors.

$CONTROL LABEL

2) MAP - this parameter sends a symbol table dump to the list-file after each program unit. This listing gives the type of all variables and their relative addresses in the program unit.

$CONTROL MAP

3) FILE=*integer* - this parameter is required for files whose FORTRAN file numbers do not explicitly appear in the I/O statement, such as READ (I,10) ID,X where I has an assigned value.

$CONTROL FILE=*integer*

e.g. $CONTROL LABEL, MAP, FILE=4

The CHARACTER declaration is a Type Statement indicating that the variables and arrays following the declaration are of type character. The length of character symbolic names can be specified in two ways:

CHARACTER (FORTRAN 4-16)

1) Through the length attribute following the CHARACTER heading (*x).

e.g. CHARACTER*80 BUFF1

2) Through individual length attributes following character symbolic names.

e.g. CHARACTER BUFF2(80)

In the first example, BUFF1 is referred to as a character variable with length 80. In the second example, BUFF2 is referred to as a character array with 80 elements.

Elements in the character array are referred to in the same way as in integer arrays.

e.g. BUFF2(9)="Z"

Elements in the character variable are referred to differently. The beginning element and number of consecutive elements must be specified.

e.g. BUFF1[3:5]="ABCDE"

This example indicates that the string, ABCDE, will be placed in the character variable BUFF1 starting at the third element.

Free Format
(FORTRAN 9-43,9-45)

Free-field input/output is formatted conversion according to format and/or edit control characters imbedded in the data. The FORMAT Statement is not required. For free-field input/output, an * instead of a FORMAT Statement indentifier is used.

Input

e.g. READ(5,*)list elements

The data items on file 5 to be read are separated by data item delimiters which may be a comma, a blank space, or any ASCII character that is not a part of the data item.

Output
(FORTRAN 9-46)

For free-field output, predefined field and edit descriptions are used.

End-of-file Test
(FORTRAN 8-3)

Action Label References may be used in read or write statements. The most commonly used action label reference is in the form:

```
READ(file, format label,
     END=label)
```

If an end-of-file is encountered then control is transferred to the statement indicated by the label after END=, at which point the programmer decides upon the course of action.

e.g. READ(5,10,END=30)

NOTE: There is also an ERR=label reference for error conditions.

Sequential Files (FORTRAN 8-1,8-2,8-5,9-2)

Two types of access to files on disc devices are available through the file system. These are sequential or direct.

In sequential access, as many records as necessary are used in sequence until the entire list of variables has been transmitted. After having read or written one record on a sequential file, the record pointer is automatically moved to the next record.

Direct-access Files (FORTRAN 8-1,8-2,8-5,9-2)

Direct-access features in FORTRAN allow the access of any record in a file at random. The form of a direct-access read is:

```
READ (file@record,10)list elements
```

where file is a positive integer constant or variable indicating the file unit number and record is an arithmetic expression, constant or integer variable, the value of which indicates a specific record.

```
e.g.  WRITE(8@ID(1),10)ID,X
```

This will cause the values of ID and X to be written to file 8 in the record specified by the value of ID(1) and according to the format referenced by label 10.

Equating logical unit numbers with actual file names (Section II-JCL)

When reading or writing to other than standard input or output files (files 5 and 6), a file equate must be done before executing the program.

The name for a FORTRAN logical file is created by concatenating the word FTN with the two digit representation of the integer name.

```
e.g.  file 8 is FTN08
```

This name, FTN08, is the Formal File Designator.

File Command

To be able to read or write on an existing permanent file or temporary file, a correspondence between the name of the file, i.e. the Actual File Designator, and the Formal File Designator must be established. The files are equated using the File Command.

Permanent Files

For example, there is an existing permanent file by the name of *XXX* and the programmer wishes to read and/or write on this file through file unit number 3.

