Burroughs

B 6700 / B 7700

SYSTEM SOFTWARE

HANDBOOK

THIS HANDBOOK REPLACES VOL. II
OF FORM NO. 5000276
DATED JANUARY 1972

$5.00

PRINTED IN U.S. AMERICA 7 - 73 5000722

LIST OF EFFECTIVE PAGES

NOTE: Insert latest changed page; dispose of superseded pages.

TOTAL NUMBER OF PAGES IN THIS MANUAL IS 436 CONSISTING OF THE FOLLOWING:

Page No.	Issue
Title	Original
A	Original
i thru xii	Original
1-1 thru 1-16	Original
2-1 thru 2-67	Original
2-68 Blank	Original
3-1 thru 3-137	Original
3-138 Blank	Original
4-1 thru 4-81	Original
4-82 Blank	Original
5-1 thru 5-30	Original
6-1 thru 6-52	Original
7-1 thru 7-18	Original
A-1 thru A-2	Original

A PAGE

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont)

PRINTED IN U.S. AMERICA 5000722

TABLE OF CONTENTS (Cont)

TABLE OF CONTENTS (Cont)

TABLE OF CONTENTS (Cont)

TABLE OF CONTENTS (Cont)

TABLE OF CONTENTS (Cont)

TABLE OF CONTENTS (Cont)

TABLE OF CONTENTS (Cont)

TABLE OF CONTENTS (Cont)

TABLE OF CONTENTS (Cont)

LIST OF ILLUSTRATIONS

TABLE OF CONTENTS (Cont)

LIST OF TABLES

The terms "Multiplexor" and "I/O Processor" are synonymous.

SECTION 1

GENERAL DESCRIPTION OF THE MCP

SECTION 1 — CONTENTS

SECTION 1

GENERAL DESCRIPTION OF THE MCP

LOCATIONS IN THE D[0] STACK

A list of the lower locations relative to the address contained in display register D[0] follows. The D[0] register points to memory address 30 hexadecimal.

HEX	DECIMAL	DECLARATION
(0,02)	(0, 2)	= ARRAY ARRAYSTACK [*,*,*]
(0,03)	(0, 3)	= WORD DO3
(0,04)	(0, 4)	= WORD ARRAY M2 [*,*]
(0,05)	(0, 5)	= SEGMENT DESCRIPTOR FOR MCP SAVE CODE
(0,06)	(0, 6)	= SLAVEQUARTERS, TOTAL CORESIZE
(0,07)	(0, 7)	= PROCEDURE ARRAYDEC
(0,08)	(0, 8)	= MONITORVALUE
(0,09)	(0, 9)	= MONITORMASK
(0,0A)	(0, 10)	= PROCEDURE BLOCKEXIT
(0,0B)	(0, 11)	= PROCEDURE GOTOSOLVER
(0,0C)	(0, 12)	= DABMEM
(0,0D)	(0, 13)	= PROCEDURE TIMETUNNEL
(0,0E)	(0, 14)	= PROCEDURE SOFTWAREINTERRUPTATTACH
(0,10)	(0, 16)	= PROCEDURE PICKASTACK
(0,11)	(0, 17)	= ARRAY INTRINSICINFO [*]
(0,13)	(0, 19)	= PROCEDURE HOLD
(0,14)	(0, 20)	= PROCEDURE MEMDUMP
(0,15)	(0, 21)	= PROCEDURE XALGOLFILLSTAT
(0,16)	(0, 22)	= PROCEDURE XALGOLSEARCHSTAT
(0,17)	(0, 23)	= PROCEDURE PROGRAMDUMP
(0,18)	(0, 24)	= REAL PROCEDURE TIMEINTRINSIC
(0,1B)	(0, 27)	= PROCEDURE CLOSE
(0,1C)	(0, 28)	= PROCEDURE FORMATINT
(0,1D)	(0, 29)	= TIMEOFDAYWORDV
(0,1F)	(0, 31)	= PROCEDURE OPEN
(0,20)	(0, 32)	= PROCEDURE FAULTDEC
(0,21)	(0, 33)	= TRANSLATETABLE BCLTOEBC [*]
(0,22)	(0, 34)	= TRANSLATETABLE EBCDICTOBCL [*]
(0,23)	(0, 35)	= DOUBLE VALUE ARRAY POTL
(0,24)	(0, 36)	= DOUBLE VALUE ARRAY POTM
(0,25)	(0, 37)	= DOUBLE VALUE ARRAY POTH
(0,26)	(0, 38)	= VALUE ARRAY TRUTHSETS
(0,27)	(0, 39)	= SAVE TRANSLATETABLE HEXTOBCL
(0,28)	(0, 40)	= PROCEDURE MESSER
(0,29)	(0, 41)	= REAL PROCEDURE ATTRIBUTEGRABBER
(0,2A)	(0, 42)	= PROCEDURE ATTRIBUTEHANDLER
(0,2B)	(0, 43)	= BOOLEAN PROCEDURE GETSTATUS
(0,2C)	(0, 44)	= BOOLEAN PROCEDURE SETSTATUS
(0,2D)	(0, 45)	= REAL PROCEDURE SYSTEMSTATUS
(0,2E)	(0, 46)	= PROCEDURE USERIOERROR
(0,2F)	(0, 47)	= SAVE PROCEDURE SWAP
(0,30)	(0, 48)	= BOOLEAN PROCEDURE WRITESPO
(0,32)	(0, 50)	= PROCEDURE LOADCONTROL

LOCATIONS IN D [0] STACK

HEX	DECIMAL	DECLARATION
(0,33)	(0, 51)	PROCEDURE FORGETSPACE
(0,34)	(0, 52)	REAL PROCEDURE DIRECTORYSEARCH
(0,35)	(0, 53)	REAL PROCEDURE WAITIO
(0,36)	(0, 54)	PROCEDURE IOREQUEST
(0,37)	(0, 55)	WORD PROCEDURE DISKIO
(0,38)	(0, 56)	PROCEDURE FORGETAREA
(0,39)	(0, 57)	REAL PROCEDURE GETUSERDISK
(0,3A)	(0, 58)	PROCEDURE PROCESSKILL
(0,3B)	(0, 59)	REAL PROCEDURE SPACEOF
(0,3C)	(0, 60)	REAL PROCEDURE SPACEN
(0,3D)	(0, 61)	PROCEDURE MOMTOVECTOR
(0,3E)	(0, 62)	PROCEDURE TURNOVERLAYKEY
(0,3F)	(0, 63)	PROCEDURE MAKEPRESENTANDSAVE
(0,40)	(0, 64)	PROCEDURE UPDATEDISKHEADER
(0,41)	(0, 65)	BOOLEAN PROCEDURE DISPLAYTOSTANDARD
(0,42)	(0, 66)	REAL PROCEDURE GETSPACE
(0,43)	(0, 67)	BOOLEAN PROCEDURE DISKWAIT
(0,44)	(0, 68)	INTEGER PROCEDURE GETAREA
(0,45)	(0, 69)	PROCEDURE SORT
(0,46)	(0, 70)	PROCEDURE RESIZEANDDEALLOCATE
(0,48)	(0, 72)	REAL TIMEBASE
(0,49)	(0, 73)	PROCEDURE PUTSPACEUSAGE
(0,4A)	(0, 74)	PROCEDURE FORGETUSERDISK
(0,4B)	(0, 75)	REAL PROCEDURE DELTA
(0,4C)	(0, 76)	PROCEDURE ANSWER
(0,4D)	(0, 77)	PROCEDURE DIAL
(0,4E)	(0, 78)	PROCEDURE SPEAK
(0,4F)	(0, 79)	PROCEDURE HANGUP
(0,50)	(0, 80)	PROCEDURE SEARCH
(0,51)	(0, 81)	PROCEDURE CHANGEINUSEID
(0,52)	(0, 82)	PROCEDURE KANGAROO
(0,56)	(0, 86)	PROCEDURE CONTROLCARD
(0,59)	(0, 89)	PROCEDURE KEYIN
(0,5A)	(0, 90)	WORD PROCEDURE EXPANDAROW
(0,5B)	(0, 91)	REAL PROCEDURE ENTERUSERFILE
(0,5C)	(0, 92)	ARRAY DISKFILEHEADERS [*,*]
(0,5D)	(0, 93)	PROCEDURE DELIVERY
(0,5E)	(0, 94)	REAL PROCEDURE DESCRIPTORSIZE
(0,5F)	(0, 95)	REAL PROCEDURE WRITEHEADER
(0,60)	(0, 96)	REAL PROCEDURE READHEADER
(0,61)	(0, 97)	PROCEDURE HEADATHAND
(0,62)	(0, 98)	PROCEDURE HEADATGRAB
(0,63)	(0, 99)	PROCEDURE BINARYIOINT
(0,64)	(0,100)	TRANSLATETABLE BCLTOHEX
(0,65)	(0,101)	TRANSLATETABLE EBCTOHEX
(0,66)	(0,102)	WORD PROCEDURE UNRAVEL
(0,67)	(0,103)	PROCEDURE MUTATE
(0,68)	(0,104)	SAVE WORD PROCEDURE MYSELFER
(0,69)	(0,105)	PROCEDURE CONTINUER
(0,6A)	(0,106)	PROCEDURE ACCEPT
(0,6B)	(0,107)	PROCEDURE DCFLUSH
(0,6C)	(0,108)	PROCEDURE DCCOMBINE

LOCATIONS IN D [0] STACK

HEX	DECIMAL	DECLARATION
(0,6D)	(0,109)	PROCEDURE DCINSERT
(0,6E)	(0,110)	REAL PROCEDURE DCREMOVE
(0,6F)	(0,111)	DOUBLE PROCEDURE DCHOLD
(0,70)	(0,112)	PROCEDURE DCCOMMUNICATE
(0,71)	(0,113)	REAL PROCEDURE DCWRITE
(0,72)	(0,114)	PROCEDURE INTERRUPTDEQUEUE
(0,73)	(0,115)	REAL PROCEDURE DCQINFO
(0,74)	(0,116)	REAL PROCEDURE DCINFOINT
(0,75)	(0,117)	PROCEDURE DCALLOCATE
(0,76)	(0,118)	PROCEDURE FORKCONTROLCARD
(0,77)	(0,119)	REAL PROCEDURE DCQUEHANDLER
(0,78)	(0,120)	WORD PROCEDURE DCATTACHDETACH
(0,79)	(0,121)	REAL PROCEDURE DCERRORLOGGER
(0,7A)	(0,122)	RESERVED FOR DATACOM
(0,7B)	(0,123)	REAL PROCEDURE DMDIRECTORYCONTROL
(0,7C)	(0,124)	INTEGER PROCEDURE DMWRITE
(0,7D)	(0,125)	BOOLEAN PROCEDURE DMINTRINSIC
(0,7E)	(0,126)	PROCEDURE OVERLAYFD
(0,80)	(0,128)	REAL PROCEDURE COPYINT
(0,81)	(0,129)	PROCEDURE SUPERMON
(0,82)	(0,130)	BOOLEAN PROCEDURE DIDDLER
(0,83)	(0,131)	PROCEDURE WORDMOVEINT
(0,84)	(0,132)	PROCEDURE EXCHANGEINT
(0,85)	(0,133)	REAL PROCEDURE SUBDIRECTORYSEARCH
(0,86)	(0,134)	REAL PROCEDURE FINDINPUT
(0,87)	(0,135)	REAL PROCEDURE FINDOUTPUT
(0,88)	(0,136)	REAL PROCEDURE FINDUNIT
(0,89)	(0,137)	PROCEDURE AMNESIA
(0,8B)	(0,139)	PROCEDURE STACKHISTORY
(0,8C)	(0,140)	PROCEDURE LOSEOLAYSPACE
(0,8D)	(0,141)	REAL PROCEDURE FINDOLAYSPACE
(0,8E)	(0,142)	BOOLEAN PROCEDURE FIXHANDLER
(0,8F)	(0,143)	DUMMY TAG-6 WORD (USED BY COMPILERS)
(0,90)	(0,144)	TAG-6 WORD (OVERLAY CONTROL)
(0,91)	(0,145)	NULL QUEUE REFERENCE (USED BY ESPOL)
(0,92)	(0,146)	EMPTY DATA DESCRIPTOR
(0,93)	(0,147)	REAL PROCEDURE MCSLOGGER
(0,94)	(0,148)	PROCEDURE LOGGER
(0,95)	(0,149)	REAL PROCEDURE DIRECTOR
(0,96)	(0,150)	PROCEDURE CAUSEP
(0,97)	(0,151)	RESERVED
(0,98)	(0,152)	PROCEDURE WAITP
(0,99)	(0,153)	PROCEDURE SETORRESET
(0,9A)	(0,154)	PROCEDURE PROCUREP
(0,9B)	(0,155)	BOOLEAN PROCEDURE MAKEUSER
(0,9C)	(0,156)	PROCEDURE LIBERATEP
(0,9D)	(0,157)	PROCEDURE ENABLEP
(0,9E)	(0,158)	PROCEDURE DISABLEP
(0,9F)	(0,159)	PROCEDURE SUPERWAIT
(0,A0)	(0,160)	BOOLEAN PROCEDURE STACKSWAPPER
(0,A1)	(0,161)	EVENT DCALGOLSPOLOCK
(0,A3)	(0,163)	WORD DCALGOLSPOQ

LOCATIONS IN D [0] STACK

HEX	DECIMAL	DECLARATION
(0,A4)	= (0,164)	= BOOLEAN PROCEDURE ACCEPTAMSG
(0,A5)	= (0,165)	= BOOLEAN PROCEDURE CHECKTASKSTATUS
(0,A8)	= (0,168)	= PROCEDURE PLCONDHANDLER
(0,AC)	= (0,172)	= BOOLEAN PROCEDURE USERDATAFREEZER
(0,AD)	= (0,173)	= REAL PROCEDURE USERDATA
(0,AE)	= (0,174)	= REAL PROCEDURE USERDATAREBUILD
(0,AF)	= (0,175)	= PROCEDURE FORKHANDLER
(0,B0)	= (0,176)	= TRANSLATETABLE BCLTOASC
(0,B1)	= (0,177)	= TRANSLATETABLE EBCTOASC
(0,B2)	= (0,178)	= TRANSLATETABLE ASCTOHEX
(0,B3)	= (0,179)	= TRANSLATETABLE ASCTOBCL
(0,B4)	= (0,180)	= TRANSLATETABLE ASCTOEBC
(0,B6)	= (0,182)	= PROCEDURE MAINTENANCE
(0,B7)	= (0,183)	= PROCEDURE CREATEDIRECTORY
(0,B8)	= (0,184)	= PROCEDURE CHANGESECURITY
(0,B9)	= (0,185)	= PROCEDURE JOBBLOCKEXIT
(0,BA)	= (0,186)	= PROCEDURE JOBROLLOUT

UNIT TABLE

FIELD	NAME AND MEANING
47:6	UNITTYPE. (in decimal) 0 = NO UNIT TYPE. 1 = DISK. 2 = DISPLAY (SPO, CONSOLE, ETC.). 3 = (DATACOM). 4 = PAPER TAPE READER. 5 = PAPER TAPE PUNCH. 6 = LINE PRINTER (BCL - BUFFERED). 9 = CARD READER. 10 = (PSEUDO-READER). 11 = CARD PUNCH. 13 = MAG TAPE (7 TRACK - NRZ). 14 = MAG TAPE (9 TRACK - NRZ). 15 = MAG TAPE (9 TRACK - PE). 38 = LINE PRINTER (EBCDIC - BUFFERED). 39 = LINE PRINTER (EBCDIC - UNBUFFERED). 49 = DISK PACK.
41:1	ABNORMALPWROFF.
40:1	NORMALPWROFF.
39:1	UNINITIALIZEDF.
36:1	ULPFMED.
35:1	UAUTOBACKUP.
34:1	ULABELREAD.
33:1	MAINTBIT.
32:1	MARGINALBITF.
31:1	UNITSECURITYF.
30:1	USTKASSIGNED.
29:4	DFONUMF.
25:1	UNITNOTREADY.
24:1	UNITERROR.
23:1	UPATHWAIT.
22:1	UNITIOBUSY.
21:1	UNITINPROCESS.
20:1	UNITRETRY.
19:1	ULOCKED.

UNIT TABLE

FIELD	NAME AND MEANING
18:1	USAVED.
17:3	DENSITYF. IF UNITTYPE = DISK: 0 = IIB-6. 1 = IA-2. 2 = IC-3. 3 = IIB-2. 4 = IC-4. 5 = IIB-4. IF UNITTYPE = MAG TAPE: 0 = 800 BPI. 1 = 556 BPI. 2 = 200 BPI. 3 = 1600 BPI.
14:1	USCRATCH.
13:1	UNITASSIGNED.
12:1	ULABELLED.
11:1	UINREWIND.
10:1	UCLOSEDNOREWIND.
9:1	UTOBEPURGED.
8:1	UWRITERING.
7:1	UCONTROLCARD.
6:1	UCLOSERRF.
5:5	UNITPATHGROUP.
0:1	ALWAYS = 1

IOAREA[•] FORMAT

HEX	DEC	NAME(S)
0	0	IOLINKAGE (USED IN IOQUE & PATHQUE)
1	1	USER
2	2	IOMASK
3	3	IORD (CONTAINS R.D. AT I/O FINISH)
4	4	AREADESC (POINTS AT IOCWP)
5	5	TIMECELL (TIME OF I/O IN 2.4 usec)
6	6	EVNT (CAUSED AT I/O FINISH)
7	7	FIBDESC
8	8	BFFREVENT
9	9	BFFREVENT2
A	10	IOAL (BUFFER LINKS)
B	11	IOAW/MSGPLC (BUFFER INFO)
C	12	IOADW (ACTUAL KEY)
D	13	ACTUALKEY2 (UPDATE READ)
E	14	DUPLCOPY (OCCURS WORD)
F	15	JUNK1
10	16	IOCWP (THE IOCW)
11	17	FRSTDATA

USER WORD OF IOCB

FIELD	NAME AND MEANING
47:10	IDNO (USER STACK NO.)
37:1	COUNTEIOF (COUNT IN STACK)
36:1	GOGETAREA (FOR DISK LOGGING)
32:9	UNITNOF (LOGICAL UNIT NO.)
19:5	MPXROUTEF (MPX DESIGNATE NORMALLY ZERO)
14:2	USERIDF (USER ID FIELD)
14:1	SIMPLEIOBIT (SIMPLE IO)
13:1	LIBMAINBIT (LIBRARY MAINTENANCE BIT)
12:1	READCHECKBIT (READ CHECK BIT)
11:1	UPDATEREAD (DO READ AFTER WRITE - UPDATE FILES)
10:1	TAPEPROCTECTIONF (USE BUF [1] IN PARITY RETRY)
9:1	USERINPROCESS (I/O IN PROCESS)
8:1	USEROUERROWF (BADLY BLOCKED DISK)
7:1	USERSPECIALIO (SPECIAL ACTION TO BE TAKEN)
6:1	USERIOBIT (USER'S I/O)
5:1	CLOSERRBIT (CLOSED BECAUSE OF ERROR)
4:1	DIRECTIOBIT (DIRECT IO INDICATOR)
3:1	IOERRORRECOVERY
2:3	USERIOFIELD
2:1	USERPARITYBIT
1:1	USEREOFORTBIT
0:1	USERIOFINISH

USER WORD OF IOCB

NOTE

THE FOLLOWING FIELDS APPLY TO MISC WORD OF IOCB.

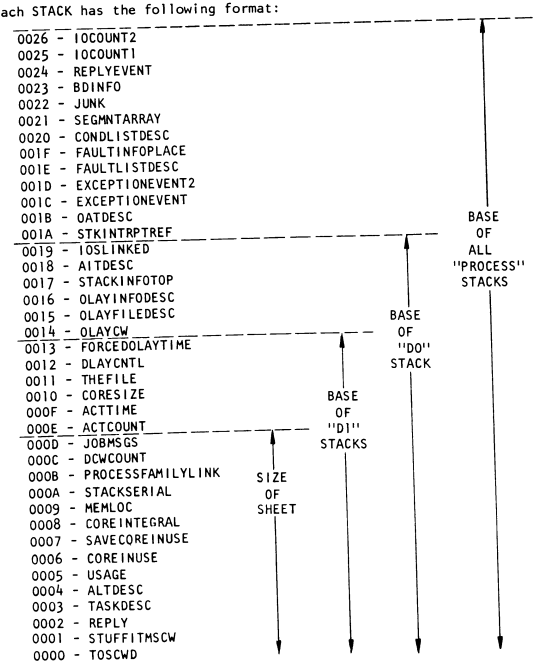
47:20	WORDCOUNTF (WORD COUNT FIELD)
27:3	RDCHRCNT (CHARACTER COUNT FIELD)
24:8	RDUNITNO (UNIT NO.)
16:17	IOERRORMASKFIELD (IOFINISH)

NOTE

AFTER SUCCESSFUL IOFINISH, IOERRORMASKFIELD CONTAINS ERROR FIELD (AFTER MASKING) OF RESULT DESCRIPTOR.