Before running the program, the following file equate is performed:

```
:FILE FTN03=XXX,OLD
```

In some cases, a temporary file is required meaning records do not have to be permanently saved. An example of this would be writing card images to a file and then reading them back as required. A typical file equate for a temporary file is:

```
:FILE FTN02;REC=-80,,,ASCII
```

Temporary Files

NOTE: An Actual File Designator was not required since a permanent file was not referenced. All records in this file will be lost upon completion of the program.

@.USER.SUPPORT

PROGRAM	SOURCE
ACUMLOG	SACUMLOG
	SCOSTING - subroutine
	SDATE - subroutine
BONETUMR	SBONETMR
CARD	SCARD
CCSS	SCCSS
	SASORT - subroutine
	SCLEAR - "
	SGP1 - "
	SGP2 - "
	SGP3 - "
	SGP4 - "
	SGPLOT - "
	SLISTS - "
	SRRCD - "
	STMN1 - "
	STRAM - "
	SBARG - "
SL (CDC Conversion Program.) (SL file)	SCDCCONV - (ASCII USE FI
	SCORRECT - subroutine
CCSS TEST FILES	CGROWTH
	DGROWTH

PROGRAM	SOURCE
CHISQ	SCHISQ
CHITREND	SCHITRND SPROB - subroutine STREND - subroutine
CPR	SCPR CPRDATA - data file
GENRLIO (RL file)	SGENIO
KSAMP	
MACMAN	SMACMAN
MACPEE	SMACPEE
MACPUF	SMACPUF
MTHLOG	SMTHLOG SDATE - subroutine
PLOTRL (RL file)	SPLOTPT
PRINTER	SPRINTER
SAMPLE	

PROGRAM	SOURCE
	SBUSY - (subroutine BUSYTEST)
THYROID	STHYROID
SUMLOG	SSUMLOG
	TRAND - subroutine.

NATIONAL BANK OF DETROIT

11/23/73

SUBJECT: PRINTF
Disc to Printer Utility Program

ORIGINATOR: Thomas M. Root
EDP Systems - 10th Floor
National Bank of Detroit
611 Woodward Avenue
Detroit, Michigan 48226
Phone (313) 225-3699

PRINTF is a utility program for printing ASCII files. It was written in order to implement a form of pseudo-spooling, in which any programatic output intended for the line printer would first be written to a disc file and printed later; but it is also useful for listing data and program source files.

The program examines the CCTL f-option of the input file to determine whether the first character of each record is to be used as a carriage control character or printed as data. If the CCTL bit is off, the entire record is printed using single spacing with automatic page eject. The file name is printed as a heading on the first page. If the CCTL bit is on, the user is asked if the records are to be pre-spaced or post-spaced and whether to use or suppress automatic page eject.

After printing one file, the program will ask for another file until a blank line (carriage return) or :EOD is entered. The user may terminate the printing of a file by typing a control Y.

```
:RUN PRINTF.PUB.SYS

FILE PRINT PROGRAM

FILE?    { filename [/lockword] [.group [.account]] }
         { *formal designator                       }
```

If the CCTL bit is off, the program will begin printing the file; otherwise it will ask: (default response is underlined)

```
PRE- OR POST-SPACING?    { PRE  }
                         { POST }

SUPPRESS AUTO EJECT?     { Y }
                         { N }
```

After each file is printed, the program will ask for another file until a blank line or :EOD is entered.

NOTES

Pre-spacing and post-spacing:

In general, output from the compilers and other MPE subsystems such as the segmenter is post-spaced. Output from a Fortran program is, by convention, pre-spaced.

Auto page eject:

The line printers used on the HP 3000 provide two techniques for slewing paper: the first is a VFU* slew in which the paper is advanced until a hole is found in the specified channel of the VFU tape, the second is a count down slew in which the paper is advanced and a line counter is decremented until the line count has been counted down to zero. Automatic page eject has been implemented by using a VFU slew on channel 3 whenever single spacing is desired. The standard VFU tape supplied by Hewlett Packard contains a hole punched in channel 3 for every line except lines 1 - 3 and 64 - 66 of an 11 inch form. Whenever a special VFU tape is made for non-standard length forms, the user must either punch channel 3 for every line or switch to a count down slew. Count down slew may be selected either explicitly by specifying a carriage control character of the form %2nn as indicated in figure 6-3 of the MPE manual (page 6-27) or by suppressing the auto eject feature of the driver by answering YES to the question "SUPPRESS AUTO EJECT?".