FORMATS OF STACKS

Each STACK has the following format:



FORMAT OF TASK VARIABLE

The following list denotes relative locations within the task variable:

0000 - TASKMSCW
0001 - AVALUE
0002 - MYBDNAME
0003 - DATACORE
0004 - CODECORE
0005 - EXCEPTIONTASK
0006 - FILESENTRY
0007 - MYFPB
0008 - MYPPB
0009 - LOCKEDEVENT
000A - LOCKFDEVENT2
000B - MAXIOTIME
000C - MAXPROCESSTIME
000D - MYNAME
000E - OPTION
000F - PARTNER
0010 - PRIORITY
0011 - RESTARTCOUNT
0012 - STACKSIZE
0013 - STATIONINFO
0014 - TARGETTIME
0015 - USERCODE
0016 - USERCODE1
0017 - USERCODE2
0018 - USERCODE3
0019 - VALIDITYBITS
001A - YOURNAME
001B - ABORTEDRCW
001C - COMPILERINFO
001D - ELAPSEDTIME
001E - ERROR
001F - HISTORY
0020 - IOTIME
0021 - PROCESSTIME
0022 - TASKTYPE
0023 - SERIAL
0024 - TIMESTARTED
0025 - ORGUNIT
0026 - MYSTACKHISTORY
0027 - BLOCK
0028 - MEMTYPE
0029 - CONTROLCARDS
002A - TASKPARAMS
002B - JOBINFO

FORMAT OF TASK VARIABLE

```
002C - ROLLOUTINFO
002D - CLOCKONTIME
002E - DISTACKINFO
002F - ENTRYPOINT
0030 - NEXTTIMER
0031 - PALACE
0032 - MSCWINTOTASK
0033 - TIMESLOT
0034 - LOCKCOUNT
0035 - TASKINFO
0036 - READYON
0037 - READYTIME
```

FIXED PART OF FILE INFORMATION BLOCK (FIB)

```
0000        SIRW SELECTOR              = FIBW[0]#;     % INDXFLD=PWRITES
0001        MSCW FIBMSCW               = FIBW[1]#;     % (1,0)
0002        REAL FIBLOCK               = FIB[2]#,      % (1,1)
0003             RECORDSTATUS          = FIB[3]#,      % (1,2)
0004             FILESTATUS            = FIB[4]#,      % (1,3)
0005             TANKDATA1             = FIB[5]#,      % (1,4)
0006             TANKDATA2             = FIB[6]#,      % (1,5)
0007             TANKDATA3             = FIB[7]#,      % (1,6)
0008             LABELATT              = FIB[8]#,      % (1,7)
0009             IOINFO                = FIB[9]#;      % (1,8)
000A        ARRAY LEB[*]               = FIBW[10]#,    % (1,9)
000B             IOAREA[*]             = FIBW[11]#,    % (1,A)
000C             BUFFDESC[*]           = FIBW[12]#;    % (1,B)
000D        WORD USEROUTINES           = FIBW[13]#;    % (1,C)
000E        FMTBUFFDESC                = FIB[14]#,     % (1,D)
000F             FILLOGLNK             = FIB[15]#,     % (1,E)
```

FIB

```
0010            FILIOTIME           = FIB[16]#;         % (1,F)
0011      WORD  PWRITES             = FIBW[17]#,        % (1,10)
0012            PREADS              = FIBW[18]#,        % (1,11)
0013            PWRITEN             = FIBW[19]#,        % (1,12)
0014            PREADN              = FIBW[20]#,        % (1,13)
0015            PSEEK               = FIBW[21]#,        % (1,14)
0016            PREADR              = FIBW[22]#,        % (1,15)
0017            PSEARCH             = FIBW[23]#,        % (1,16)
0018            PLOCKER             = FIBW[24]#,        % (1,17)
0019            PRELEASE            = FIBW[25]#,        % (1,18)
001A            FILEEVENT1          = FIBW[26]#,        % (1,19)
001B            FILEEVENT2          = FIBW[27]#;        % (1,1A)
001C      REAL  FILEACCESS          = FIB[28]#,         % (1,1B)
001D            DISKBLOCK           = FIB[29]#,         % (1,1C)
001E            PCWCONTROL          = FIB[30]#,         % (1,1D)
001F            OFFSET              = FIB[31]#,         % (1,1E)
0020            RECORDCOUNT         = FIB[32]#,         % (1,1F)
0021            BLOCKCOUNT          = FIB[33]#,         % (1,20)
0022            LOWER               = FIB[34]#,         % (1,21)
0023            UPPER               = FIB[35]#,         % (1,22)
0024            ACTNUM              = FIB[36]#,         % (1,23)
0025            R                   = FIB[37]#,         % (1,24)
0026            I                   = FIB[38]#,         % (1,25)
0027            T                   = FIB[39]#;         % (1,26)
0028      WORD  AEXP                = FIBW[40]#;        % (1,27)
0029      ARRAY DHEADER[*]          = FIBW[41]#;        % (1,28)
002A      REAL  FIBEOF              = FIB[42]#,         % (1,29)
002B            PAGESPEC            = FIB[43]#,         % (1,2A)
002C            SBLOCKING           = FIB[44]#,         % (1,2B)
002D            SIOINFO             = FIB[45]#;         % (1,2C)
002E      ARRAY SIOAREA[*]          = FIBW[46]#;        % (1,2D)
002F      REAL  SOFFSET             = FIB[47]#;         % (1,2E)
0030      REAL  CURRENTBLOCK        = FIB[48]#;         % (1,2F)
0031      WORD  PMOVEIN             = FIBW[49]#,        % (1,30)
0032            PMOVEOUT            = FIBW[50]#,        % (1,31)
0033            PWAIT               = FIBW[51]#;        % (1,32)
0034      REAL  MINRECSZ            = FIB[52]#;         % (1,33)
0035      REAL  FIBMLOGCNT          = FIB[53]#;         % (1,34)
0036      WORD  INPUTTRANSLATION    = FIBW[54]#,        % (1,35)
0037            OUTPUTTRANSLATION   = FIBW[55]#;        % (1,36)
0038      REAL  FMTLOCK             = FIB[56]#,         % (1,37)
0039            ATTVALUE            = FIB[57]#;         % (1,38)
```

MEMORY LINKS

All areas in memory (whether in use or not in use, overlayable or non-overlayable) are surrounded by memory link words. When getting space, the MCP uses the Linked List Lookup operator to find the first area which is equal to or greater than the size needed. The MCP begins with Word 0 or 1 depending on the type of area needed.

In-Use Areas:

In-use areas both in overlayable and nonoverlayable memory are surrounded by four memory links, as follows: LINKA, LINKB, LINKC, the in-use area, and LINKZ.

Available Areas:

Areas that are not in use are placed in two available lists; all items in these lists are linked together in the order of their size, smallest first. Available areas are surrounded by four memory links, as follows: AVAILA, AVAILB, the available area, AVAILY, and AVAILZ.

FORMATS OF MEMORY WORDS 0 AND 1

FORMATS OF MEMORY WORDS 0 AND 1

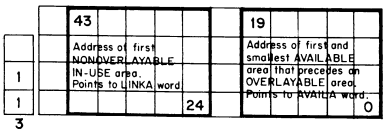
The first two words in memory, words 0 and 1, contain pointers to each of the types of areas contained in memory:

a. Nonoverlayable in-use areas.

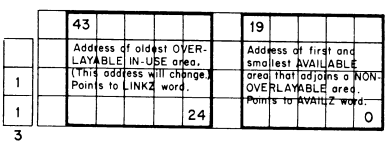
b. Overlayable in-use areas.

c. Available areas.

MEMORY WORD 0

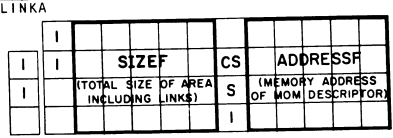
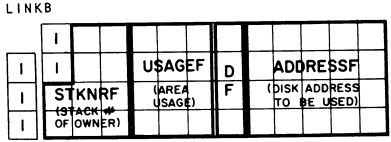
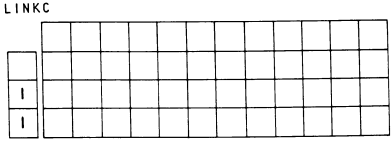
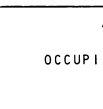
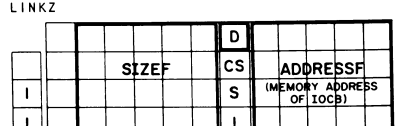


MEMORY WORD 1



"LEFTOFF"

Formats of Links: In-Use and Available Areas

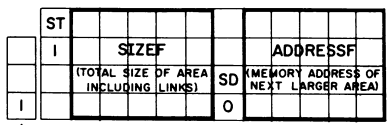
WORD	IN-USE
1	LINKA  47:2 = OVERLAYCF = Status of area during overlay process. CS = CURSAVF = 1 if area is temporary save. S = SAVEF = 1 if area is non-overlayable.
2	LINKB  DF = DELTAF = Number of unused words not sent back to available list
3	LINKC 
	
LAST	LINKZ  D = 1 if area contains a DELTA area.

MEMORY LINKS
Formats of Links: In-Use and Available Areas (Cont)

AVAILABLE

WORD

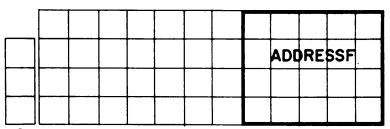
AVAILA



ST = STOPPERF = 1 if last item in available list
SD = SAVEDF = 1 if memory mod is "saved"

1

AVAILB



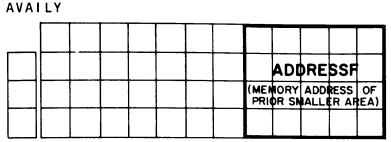
(Used for linked list of available areas that precede overlayable areas).

2

AVAILABLE AREA

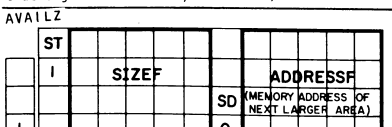
3

AVAILY



(Used for linked list of available areas that adjoin non-overlayable areas).

AVAILZ



LAST

SECTION 2
COMPILERS

SECTION 2 — CONTENTS

SECTION 2

COMPILERS

The B 6700 language compilers are special-purpose computer programs used to syntactically check source code and to translate this code into object code capable of being executed under control of the B 6700 MCP.

All B 6700 compilers are written in B 6700 Extended ALGOL. Each compiler source file has the title: SYMBOL/<compiler-name>, where <compiler-name> is identical to the name of the language in which the program to be compiled is written (e.g., SYMBOL/ALGOL, SYMBOL/FORTRAN). Similarly, each compiler object file has the title: SYSTEM/<compiler-name> (e.g. SYSTEM/ALGOL, SYSTEM/XALGOL).

Each compiler accepts as input the appropriate language coded in either EBCDIC or BCL characters. In addition to these two character formats, the FORTRAN compiler accepts BCD characters. Each compiler, except ESPOL and XALGOL, internally processes characters in EBCDIC, with BCL characters being translated to EBCDIC via the MCP translation table, BCLTOEBCDIC, and BCD characters being translated via a modified BCLTOEBCDIC table. The ESPOL and XALGOL compilers internally process characters in BCL, with EBCDIC characters being translated to BCL via the MCP translation table, EBCDICTOBCL.

COMPILER FILES

Compiler communication is handled through various input and output files; these files are described in the following paragraraphs.

Input Files

The primary compiler input file is a card file with the internal name CARD; the secondary input file is a serial disk file with the internal name TAPE. The primary file (CARD) is required; the secondary file (TAPE) is optional. File CARD can be either BCL-coded (or BCD-coded for FORTRAN) or EBCDIC-coded and may be blocked or unblocked. File TAPE can be either BCL-coded or EBCDIC-coded and may be blocked or unblocked. (See table 2-1.) Both file CARD and file TAPE may be label-equated to change file TITLE and/or file KIND. Additional files may be input to all compilers except XALGOL, BASIC, and COBOL through use of the INCLUDE compiler control option. These files may be either BCL-coded or EBCDIC-coded and may be blocked or unblocked.

The COBOL compiler may also access permanent library files as compiler input through use of the FROM compiler option and the COPY statement. The files thus accessed may be either BCL-coded or EBCDIC-coded and may be either blocked or unblocked.

NOTE

Usually BCL-coded records are 10 or 11 words and EBCDIC-coded records are 14 or 15 words, but this is not a strict requirement of compiler input files.

COMPILER FILES

Output Files

Output files produced by the compilers may include the object code file (on disk), an updated symbolic code file, a line printer listing, or various special-purpose files produced by the ESPOL and COBOL compilers. The object code file has the internal name CODE and is saved on disk unless the COMPILE card specifies compilation for syntax only or unless syntax errors are detected in the source input data by the compiler. If compile-and-go is specified by the COMPILE card, then the object file is discarded following execution of the program. (This concept is meaningless for ESPOL programs since such programs cannot be "executed".) If compilation for library is specified, then the object code file is saved (with a SAVEFACTOR assigned by the compiler involved). The TITLE of the saved object code file is identical to the program name appearing on the COMPILE card (except in the case of separately-compiled procedures). When procedures are compiled separately, the TITLE of the resultant object code library file consists of the program name appearing on the COMPILE card with the rightmost file identifier in the program name replaced by the procedure identifier. If there is only one directory level in the program name, this program name is assigned as the file TITLE.

The updated symbolic file is a serial head-per-track disk file (unless the file KIND is altered) generated only if the compiler option NEW is set. This file has the internal file name NEWTAPE and contains EBCDIC-coded 14 or 15-word records in 420-word blocks (except when generated by the XALGOL or ESPOL compilers in which case the file contains BCL-coded 10 or 11-word records in 150-word blocks). A SAVEFACTOR is assigned to the file by the compiler. (Refer to table 2-2.)

The output listing is an optional print file that is created unless the compiler option LIST (and LISTP and LISTI for COBOL) is reset. The file has the internal name LINE and contains the following items:

Source input
Code segmentation information
Error messages
Elapsed compilation time
Processing time
Number of cards scanned
Number of syntactic items
Stack estimate (in words)
Required core estimate (in words)
Number of errors detected
Sequence number of last error detected
Generated object code (if option CODE is set)
Stack address assignments (if option STACK is set)

For FORTRAN compilations the listing also includes:

File size (in words)
Array storage (in words)

For COBOL compilations the listing also includes a display of elementary data items being referenced by instructions employing the CORRESPONDING expression (unless compiler option SPEC is set).

The output error message listing with the internal name ERRORFILE is an optional line printer file which is created when the compiler option ERRLIST is set. This file is not available to ESPOL and is normally employed when jobs are run from a remote terminal using CANDE. In such cases, ERRLIST is automatically set and the file is assigned to the remote device. File ERRORFILE may also be used with jobs input from the console or card reader. The file is limited to a width of 72 characters thus allowing it to be output to the remote device (or card punch) without truncation of characters. When a syntax error is detected, the offending card image is written on this file with an error message and pointer to the syntactical item in question being written on the following line of text.

The ESPOL compiler may be used to produce an object code output file on a deck of EBCDIC-coded punched cards or on a tape. This compiler file, with an internal name of DECK, is created when the compiler option DECK is set. File DECK is an EBCDIC-coded punch file (14-word records) which may be label-equated to a tape file if no card punch output is desired. For example:

```
<1>ESPOL FILE DECK(KIND=TAPE)
```

The COBOL compiler may also produce three additional optional output files: a card punch file containing all source-language card images which contain syntactical errors and two library files containing portions (or all) of the source input. When the compiler option PUNCH is set, all card images containing syntax errors are written to the file with the internal name of PUNCH. These cards will be punched in EBCDIC even if the input file is BCL-coded.

When the SAVE compiler control card is employed in a COBOL program with a file TITLE immediately following the word SAVE, a permanent library file with that TITLE is created. This file has the internal name SAVEPERM, and records from this file may be recalled, either in the same compilation or by other programs in other compilations, by use of the FROM <file-title> compiler option where <file-title> is the same title employed on the SAVE card creating the file. The permanent library file is EBCDIC-coded, contains 14-word records in 420-word blocks, and has a SAVEFACTOR of 999 days.

When the SAVE compiler control card is employed in a COBOL program with no file title following the word SAVE, then a temporary serial disk library file is created to save portions of the source file to be recalled and inserted in a later part of the same compilation. Resetting

TABLE 2-1. COMPILER INPUT FILES

COMPILER	PURPOSE	KIND	FILE NAME	CODE	RECORD SIZE	BLOCK SIZE	COMMENTS
ALGOL DCALGOL XALGOL BASIC COBOL ESPOL FORTRAN PL/I	Input Card File	Card Reader	CARD	EBCDIC	14 or 15 words	Blocked or Unblocked	Required. Primary compiler input file. May be label-equated to another file. CANDE file is equated to this file automatically by CANDE. FORTRAN CARD file may also be BCD-coded.
				BCL	10 or 11 words		
ALGOL DCALGOL XALGOL BASIC PL/I	Input Disk File	Serial Disk	TAPE	EBCDIC	14 or 15 words	420 words	Optional. Secondary compiler input file. Selected by setting compiler option MERGE. May be label-equated to another file, but may not be used with CANDE. COBOL TITLE is "COBOL/IMAGE". FORTRAN default external name is FORSYM.
				BCL	10 or 11 words	150 words	
COBOL ESPOL FORTRAN	Input Disk File	Serial Disk	TAPE	EBCDIC	14 or 15 words	Blocked or Unblocked	
				BCL	10 or 11 words		
COBOL	Permanent Library File called by FROM option	Serial Disk	SAVEPERMIN	EBCDIC	14 or 15 words	Blocked or Unblocked	Optional. TITLE is not label-equatable and is specified on FROM card.
				BCL	10 or 11 words		

TABLE 2-1. COMPILER INPUT FILES (Cont)

COMPILER	PURPOSE	KIND	FILE NAME	CODE	RECORD SIZE	BLOCK SIZE	COMMENTS
COBOL	Permanent Library File called by COPY statement	Serial Disk	LIBRARY	EBCDIC	14 or 15 words	Blocked or Unblocked	Optional. TITLE is not label-equatable. Called by "COPY <library-name>", where <library-name> is the file title.
				BCL	10 or 11 words		
ALGOL DCALGOL FORTRAN PL/I	Library include file	Serial Disk	Internal or external name may be given on $INCLUDE card	EBCDIC	14 or 15 words	420 words	Optional. File opened by INCLUDE card bearing the appropriate external name (title) or internal name of the file. Maximum of five levels of nesting permitted.
				BCL	10 or 11 words	150 words	
ESPOL	Library include file	Serial Disk	Internal or external name may be given on $INCLUDE card	EBCDIC	14 or 15 words	Blocked or Unblocked	
				BCL	10 or 11 words		

TABLE 2-2. COMPILER OUTPUT FILES

COMPILER	PURPOSE	KIND	FILE NAME	CODE	RECORD SIZE	BLOCK SIZE	COMMENTS
ALGOL DCALGOL XALGOL BASIC COBOL FORTRAN PL/I	Object code file	Serial Disk	CODE	Hexa-decimal	30 words	150 words	Saved if compilation for library specified. SAVEFACTOR=999 (30 for FORTRAN). Discarded after execution if compile and go specified. Discarded if syntax errors are discovered, not opened or if compile for syntax is specified. The default file TITLE after compilation is the program name on COMPILE card (modified by subprogram ID when separate compilation is used). For CANDE, file TITLE becomes CANDE <file-name> OBJECT. File TITLE during COBOL compilation is "COBOL/CODE".
ESPOL	Object code file	Serial Disk	CODE	BCL	30 words	300 words	

TABLE 2-2. COMPILER OUTPUT FILES (Cont)

COMPILER	PURPOSE	KIND	FILE NAME	CODE	RECORD SIZE	BLOCK SIZE	COMMENTS
ALGOL DCALGOL BASIC COBOL FORTRAN PL/I	Updated symbolic code file	Serial Disk	NEWTAPE	EBCDIC	14 words except 15 words for ALGOL, DCALGOL	420 words	Optional output produced when compiler option NEW is set. Various SAVEFACTORs apply: ALGOL, DCALGOL, XALGOL, BASIC - 999; COBOL - 99; ESPOL, FORTRAN - 10. This file is label-equatable, but is not used with CANDE jobs. FORTRAN default external name; FORSYM.
XALGOL ESPOL	Update symbolic code file	Serial Disk	NEWTAPE	BCL	10 words	150 words	
ALGOL DCALGOL BASIC COBOL FORTRAN	Line printer listing	Line Printer or Backup Disk	LINE	EBCDIC	22 words	22 words	Optional and label-equatable. Produced by setting LIST, LISTP, or LIST1 compiler options. The FORTRAN default external file name is LINE.
XALGOL ESPOL	Line printer listing	Line printer or Backup Disk	LINE	BCL	17 words	17 words	

TABLE 2-2. COMPILER OUTPUT FILES (Cont)

COMPILER	PURPOSE	KIND	FILE NAME	CODE	RECORD SIZE	BLOCK SIZE	COMMENTS
ALGOL DCALGOL XALGOL BASIC COBOL FORTRAN ESPOL PL/I	Error listing	Line Printer or Backup Disk	ERRORFILE	EBCDIC	12 words	12 words	Optional. Selected when ERRLIST compiler option is set. Provides error messages and erroneous-card images. For CANDE jobs, ERRORFILE is automatically label-equated to the remote terminal entering the job and the file is automatically provided. The FORTRAN default external name is ERRORFILE.
ESPOL	Tape file containing object code; used to produce card deck	Card Punch	DECK	EBCDIC	14 words	14 words	Selected by setting compiler option DECK. The tape is suitable for input to SYSTEM/PUNCH; it may be label-equated to tape punch during compilation.
COBOL	Punch input records containing syntax error	Card Punch	PUNCH	EBCDIC	10 words	Unblocked	Optional. Selected by setting PUNCH compiler option. All card images containing syntax errors are placed in EBCDIC on this file to be punched.

TABLE 2-2. COMPILER OUTPUT FILES (Cont)

COMPILER	PURPOSE	KIND	FILE NAME	CODE	RECORD SIZE	BLOCK SIZE	COMMENTS
COBOL	Permanent library file	Serial Disk	SAVEPERM	EBCDIC	14 words	Blocked or unblocked	Optional. Permanent library file created when SAVE <file-title> option is used. SAVEFACTOR = 999. TITLE of file is specified on the SAVE card.
				BCL	10 words		
COBOL	Temporary library file	Serial Disk	SAVEFILE	EBCDIC	14 words	210 words	Optional. Discarded after compilation is terminated. TITLE = "COBOL/SAVE". Created by use of SAVE card (with file-name omitted). Recalled by FROM option.

COMPILER CONTROL CARDS

SAVE delimits the portions of the source input to be saved, and a FROM <sequence-number> card is used to recall this saved input where <sequence-number> is the sequence number of the SAVE option card initiating the save of the source code desired. The temporary file has the internal name SAVEFILE and the TITLE COBOL/SAVE, and contains EBCDIC-coded 14-word records in 210-word blocks. Following compilation of the program, the temporary library file is discarded.

Compiler File Tables

The available compiler input files are listed in table 2-1, and the available compiler output files are listed in table 2-2. These tables denote the compilers which are associated with each file, the purpose served by the file, the file KIND, the file name, the character code employed in the file, and the size of the file record and block. These tables also include brief commentarles of each file.

COMPILER CONTROL CARDS

The manner in which B 6700 compilers handle source language input may be controlled by the user. To a limited degree the user may specify the extent to which the input source code is to be examined for errors, the manner in which the compiler is to receive this source input, and the form of the generated compiler output. The controlling information may be input to the compiler at any point within the primary or secondary input files by means of a compiler control card (or card image in the case of disk or tape input). A compiler control card is identified by the appearance of a dollar sign ($) in an appropriate column of the card. The control information is punched on this card following the dollar sign. For PL/I, control card specifications are enclosed in quotation marks. For FORTRAN input the dollar sign must appear in columns 1 or 2 of the card, with control information being punched in columns 2 or 3 through 72 (sequence numbers fall in columns 73 through 80).

For ALGOL, DCALGOL, ESPOL, and XALGOL input the same compiler control card format is followed as for FORTRAN input except that the dollar sign may be punched on the card in either column 1 or on the first nonblank character position after column 1. The control information may then be punched in the succeeding columns (through column 72); sequence numbers fall in columns 73 through 80. When the dollar sign appears in a card column other than column 1, this compiler control card image will be included in the new source language files as well as in the output listing if such output files are generated. If the dollar sign appears in column 1, the control card image will not be included in the new source language file; this card image will appear on the output listing only if the compiler option "$" is set. Regardless of the column in which the dollar sign is placed on a compiler control card, the compiler control information following that dollar sign will be obeyed for the compilation in progress. No part of the compiler control information (including the dollar sign) may be placed after column 72 of the card.