*Vertical Forms Unit

USER'S GROUP SOFTWARE LIBRARY

ACCOUNT= SUPPORT GROUP= USERS

FILENAME	CODE	LOGICAL RECORD SIZE	TYP	EOF	LIMIT	R/B	SPACE SECTORS	#X	MX	ACC
ACSCHEMA		80B	FA	315	315	3	106	16	16	???
ACUMLOG	PROG	256B	FB	19	19	1	20	1	1	???
ANACOMS		80B	FA	51	816	3	18	1	16	???
ANSWER1		80B	FA	10	110	3	38	1	1	???
ANSWER2		80B	FA	22	122	3	42	1	1	???
ASCREAL		80B	FA	5	105	3	36	1	1	???
BESSELF0		80B	FA	26	416	3	18	2	16	???
BESSELF1		80B	FA	28	448	3	20	2	16	???
BESSELM0		72B	FA	26	416	3	18	2	16	???
BONETUMR	PROG	256B	FB	29	29	1	30	1	1	???
CALENS		80B	FA	222	3552	3	75	1	16	???
CARD	PROG	256B	FB	8	8	1	9	1	1	???
CCSS	PROG	256B	FB	326	326	1	327	1	1	???
CDCCONV		72B	FA	13	13	3	6	6	6	???
CGROWTH		640B	FB	3	3	2	15	3	3	???
CHISQ	PROG	256B	FB	61	61	1	62	1	1	???
CHITREND	PROG	256B	FB	35	35	1	36	1	1	???
CPR	PROG	256B	FB	12	12	1	13	1	1	???
CPRDATA		80B	FA	810	811	1	812	8	8	???
DATA1	BASD	102B	FB	15	15	1	16	8	8	???
DATA2	BASD	40B	FB	5	5	1	6	6	6	???
DGROWTH		640B	FB	250	250	2	630	8	8	???
FCOPY	PROG	256B	FB	77	1232	1	78	1	16	???
FOOTBALL	BASP	256B	FB	46	46	1	47	8	8	???
FTNCOMPS		88B	FA	52	832	2	27	1	16	???
GAUSS		72B	FA	24	384	3	9	1	15	???
GENRLIO	RL	256B	FB	35	35	1	36	1	1	???
ISAMS		80B	FA	1711	27376	3	1142	2	16	???
ISDUMPS		80B	FA	147	2352	3	50	1	16	???
ISORT		88B	FA	30	480	2	16	1	16	???
KSAMP	PROG	256B	FB	73	73	1	74	1	1	???
LASCII		80B	FA	101	102	3	35	1	1	???
LBINARY		80B	FA	87	187	3	64	1	1	???
LDSCHEMA		80B	FA	161	161	3	55	14	14	???
LESTHAN3		80B	FA	48	48	3	17	1	1	???
LOGASRC		80B	FA	194	194	3	66	1	1	???
LOGRSRC		80B	FA	430	430	3	145	1	1	???
MACMAN	PROG	256B	FB	47	47	1	48	1	1	???
MACPEE	PROG	256B	FB	76	76	1	77	1	1	???
MACPUF	PROG	256B	FB	110	110	1	111	1	1	???
MAXMIN		72B	FA	29	464	3	20	2	16	???
MTHLOG	PROG	256B	FB	16	16	1	17	1	1	???
MULTPLY3		80B	FA	113	213	3	72	1	1	???
NFACT		88B	FA	41	141	2	72	1	1	???
NUMRECS		80B	FA	14	224	3	10	2	16	???
PASSCOMS		80B	FA	100	1600	3	68	2	16	???
PLOT7202		80B	FA	47	752	3	32	2	16	???
PLOTRL	RL	256B	FB	23	23	1	24	1	1	???
POWER2		88B	FA	41	141	2	72	1	1	???
PRINTER	PROG	256B	FB	9	9	1	10	1	1	???
PRINTF	PROG	256B	FB	12	15	1	16	1	1	???
PTAPEIN	PROG	256B	FB	30	480	1	31	1	16	???
PTAPEINS		80B	FA	459	7344	3	154	1	16	???