For COBOL input the dollar sign may appear in either column 7, 8 or 1. If it appears in column 7 or 8, columns 1 thru 6 are to be either a sequence number or all blanks. If the $ appears in column 8, the $ card image is placed in the new symbolic. If a $ appears in column 1 or column 7 that $ card is not written to the new symbolic. Control card information follows the dollar sign and may not extend past column 68.

For BASIC input the dollar sign may appear in the first nonblank column of the card or in the first nonblank column of the card following the statement number. Compiler control information may be inserted in a CANDE file at any point in that file by creating records which have the dollar sign in the first nonblank character position following the statement number. The compiler control information follows the dollar sign.

The compiler control information may consist of an option action (or actions), an option (or options), and/or option parameters (i.e., literals associated with various options). These items are discussed in the following paragraphs.

Option Actions

Option actions include SET, RESET, POP, SET =, and may be omitted (not specified). (The "SET =" option action is not provided by the COBOL, BASIC, or FORTRAN compilers.) The range of the option action is the option list which follows it on the compiler control card. If the option action is omitted, the option(s) which follow will be set and all other options (except ERRLIST for ALGOL and DCALGOL, LIBDOLLAR and B6700 for COBOL, and user options) will be reset.

This does not apply for special-action compiler options such as AREACLASS, FROM, GO TO, INCLUDE, LEVEL, LIMIT, and PAGE. These seven options cannot be set or reset in the normal sense, and their appearance on compiler control cards with omitted option actions causes no other option to be reset.

If no option action and no options or parameters appear on a compiler control card, then the resulting action will be to delete from the secondary input file the card image with a sequence number identical to the sequence number on this compiler control card. For ALGOL, DCALGOL, ESPOL, and XALGOL, the dollar sign on this control card must appear in column 1.

If the option action is SET, the option(s) following the option action will be set and the state of the other compiler options will be unchanged (the default setting if not previously specified). If the option action is RESET, the option(s) following it will be reset and the state of all other options will be unchanged. If the option action is POP, then the option(s) which follow it will revert to their immediately previous setting. The "SET =" option action will be discussed in a later paragraph.

COMPILER CONTROL CARDS

An option whose default state is RESET is initially assigned a 48-bit stack filled with 0's; an option whose default state is SET is initially assigned a 48-bit stack with a 1 on top and 0's in the remaining positions. Each SET option action then causes the stacks allocated to the designated options to be pushed down one bit and a 1 to be placed on the top of each of these stacks. Each RESET causes the appropriate stacks to be pushed down one bit and a 0 to be placed on the top of these stacks. POP causes the stacks corresponding to the designated options to be popped up one bit, causing the associated options to revert to their previous state. Since the size of these option stacks is 48 bits, a maximum history of 48 states may be recorded. When a compiler control card appears with an option and an omitted option action, then the stack record of option state histories is discarded, negating the accuracy of the POP option action.

The "SET =" option section allows the state of one compiler option to control the setting or resetting of another compiler option. This construct is known as an "explicit set," and has the following syntax:

```
<explicit set> ::= SET <option> = <condition><user option>

<condition> ::= NOT <empty>
<option> ::= <standard option><user option>

<user option> ::= {an identifier, other than a standard option or a
                   parameter, which may have previously appeared on
                   a compiler control card}

<standard option> ::= {any option provided by the compiler}

<empty> ::= {blank}
```

An example of the explicit set is:

```
SET LIST = NOT MYOPT
```

which sets the standard option LIST to the opposite state of the user option MYOPT.

The explicit set does not apply to the COBOL, BASIC, or FORTRAN compilers, and user options are not implemented for these two languages. Any user option may be used to control the setting of any other option (user or standard) by use of an explicit set. When an explicit set appears in a compilation deck, the state (set or reset) of the user option following the equal sign is evaluated. The option preceding the equal sign is then set to the same state (if <condition> is <empty>) or to the opposite state (if <condition> is NOT) and this setting is recorded in the option's history stack. If the user option has not previously been either set or reset, it is assumed to be reset.

Options

B 6700 compiler options are presented in table 2-3 and discussed in later paragraphs. PL/I compiler options are discussed at the latter portion of this section.

TABLE 2-3. COMPILER OPTIONS

OPTION	COMPILERS								
	ALGOL	BASIC	COBOL	DCALGOL	ESPOL	FORTRAN	XALGOL	NDL	BINDER*
AREACLASS	X		X	X					
ASCII						X			
AUTOBIND	X			X		X			
BCD						X			
BCL	X			X		X			
BEGINSEGMENT	X			X	X				
BIND	X			X		X			
BINDER	X			X		X			
BINSEQ							X		
BUMP					X				
B2500			X						
B5500						X			
B5700			X			X			

*Despite the fact that the BINDER is not a compiler, BINDER options are listed here as an easily available reference.

TABLE 2-3. COMPILER OPTIONS (Cont)

OPTION	COMPILERS								
	ALGOL	BASIC	COBOL	DCALGOL	ESPOL	FORTRAN	XALGOL	NDL	BINDER*
B6700			X						
B7700						X			
CHECK	X	X	X	X	X	X	X		
CODE	X	X	X	X	X	X	X	X	X
CODEN									X
COMP			X						
COPY						X			
COUNT					X				
DEBUG			X			X			
DECK					X				
DUMPINFO	X			X	X				
EBCDIC						X			
ENDSEGMENT	X			X	X				

*Despite the fact that the BINDER is not a compiler, BINDER options are listed here as an easily available reference.

TABLE 2-3. COMPILER OPTIONS (Cont)

OPTION	COMPILERS								
	ALGOL	BASIC	COBOL	DCALGOL	ESPOL	FORTRAN	XALGOL	NDL	BINDER*
ERRLIST	X	X	X	X		X	X		X
EXTERNAL	X			X		X			
FORMAT	X			X					
FREE			X			X			
FREETAPE						X			
FROM			X						
GLOBAL			X						
GO TO	X			X	X				
GRAPH						X			
HOST	X			X		X			X
<identifier>	X			X					
INCLNEW	X			X	X	X			
INCLUDE	X			X	X	X			
INFO			X						

*Despite the fact that the BINDER is not a compiler, BINDER options are listed here as an easily available reference.

TABLE 2-3. COMPILER OPTIONS (Cont)

OPTION	COMPILERS								
	ALGOL	BASIC	COBOL	DCALGOL	ESPOL	FORTRAN	XALGOL	NDL	BINDER*
INITIALIZE	X			X		X	X		
INSTALLATION	X			X		X			
INTRINSICS	X			X	X				X
LEVEL	X		X	X	X	X			
LIMIT	X	X	X	X	X	X	X		
LINEINFO	X	X	X	X	X	X	X		X
LINESIZE		X							
LIST	X	X	X	X	X	X	X	X	X
LISTDELETED	X			X					
LISTP	X			X	X		X		
LISTI			X						
LOADINFO	X			X	X				
LONG						X			
MERGE	X	X	X	X	X	X	X	X	

*Despite the fact that the BINDER is not a compiler, BINDER options are listed here as an easily available reference.

TABLE 2-3. COMPILER OPTIONS (Cont)

OPTION	COMPILERS								
	ALGOL	BASIC	COBOL	DCALGOL	ESPOL	FORTRAN	XALGOL	NDL	BINDER*
MONITOR					X	X			
NEW	X	X	X	X	X	X	X	X	
NEWID			X						
NEWSEGMENT						X			
NEWSEQERR	X		X	X	X	X	X		
NORMAL					X				
NOWARN						X			
OLDBASIC		X							
OLDEXPO					X				
OMIT	X			X	X		X		
OMITDEBUG						X			
OPT						X			
OPTIMIZE			X						
OWN			X			X			

*Despite the fact that the BINDER is not a compiler, BINDER options are listed here as an easily available reference.

TABLE 2-3. COMPILER OPTIONS (Cont)

OPTION	COMPILERS								
	ALGOL	BASIC	COBOL	DCALGOL	ESPOL	FORTRAN	XALGOL	NDL	BINDER*
OWNARRAYS						X			
PAGE	X		X	X	X	X	X		
POOL					X				
PUNCH			X						
PURGE									X
READLOCK					X				
SAVE			X		X				
SECGROUP			X						
SEGS	X			X	X		X		X
SEPARATE	X			X					X
SEQ	X	X	X	X	X	X	X	X	
SEQERR	X	X	X	X	X	X	X		
SINGLE	X	X	X	X	X	X	X	X	
SPEC			X						

*Despite the fact that the BINDER is not a compiler, BINDER options are listed here as an easily available reference.

TABLE 2-3. COMPILER OPTIONS (Cont)

OPTION	COMPILERS								
	ALGOL	BASIC	COBOL	DCALGOL	ESPOL	FORTRAN	XALGOL	NDL	BINDER*
STACK	X	X	X	X	X	X	X		X
STOP						X			X
SUPRS						X			
SYNTAX								X	
S360			X						
TIME	X			X		X	X		X
TRACE			X			X			X
USE									X
USASI			X						
VECTORMODE						X			
VERSION	X			X	X				
VOID	X	X	X	X	X	X	X	X	
VOIDT	X	X	X	X	X	X	X	X	
WARN									X

*Despite the fact that the BINDER is not a compiler, BINDER options are listed here as an easily available reference.

TABLE 2-3. COMPILER OPTIONS (Cont)

OPTION	COMPILERS								
	ALGOL	BASIC	COBOL	DCALGOL	ESPOL	FORTRAN	XALGOL	NDL	BINDER*
XDECS	X			X	X				
XREF	X	X	X	X	X	X			
XREFS	X			X	X				
$	X	X	X	X		X	X		

*Despite the fact that the BINDER is not a compiler, BINDER options are listed here as an easily available reference.

Options offered by the B 6700 compilers are discussed alphabetically in the following paragraphs. The default setting of each option is shown in parentheses following the name of the option; the effect of setting the option is described in the accompanying paragraph. (The default state of the LIST option is SET and the default states of all other options are RESET unless the compiler is utilized through the CANDE language. If the compiler is called from CANDE, the default state of the LIST option is RESET and the default state of the ERRLIST option is set.) The compiler options are:

AREACLASS (cannot be SET or RESET)

The proper format for this compiler option is: AREACLASS <integer>, where <integer> is any non-negative integer less than 256. This option assigns the value of <integer> to the AREACLASS attribute of the object code file when such a file is produced by the compiler. Option AREACLASS should appear on a compiler control card with an omitted option action.

ASCII

The capability of handling data and source programs in ASCII has been added to the software. In general, all compilers (excepting ESPOL and XALGOL) will accept source programs in ASCII. By use of the MCP soft translation feature, data files can be accepted by setting filetype to eight on label equation cards. Some specific changes to individual compilers follow.

For ALGOL, a new type of ASCII has been implemented. Character arrays may be declared to be of ASCII type. Pointers become ASCII pointers by giving them a length attribute of seven (although each ASCII character still takes up eight bits).

ASCII may be used in the TRUTHSET and TRANSLATETABLE constructs to permit software comparisons and translations within the ASCII character set. Because of hardware limitation, however, it is not permitted to replace an ASCII pointer for a specified number of digits. In addition, the integer and double type transfer functions for pointers are not available for ASCII pointers. These may result in errors at execution time.

The implementation of ASCII in COBOL permits internal usage of ASCII data. This may be done by adding the declarative USAGE IS ASCII. Such usages for fields declared at other than the 01 level must be contained within 01 level fields declared as ASCII usage.

All legitimate non-mixed mode comparisons will work correctly. In addition, certain mixed mode comparisons have been implemented. ASCII-BCL comparisons will use the ASCII collating sequence for comparisons, while ASCII-EBCDIC comparisons will use the EBCDIC collating sequence. Pure ASCII operations will operate according to the ASCII collating sequence.

COMPILER CONTROL CARDS

Numeric and numeric edited items are not permitted in ASCII. A USAGE IS ASCII field may not have subordinate COMP, COMP-2, or DISPLAY-1 items.

Files for which the first RECORDAREA is declared as USAGE IS ASCII will be assigned an internal mode of ASCII. These files will, therefore, return ASCII data.

The string delimiting character in ASCII FORTRAN source decks remains the apostrophe. Two apostrophes must appear adjacent to each other inside a string to indicate one apostrophe in the resulting datum. All comparisons are based upon the EBCDIC collating sequence.

AUTOBIND (SET or RESET)

Autobinding is a compiler option which combines the processes of compiling and program binding into one job. During compilation, the compiler produces a set of instructions to be passed to the BINDER. In many cases, these BINDER instructions are self-sufficient for binding purposes and the user need not be concerned with BINDER control cards. In those cases where BINDER instructions are required, the user may insert his own BINDER control cards.

The autobinding feature of the compiler is invoked by the dollar option AUTOBIND. The AUTOBIND option may be set or reset at any point throughout compilation. However, it is recommended that it be set or reset only once at the beginning of compilation for the following two reasons:

a. Only the status of AUTOBIND at the end of compilation is significant. Specifically, if four procedures are being compiled, the first three with AUTOBIND reset and the last one with AUTOBIND set, the binder will still attempt to bind all four procedures to the specified HOST.

b. Compile and go on a separate procedure with AUTOBIND reset, will result in the MCP attempting to execute the procedure as soon as it is compiled. This may result in errors as the procedure has not yet been bound into a HOST. If AUTOBIND is set throughout compilation, execution of the resultant program will take place after binding.

   In ALGOL a separate procedure compiled at level two or an outer block may serve as a HOST for binding. In FORTRAN, a main program may serve as a HOST. Separate ALGOL procedures compiled at level three (default level) or FORTRAN subprograms may be bound into a HOST. At most, one HOST may be compiled in a job along with any number of separate procedures or subroutines. In ALGOL, the HOST must be the last program unit compiled. If an appropriate HOST file is compiled with AUTOBIND set, it is assumed to be the HOST for binding. This assumption cannot be overridden by either of the methods given next for specifying a HOST.

If no eligible HOST is being compiled, a HOST must be specified. Two methods are available:

```
<1> COMPILER FILE HOST (TITLE = FILE1D1/----/FILE1DN)
```

or a BINDER HOST card, such as

```
$ HOST IS FILE1D1/----/FILE1DN;
```

The code file of any level three procedure compiled with AUTOBIND set is marked as (non-executable). If not executed via inter-program communication, the procedure must be bound into a HOST file by the BINDER before being executed.

Code files of any level three procedures (or higher) compiled are purged after being bound into a HOST by AUTOBIND. To retain such as code file, it is necessary to refer to it specifically in a BINDER control statement. Either of the following statements will allow the procedures code file to remain:

```
$ BIND PROCEDURENAME

        or

$ EXTERNAL PROCEDURENAME
```

The first statement performs the same function as the default compiler-generated bind statement, except the code file will not be purged. The second statement instructs the BINDER not to bind the procedure into the HOST even though it has been compiled with AUTOBIND set and there is an external reference to it in the HOST.

BCD (RESET)

The FORTRAN compiler accepts 6-bit BCD-coded characters as input if the input file is labeled using a <1> BCL card (not the BCL compiler option) and if the compiler option BCD is set. Once BCD is set within a file, it may not be reset within that file.

BCL (RESET)

When the compiler option BCL is set, the default character size is 6-bit; otherwise, the default character size is 8-bit.

BEGINSEGMENT

Dollar options BEGINSEGMENT and ENDSEGMENT have been implemented in ALGOL and ESPOL to allow further user control of procedure segmentation. Procedures encountered between the BEGINSEGMENT and ENDSEGMENT will all be placed in the same segment.

COMPILER CONTROL CARDS

The ENDSEGMENT must appear after the last source image of the last procedure in the user segment.

Only procedures and blocks completely contained inside a procedure in the segment may be included in a user segment.

User segments may be nested; that is, a BEGINSEGMENT may appear in a user segment. In this case, an ENDSEGMENT applies to the user segment currently being compiled.

If a $ BEGINSEGMENT appears before the beginning of a separately compiled procedure, a $ ENDSEGMENT is assumed at the end of the procedure even if none appears. The driver procedure created for procedures compiled at level three will always be in a different segment.

The printout for segment information is modified for user segments. User segments will be numbered consecutively in a program beginning with one; that is, the first BEGINSEGMENT will create USERSEGMENT1, the second BEGINSEGMENT will create USERSEGMENT2. At the beginning of a user segment, its segment number will be printed out. The length of each user segment will be printed at its end. Procedures that would normally be segmented, but are not because of user segmentation, will print out as being "IN" a particular segment.

External procedures may not be declared in a user segment.

Forward procedure declarations are not affected by user segmentation.

A procedure may not be split across user segments.

In ESPOL only, user segmentation should not be used if stack building code in a USERSEGMENT invokes a procedure in that USERSEGMENT. For example:

```
$ BEGINSEGMENT
REAL PROCEDURE P;
   BEGIN
     P:=1;
   END;
REAL R:=P;
$ ENDSEGMENT
```

will result in incorrect code.

Warning Messages:

If more than one $ BEGINSEGMENT appears before a procedure, the warning message EXTRA $BEGINSEGMENT IGNORED will be printed.

If a $ ENDSEGMENT appears when the user is not controlling segmentation, the warning message EXTRA ENDSEGMENT IGNORED will be printed.

Explanation of Error Messages:

1. External procedures illegal in a user segment:

   An external procedure declaration has appeared in a user segment.

2. User segments may contain only procedures:

   An attempt has been made to include a non-procedural block in a user segment.

3. Illegal BEGINSEGMENT or ENDSEGMENT:

   A BEGINSEGMENT or ENDSEGMENT card has been seen outside declarations.

4. A procedure may not be split across segments:

   ENDSEGMENT missing after nested USERSEGMENTED procedure.

5. ENDSEGMENT required:

   A BEGINSEGMENT appeared and there was no ENDSEGMENT.

6. User segment may not have save procedures (ESPOL only):

   An attempt was made to include a save procedure in a user segment.

BIND (AUTOBINDING Only)

Format is similar to dollar card, but it is used to pass control statements to BINDER when AUTOBINDING to ALGOL, DCALGOL, and FORTRAN.

BINDER (AUTOBINDING Only)

Allows passing of dollar cards to BINDER when AUTOBINDING in ALGOL, DCALGOL, and FORTRAN when immediately following the $ sign. The compiler passes the remainder of the card intact with a $ sign in front.

BINSEQ (RESET)

When set, the XALGOL compiler option, BINSEQ, causes the character comparisons in string scan statements, string transfer statements, and conditional expressions to be made according to the B 6700 BCL collating sequence. When BINSEQ is reset, then such comparisons will be made on the basis of the XALGOL (B 5700 BCL) collating sequence. (Character comparisons will be considerably faster when BINSEQ is set.)

COMPILER CONTROL CARDS

BUMP TO

The BUMP TO dollar parameter provides for incrementing the resequence number to produce a specified contents for the least significant one to seven digits of the resequence number. The syntax is: BUMP TO <digit string>: the succeeding sequence number will be bumped to the lowest number ending in the specified string. Normal resequencing will then continue (BUMP TO is meaningful only when resequencing).

The following examples show the current reseq number, the BUMP argument, and the new reseq number:

OLD:	12340000	12345678	12345678	12345678
BUMP TO:	0000	0000	5000	6000
NEW:	12340000	12350000	12755000	12346000

BUMP or BUMP TO may be used for the $ card BUMP option.

B2500 (RESET)

B2500 is a COBOL compiler option pertaining to all models of the B 2500, B 3500, B 3700, and B 4700 systems. (See B6700 option for additional system option descriptions.)

B5500 (RESET)

When set, the B 5500 option causes the B 6700 compiler to become compatible with the B 5500 (and B 5700) system.

B5700 (RESET)

B5700 is a COBOL compiler option pertaining to all models of the B 5500 and B 5700 Systems. (See B6700 option for additional system option descriptions.)

B6700 (SET)

B6700 is a COBOL compiler dollar option which restricts recognition of reserved words. Setting of a system option excludes from recognition as a reserved word any word which is reserved on the B 6700 compiler, but not reserved on the system indicated by the system option which is then set. Thus, the word MODIFY is reserved only when the system option B6700 is set, but when B6700 is not set, the word MODIFY may be declared to be (and used as) a data-name, procedure-name, etc.

A system option may be set by a $ CARD or by a $ SET CARD. Any attempt to POP or RESET a system option will result in a warning message and the system option will be ignored. Setting of a system option causes all other system options to be reset. A $ CARD without SET, POP, or RESET will reset all options except those specified on the $ CARD and will leave the system option set to B6700 unless some other system option is specified by that $ CARD. If no $ CARD is used, then B6700 is assumed.

Under control of the system options, several new words have been added to the reserved word list. The uses of these words are described in the COBOL manuals for the applicable systems. The following is a list of these new words with indication of the system option setting(s) which cause them to be recognized as reserved words:

B2	B5700, B2500
CMP	B5700, B2500
CMP-1	B5700, B2500
PC	B5700, B2500
MD	B5700
SY	B5700, B2500
REMARKS	B5700, B2500, USASI, S360
NOTE	B5700, B2500, USASI, S360
UNIT	B6700, B2500, USASI, S360
PROCESSING	B2500, USASI, S360
OC	B6700, B5700, B2500
VA	B6700, B5700, B2500
VALUES	B6700, B5700, USASI, S360
OTHERWISE	B6700, B2500, USASI
PT-READER	B2500
PT-PUNCH	B2500
TAPE-7	B2500
TAPE-9	B2500
TAPE-PE	B2500
DISK	B2500

B7700 (RESET)

B7700 is a FORTRAN compiler option which specifies that code is to be tuned for the B 7700 execution rather than tuned for B 6700 execution.

CHECK (RESET)

When set, the CHECK compiler option causes the source-language input to be sequence-checked and sequence errors to be listed on the output listing. In the ALGOL and DCALGOL compilers, the sequence numbers of the output symbolic file are also checked for errors and handled in the same manner as input sequence errors. When CHECK is reset the compilers make no check of sequence numbers. (See SEQERR and NEWSEQERR.)

CODE (RESET)

When set, the CODE compiler option causes the output listing to contain the object code generated by the compiler.

CODEN (RESET)

CODEN is a BINDER option which, when set, causes the input object code to be included in the output listing.

COMPILER CONTROL CARDS

COMP (RESET)

COMP is a COBOL compiler option which controls the manner in which 77-level COMPUTATIONAL items (not COMP-1 items) are stored in memory. When COMP is set, all such COMPUTATIONAL items are grouped into a data array. A data descriptor is placed in the program stack, and all 77-level COMPUTATIONAL items are referenced by means of this descriptor. When COMP is reset, each COMPUTATIONAL item is assigned a location in the program stack (just as COMPUTATIONAL-1 items are) and each COMPUTATIONAL item may then be accessed directly.