ACCOUNT= SUPPORT GROUP= USERS (CONT.)

FILENAME	CODE	LOGICAL RECORD SIZE	TYP	EOF	LIMIT	R/B	SPACE SECTORS	#X	MX	A
RANDG		72B	FA	16	256	3	12	2	15	???
SACUMLOG		80B	FA	415	415	3	140	8	8	???
SAMPLE	PROG	256B	FB	12	12	1	13	1	1	???
SASORT		80B	FA	1035	1035	1	1036	8	8	???
SBARG		80B	FA	361	361	3	122	16	16	???
SBONETMR		80B	FA	185	185	3	63	8	8	???
SBUSY		80B	FA	72	72	3	25	7	7	???
SCARD		80B	FA	84	84	3	29	8	8	???
SCCSS		80B	FA	249	249	1	250	8	8	???
SCHISQ		80B	FA	641	641	1	642	8	8	???
SCHITRND		80B	FA	181	181	1	182	8	8	???
SCLEAR		80B	FA	154	154	1	155	8	8	???
SCORRECT		80B	FA	18	18	3	7	7	7	???
SCOSTING		80B	FA	28	28	3	11	6	6	???
SCPR		80B	FA	100	100	1	101	8	8	???
SDATE		80B	FA	34	34	3	13	7	7	???
SESSION		80B	FA	46	736	3	32	2	16	???
SETSAMS		80B	FA	360	5760	3	121	1	16	???
SGENIO		88B	FA	715	715	2	359	8	8	???
SGP1		88B	FA	203	203	2	103	8	8	???
SGP2		96B	FA	224	224	2	113	8	8	???
SGP3		96B	FA	335	335	2	169	8	8	???
SGP4		88B	FA	286	286	2	144	8	8	???
SGPLOT		80B	FA	1054	1054	1	1055	8	8	?'
SL	SL	256B	FB	7	7	1	8	1	1	??.
SLISTS		80B	FA	1011	1011	1	1012	8	8	???
SLPUBSRC		80B	FA	367	467	3	130	13	16	???
SMACMAN		80B	FA	701	701	1	702	8	8	???
SMACPEE		80B	FA	1262	1262	1	1263	8	8	???
SMACPUF		80B	FA	1384	1384	1	1385	8	8	???
SMTHLOG		80B	FA	238	238	3	81	8	8	???
SPLCOMPS		88B	FA	52	832	2	27	1	16	???
SPLOTPT		80B	FA	319	319	1	320	8	8	???
SPRINTER		80B	FA	140	140	1	141	8	8	???
SPRINTF		80B	FA	155	162	3	55	1	1	???
SPROB		80B	FA	36	36	1	37	8	8	???
SRRCD		88B	FA	674	674	2	338	8	8	???
SSUMLOG		80B	FA	238	238	3	81	8	8	???
STHYROID		80B	FA	181	181	1	182	8	8	???
STMN1		80B	FA	1209	1209	3	404	8	8	???
STRAM		80B	FA	1089	1089	3	364	8	8	???
STREND		80B	FA	83	83	1	84	8	8	???
SUBTRCT3		80B	FA	60	60	3	21	1	1	???
SUM3		80B	FA	53	153	3	52	1	1	???
SUMLOG	PROG	256B	FB	16	16	1	17	1	1	???
TANACOMS		80B	FA	26	416	3	18	2	16	???
TAYLOR		72B	FA	54	864	3	19	1	16	???
THYROID	PROG	256B	FB	28	28	1	29	1	1	???
TRAND		80B	FA	16	16	1	17	6	6	???
WHOSRC		80B	FA	50	50	3	18	1	1	???