COPY (RESET)

COPY is a FORTRAN option which when set causes symbolic card images to be included from a named file. The file name appearing in the copy control statement is prefixed by "FTNLIBR" to obtain the title of the file to be included.

COUNT (RESET)

COUNT is an ESPOL compiler option which, when set, counts entries to MCP procedures. This information is placed on the file titled COUNTS/<MCP codefile TITLE> (e.g., COUNTS/SYSTEM/MCP) which may be processed by the SYSTEM/COUNTANALYZER program. This program may be initiated by the "CA" system input message.

DEBUG (RESET)

DEBUG is a COBOL and FORTRAN compiler option which, when set, causes lists of stack locations and names, scanned entities, and compiler procedure entries and exits to be included in the output listing. This option is intended primarily to facilitate compiler development and its function may change without notice.

DECK (RESET)

The ESPOL compiler option, DECK, when set causes a code file to be generated on compiler output file DECK. This is an EBCDIC-coded magnetic tape file that may be used as input to utility program SYSTEM/PUNCH to create a card deck. (File DECK may be label-equated to a card punch to directly produce a card deck.) When set, option DECK causes option POOL to be set also; thus, data pools will be made SAVE arrays and included in the output file.

DUMPINFO

This dollar option which is used with the LOADINFO dollar option and INFO file, enables the user to save or load the contents of the main table in the ALGOL and ESPOL compilers via the file. The tables saved include INFO, ADDL, TEXT, and STACKHEAD arrays, plus several simple variables.

Expected usage is in conjunction with separate compilation of procedures. One would typically compile all GLOBAL declarations and dump the tables to file INFO. Subsequent compilations of procedures merely load the INFO and GO TO the start of the procedure symbolic.

Example (ESPOL):

```
<I>COMPILE MAKE/INFOFILE  ESPOL LIBRARY
<I>ESPOL FILE TAPE=SYMBOL/MCP
<I>ESPOL FILE INFO=MY/INFO
<I>BCL
   $ MERGE
   [
   <GLOBAL DECLARATIONS>
   $ DUMPINFO
   ]
<I>END

<I>COMPILE MY/MCP  ESPOL LIBRARY
<I>ESPOL FILE TAPE=SYMBOL/MCP
<I>ESPOL FILE INFO=MY/INFO
<I>BCL
   %%%       LOAD THE GLOBALS INTO TABLES.
   [
   $ LOADINFO
   <ADDITIONAL GLOBAL DECLARATIONS>
   ]
   $ SET LIST STACK <ONE OR MORE SEPARATE PROCEDURE
   DECLARATIONS>
<I>END
```

Example (ALGOL):

```
<I>COMPILE MAKE/INFOFILE  ALGOL LIBRARY
<I>ALGOL FILE INFO=MY/INFO
<I>DATA
   [
   <GLOBAL DECLARATIONS>
   $ DUMPINFO
   ]
END.
<I>END

<I>COMPILE MY/PROGRAM  ALGOL LIBRARY
<I>ALGOL FILE INFO=MY/INFO
<I>DATA
   %%%     LOAD THE GLOBALS INTO TABLES.
   [
   $ LOADINFO
   <ADDITIONAL GLOBAL DECLARATIONS>
   ]
   <SEPARATE PROCEDURE DECLARATION>
<I>END
```

COMPILER CONTROL CARDS

The ALGOL dollar card options, DUMPINFO and LOADINFO, have been modified to facilitate their use with intermediate level global binding. These changes are:

1. The DUMPINFO and LOADINFO options may be followed by either an internal file name or an external file name and terminated with a period. This file name information is in a format similar to the include dollar option. This permits selective INFO dumping at several points and selective INFO loading more than once throughout a compile.

2. DUMPINFO and LOADINFO must now be the last option appearing on a dollar card.

3. When a new LOADINFO is done, all old INFO structure in ALGOL is removed. Thus, compiling different portions of the same program, even if they operate in different environments, may now be done in the same compilation.

4. LOADINFO changes all variables in INFO to be globals and all procedures already compiled to be forward. This means that an INFO file created by a DUMPINFO done immediately before a procedure in a normal compile will be suitable for future use as globals if one wishes to separately compile that procedure.

In general, the effect of these changes is to considerably increase the number of places where DUMPINFO and LOADINFO may appear in order to produce and use an INFO file suitable for separate compilation. Caution is generally required only when variables with the same name are declared at different levels; a separate compilation will only be able to access the last such variable seen before the DUMPINFO occurred.

EBCDIC (SET)

When the FORTRAN compiler option EBCDIC is set the default character size is 8-bit; otherwise, the default character size is 6-bit.

ENDSEGMENT

See BEGINSEGMENT description.

ERRLIST (RESET, SET when used with CANDE)

When set, the ERRLIST compiler option causes an error listing to be generated on compiler output file ERRORFILE. When an error is detected in input source code, then the offending line of code, the error message, and a pointer to the syntactical item in question will be written on two lines in the file ERRORFILE. This option is provided primarily for use when the compiler is called from a remote terminal by the CANDE language, but it may be used regardless of how the compiler is called. When the compiler is called from CANDE, option ERRLIST is automatically set and ERRORFILE is automatically equated to the remote device involved. (For the ALGOL and DCALGOL compilers, once set, ERRLIST may be reset only through a RESET, POP, or "SET" control statement.)

EXTERNAL (AUTO BINDING ONLY)

EXTERNAL causes designated program units to remain external to the program. (BINDER will normally attempt to bind all external program units.)

FORMAT (RESET)

FORMAT is an ALGOL and DCALGOL compiler option which, when set, causes several blanks to be included in the output listing after each procedure declaration to aid reading of the listing.

FREE (RESET, SET for CANDE)

This compiler option, when set, causes the compiler to accept free form input from the source file called CARD. This option is initially set for all jobs initiated through CANDE, and reset otherwise.

A dollar sign ($) in column one or a dollar sign in column two with a blank in column one causes the FORTRAN compiler to interpret this card image as a compiler option card.

The FREE option in COBOL allows all input to begin in column seven of the card or tape image. A hyphen in column seven indicates continuation, while a $ in column seven indicates a dollar card.

To put a comment in the symbolic file, an asterisk (*) or a slash (/) indicates that the rest of the card is a comment, provided that it appears in column seven.

All PARAGRAPH, SECTION, and DIVISION headings must begin in columns 7-11 inclusive. All other source images may begin at any position in the record. Under this option, columns 73 - 80 are still treated as comments.

When the FREE option is used with CANDE, certain conventions are followed by CANDE in mapping the input line into a record. For COBOL symbolic files, the sequence number is right-justified in columns one through six; the text field (beginning with the first non-numeric character or the seventh character, whichever occurs first), is placed in the record beginning at column seven. For FORTRAN, the sequence number is right-justified in columns 73 through 80; the text field (beginning with the first non-numeric character, or the eighth character, whichever occurs first), is placed in the record beginning at column one. Thus, the first text character typed by the CANDE user is the first character examined by each compiler to analyze free form records.

FREETAPE (RESET)

This FORTRAN option, when set, causes the compiler to accept free form input from the TAPE file. This option is initially reset. If NEW is set and either FREE or FREETAPE is set, the compiler will perform a minor amount of editing on the output NEWTAPE file, to ensure that it is acceptable as either free or fixed form input.

COMPILER CONTROL CARDS

FROM (cannot be SET or RESET)

The COBOL compiler option, FROM, when used, must be the first option on a compiler control card, with the option action omitted. The proper format for the FROM option is: FROM <file part>, where <file part> may be either a file title, a file title followed by a sequence number, a file title followed by two sequence numbers separated by a hyphen, or a single sequence number. The FROM option permits the recall of records from permanent or temporary library files to be inserted into the source input file. If the word FROM is immediately followed by a file title, then the records will be recalled from a permanent file with this title. The entire permanent file will be recalled. The file title may optionally be followed by a sequence number. In this case, recall will begin at the indicated sequence number in the permanent file (or at the next highest sequence number if the given sequence number is not in this permanent file) and will continue until the end of the permanent file. If the file title is followed by a sequence number which is in turn followed by a hyphen and a second sequence number, then recall will begin with the first sequence number and will continue through the second number.

If the file title is omitted and the word FROM is immediately followed by a sequence number, this sequence number must be the sequence number of a SAVE compiler option card appearing in the same program deck. All records (stored in a temporary library file earlier in the compilation by the SAVE card with the given sequence number) will be recalled and inserted in place of the FROM card. If the sequence number given does not correspond to any such SAVE compiler control card, then a warning message is printed and the recall is not performed.

GLOBAL (RESET)

This option causes all 01 and 77 level items in the WORKING STORAGE SECTION of a COBOL program to assume the descriptive clause GLOBAL.

GO TO (cannot be SET or RESET)

This compiler option, when used, should appear with no other options on a compiler control card and must not be preceded by an option action. The proper format for the GO TO option is: GO TO <sequence number>, or GO <sequence number>, where <sequence number> is the sequence number appearing on a card image in the TAPE file. The GO TO compiler option causes the input TAPE file to be repositioned so that the next card image used from this file by the compiler will be the first card image with a sequence number greater than or equal to <sequence number>. This option cannot be used in a DEFINE declaration or in INCLUDED text. (See INCLUDE.) File TAPE must be properly sequenced in ascending order; that is, each sequence number on each card image in the file must be greater than the preceding sequence number. (Must be disk or disk pack.) One can GO TO a lower sequence number.

GRAPH (RESET)

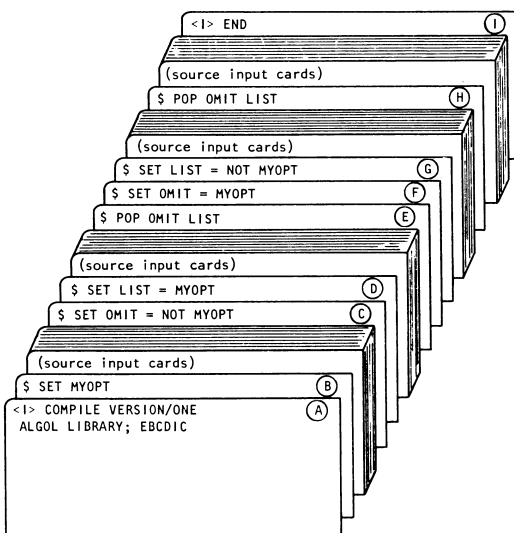
This option causes the FORTRAN compiler to draw a graph of the program only if OPT=1 is set.

HOST

See AUTOBIND description.

<identifier>(RESET)

The <identifier> compiler option allows the user to specify user options to be employed with the explicit set (SET =) option action. <identifier> must not be any identifier which might normally appear on a compiler control card bearing standard options, option actions, or parameters. When this identifier first appears on a compiler control card, it will be recognized as a user option by the compiler. This user option may then be used to explicitly set any other option. The following example shows how to organize a card deck so that specific portions of source input code can be compiled and listed simply by setting a single user option. (See also Sample Card Input Decks.)



COMPILER CONTROL CARDS

In the example deck displayed (assuming no $ in source input cards):

(A) denotes the card bearing the COMPILE and EBCDIC MCP control statements.

(B) defines and sets a user option called MYOPT.

(C) resets the standard compiler option OMIT.

(D) sets the standard compiler option LIST.

(E) returns OMIT and LIST to their states immediately prior to (C).

(F) sets OMIT.

(G) resets LIST.

(H) returns OMIT and LIST to their states immediately prior to (F).

(I) denotes the end of the compilation deck.

Thus, when MYOPT is set (as it is in this example), the source language information on the cards following (D) will be included in the compilation and in the output listing, and the source language information on the cards following (G) will not be included in either compilation or listing. Conversely, if MYOPT is reset (accomplished by removing card (B)), the information on the cards following (D) will not be included and the information on the cards following (G) will be included in the compilation and listing. The source language information on the cards immediately following (B) and (H) will be included in both compilation and output listing whether or not MYOPT is set.

INCLNEW (RESET)

When set, the INCLNEW compiler option causes source text which is included by use of INCLUDE compiler control cards to be output to file NEWTAPE (if NEW is set). If INCLNEW is reset, then the included text will not be output to NEWTAPE. If NEW is reset, the state of INCLNEW is ignored.

INCLUDE (cannot be SET or RESET)

The INCLUDE compiler option must appear alone on a compiler control card with the option action omitted. The INCLUDE card permits indirect source language input to the compiler from files other than CARD or TAPE. This card, therefore, is used to specify that a portion of another file is to be included in the source language input at the point the INCLUDE appears. It is possible for included card images themselves to contain INCLUDE cards; however, included text may be nested no deeper than five levels. FILETYPE is 7.

The proper syntax for the INCLUDE compiler control statement is:

<include statement> ::= INCLUDE <file option> <sequence option>

<file option> ::= <empty> | <internal name> | <title>

<sequence option> ::= <empty> | <start option> | <start option> <stop option> | <stop option>

<internal name> ::= {program file internal name}

<title> ::= {file TITLE enclosed in quotes}

<start option> ::= * | <sequence number>

<stop option> ::= <dash> <sequence number>

<sequence number> ::= {unsigned integer up to 8 digits long}

<dash> ::= -

In the INCLUDE control statement, <file option> specifies the file to be included. The use of <internal name> allows the name to be used for label equation purposes. The use of <title> allows the actual title of the file to be included. If <file option> is empty, then the same file as the one specified on the previous INCLUDE card at the same level of nesting is included. Thus, the first INCLUDE card at any of the five possible levels of nesting must contain either a <title> or an <internal name>.

<start option> specifies the sequence number of the first card image to be included from the file. If <start option> is missing, inclusion will begin with the first record of the file; if <start option> specifies a <sequence number>, inclusion will begin with that record. If the * form of <start option> is used, inclusion will begin at the point at which it left off the previous time that inclusion took place on this file at this level of nesting.

<start option> specifies the sequence number of the last card image to be included from this file. If <stop option> is missing, then the last record of the file will be the last record included.

INFO (RESET)

INFO is a COBOL compiler option which, when set, causes the compiler's INFO array to be printed whether or not LIST is set. (The COBOL compiler INFO array contains six words for each identifier in a program.) This option is intended primarily for compiler development use.

INITIALIZE (AUTOBINDING Only)

INITIALIZE is used in intrinsic binding for the purpose of allowing non-ESPOL intrinsics to refer to MCP procedures with fixed addresses.

COMPILER CONTROL CARDS

INSTALLATION (RESET)

When set, the INSTALLATION option allows the user to use installation (nonstandard) intrinsics. If used, this option must be set before the beginning of the source input, and when this option is reset, only standard intrinsics may be employed.

INTRINSICS (RESET)

The INTRINSICS option causes compilation to occur at level two and allows for global declarations. Thus, the separate procedures being compiled may be bound afterwards into the intrinsic file. When used with the BINDER, the BINDER generates an intrinsic file from scratch as opposed to binding to a HOST intrinsic file. INTRINSICS must be set before the first source statement.

LEVEL (cannot be SET or RESET)

The LEVEL option, when used, controls the lexicographical level at which the compilation is to occur. The proper format for this option is: LEVEL <integer>, where <integer> corresponds to the desired level number. The LEVEL option should not be preceded by an option action. This option allows the user to override the lexicographical levels assigned by the compiler. The default levels are 2 for programs and 3 for separately compiled procedures. Only LEVEL cards which appear befor the start of source language input are considered by the compiler.

LIMIT (cannot be SET or RESET)

The LIMIT option, when used, allows the user to control compiler error termination. The proper format for the LIMIT option is: LIMIT integer . Compilation will be terminated if the number of errors detected by the compiler equals or exceeds integer . (For the COBOL compiler, this check occurs on pass two.) If a limit is not specified by a LIMIT card, compilation will terminate after a default maximum error limit of 150 is detected. If the compiler is called from CANDE, compilation will terminate by default after 10 such errors. For BASIC, limit by default is 100.

LINEINFO (RESET, SET for CANDE)

The dollar card option LINEINFO has been added to all compilers to retain source line identification information for diagnostic purposes. This saves sequence or line number information at compile time and its relation to the code emitted at compile time. If a program should then terminate abnormally or monitoring is occurring, the line information will be displayed or printed, as appropriate.

When a program is discontinued for any reason or when a program dump is obtained, it is now possible to receive source line identification of the point of the program where the error occurred. This facility is also provided with the MONITOR and DUMP statements.

The LINEINFO information may require a significant additional amount of disk storage in the code file of a compiled program. For this reason, it may not be desirable to retain the option set after debugging is completed.

LINEINFO is automatically set for all compilations ordered through CANDE. When binding, the LINEINFO dollar option is set by default so that line information is preserved. Program units with and without LINEINFO may be combined. Resetting LINEINFO will cause the binder to discard all line information. This may be done whether or not any binding is done, even if the HOST is a complete program.

At occurrence of a memory dump under MCP and programs where LINEINFO was reset, SYSTEM/DUMPANALYZER can be FAKED OUT to give a dump output with benefit of LINEINFO by:

1. Recompiling MCP and the program that was running with everything the same except LINEINFO set.

2. Having these recompiled code files present in the disk directory with the exact same title.

LINESIZE (SET, RESET, or POP)

The BASIC compiler option LINESIZE allows the user to specify the size in words of his output (print) line for particular devices. The allowable size range is (5 LEQ LS LEQ 22). This option should be considered a program option. Hence, it must appear before the first source image. Any setting of this option thereafter will produce an error.

LIST (SET, RESET for CANDE and BINDER)

When set, the LIST compiler option causes an output listing to be generated on the compiler output file LINE. (This listing is generated on pass two for the COBOL compiler.) If LIST is reset, then only syntax error messages and the offending card images which caused the errors will be listed.

LISTDELETED (RESET)

When set, the ALGOL and DCALGOL compiler option LISTDELETED causes the inclusion in the output listing of card images from the secondary input file (TAPE) which are replaced, voided, or deleted during the compilation. Four asterisks will appear on the listing to the left of each of these card images. To the right of the card images will appear the words: REPLACED (if the card image was replaced by a card in the primary input file), VOIDT (if the card image was voided from the input file by the VOIDT compiler option), or DELETED (if the card image was deleted by a compiler control card with a dollar sign in column 1 and no option actions, options, or parameters).

COMPILER CONTROL CARDS

LISTP (RESET)

When set, the LISTP option causes patches (input records from compiler file CARD) to be included in the output listing while records from compiler file TAPE are not included. Thus, patches only will be listed when LISTP is set. This option is effective only if the option LIST is reset. If LIST is set, then the state of LISTP is ignored. Therefore, LISTP or LIST causes an output listing to be generated when set.

LIST1 (RESET)

The COBOL compiler LIST1, when set, causes an output listing to be generated on pass one of the compiler. This listing will contain the source language input and syntax error numbers when such errors occur. This listing may be produced whether or not LIST is set.

LOADINFO

See DUMPINFO description.

LONG (RESET)

The FORTRAN compiler option LONG, when set, causes the compiler to suppress segmentation of program arrays even though they are longer than 1024 words. When LONG is reset, the compiler segments arrays greater than 1024 words in length into 256-word segments. (No array may exceed 65,535 words in length.)

MERGE (RESET)

When set, the MERGE compiler option causes primary input (file CARD) to be merged with secondary input (file TAPE) to form the total input to the compiler. If matching sequence numbers occur in the two input files, the primary input overrides the secondary input. When MERGE is reset, only primary input is used and secondary input is totally ignored.

MONITOR (RESET)

This ESPOL compiler option, when set, allows the user to employ the monitoring facilities of the MCP. Existing monitoring procedures of the MCP may be used or the programmer may recompile an MCP with his own monitoring procedure declarations included. (See the B 6700 ESPOL Language Information Manual, 5000094.)

NEW (RESET)

When set, the NEW compiler option causes a new symbolic file generated by the compiler to be placed on compiler output file NEWTAPE. This new file is created even if syntax errors are detected in the symbolic input (from which the new file is constructed) during compilation and may be used as a secondary input file by a later compilation.

NEWID (RESET)

The proper format for this COBOL compiler option is: NEWID <string>, where <string> is an alphanumeric string one to eight characters long enclosed by quotation marks ("). When NEWID is set with this string as a parameter, the output listing and the new source language file (file NEWTAPE) will contain this string in columns 73 thru 80 of each card image as a new program identification. When NEWID is being reset, the string parameter need not appear on the compiler control card resetting the option.

NEWSEGMENT

The FORTRAN compiler option NEWSEGMENT, when it appears on a dollar card, causes the FORTRAN compiler to break the next statement out into a new code segment; thus effectively eliminating size restrictions on program sub-units. Since getting from one segment requires branching, it is suggested that this option be used (if needed) at nodes in the sub-program that are passed infrequently (that is, do not segment in the middle of a loop).

NEWSEQERR (RESET)

When set, the NEWSEQERR compiler option causes the records written to file NEWTAPE (the new symbolic file) to be checked for sequence errors. As each sequence error is discovered, a warning message is placed on the output listing. At the end of the compilation the new source file (NEWTAPE) will not be locked (and thus will not be available to the user) and a message indicating this will be placed on the output listing. Sequence errors detected as a result of NEWSEQERR being set will not interfere with a successful compilation (one in which no syntax errors are detected). NEWSEQERR is effective even if CHECK is reset. (See also CHECK and SEQERR.)

NORMAL (RESET)

Enforces gross organization of MCP symbolic file. That is, procedure FORWARD by declarations, followed by non-procedure declarations, followed by actual procedure declarations.

NOWARN (RESET)

The FORTRAN compiler option NOWARN, when set, causes source input file errors which are flagged as warnings to be considered syntax errors.

OLDBASIC (RESET)

The dollar card option OLDBASIC has been provided so that when methods of handling certain BASIC constructs are changed in B 6700 BASIC, the user may avoid these changes by setting the OLDBASIC option. When OLDBASIC is set, programs will compile and execute as if the change had never been made.

COMPILER CONTROL CARDS

OMIT (RESET)

When set, OMIT causes card images from both file CARD and file TAPE to be ignored. That is, these card images will not be compiled, although they may be listed and/or included in a new source file. On the output listing these images are flagged by the word OMIT. (See also VOID and VOIDT.)

OMITDEBUG (RESET)

When this FORTRAN compiler option OMITDEBUG is set, all debugging statements are to be treated as if omitted. This also applies to continuation cards of debugging statements.

OPT (RESET)

The value of the OPT compiler option specifies the level of optimization to be performed on a given source program. The legal value are 1, 0, and -1.

OPTIMIZE (RESET)

This option when set, causes additional optimizations of object code to be done at compile time.

OWNARRAYS (RESET)

OWNARRAYS applies the OWN function to arrays.

OWN (RESET)

Subroutine variables which are not formal parameters and which are declared or first appear in a portion of source input where the FORTRAN compiler option OWN is set are allocated locations in the D2 program stack. The result of this action is to allow a subroutine variable to contain the value it held at the end of the last call on the subroutine when that subroutine is called again.

For example, consider this subroutine portion of a source language input file:

```
                SUBROUTINE AUGMNT (/X/,/Y/,Z)
                DIMENSION Z(5), HOLD(5)
          $SET OWN
                INTEGER   A
                DIMENSION B(5)
                I=I+1
          $RESET OWN
                X=X+I
                HOLD(1)=Y
                Y=Y+A
                A=HOLD(1)
                DO 1 J=1,5
                HOLD(J)=Z(J)
                Z(J)=Z(J)+B(J)
            1   B(J)=HOLD(J)
                RETURN
                END
```

In this example, the OWN option is used to indicate that the variables A and I and the elements of the array B are to retain their values between subroutine calls. When the subroutine is first called, these items are initialized to zero as is normally the case with local variables. Thus, the function of this subroutine is to add 1 to X during the first call, 2 during the second call, 3 during the third call, etc. During each call, the variable Y and the elements of the Z array are increased by the values passed to these formal parameters when the AUGMNT subroutine was last called. During the first call on AUGMNT, these six parameters are increased by zero.

A further restriction is that an array will be affected by the OWN option only if the specification statement (e.g., the DIMENSION statement) associated with the array lies in a region of code where the OWN option is set.

PAGE (cannot be SET or RESET)

The PAGE compiler option should appear on a compiler control card without an option action preceding it. When a PAGE card appears, the output listing is skipped to the top of the next page.

POOL (RESET)

POOL is an ESPOL compiler option which, when set, causes the compiler to make all data pools SAVE (nonoverlayable) arrays. Thus, if a program does not have presence bit capability, the data pools will be available to the program. (If option DECK is set, POOL is automatically set.) If POOL is reset, the compiler will generate non-present descriptors pointing to the data pools in the code file.

COMPILER CONTROL CARDS

PUNCH (RESET)

PUNCH is a COBOL compiler option which, when set, causes all source input card images containing syntax errors to be written on compiler output file PUNCH. The card images are punched in EBCDIC format even if the input file is coded in BCL format.

PURGE (AUTOBINDING Only)

PURGE causes all input files specified in the <file list> to be removed from the disk directory after binding. Only files opened by BINDER will be surged.

READLOCK (RESET)

The ESPOL compiler option READLOCK, when set, causes the BUZZ intrinsic to emit an increased amount of debugging code. Normally used in the case where the MCP gets hung in a READLOCK loop.

SAVE (RESET)

When set, SAVE specifies that all source language card images between the control card setting the SAVE option and the first control card resetting the SAVE option are to be placed in a permanent or temporary file. If the file is to be a permanent library file, the title of the file must immediately follow the word SAVE on the control card. If no title appears following SAVE, the library file will be discarded when the compilation is ended. Once set, SAVE may be explicitly reset by another compiler control card of the form "$ RESET SAVE" or "$ POP SAVE". SAVE may also be reset implicitly by another compiler card of the form "$ FROM <sequence number>" or "$ FROM <file title>". SAVE may even be reset implicitly by a compiler control card of the form "$ SET SAVE", which terminates one group of records and starts a new group of records to be saved.

SECGROUP (RESET)

This option causes COBOL sections to be combined together to create larger code segments.

SEGS (RESET)

When set, the SEGS compiler option causes an output listing to be generated (when LIST is reset) containing the beginning and ending segmentation messages. Since these messages are printed when LIST is set, the SEGS option has effect only when LIST is reset.

SEPARATE (RESET)

When set, the FORTRAN compiler option SEPARATE causes a separate object code file to be generated during compilation for each complete program unit in a FORTRAN program (for compile-to-library jobs). A FORTRAN

program unit may be a main program, a subroutine subprogram, or a function subprogram. SEPARATE can only be set before the first noncomment card in the source deck, and once this option is set it cannot be reset. The files generated by the SEPARATE option will have TITLE's based upon the program name appearing on the COMPILE control card. The TITLE of the file containing the main program object code will be identical to the program name. The TITLE of each code file corresponding to subprograms will be constructed from the program name with the right-most identifier in this name (e.g., B in A/B) being replaced by the subprogram name.

SEQ (RESET)

When set, the SEQ compiler option causes new sequence numbers to be assigned to the card images on the output listing file and on the new symbolic file when LIST and NEW are also set. The proper format for the SEQ option is: SEQ <sequence base> <sequence increment>, where <sequence base> is an unsigned integer and <sequence increment> is a plus sign (+) followed by an unsigned integer. Either one or both of these two parameters may be omitted (empty) and may appear on a compiler control card other than the SET SEQ card. The starting sequence number is specified by <sequence base>, and each successive sequence number assigned exceeds the previous sequence number by the amount specified by the integer part of <sequence increment>. When one or both of these parameters are missing on a SEQ card, then default values are assigned to these items. The default values for both <sequence base> and <sequence increment> are both 1000 (except for the COBOL compiler, for which the default values are both 10).

SEQERR (RESET)

When set, SEQERR causes the input source language to be checked for sequence errors. If any sequence errors are detected they are indicated on the output listing with warning messages. If, at the end of a compilation, any sequence errors have been detected as a result of SEQERR being set, then the object code file will not be locked and consequently will not be available to the user. Thus, although a program may compile successfully, it will not create a code file if sequence errors are detected. The setting of option CHECK has no effect on the operation of SEQERR. (See also CHECK, NEWSEQERR.)

SINGLE (RESET)

When set, the SINGLE compiler option causes the output listing to be single-spaced. When SINGLE is reset, then the output listing is double-spaced.

COMPILER CONTROL CARDS

SPEC (RESET)

The SPEC option for the COBOL compiler, when set, suppresses the inclusion in the output listing of warning messages, sequence error warning messages, and the names of elementary data items being referenced by statements containing the CORRESPONDING clause.

STACK (RESET)

When set, the STACK compiler option causes relative stack addresses (address couples) for all program items to be included in the output listing.

STOP (AUTOBINDING Only)

STOP causes the BINDER to stop interpreting input statements and option cards, causing them to be flushed out.

SUPRS (RESET)

The FORTRAN compiler option SUPRS, when set, causes the suppression of warning messages in the output listing.

SYNTAX (RESET)

The NDL option SYNTAX, when set, prohibits creation of the SYSTEM/NIF, SYSTEM/REQUESTIMAGE, and DC/CODE files.

S360 (RESET)

S360 is a COBOL compiler option which is to be used to improve compatibility with the IBM-360 and IBM-570 systems. (See B6700 option for additional system option descriptions.)

TIME (RESET)

The TIME compiler option, when set with LIST reset, causes an output listing to be generated containing compilation "trailer" information (e.g., number of errors, number of segments, compilation time, etc.). Since this information is printed when LIST is set, the TIME option has effect only when LIST is reset.

TRACE (RESET)

The TRACE option, when set:

a. sets LIST, CODE, STACK, and DEBUG options when used with the FORTRAN compiler.

b. when used with the BINDER, causes the listing to contain a record of BINDER procedure entries and exists as well as the contents of certain BINDER arrays.

c. when used with the COBOL compiler, causes the contents of the first two words of that compiler's ACCUM array to be included in the output listing for each syntactic item scanned in the compiler'f first pass. This information is produced by the COBOL compiler's scanner, and this option is primarily intended for compiler development use only.

USASI (RESET)

USASI is a COBOL compiler option which is used for programs which conform to the COBOL standard as specified in the "USASI COBOL, X3-23, 1968" publication. (See B6700 option for additional system option descriptions.)

USE (AUTOBINDING Only)

USE provides to BINDER a technique for matching identifiers in a host with differing identifiers in a separate program unit.

VECTORMODE (RESET)

When set and only if the FORTRAN compiler is capable, this option provides the mode compiler with the vector mode capability.

VERSION vv.ccc (RESET)

When SET, the ALGOL and ESPOL VERSION compiler option allows the user to have the patch version specified on the printout. As used by the system software, the first v represents the mark number, the second v represents the level number, and the ccc represents the cycle.

For a user to gain access to these numbers, two additional COMPILETIME options have been implemented in ALGOL and ESPOL:

a. COMPILETIME (20) yields the version in integer.

(1) COMPILETIME (20) DIV 10 yields the mark number (for system software).

(2) COMPILETIME (20) MOD 10 yields the level number (for system software).

b. COMPILETIME (21) yields the cycle in integer.

When compiling from patch with mark set, then:

a. If on the $# CARD the first string after the noise word is a string of three or fewer digits, then

(1) The ALGOL compiler will replace the three digit patch number with the string of 10 characters as follows: VERSION (2 characters), PERIOD CYCLE (3 characters), PERIOD CYCLE (3 characters), PERIOD PATCH (3 characters); i.e., VV.CCC.PPP.

(2) The ESPOL compiler will replace the string three digit patch number with the string of eight characters as follows: VERSION (2 characters), CYCLE (3 characters), PATCH (3 characters). When printed on a listing, the ESPOL compiler will insert periods between the VERSION and CYCLE, CYCLE and PATCH.

b. If on the $# CARD the first string after the noise word is a string other than three or fewer digits, then the eight-character string will be used and the VERSION information will be ignored.

When compiling with NEW SET and a $ VERSION card appears in the symbolic, then if patch deck contains a $ VERSION card, the new symbolic will be updated to the version and cycle on the last $ VERSION card in the patch deck if the sequence number is less than the one on the symbolic.

VOID (RESET)

When set, VOID causes all input from files CARD and TAPE (except compiler control cards) to be ignored by the compiler until VOID is reset or is popped into a reset state. The input ignored will not be listed or included in a new symbolic file even with LIST and NEW set. (See also VOIDT and OMIT.)

VOIDT (RESET)

When set, VOIDT causes only primary input to be compiled; all secondary input from file TAPE is ignored (except compiler control cards) until VOIDT is reset. This option is effective only when the option MERGE is set, since secondary input is ignored unless MERGE is set. The ignored input will not be listed or included in a new symbolic file even when LIST and NEW are set. (See also VOID and OMIT.)

WARN (RESET)

When set, the BINDER option WARN generates an output listing containing warning messages when LIST is reset. These messages are automatically provided on the output listing when LIST is set.

XDECS (RESET)

The XDECS compiler option, when set, causes the occurrences of program declarations to be recorded for cross-referencing purposes when XREF is set. This option is initially set when XREF is set and may be set, reset, popped, etc., as many times as desired. The XDECS option is used in conjunction with XREF to select the portions of source code to be examined for information concerning declaration locations. (See XREF.)

XREF (RESET)

The XREF compiler option, when set, causes (in the event of successful compilation) an index of all identifiers used in the compiled program to be written on the file LINE. This operation is accomplished by the automatic initiation of a separate program (called SYSTEM/XREFANALYZER) at the end of the compilation and giving to it a file containing the necessary information. These identifiers are arranged according to the EBCDIC collating sequence, and each identifier is followed by a list of the sequence numbers of the card images on which the identifier appears. The LIST option need not be set to generate an output listing of this cross-reference information. XREF should be set before any of the source input is processed, and once set, this option may not be reset. (See also XDECS and XREFS.)

XREFS (RESET)

The XREFS compiler option, when set, causes program identifier references to be noted for cross-referencing purposes when XREF is set. This option is initially set when XREF is set and may be set, reset, popped, etc., as many times as desired. The XREFS option is used in conjunction with XREF to select the portions of source code to be examined for information concerning the location of identifier references. Only references of identifiers which are declared in portions of the source file where XDECS is set will be cross-referenced. (See XREF and XDECS.)

$ (RESET)

When set, the $ compiler option causes images of all compiler control cards in the input file(s) to be included in the output listing.

PL/I COMPILER OPTIONS

The PL/I compiler options are classed as Boolean options, parameter options, and value options. A description is provided for each of these options.

Boolean Options

For each <Boolean option> there is within the PL/I compiler a 42-bit stack in which the current and previous settings of the option (i.e., on or off) are stored. The bit at the top of the stack represents the current state of the option (1 indicates the option is "on", and 0 indicates "off"). When an option is "SET", its stack is pushed down one bit and a 1 is put on top of the stack. "RESET"ing an option causes the corresponding option stack to be pushed down one bit and a 0 to be put on top of the stack. "POP" causes the option stack to be pushed up one bit, thus causing the current setting to be lost and the last previous setting to become the current setting. For the initial release, the default state of ERRORS, LIST1, MODEL11, SIXTY, and STMTNO is "SET", and all other Boolean options are reset by default. ERRLIST is set by default for CANDE jobs.

COMPILER CONTROL CARDS

ATTRIB

When set, ATTRIB will cause an output listing to be generated which contains each identifier declared in the program to be listed along with the explicitly declared attributes for each identifier and those attributes which are given to each identifier by default. ATTRIB may be set selectively for a given block rather than the entire program. This is done by setting ATTRIB before the block's BEGIN or PROCEDURE statement and then popping ATTRIB after the corresponding END statement.

CODE

When set, CODE causes an output listing to be generated which contains the object code generated by the compiler.

CONTROLS

When set, CONTROLS will cause compiler control cards which begin in column one to be included in the LISTI output, if such an output file is generated. Control cards which begin in a column other than one will always appear on LISTI output, as well as any new source language files (NEWTAPE1 or NEWTAPE2) if such files are generated.

ERRLIST

The ERRLIST option is intended for use with compilations initiated through CANDE. When this option is set (as it is by default for CANDE jobs) card images and error messages corresponding to syntax errors in the program are placed in a file assigned to the remote terminal being used. (See "WARN".)

ERRORS

Setting the option ERRORS will cause an output listing of errors detected during compilation to be generated. For each statement which contains an error, the statement is first listed as it appeared on the intput card image and is preceded by its statement number. Then, if there was any preprocessor expansion necessary to the statement, the expanded statement is listed, otherwise the statement (unexpanded) is listed. This second listing of the statement is then marked where the error occurred and an explanation of the error is printed below the statement.

EXTERN

Setting EXTERN will produce a listing similar to the listing produced by setting ATTRIB except that only those identifiers which are declared with the attribute external will be included in the listing.

FLEVEL

The FLEVEL option specifies that if during compilation the compiler encounters a PL/I construct which the IBM F-LEVEL PL/I compiler would handle differently from the B 6700 PL/I compiler, then it should be handled according to F-LEVEL.

LIST

Setting and resetting the LIST option sets and resets, respectively both LIST1 and LIST2.

LIST1

LIST1, when set, causes a card image listing consisting of the merged card and tape input files to be generated.

LIST2

LIST2, when set, causes a listing consisting of the merged card and tape input files plus all expansions of preprocessor (i.e., compile-time) statements to be generated.

MERGE

When set, MERGE causes primary input (file CARD) to be merged with secondary input (file TAPE). If matching sequence numbers occur in the two input files, the primary input overrides the secondary input.

MODEL11

MODEL11, when set, causes code to be generated suitable for execution on a Model 11 processor. When MODEL11 is reset, the generated code is suitable for execution by a Model 1 processor.

MULTIPLE

When MULTIPLE is set, causes the last name of the program file name to be replaced by the name of the main entry point of the procedure. If SEPARATE is not set, the name of the first procedure will have the last name of the program file name replaced by the name of the main entry point of the procedure.

NEW or NEW1

When set, NEW (or NEW1) causes a new symbolic file generated by the compiler to be placed on the output file NEWTAPE1. This new symbolic file consists of the merged card and tape input files. This new file may be used as a secondary input file by a later compilation.

COMPILER CONTROL CARDS

NEW2

When set, NEW2 causes a new symbolic file generated by the compiler to be placed on the output file NEWTAPE2. This new symbolic file consists of the merged card and tape input files plus all expansions of pre-processor (i.e., compile-time) statements. This new file may be used as a secondary input file by a later compilation.

NEW1SEQERR

When set, NEW1SEQERR will cause records written to the file NEWTAPE1 to be checked for sequence errors. As each sequence error is discovered, a warning message is placed on the output listing and also displayed on the SPO. If such errors occur, then at the end of compilation the file NEWTAPE1 will not be locked and thus will not be available to the user.

NEW2SEQERR

The option NEW2SEQERR performs the same function for the NEWTAPE2 file as the NEW1SEQERR option performs for the NEWTAPE1 file.

NOBINDINFO

When NOBINDINFO is set, information for the BINDER is not placed into the code file. NOBINDINFO is reset by default.

SEG or SEGS

When set, SEG (or SEGS) will cause beginning and ending segmentation messages to be included in the output listing.

SINGLE

Setting and resetting the SINGLE option sets and resets, respectively, both SINGLE1 and SINGLE2.

SINGLE1

When set along with the option LIST1, SINGLE1 will cause the listing produced by LIST1 to be single spaced. If SINGLE1 is not set, the listing will be double spaced.

SINGLE2

When set along with the option LIST2, SINGLE2 will cause the listing produced by LIST2 to be single spaced. If SINGLE2 is not set, the listing will be double spaced.

SIXTY

When set, SIXTY specifies that the PL/I program will use the 60-character set, otherwise the 48-character set will be used. SIXTY is set by default.

STACK

When set, STACK causes relative addresses (address couples) for all program items to be included in the output listing.

STMTNO

When set, the STMTNO option causes the sequence number of the line causing fault termination of program execution to be included in the generated error message. Setting STMTNO produces a larger code file than would otherwise be created by the compiler.

TIME

When set, the TIME option causes trailer information (i.e., number of errors, compilation times, etc.) to be placed in the compiler output file. This option has effect only when LIST is reset as this trailer information appears automatically when LIST is set.

TRACE

Setting and resetting the TRACE option sets and resets, respectively, both TRACEENTRYS and TRACELABELS.

TRACEENTRYS

When set, the TRACEENTRYS option causes execution-time monitoring of procedure entries to be performed. Each procedure entry is indicated by a message in the SYSPRINT file.

TRACELABELS

When set, the TRACELABELS option causes execution-time monitoring of program flow through labels on executable statements. Each such label encounter is indicated by a message in the SYSPRINT file.

VOID

When set, VOID will cause all input from files CARD and TAPE (except compiler control cards from the file CARD) to be ignored until VOID is RESET or POP'ed into a reset state. The ignored input will not be listed or included in a new symbolic file.

VOIDT

When set, VOIDT causes only primary input to be compiled; all secondary input is ignored until VOIDT is reset. The ignored input will not be listed or included in a new symbolic file.

Example:

```
"SET NEW1"
"SET CONTROLS LIST1 RESET LIST2"
```

COMPILER CONTROL CARDS

Parameter Options

For each <parameter> of a <parameter option> there is within the PL/I compiler a 42-word stack in which the history of the <parameters>s value are stored. Setting (or resetting) a <parameter option> causes each <parameter> stack associated with that <parameter option> to be pushed down one word and the new value of that parameter to be placed on the top of that stack. If one or more of the <parameter>s on the control card is <empty>, then the last previous setting is used and placed in the appropriate stack. Popping a <parameter option> causes all <parameter>s associated with the option to be discarded and to be replaced by the prior value from the appropriate stack.

The <parameter option>s CARDCOL, TAPECOL, INCLCOL, COL1 and COL2 are used to define how the columns of input card images and the columns of output images for new symbolic source files will be used. By using these options the user can specify which columns are to be used for program text, which columns are to be used for a sequence number prefix and which columns are to be used for the sequence number. However, the following restrictions are placed upon the user:

1. The program text must occupy at least one column but no more than 80 columns.

2. The sequence number prefix may not occupy more than six columns.

3. The sequence number may not occupy more than 12 columns.

4. The combined length of the columns for program text, sequence number prefix and sequence number must be at least one column but no more than 80 columns.

The default column settings are:

1. Columns 1-72 for program text.

2. Columns 73-80 for sequence number.

3. No columns for sequence number prefix.

COL1 causes the specified <parameter>s to be changed for the symbolic output files LIST1 and NEWTAPE1 when such files are generated.

COL2 causes the specified <parameters>s to be changed for the symbolic output files LIST2 and NEWTAPE2 when such files are generated.

CARDCOL causes the specified parameters to be changed for reading input from cards.

TAPECOL causes the specified parameters to be changed for reading input from file TAPE.

INCLCOL causes the specified parameters to be changed for reading from pre-processor (i.e., compile-time) INCLUDE-files.

Example:

"SET CARDCOL (TEXT = 1 FOR 72 SEQ = 73 FOR 2, 6)"
This control card specifies that for input from cards the program text will be in columns 1-72, sequence numbers will start in column 73 with columns 73 and 74 being used for a sequence number prefix and the next six columns (75-80) for the actual sequence number.

Since the only difference between these column specifications and the default value is that these allow for a sequence number prefix in columns 73 and 74, the above control card could have been written.

```
"SET CARDCOL (SEQ = 73 FOR 2, 6)"
             or
"SET CARDCOL (SEQ =       FOR 2, 6)"
```

In these cases the compiler would have used the default <program string parameter> of TEXT = 1 FOR 72 and the default sequence <starting column> value of 73.

The <parameter option>s SEQ1, SEQ2, and SEQ are used to define how the source language symbolic files are to be sequenced. Associated with these options are <sequence base>, <sequence increment> and <prefix>. The <prefix> is a string of six or less characters which precedes the sequence number and is treated as a comment for sequence checking and patching purposes. The default for <prefix> is the null character string. The <sequence base> is an <unsigned integer> which specifies the starting sequence number for resequencing. The default value of the <sequence base> is zero. The <sequence increment> specifies the amount by which each new sequence number will exceed the previous sequence number. The default <sequence increment> is zero.

SEQ1 causes <sequence base>, <sequence increment> and <prefix> to change as specified for the symbolic files LIST1 and NEWTAPE1 when such files are being generated.

SEQ2 causes <sequence base>, <sequence increment> and <prefix> to change as specified for the symbolic files LIST2 and NEWTAPE2 when such files are being generated.

SEQ specifies both SEQ1 and SEQ2.

Example:

"SET SEQ1 ('AB' 1000 + 100)"
This control card specifies that the sequence number field for files LIST1 and NEWTAPE will consist of a two letter <prefix> and that the sequence number will start at 1000 and each following sequence number will exceed the previous number by 100.

COMPILER CONTROL CARDS

Value Options

With each <value option> there is associated a single value which, when set, remains unchanged throughout the entire compilation. Thus, resetting or popping one of these options once it has been set will have no effect.

TITLE

This option is set by specifying a character string containing less than 60 characters. This character string will then be used as the title for the program and will appear in the heading for each listing specified (e.g., LIST1, ATTRIB, ERRORS) as well as on the dateline which appears at the top of each page of listing. If this option is not specified, the program's object code file name (i.e., the name given on the COMPILE card of the program) will be used. TITLE must be set on an input card appearing before any input card causing output on LIST1 before that title will appear in the heading of the LIST1 file.

OPTIMIZE or OPT

This option is set by specifying an <unsigned integer> which indicates the degree of optimization which the user wants the compiler to use. The lowest and default optimization value is 1 and the maximum optimization value is 3.

WARN

This option is set by specifying an <unsigned integer> which indicates the lowest level of error message to be included in the ERROR LISTING file and in the file generated by the ERRLIST option. Syntax errors are assigned level numbers according to their severity from 0 to 9, with 9 being the most severe. The default is WARN=0.

EXECUTION or EXEC

This option is set by specifying an <unsigned integer> which indicates the lowest level of error message which will cause the compilation to be aborted. When an error occurs at this level or higher (as described in the discussion of "WARN") the code file is not locked and compilation is terminated with a "SNTX" message. The default is EXEC=7.

Example:

The following compiler cards and B 6700 PL/I compiler control cards illustrate a deck which might be used to create a new symbolic file called NEW/PROG. The <!> represents an invalid punch in column one. The numbers to the left are not part of the cards but appear only as reference numbers for the explanation which follows the example.

```
          <1> COMPILE PROG PL/I LIBRARY
(1)       <1> PL FILE NEWTAPE1 (TITLE = NEW/PROG)
          <1> EBCDIC
(2)           "SET TITLE (PL/I PROGRAM)"
(3)           "SET NEW1 NEW1SEQERR LIST1"
(4)           "SET COL1 (TEXT = 1 FOR 70 SEQ = 71 for 3, 7)"
(5)           "SET SEQ1 ('XYZ' 1000 + 1000)"
                .
                .
                .
          <1> END
```

Card (1) specifies the title for the new symbolic file which is to be created.

Card (2) specifies that the title which will appear on all the listings which are generated will be PL/I PROGRAM.

On card (3) NEW1 specifies that a new symbolic file consisting of the card images will be created on file NEWTAPE1. NEW1SEQERR specifies that as records are written to NEWTAPE1 they will be checked for sequence errors. LIST1 specifies that as the card images are written to NEWTAPE1, they will also be written to an output listing.

Card (4) specifies how the card image columns for NEWTAPE1 and LIST1 will be used.

```
Columns 1-70 program text.
Columns 71-73 sequence prefix.
Columns 74-80 sequence number.
```

Card (5) specifies the prefix for each sequence number to be XYZ and the sequence base and sequence increment to both be 1000.

If this newly created symbolic file then needed to be patched later, the following deck could be used:

```
          <1> COMPILE PROG PL/I LIBRARY
(6)       <1> PL FILE TAPE (TITLE= NEW/PROG)
(7)       <1> PL FILE NEWTAPE1 (TITLE = NEW/PROG)
          <1> EBCDIC
              "SET TITLE (PL/I PROGRAM)"
(8)           "SET MERGE NEW1 NEW1SEQERR"
(9)           "SET COL1 (TEXT = 1 FOR 70 SEQ = 71 FOR 3, 7)"
(10)          "SET TAPECOL (TEXT = 1 FOR 70 SEQ = 71 FOR 3, 7)"
(11)          "SET CARDCOL (TEXT = 1 FOR 70 SEQ + 71 FOR 3, 7)"
                .
                .          (patch deck)
                .
          <1> END
```

Card (6) specifies that the secondary input file TAPE will be the disk file NEW/PROG.

COMPILER CONTROL CARDS

Card (7) specifies that the merged file that will be created will be placed on the NEWTAPE1 file and given the name NEW/PROG.

MERGE on card (8) specifies that the secondary input file TAPE is to be merged with the primary input file CARD.

Card (9) specifies the column arrangement for the output on the new symbolic NEWTAPE1.

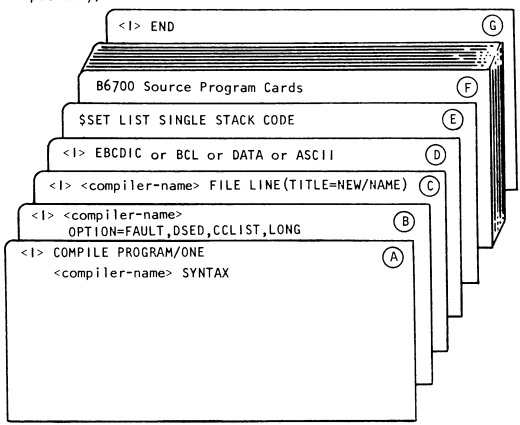
Cards (10) and (11) specify the column arrangements for the secondary and primary inputs respectively. This means that the TAPE input and the card patch deck must both be in the same card format. That is: the first 70 columns contain program text, the next three columns contain a sequence prefix and the last seven columns contain a sequence number.

SAMPLE CARD INPUT DECKS

For program compilation the only required cards are a COMPILE card (specifying the language in which the program is written), a DATA, BCL, or EBCDIC card, source program cards, and an END card, in that order. For program execution (ESPOL excluded) the only required cards are the RUN (or EXECUTE) card, DATA, EBCDIC, BCL, or ASCII card if data are to be supplied (dependent upon the format of the data cards), and an END card. Typical decks for program compilation and execution, together with explanations of the cards used, are presented in the following paragraphs. When <!> appears in column 1 of an MCP control card, the <!> represents an invalid character punched in that column. (One combination of punches that will produce such an invalid character is 1-2-3.)

COMPILE FOR SYNTAX

(Card Input Only)



(A) causes syntax check of PROGRAM/ONE written in one of the B 6700 languages. <compiler-name> must be supplanted by one of the following names according to the language in which the source program is written: ALGOL, DCALGOL, XALGOL, BASIC, COBOL, ESPOL, or FORTRAN.

(B) causes memory dump on abnormal termination (FAULT and DSED options), listing of control cards (CCLIST option), and suppression of segmentation for large arrays (LONG option). <compiler-name> must be replaced by any standard compiler name or by COMPILER.

(C) assigns the title NEW/NAME to the compiler output file LINE. <compiler-name> must be replaced by any standard compiler name or by COMPILER.

(D) indicates that the source program cards which follow this card are punched in EBCDIC, ASCII, or BCL format. (Note that DATA and EBCDIC are synonymous.)

(E) is a compiler control card which causes output listing of source cards (LIST option), single-spaced (SINGLE option), containing listing of stack assignments (STACK option), and program code (CODE option).

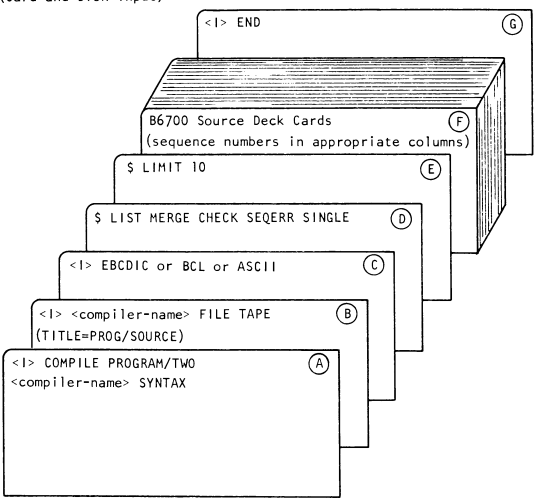
(F) denotes source cards punched in EBCDIC or BCL (as indicated by card (D)) containing a program written in the language indicated on card (A).

(G) indicates an end of the cards to be compiled.

SAMPLE DECKS

COMPILE FOR SYNTAX

(Card and Disk Input)



(A) causes syntax check of PROGRAM/TWO written in one of the B 6700 languages. <compiler-name> must be supplanted by one of the following names according to the language in which the source program is written: ALGOL, DCALGOL, XALGOL, BASIC, COBOL, ESPOL, or FORTRAN.

(B) declares that the code to be patched is contained in the disk file titled PROG/SOURCE. <compiler-name> must be replaced by the same name used on card (A) or by COMPILER.

(C) indicates that the source patch cards and compiler control card which follow this card are punched in ASCII, EBCDIC, or BCL format.

(D) is a compiler control card which causes single-spaced listing of PROGRAM/TWO source file and patch data (LIST and SINGLE options), the compilation of both source file data and patches (MERGE option), sequence checking to be performed (CHECK option), with sequence errors causing the code file to not be locked (SEQERR option).

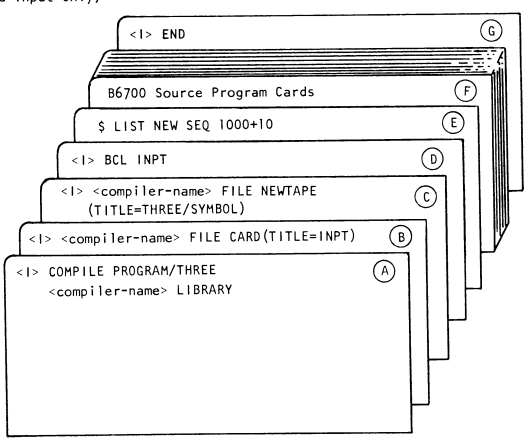
(E) signals the compiler that compilation is to be terminated when 10 errors are detected in the input.

(F) denotes cards bearing patches to the file PROG/SOURCE written in the language indicated on card (A). The sequence numbers contained on these cards (starting in column 1 for BASIC, contained in columns 1 through 6 for COBOL, and contained in columns 73 through 80 for the other languages) indicate the point of insertion for each patch card within the source file data.

(G) indicates an end of the compilation deck.

COMPILE FOR LIBRARY

(Card Input Only)



(A) causes program PROGRAM/THREE written in one of the B 6700 languages to be compiled and the resultant object code to be saved. <compiler-name> must be supplanted by one of the following names according to the language in which the source program is written: ALGOL, DCALGOL, XALGOL, BASIC, COBOL, ESPOL, or FORTRAN.

(B) declares that the input source code is contained in the card file titled INPT. <compiler-name> must be replaced by the same name used on card (A) or by COMPILER in order to indicate that the file involved is a compiler file and not a program file.

(C) causes the compiler file NEWTAPE to be titled THREE/SYMBOL. This file will contain the symbolic data written on the source program cards, and <compiler-name> must be replaced in the same manner as on card (B).

(D) indicates that the following source program cards constitute the file titled INPT and that these cards and the compiler control card are punched in BCL format.

SAMPLE DECKS

(E) is a compiler control card which causes listing of PROGRAM/THREE source cards (LIST option), creation of the symbolic file referred to on card (C) (NEW option), with card images sequenced from 1000 in increments of 10 (SEQ option with a parameter of 1000+10).

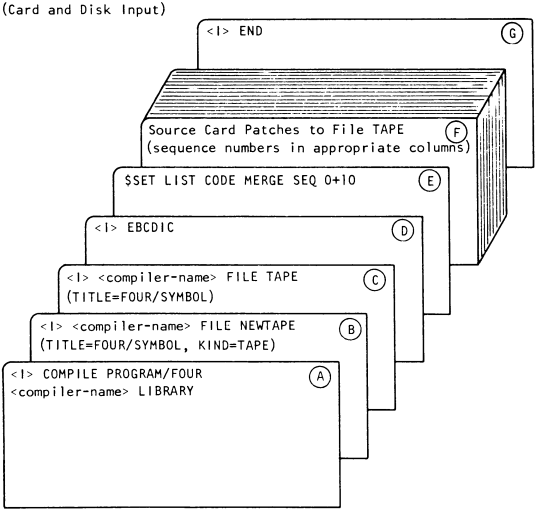
(F) denotes source program cards to be compiled, punched in BCL (as indicated by card (D)) and containing a program written in the language indicated on card (A).

(G) indicates an end of the cards to be compiled.

Note: The creation of the symbolic file (THREE/SYMBOL) is not a necessary operation, but is caused by card (C) and the NEW option on card (E). Similarly, the labeling of the card file INPT is not necessary, but is caused by card (B) and the INPT label on card (D)

COMPILE FOR LIBRARY

(Card and Disk Input)



(A) causes PROGRAM/FOUR written in one of the B 6700 languages to be compiled and the resultant object code to be saved. <compiler-name> must be supplanted by one of the following names according to the language in which the source input is written: ALGOL, DCALGOL, XALGOL, BASIC, COBOL, ESPOL, or FORTRAN.

(B) causes the compiler file NEWTAPE to be titled FOUR/SYMBOL and assigned to magnetic tape. This file will contain the symbolic data which is the source input for this compilation, and <compiler-name> must be replaced by the same name used on card (A) or by COMPILER in order to indicate that the file involved is a compiler file and not a program file.

(C) declares that the input source code to be patched is contained in the disk file titled FOUR/SYMBOL. <compiler-name> must be replaced in the same manner as on card (B).

(D) indicates that the source patch cards and compiler control card which follow this card are punched in EBCDIC format.

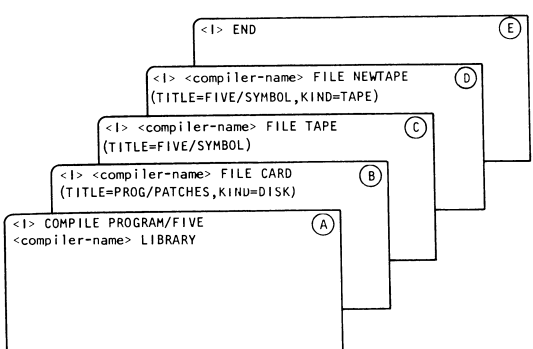
(E) is a compiler control card which causes listing of PROGRAM/FOUR source input (LIST option) containing a listing of the program object code (CODE option), the compilation of both source file data and patches (MERGE option), and the creation of the symbolic file referred to on card (B) (NEW option), with new source file card images sequenced from 0 in increments of 10 (SEQ option with a parameter of 0+10).

(F) denotes cards bearing patches to the input file FOUR/SYMBOL written in the language indicated on card (A). The sequence numbers contained on these cards (starting in column 1 for BASIC, contained in columns 1 through 6 for COBOL, and contained in columns 73 through 80 for the other languages) indicate the point of insertion for each patch card within the source file data.

(G) indicates an end of the compilation deck.

COMPILE FOR LIBRARY

(Disk Input, Disk File Patches)



SAMPLE DECKS

(A) causes PROGRAM/FIVE written in one of the appropriate B 6700 languages to be compiled and the resultant object code to be saved. <compiler-name> must be supplanted by one of the following names according to the language in which the source input is written: ALGOL, DCALGOL, XALGOL, BASIC, COBOL, ESPOL, or FORTRAN.

(B) declares that patches are to be input from the serial disk file titled PROG/PATCHES. <compiler-name> must be replaced by the same name used on card (A) or by COMPILER in order to designate it as a compiler input file.

(C) declares that the source input to be patched is contained on the serial disk file titled FIVE/SYMBOL. <compiler-name> must be replaced in the same manner as on the previous card.

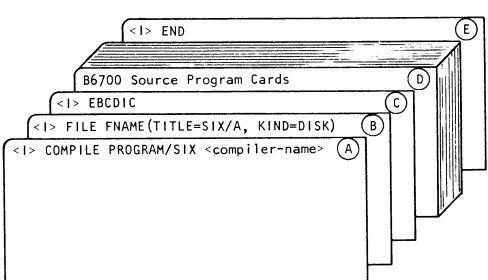
(D) declares that magnetic tape file titled FIVE/SYMBOL is to contain the symbolic data consisting of the source input after patching. <compiler-name> must be replaced in the same manner as on card (B).

(E) indicates an end of the compilation deck.

Note: The first card image in the patch file (PROG/PATCHES) must be a compiler control card. The compiler option NEW must be set if the new source data file is to be saved on the FIVE/SYMBOL tape. The option MERGE must be set to permit file TAPE to be read.

COMPILE AND GO

(Card Input Only, No Execution Data)



(A) causes the program named PROGRAM/SIX written in one of the B 6700 languages to be compiled and the resultant object code to be executed. <compiler-name> must be replaced by one of the following names according to the language in which the source program is written: ALGOL, DCALGOL, XALGOL, BASIC, COBOL, or FORTRAN.

(B) assigns the title SIX/A to the program file named FNAME and declares it as a serial disk file.

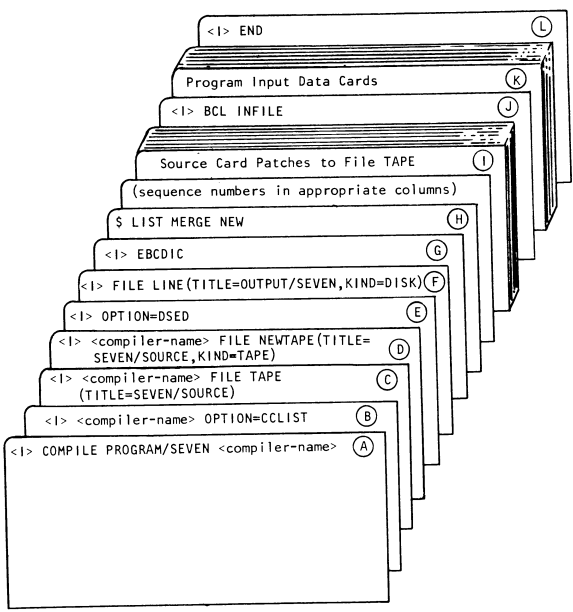
Ⓒ indicates that the source program cards (and any compiler control cards) which follow this card are punched in EBCDIC format.

Ⓓ denotes source program cards punched in EBCDIC format containing a program written is the language indicated on card Ⓐ.

Ⓔ indicates an end of the cards to be compiled and executed.

COMPILE AND GO

(Card and Disk File Input With Input Data on Cards)



Ⓐ causes the program named PROGRAM/SEVEN written in one of the appropriate B 6700 languages to be compiled and the resultant object code to be executed. <compiler-name> must be replaced by one of the following names according to the language in which the source program is written: ALGOL, DCALGOL, XALGOL, BASIC, COBOL, or FORTRAN.

SAMPLE DECKS

(B) causes the compilation option CCLIST to be set, resulting in the listing of control cards. <compiler-name> must be replaced by the name used on the preceding card or by COMPILER.

(C) causes the compiler input file TAPE to be equated to the serial disk file titled SEVEN/SOURCE. <compiler-name> must be replaced by the compiler name used on card (A) or by COMPILER. (The indicated file contains source data to be patched before compilation by the source card patches.)

(D) causes the compiler file NEWTAPE to be equated to the magnetic tape file titled SEVEN/SOURCE. <compiler-name> must be replaced as on the preceding card. (The indicated file will contain the symbolic data which was used as source input by the compiler.)

(E) causes the execution option DSED to be set, resulting in a program dump if the job is operator DSed.

(F) declares the program file named LINE a serial disk file titled OUTPUT/SEVEN.

(G) indicates that the compiler control card and source program patch cards which follow this card are punched in EBCDIC format.

(H) is a compiler control card which causes listing of PROGRAM/SEVEN source input (LIST option), the compilation of both source file (TAPE) data and patches (MERGE option), and the creation of the symbolic file referred to on card (D) (NEW option).

(I) denotes cards bearing patches to the input file SEVEN/SOURCE written in the language indicated on card (A). The sequence numbers appearing on these cards (placed according to the language used) indicate the point of insertion for each patch card within the source file data.

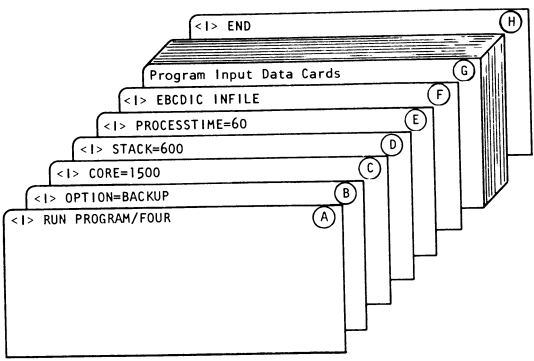
(J) indicates that the following cards are punched in BCL format and comprise records for the program input card file titled INFILE. This card terminates the compilation portion of the deck.

(K) denotes cards punched in BCL format and bearing input data for the execution of PROGRAM/SEVEN.

(L) indicates an end of the execution portion of the deck by terminating the input data cards.

PROGRAM EXECUTION

(Input Data on Cards)



(A) causes the previously-compiled-to-library program, PROGRAM/FOUR, to be executed.

(B) causes the execution option BACKUP to be set, resulting in the output of print files to backup disk.

(C) declares that the program requires a maximum of 1500 words of core memory.

(D) declares that the program requires a maximum of 600 words of working stack in main memory.

(E) specifies that the maximum processing time allowed for this job is 60 seconds.

(F) indicates that the following cards are punched in EBCDIC format and comprise records for the program input card file titled INFILE.

(G) denotes cards punched in EBCDIC format and bearing input data for the execution of PROGRAM/FOUR.

(H) indicates an end of the card deck.

SAMPLE DECKS

PROGRAM EXECUTION

(No Input Data)

```
<!> RUN PROGRAM/EIGHT; END                    (A)
```

(A) causes the object code file of the previously-compiled program, PROGRAM/EIGHT to be executed. The program in question requires no input data.

PROGRAM BINDING

Although this section is primarily concerned with B 6700 program compilation and execution decks, an example of a B 6700 BINDER deck is provided here as a reference tool.

```
<!> END                                                    (G)
BIND = FROM CODEFILE/=;                                    (F)
BIND SUBA FROM VERSION/SUBA;                               (E)
$ LIST CODE                                                (D)
<!> EBCDIC                                                 (C)
<!> BINDER FILE HOST (TITLE = HOST/FILE)                   (B)
<!> BIND BOUND/PROGRAM1 WITH BINDER FOR LIBRARY            (A)
```

(A) causes the binding of a program named BOUND/PROGRAM1 to occur, with the resultant object code being placed in a library file with the TITLE of BOUND/PROGRAM1.

(B) equates the internal BINDER file HOST with the library (serial disk) file with the TITLE of HOST/FILE.

PROGRAM BINDING

(C) indicates that the following cards in this deck are punched in EBCDIC format.

(D) is a BINDER option card which specifies that an output listing be generated (LIST option) containing a listing of the BOUND/PROGRAM1 object code (CODE option).

(E) is a BINDER source card declaring that the separately-compiled code contained in the library file VERSION/SUBA be bound into the host in place of the program unit named SUBA.

(F) is a BINDER source card declaring that the separately-compiled code contained in library files with CODEFILE as the first (left-most) identifier of their TITLE's be bound into the host in place of program units which have names identical to the rightmost identifiers of the TITLE's of these CODEFILE files.

(G) indicates an end of the BINDER deck.

SECTION 3
MISCELLANEOUS SOFTWARE INFORMATION

SECTION 3 — CONTENTS

SECTION 3
MISCELLANEOUS SOFTWARE INFORMATION

FILE IDENTIFICATION AND FILE STRUCTURE

Format of File Names

File names are composed of a series of identifiers (file identifiers, volume identifiers, and file directory identifiers) separated by slashes. An identifier is delimited by either a blank or a slash and may be of any length, although only the first 17 characters of an identifier are used if the identifier is more than 17 characters long. As many as 14 identifiers may be used in constructing a file name.

Examples of file names follow:

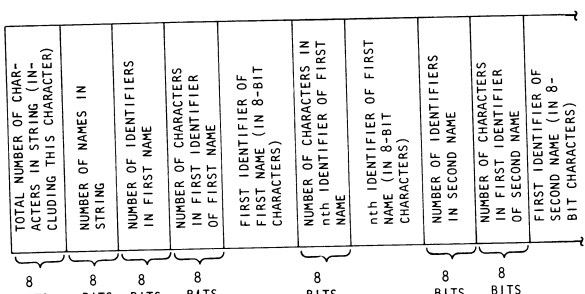
```
A
B/C
D/E/F
G/H/I/J
```

In the preceding examples, A, C, F, and J are file identifiers; B, E, and I are volume identifiers; and D, H, and G are file directory identifiers.

Note that two file names such as A/B and A/B/C cannot both be used, since A/B would then denote both a file and a file directory.

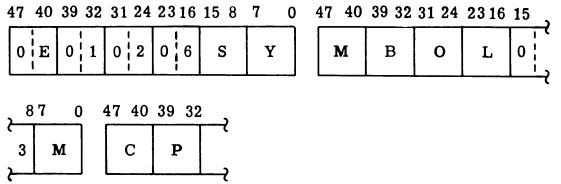
Standardform Representation of File Names

Standardform is a shorthand, convenient form for representing and storing a string of file names. The general form of a string of Standardform file names follows.



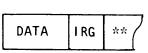
FILE IDENTIFICATION AND STRUCTURE

The Standardform representation of the file name SYMBOL/MCP is, for example, stored in 8-bit characters as follows.

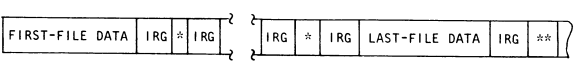


Format of Unlabeled Tapes

SINGLE FILE REEL



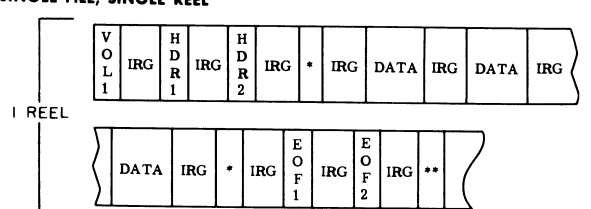
MULTIFILE REEL



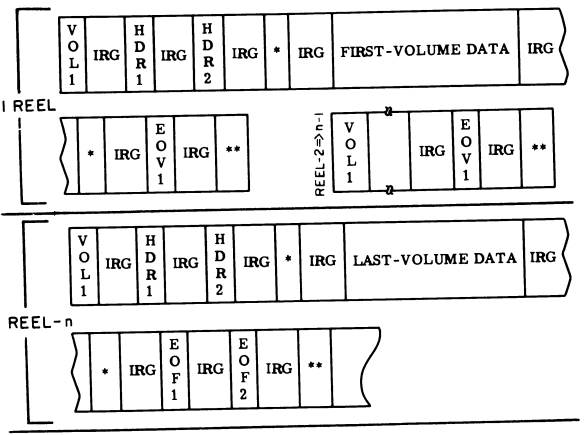
Note: Symbol * denotes a tape mark. A double tape mark consists of two tape marks separated by an interrecord gap.

Format of Labeled Tapes

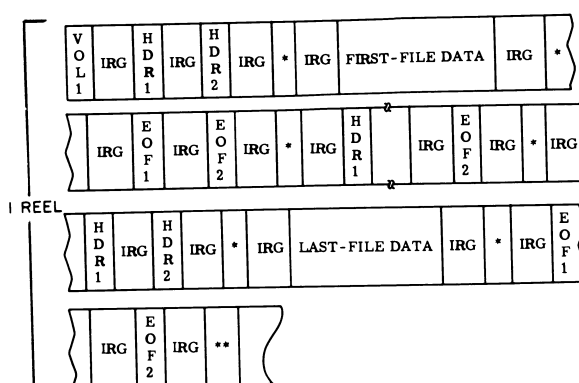
SINGLE FILE, SINGLE REEL



MULTIREEL FILE

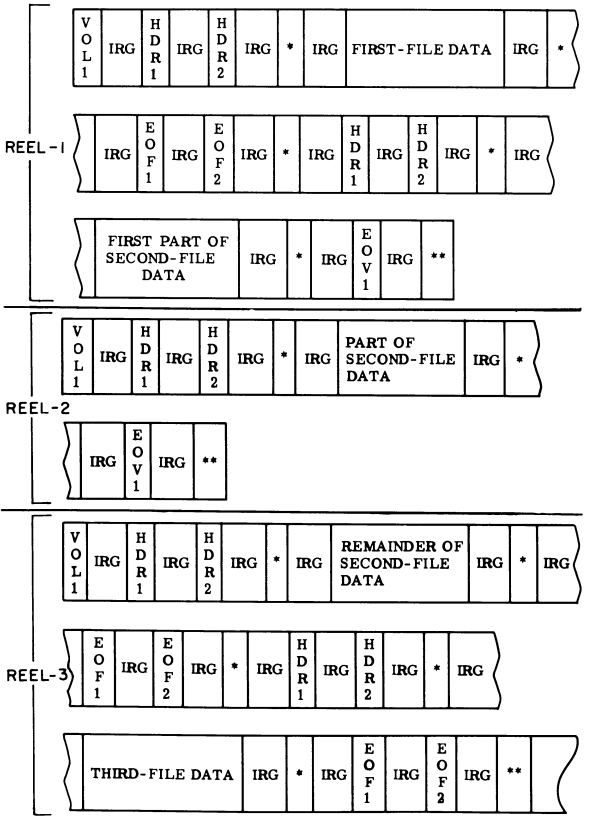


MULTIFILE REEL



FILE IDENTIFICATION AND STRUCTURE

MULTIFILE, MULTIREEL

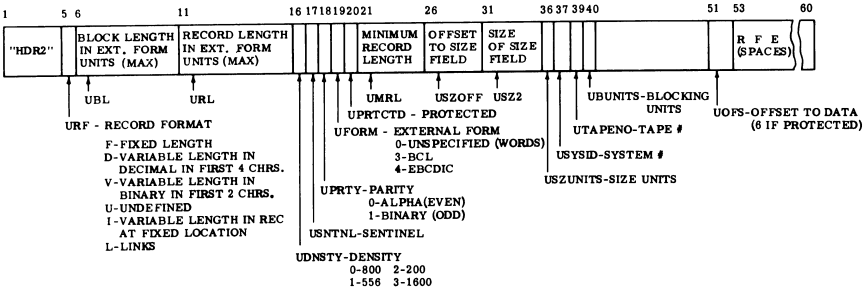


NOTES: 1. Symbol * denotes a tape mark.
2. User's header labels may appear immediately after HDR2, and user's trailer labels may appear after either EOF2 or EOV1.

VOLUME HEADER

FIRST FILE HEADER

SECOND FILE HEADER



FIRST END-OF-FILE LABEL

Same as first file header, except for first four characters: "EOF1".

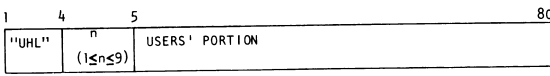
SECOND END-OF-FILE LABEL

Same as second file header, except for first four characters: "EOF2".

END-OF-VOLUME LABEL

Same as first end-of-file label, except for first four characters: "EOV1".

USERS' HEADER LABEL



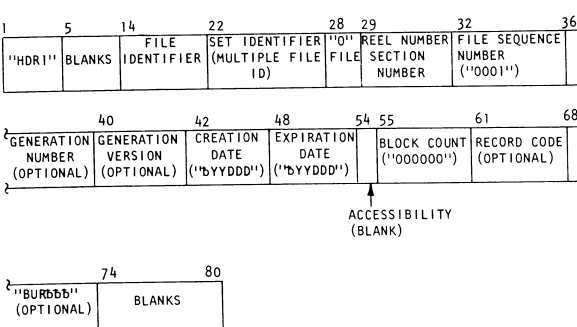
USERS' TRAILER LABEL

Same as users' header label, except for first three characters: "UTL".

B 3500 USASI Tape Label

VOLUME HEADER

FIRST FILE HEADER



FILE IDENTIFICATION AND STRUCTURE

B 5500 Tape Label

Word	Character (Word)	Character (Record)	Field Description
1	1-8	1-8	Must contain "bLABELbb".
2	1	9	Must be 0.
2	2-8	10-16	Multifile id.
3	1	17	Must be 0.
3	2-8	18-24	File id.
4	1-3	25-27	Reel-number (within file).
4	4-8	28-32	Date-written (creation date).
5	1-2	33-34	Cycle-number (to distinguish between identical runs on the same day).
5	3-7	35-39	Purge-date (date this file can be destroyed).
5	8	40	Sentinel (1 = end-of-reel, 0 = end-of-file).
6	1-5	41-45	Block-count.
6-7	6-8/1-4	46-52	Record count.
7	5	53	Memory-dump-key (1 = memory dump follows label).
7-8	6-8/1-2	54-58	Physical tape number.

The remainder of the information contained in the label record varies for ALGOL and COBOL files as follows:

ALGOL FILES

Word	Character (Word)	Character (Record)	Field Description
8	3	59	Blocking indicator (3 = blocked, 0 = not blocked).
8	4-8	60-64	Buffer size (number of words).
9	1-5	65-69	Maximum record size (number of words).
9	6-8	70-72	Zeros.

COBOL FILES

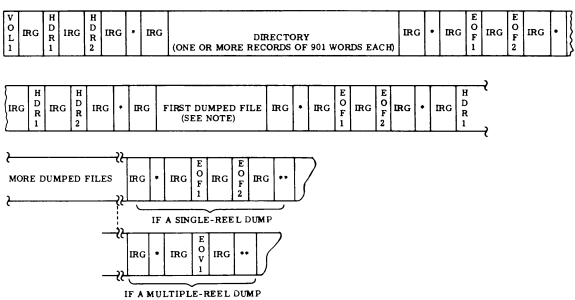
Word	Character (Word)	Character (Record)	Field Description
8	3-8	59-64	Reserved for file-control-routine -- not currently being used.
9-?	1-?	65-??	Users' portion (may be of any format desired by user and may be up to 8,120 characters in length for tape files, up to 16 characters in length for card files, and up to 56 characters in length for printer files.

B 6700/B 7700 Library Tapes

Library Tape Format:

Basically, a B 6700/B 7700 library tape is a USASI multifile volume that contains n+1 files: a tape directory and n files that have been copied to tape from disk. If the tape name is MSA, the files are named MSA/FILE000, MSA/FILE001, etc.; MSA/FILE000 is the name of the tape directory.

The general layout of a library tape is as follows:



Note: The first record of a dumped file is the disk header followed by the standard form of the file title. This is followed by the actual file data in 901-word records. The last of these records may contain fewer than 901 words; there is no padding of short records.

FILE IDENTIFICATION AND STRUCTURE

Library Tape Directory:

The directory consists of one or more records, each 901 words long. The first word in all directory records is a transaction number, that is, the number of the data record written out. (This word is not included on label records but the transaction count is increased for them.) There then follows in each directory record one word of link information followed by up to 899 words of file names in Standardform. (Refer to Standardform.) All records are completely packed; a file name may be split across directory records. The link word is used only for multiple-reel library dumps and is described in the next paragraph. The end of the directory is flagged by a name character count field of 0.

Multiple-Reel Dumps and Link Words:

In order to facilitate reloading of specified files, each new reel of a multiple-reel dump contains a partial copy of the tape directory; this copy contains the names of those files not dumped on the preceding reels. Thus, to load a file dumped to reel 3, for example, the directory of only the third reel need be examined to find the desired file.

The link word consists of the following three fields.

CHARCNT	47:16
SKIP	25:10
MOREINFO	12:13

CHARCNT and SKIP are needed because the directories of the succeeding reels are simply copied from the pertinent records of the master directory on the first reel. Thus, the first directory record on any but the first reel will generally begin with some names that do not pertain to the reel. CHARCNT tells how many characters must be jumped over to get to the first complete file name. SKIP tells how many complete file names must be skipped over in order to get to the name of the first complete file dumped on a reel.

MOREINFO contains the relative number of the file the name of which immediately precedes the first complete Standardform name in a directory record. This field is used in determining which directory records must be rewritten at reel-switching time.

The file names are described as follows (standardform):

CHAR. 1 - Total number of characters in the whole string (self-inclusive)
CHAR. 2 - Number of identifiers
CHAR. 3 - Number of names in first identifier
CHAR. 4... - File's names, each preceded by one character giving the length of that name (not self-inclusive)

For example: "SYMBOL/MCP"
becomes: 48 "0E010206" 8 "SYMBOL" 48 "03" 8 "MCP"
where "48" means to construct the following string from 4-bit input but treat the results as 8-bit.

Example of a Multiple-Reel Dump:

Suppose that

a. N files were dumped.

b. Two reels were used.

c. The file K-1 was being written at the time of reel change.

d. Two directory records were written on the first reel.

e. The name of file K is the third complete name in the second directory, and there are six characters (the end of a previous Standardform name) before the first complete name.

The formats of the two reels are as follows.

Reel 1:

first directory record

0	0	0	NAME OF FILE 1 THRU FIRST PART OF NAME OF FILE K-3

second directory record

6	0	K-3	LAST PART OF NAME OF FILE K-3, NAME OF FILE K-2, NAME OF FILE K-1, NAME OF FILE K THRU NAME OF FILE N

CHARCNT SKIP MOREINFO

file 1
.
.
.
file K-1 (first part)

Reel 2:

file K-1 (last part)

second directory record

6	2	K-3	LAST PART OF NAME OF FILE K-3, NAME OF FILE K-2, NAME OF FILE K-1, NAME OF FILE K THRU NAME OF FILE N

FILE IDENTIFICATION AND STRUCTURE

```
file K
  .
  .
  .
file N
```

Note that on any but the first reel, the directory is not the first item on the reel. Reel switching occurs in the middle of a file, and the remainder of a file that is split between two reels is written out before the directory is put on the new reel. The split file is logically considered to be on the reel on which it was begun.

The directory on each reel of a multiple-reel library tape named MSA is called MSA/FILE000. The numbers of files following the directory are FILE001, etc. In the headers of these files are reel-1 designations. Only files actually split over two reels have reel-2 designations in their headers.

Format of a Disk File Header

A disk file header consists of 15 header words immediately followed by row address words constructed according to standard mass-address word format. There are as many row address words as there are rows in the file as specified by the header field NUMROWSF (maximum = 1023). The file security information words, if any, follow the row addresses.

For directory files, the 15 header words, row addresses, and security words are immediately followed by the first record of the directory file. The format of a disk file header is as follows:

FIELD	FIELD NAME	CONTENTS
WORD 0		Contains core address or disk address. Disk address is defined as follows:
47:1	DKIADORWLOF	Write lockout bit
46:1		Undefined
45:1	DKNOTREADYF	Not-ready bit
44:6		Undefined
38:8	ROWCLASSF	Row class (AREACLASS) value
30:1	PACKBITF	Address is from pack header
29:8	EUNOF	EU unit number (0 to 255)
21:22	DISKADDRF	Address field (0 to 4,194,303)
WORD 1		Header Information
47:1	UPDATEBITF	True (1) if file updated
46:11	OPENCOUNTF	Number of processes looking at the header (0 to 2047)
35:4	DISKSPEEDF	Disk file speed (0 to 15)
31:8	FILEKINDF	Value of file FILEKIND attribute (0 to 255)

FILE IDENTIFICATION AND STRUCTURE

FIELD	FIELD NAME	CONTENTS
23:16	HEADERSIZEF	Size of this header in words (0 to 1038)
7:8	SECINFOSIZEF	Size of trailing header security information in words (0 to 255)
WORD 2		File Organization Information
47:1	IADBIT	True (1) if an Installation-Allocated Disk file
46:1	PROTECTIONF	True (1) if file is protected
45:1	PERMANENCYF	False (0) if file is temporary
44:1	TEMPORARYF	Temporary. (Turned on when a file is opened, and left on until the file is locked.)
43:1	CRUNCHEDF	True (1) if file is crunched
42:3	PHYSICALMODEF	EXTMODE at file creation
39:1	RCDUNTS	File Record Units 0 = words 1 = characters
38:4	RCRDTYPE	File Record Type 0 - fixed length 1 = variable length (length in decimal in first 4 characters of record) 2 = variable length (length in binary in first 2 characters of record) 3 = undefined, size provided explicitly 4 = variable length (length in record at fixed location) 5 = linked 6 = FORTRAN 7 = dependent specification fixed up at OPEN
34:3	SIZEMODE	File Size Mode
31:16	SIZEOFF	File Size Offset
15:16	SIZESZ	File Size Size
WORD 3		File Specifications - FIB TANKDATA2
47:16	BLOCKSIZEF	File Blocksize
31:16	MINRECSIZEF	File Minimum Record Size
15:16	MAXRECSIZEF	File Maximum Record Size
WORD 4		Contains the End of File Count
WORD 5		Row and Security Information
47:4	HEADERVERSIONF	Header format identification 0000 = old version header 0001 = new (II.2) version header
43:1	PRIVUSERF	True (1) for privileged user header
42:1	PASSWORDPRESENTF	True (1) if there exists a password for this header

FILE IDENTIFICATION AND STRUCTURE

FIELD	FIELD NAME	CONTENTS
41:2	SECURITYCODEF	Class of security on this file 0 = CLASSA 1 = CLASSB 2 = CLASSC 3 = PRIVATE
39:2	READWRITEF	Read/Write security field
37:14	NUMROWSF	Number of rows for which row address words are assigned (0 to 16383)
23:24	ROWSIZEF	Size of each file row in segments (0 to 16,777,215)
WORD 6		Name Qualification Information
47:10	SAVEFACTORF	File save time
37:18	CREATEDDATEF	File creation date
19:15	GENERATIONNOF	USASCI generation number
4:1	DUPLICATEBIT	True (1) for duplicated header
3:4	COPYNUMBERF	File copy number
WORD 7		Access Information
47:1	DISKPACKF	True (1) for disk pack file
46:1	NAMEDPACKF	True (1) for a named pack
42:1	CONTENDORSF	Number of contendors
31:3	MODEF	Disk file mode
17:18	LASTACCESSDATEF	Date file was last used
WORD 8		Is used for various purposes depending on the file kind. For directories, it contains the scramble modulus. For code files, it contains the stack number for the DI stack when a copy of the program is being executed. For pseudo reader deck files, it contains the deck number.
WORD 9		Is used for various purposes depending on the file kind. For directories, it is the next available record number. For BDFILES, it is the stack number of BDBASE stack.
WORD 10		Is used for various purposes depending on the file kind. For directories, it contains a count of empty scramble links. For variable length files, it contains the number of units in the last block of the file.
WORD 11		Is FIRSTDATA for control deck files.
WORD 12		Contains the core index of the file header.

FIELD	FIELD NAME	CONTENTS
WORD 13		Disk Pack Information
47:1	INTERCHANGEF	Pack interchange flag
46:1	SINGLEF	True (1) for a single pack file
45:1	SYSRESF	System resource bit
43:20	SERNUMF	Pack serial number
23:8	BASEUNITF	Location of base pack
15:8	LASTF	Last unit of file
7:8	PACKSF	Number of packs
WORD 14		End of File Word
47:20	DSKEOFU	Number of bits of valid data in the segment referenced in DSKEOFV (0 to 1,048,575)
27:28	DSKEOFV	Relative segment number containing EOF (0 to 3,026,531,840)

Structure of Disk Directory

The disk directory is a collection of files organized as a tree-like structure. Each of these files is referred to as a directory; the directory at the origin of the tree structure is referred to as the master directory.

A directory is composed of a header (refer to Format of Disk File Header) and a directory body. The directory body is made up of records that are 90 words (3 segments) long. Each of these records contains the following information in the following order.

a. Four words (0 through 3) that contain the following information:

   WORD 0 - A pointer to the successor record

   WORD 1 - A pointer to the predecessor record

   WORD 2 - An index to the first available name block

   WORD 3 - The record number of this record.

b. At most, 17 name block entries of five words (0 through 4) each. A description of a name block entry follows:

```
WORD 0      - Direct entry information
     47:5   - Storage medium where file is supposed to be when
              it is in use
     36:8   - File type for the file (program, directory,...)
     28:10  - Size (in words) of the header for this file
      1:2   - Directory end marker:
              3=end of record
              1=end of file
WORD 1      - Address of file header (standard mass storage
              address format)
     47:3   - Volume type
     44:25  - Volume or EU unit
     19:20  - Disk address
WORD 2      - An identifier that is part of a file name
     47:8   - Length of identifier, in characters
     39:40  - First five EBCDIC characters of the identifier
              (left justified with trailing 0's if necessary)
WORDS 3 and
       4    - Remainder of the identifier (up to 17 characters)
```

c. One word used as either an end-of-record or an end-of-file marker. (See description of field 1:2 of word 0 of a name block entry.)

Backup Files

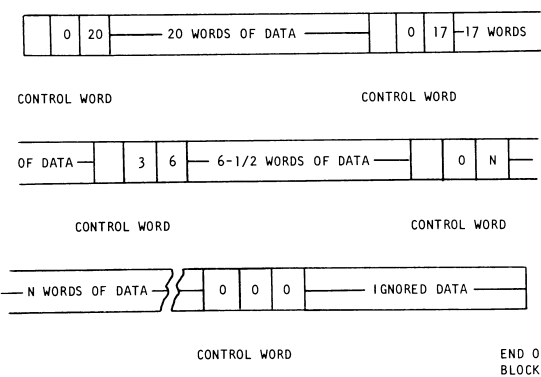
System BACKUP files are variable-length record, fixed-length-block files. Each block is 300 words long. Within a block, each logical record is composed of one control word followed by 0 or more words of data. A terminal control word of all zeros indicates that there are no more records in the present block.

FILE IDENTIFICATION AND STRUCTURE

Each control word is divided into specific fields. These fields are as follows:

Field	Contents
47:28	Identical with the corresponding portion of an I/O CONTROL WORD.
19:3	Character count residue for the data record (if the record to be printed consists of complete words, the value of this field will be zero).
16:17	Word count for the following data in the record in full words, not counting the control word.

Records in a BACKUP file are not split across blocks. Thus, if the last record in a given block ended in word 297 and the next record were 10 words long, then word 298 would be a control word containing all zeros. This control word indicates an end of the present block and the next record will begin at the start of the next block. The figure below illustrates a typical block of BACKUP records.



Format of System Backup Files

FILE ATTRIBUTES

Attributes are maintained in the FIB (File Information Block) and LEB (Label Equation Block), and used by the MCP when dealing with that file.

Buffer or DIRECT I/O attributes are related to a given I/O area rather than to the file itself. For this reason, when setting or reading these attributes, reference should be made to the buffer. Buffer attributes should only be used with a file on which DIRECT I/O is being performed.

Datacom attributes are related to files with KIND = REMOTE. Their values reflect the NETWORK DEFINITION LANGUAGE description of the file or are collected while the file is open in the station list of the remote file.

The following table lists the file attributes and some associated information. The column headed "NUM" gives the number that the MCP uses when accessing or setting an attribute. This is also the number which will be displayed when an attribute error occurs (except for DIRECT I/O attributes). The column headed "NAME" is the symbolic name that the compilers recoginize for this attribute. "R/W" column describes the ability of the attribute to be accessed (read) or set (written). RW means read and write capability; RO means the capability to read only and WO means the capability to write only. "TYPE" column gives the data type required when using this attribute. P stands for pointer attribute, I for integer, and B for Boolean.

The "R" column gives the conditions under which the attribute may be read (i.e., accessed), and the "W" column gives the conditions under which attribute may be written (i.e., set). The following codes are used for these columns:

C - FILE MUST BE CLOSED
O - FILE MUST BE OPEN
A - PHYSICAL FILE MUST BE ASSIGNED TO LOGICAL FILE
V - VARIABLE - MAY BE ACCESSED OR SET ANY TIME

A dash (-) in any column indicates that this column is not applicable, DIO - DIRECT I/O.

NUM	NAME	R/W	TYPE	R	W
000	TITLE	RW	P	V	-
001	REEL	RW	I	V	V
001	IOMASK (DIO)	RW	I	V	V
002	DATE	RW	I	V	C
002	IOCW (DIO)	RW	*R	V	V
003	CYCLE	RO	I	V	C
003	IOCANCEL (DIO)	RO	B	V	-
004	VERSION	RW	I	V	C
004	IOERRORTYPE (DIO)	RO	I	V	-
005	SAVEFACTOR	RW	I	V	V
005	IORESULT (DIO)	RO	B	V	-
006	DENSITY	RW	I	V	C
006	IOEOF (DIO)	RO	B	V	-
007	PARITY	RW	I	V	V
007	IOPENDING (DIO)	RO	B	V	-
008	KIND	RW	I	V	C

NUM	NAME	R/W	TYPE	R	W
008	IOCOMPLETE (DIO)	RO	B	V	-
009	LABELTYPE	RW	I	V	C
009	IOTIME (DIO)	RO	I	V	-
010	EXTMODE	RW	I	V	C
010	IORECORDNUM (DIO)	RO	I	V	-
011	OPTIONAL	RW	B	V	C
011	IOWORDS (DIO)	RO	I	V	-
012	PROTECTION	RW	I	V	C
012	IOCHARACTERS (DIO)	RO	I	V	-
013	FILETYPE	RW	I	V	C
013	IOADDRESS (DIO)	RO	I	V	-
014	BLOCKSIZE	RW	I	V	C
015	MAXRECSIZE	RW	I	V	C
016	MINRECSIZE	RW	I	V	C
017	AREASIZE	RW	I	V	C
018	AREAS	RW	I	V	C
020	MYUSE	RW	I	V	C
021	OTHERUSE	RW	I	V	V
022	FLEXIBLE	RW	B	V	C
024	SPEED	RW	I	V	V
026	BUFFERS	RW	I	V	C
027	DIRECTION	RW	I	V	C
028	CURRENTBLOCK	RO	I	V	V
029	INTMODE	RW	I	V	C
030	OPEN	RW	B	V	V
031	PRESENT	RO	B	V	-
032	SIZEMODE	RW	I	V	C
033	SIZEOFFSET	RW	I	V	C
034	SIZE2	RW	I	V	C
035	STATE	RO	I	A	-
036	EOF	RO	B	A	-
039	PACKNAME	RO	P	V	-
040	SINGLEPACK	RW	B	V	C
041	CYLINDERMODE	RW	B	V	C
042	RESIDENT	RO	B	V	-
044	BLOCK	RO	I	V	-
045	CARRIAGECONTROL	RW	I	V	C
046	RECORD	RO	I	V	-
049	LASTRECORD	RO	I	A	-
050	PAGESIZE	RW	I	V	V
051	PAGE	RW	I	A	V
052	LINENUM	RW	I	A	V
054	FORMMESSAGE	WO	P	-	V
057	UPDATED	RO	B	A	-
058	FILEKIND	RO	I	A	-
060	ROWINUSE	RO	I	A	-
061	USEDATE	RO	I	V	-
062	SERIALNO	RO	I	V	-
064	ROWADDRESS	RO	I	A	-

FILE ATTRIBUTES

NUM	NAME	R/W	TYPE	R	W
066	INTERCHANGE	RW	B	V	C
067	DUPLICATED	RW	B	V	C
069	COPIES	RW	I	V	C
072	INTNAME	RW	P		
073	UNITS	RW	I	V	C
074	ATTERR	RO	B	V	-
075	ATTYPE	RO	I	V	-
076	ATTVALUE	RO	I	V	-
078	UNITNO	RO	I	O	C
079	SECURITYGUARD	RW	P	V	V
080	SECURITYTYPE	RW	I	O	V
081	SECURITYUSE	RW	I	O	V
082	AREACLASS	RW	I	V	
083	READCHECK	RW	B	V	V
084	RECORDINERROR	RO	I	V	-
085	ERRORTYPE	RO	I	O	-
086	RECORDKEY	RO	I	V	-
087	IOINERROR	RO	B	V	-
088	TAPEREELRECORD	RO	I	A	-
099	FAMILYSIZE	RO	I	O	-
100	POPULATION	RO	I	O	-
101	FAMILY	WO	P	-	O
102	ENABLEINPUT	RO	B	O	-
105	TIMELIMIT	RW	*R	V	V
106	LASTSTATION	RW	I	O	O
107	RECEPTIONS	RO	I	O	-
108	CENSUS	RO	I	O	-
111	DISPOSITION	RO	I	O	-
112	TRANSMISSIONS	RO	I	O	-
113	ASSIGNTIME	RO	I	O	-
116	SCREEN	RO	B	O	-
117	TRANSMISSIONO	RO	I	O	-
118	WIDTH	RO	I	O	-

* Positive Real No.

COMMENT:

The attributes TITLE and PAGESIZE are general file attributes which have special meaning as datacom station attributes.

FILE ATTRIBUTES - ALPHABETICAL

The column headed "KIND" gives a code for kind of file attribute: General File Attribute (G); Direct I/O (DIO); Datacom Attributes (D).

NAME	NUM	R/W	TYPE	R	W	KIND
AREACLASS	082	RW	I			G
AREAS	018	RW	I	V	C	G
AREASIZE	017	RW	I	V	C	G
ASSIGNTIME	113	RO	I	O	-	D
ATTERR	074	RO	B	V	-	G

FILE ATTRIBUTES

NAME	NUM	R/W	TYPE	R	W	KIND
ATTVALUE	076	RO	I	V	-	G
ATTYPE	075	RO	I	V	-	G
BLOCK	044	RO	I	V	-	G
BLOCKSIZE	014	RW	I	V	C	G
BUFFERS	026	RW	I	V	C	G
CARRIAGECONTROL	045	RW	I	V	C	G
CENSUS	108	RO	I	O	-	D
COPIES	069	RW	I	V	C	G
CURRENTBLOCK	028	RO	I	V	V	G
CYCLE	003	RO	I	V	C	G
CYLINDERMODE	041	RW	B	V	C	G
DATE	002	RW	I	V	C	G
DENSITY	006	RW	I	V	C	G
DIRECTION	027	RW	I	V	C	G
DISPOSITION	111	RO	I	O	-	D
DUPLICATED	067	RW	B	V	C	G
ENABLEINPUT	102	RO	B	O	-	D
EOF	036	RO	B	A	-	G
ERRORTYPE	085	RO	I	O	-	G
EXTMODE	010	RW	I	V	C	G
FAMILY	101	WO	P	-	O	D
FAMILYSIZE	099	RO	I	O	-	D
FILEKIND	058	RO	I	A	-	G
FILETYPE	013	RW	I	V	C	G
FLEXIBLE	022	RW	B	V	C	G
FORMMESSAGE	054	WO	P	-	V	G
INTERCHANGE	066	RW	B	V	C	G
INTMODE	029	RW	I	V	C	G
INTNAME	072	RW	P			G
IOADDRESS	013	RO	I	V	-	DIO
IOCANCEL	003	RO	B	V	-	DIO
IOCHARACTERS	012	RO	I	V	-	DIO
IOCOMPLETE	008	RO	B	V	-	DIO
IOCW	002	RW	I	V	V	DIO
IOEOF	006	RO	B	V	-	DIO
IOERRORTYPE	004	RO	I	V	-	DIO
IOINERROR	087	RO	B	V	-	G
IOMASK	001	RW	I	V	V	DIO
IOPENDING	007	RO	B	V	-	DIO
IORECORDNUM	010	RO	I	V	-	DIO
IORESULT	005	RO	I	V	-	DIO
IOTIME	009	RO	I	V	-	DIO
IOWORDS	011	RO	I	V	-	DIO
KIND	008	RW	I	V	C	G
LABELTYPE	009	RW	I	V	C	G
LASTRECORD	049	RO	I	A	-	G
LASTSTATION	106	RW	I	O	O	D
LINENUM	052	RW	I	A	V	G
MAXRECSIZE	015	RW	I	V	C	G
MINRECSIZE	016	RW	I	V	C	G
MYUSE	020	RW	I	V	C	G

FILE ATTRIBUTES

NAME	NUM	R/W	TYPE	R	W	KIND
OPEN	030	RW	B	V	V	G
OPTIONAL	011	RW	B	V	C	G
PACKNAME	039	RO	P	V	-	G
PAGE	051	RW	I	A	V	G
PAGESIZE	050	RW	I	V	V	G
PAGESIZE	050	RO	I	O	-	D
PARITY	007	RW	I	V	V	G
POPULATION	100	RO	I	O	-	G/D
PRESENT	031	RO	B	V	-	G
PROTECTION	012	RW	I	V	C	G
READCHECK	083	RW	B	V	V	G
RECEPTIONS	107	RO	I	O	-	D
RECORD	046	RO	I	V	-	G
RECORDINERROR	084	RO	I	V	-	G
RECORDKEY	086	RO	I	V	-	G
REEL	001	RW	I	V	V	G
RESIDENT	042	RO	B	V	-	G
ROWADDRESS	064	RO	I	A	-	G
ROWINUSE	060	RO	I	A	-	G
SAVEFACTOR	005	RW	I	V	V	G
SCREEN	116	RO	B	O	-	D
SECURITYTYPE	080	RW	I	O	V	G
SECURITYUSE	081	RW	I	O	V	G
SERIALNO	062	RO	I	V	-	G
SINGLEPACK	040	RW	B	V	C	G
SIZE2	034	RW	I	V	C	G
SIZEMODE	032	RW	I	V	C	G
SIZEOFFSET	033	RW	I	V	C	G
SPEED	024	RW	I	V	V	G
STATE	035	RO	I	A	-	G
TAPEREELRECORD	088	RO	I	A	-	G
TIMELIMIT	105	RW	*R	V	V	D
TITLE	000	RW	P	V	-	G
TITLE	000	RO	P	O	-	D
TRANSMISSIONO	117	RO	I	O	-	D
TRANSMISSIONS	112	RO	I	O	-	D
UNITNO	078	RO	I	O	C	G
UNITS	073	RW	I	V	C	G
UNITSLEFT	055	RO	I	V	-	G
UPDATED	057	RO	B	A	-	G
USEDATE	061	RO	I	V	-	G
VERSION	004	RW	I	V	C	G
WIDTH	118	RO	I	O	-	D

* Positive Real no.

General File Attributes

A list of the file attributes (in alphabetical order), including Buffer and Datacom, with a brief explanation follows. See appropriate compiler documents for correct syntax for referencing attributes.

Buffer attributes should be used with DIRECT I/O files.

Remote files (i.e., files with KIND = REMOTE) are used in DATACOM I/O. Remote files are opened initially to include all the stations subsumed in the NETWORK DEFINITION LANGUAGE (NDL) file section description of that file. Therefore, a remote file may comprise a collection of stations of many different terminal types. One of the problems related to multi-station remote files is distinguishing between the member stations. This is accomplished by assigning a "relative station number" (RSN) to each station which is a member of the file. The RSN is in effect an index into the station list of the remote file, which is created when the file is opened. The RSN is not unique to the station, it is in fact reusable, and after extensive use of the family attribute, an RSN may or may not point to a valid (that is to say non-empty) entry in the station list.

There are both file and station DATACOM attributes. The FILE ATTRIBUTES deal with characteristics of the remote file as a whole and the STATION ATTRIBUTES deal with the characteristics of the specific stations in the file.

AREACLASS (82)

Integer. Used in disk file reconstruction. Permits user control of disk file row distribution among EU's. Must be set before disk address is assigned to row. Accessible at anytime. Can have value of zero to 255 with zero the default. Not label equatable. Parameter (row number) required when referencing. (See disk file data recovery document.)

AREAS (18)

Integer. Number of rows in a disk file. Accessible at any time. Set only when file is closed. Value = zero to 1000. Default value is 1. Label equatable (AREAS = <integer>). Initialized by file declaration (ALGOL), File-Control paragraph, "ASSIGN TO INTEGER-1 * INTEGER-2 DISK" Clause (COBOL), FILE card (FORTRAN).

AREASIZE (17)

Integer. Number of records per disk file row (AREA). Accessible at any time. Set only when file is closed. Label equatable (AREASIZE = <integer>). Initialized by file declaration (ALGOL), File-Control paragraph, "ASSIGN TO INTEGER-1 * INTEGER-2 DISK" Clause (COBOL), FILE card (FORTRAN). Meaningful only for output files.

FILE ATTRIBUTES

ASSIGNTIME (113)

Datacom station attribute. Integer. Returns the time at which the station was assigned to the file (not the length of time that the file has been open). Accessible at any time that the file is open. Read-only. Value is the number of seconds elapsed from midnight until the time that the station was assigned.

ATTERR (74)

Boolean. A read-only attribute which indicates file attribute error. Values = TRUE or FALSE. TRUE only if a file attribute action caused an error. Reset to false after a successful attribute action. May be tested by conditional statements (ALGOL), PROCEDURE DIVISION statements (COBOL).

ATTVALUE (76)

Integer. Returns value of last file attribute in error. Read-only attribute which may be tested by conditional statements (ALGOL), PROCEDURE DIVISION statements (COBOL).

ATTYPE (75)

Integer. Returns attribute number of the last file attribute in error. Read-only attribute which may be tested by conditional statements (ALGOL), PROCEDURE DIVISION statements (COBOL).

BLOCK (44)

Integer. Number of the current block. Disk file use only. Not label equatable. May be tested by conditional statements (ALGOL), PROCEDURE DIVISION statements (COBOL). Not specifiable (FORTRAN).

BLOCKSIZE (14)

Integer. Physical size of a block of records. May be read at any time. May be set only when file is closed. Specified in words (file attribute UNITS = 0) or INTMODE units (file attribute UNITS = 1). If UNITS is not specified BLOCKSIZE = words. Value may range from minimum size to 65,535 INTMODE units or words depending on UNITS. Default value = Record size. Disk files should be blocked in multiples of 30 words. Minimum block size for tape is six words. Label equatable (BLOCKSIZE = <integer>, UNITS = units mnemonic). Initialized by file declaration (ALGOL), DATA DIVISION, FILE DESCRIPTION, BLOCK CONTAINS clause (COBOL), not specifiable (FORTRAN).

BUFFERS (26)

Integer. Number of buffers assigned to the file. May be read at any time and may be set only when file is closed. May have value of one to 63. Default = 1. (COBOL default = 2, FORTRAN default = 2). Actual number of buffers may be less than amount requested if MCP determines that the space is not available. Label equatable (BUFFERS = <integer>). Initialized by file declaration (ALGOL), FILE card (FORTRAN), File-control paragraph, RESERVE clause (COBOL).

CARRIAGECONTROL (45)

Integer. Type of carriage control to be employed on this line printer file. May be read at any time; must be set only when file is closed. The value of this attribute indicates the effect the first character of each record is to have upon printer carriage control.

The values and mnemonics of this attribute are as follows:

0	STANDARD
1	CTLASA
2	CTL360

STANDARD is the default value. CTLASA is simulated for FORTRAN by the FORTRAN formatter.

Standard carriage control makes no attempt to interpret the first character of a printer record. Carriage control is handled through the language syntax.

NOTE

Nonstandard carriage control is only allowed for character-oriented (UNITS=1) EBCDIC (INTMODE=EBCDIC) printer files.

CTLASA indicates that the line is to be printed after carriage motion has been completed according to the first character of the line as follows:

First Character of Record	Carriage Action
blank	single space
0	double space
-	triple space
+	no carriage motion
n, where n is a non-zero digit	skip to channel n

CTL360 indicates that the first character (8-bit) of the line be divided into the following fields, the values of which have the following effect upon carriage control:

FILE ATTRIBUTES

Field	Carriage Action
[0:1]	If this bit is on, printing occurs after carriage motion, else before the motion.
[1:1]	If this bit is on, no printing occurs - only carriage control.
[2:1]	Ignored.
[6:4]	The value of this field is either the channel number (if skipping) or the line count (if spacing), according to the following bit.
[7:1]	If this bit is on, then a skip to channel n is to be performed, else space ahead the number of spaces indicated in [6:4].

CENSUS (108)

Either file or station datacom attribute. Integer. File attribute returns the number of messages currently in the input queue of the file. Station attribute returns the number of messages currently in the input queue of the station. Accessible whenever the file is open. Read-only. Meaningful only for remote files with MYUSE = 1 (IN) or MYUSE = 3 (I/O).

If the MCS is participating in I/O, there may or may not be a station input queue assigned and only the file attribute CENSUS should be referenced. If there is no input queue for the station, the station attribute CENSUS will return zero and the search will have caused a nonfatal attribute error.

COPIES (69)

Integer. Accepts or returns a value indicating the total number of copies within a duplicated file. The maximum number of copies is 16. The default value is two if a duplicated file is created, but the COPIES attribute has not been specified.

CURRENTBLOCK (28)

Integer. A read only integer which is accessible only when the file is open. The value returned is the size of the block currently in use, in logical units (i.e., intmode units if it is a character-oriented file, otherwise words).

CYLINDERMODE (41)

Boolean. Disk packs only. When true, this attribute provides for assignment of space by cylinders; i.e., space assigned to each area lies within the bounds of a single cylinder. Default value is false which provides space assignment without regard for cylinder bounds but rather on a space available basis. AREASIZE must not be greater than cylinder size when CYLINDERMODE is set; an attempt to introduce this condition will result in an "OPEN ERROR # 20" system message.

DATE (2)

Integer. Creation date of the file. Accessible at any time. May be set only when file is closed. Value = YYDDD, where YY = year and DDD = day of the year (JULIAN). Default value = <current date> (some other date may be specified if desired). Label equatable (DATE = <integer>). Initialized by file declaration (ALGOL), LABEL clause and USE routines (COBOL). Not specifiable (FORTRAN).

DENSITY (6)

Integer. Allows the user to specify the recording density when creating a magnetic tape file. Accessible at any time. May be set only when file is closed. Values range zero thru three. (0 (HIGH) = 800 BPI, 1 (MEDIUM) = 556 BPI, 2 (LOW) = 200 BPI, and 3 (SUPER) = 1600 BPI). 556 BPI not valid for 9-track tape; 1600 BPI not valid for 7-track tape. Label equatable (DENSITY = <density mnemonic>). Default = Selected by setting of available unit. Initialized by file declaration (ALGOL), (may be specified either by mnemonic or value). Not specifiable in COBOL or FORTRAN.

DIRECTION (27)

Integer. Indicates the direction (forward or reverse) in which records on tape files are accessed or will be accessed. Accessible at any time, and should be set when file is closed. May have value of zero (FORWARD) or one (REVERSE). Default is normally zero (FORWARD). Default is one (REVERSE) if file is opened reverse. Label equatable (DIRECTION = <direction mnemonic>). Initialized by file declaration (ALGOL) (DIRECTION = <integer>). Initialized by OPEN statement (COBOL). Not specifiable (FORTRAN). REVERSE cannot be specified for output files.

DISPOSITION (111)

Datacom station attribute. Integer. Returns the disposition of the station in the remote file. Accessible whenever the file is open. Read-only.

FILE ATTRIBUTES

Values and meanings:

0	Unknown
1	Assigned
2	Denied
4	Postponed
6	Denied, illegal use attempted

DUPLICATED (67)

Boolean. Accepts or returns a true (one) if the file is duplicated and a false (zero) if the file is not duplicated. The default value is false.

ENABLEINPUT (102)

Either station or file datacom attribute. Boolean. Station attribute returns TRUE if the station is both assigned and enabled for input. File attribute returns TRUE if at least one of the assigned stations in the remote file's station list is also enabled for input. Valid only for files with MYUSE = 1. Accessible only when file is open. Read-only.

EOF (36)

Boolean. Indicates whether or not (TRUE or FALSE) the end of the file has been reached. Read-only and may be read at any time (physical file must be assigned to logical file). Values of TRUE or FALSE. Default is FALSE. Not label equatable. May be interrogated programmatically.

ERRORTYPE (85)

Integer. Read-only and may be read at any time on DISK files only. Describes the type of error that most recently occurred. Its values are:

0 NO ERROR
1 SUNOTREADY
2 READPARITYERROR
3 READCHECKFAILURE

Default value = 0 (no error has occurred). Not label equatable. Can be interrogated only when disk file is open. Either the value or the mnemonic may be specified when testing this attribute. See disk file data recovery document for additional information concerning disk errors.

EXTMODE (10)

Integer. Specifies the character size, if any, required on an external I/O device. Accessible at any time. May be set only when file is closed. This value is ignored for input files; a labeled input file will have the external mode specified in the label.

External mode is compared with the internal mode (INTMODE) to decide whether or not translation is needed. The values of EXTMODE and their associated mnemonics are as follows:

0 SINGLE (word mode)
2 HEX
3 BCL
4 EBCDIC
5 ASCII
6 BINARY (card files)

Default value = value of INTMODE (output files). Label equatable (EXTMODE = <extmode mnemonic>). May be initialized in ALGOL with file declaration (EXTMODE = <mnemonic or value>). May not be initialized programmatically in COBOL or FORTRAN. Can be interrogated or changed in ALGOL (statements) or COBOL (PROCEDURE DIVISION statements). EXTMODE specifies, for input files, the character size of the physical file. For output files, EXTMODE specifies the character size desired on the physical file. If EXTMODE (external character size) and INTMODE (internal character size) differ, translation may be supplied in certain cases.

FAMILY (101)

Datacom file attribute. Pointer. The file attribute family may be used to add or subtract a station or a set of stations (the family of a file described in NDL) from a remote files station list. Write-only and may be set only when file is open. The FAMILY attribute makes it possible to alter dynamically the station list associated with an open remote file from its current value. However, if the file (or station) to be added or subtracted is not described in NDL, or is/is-not already in the station list, then no change is made to the remote files station list. Attempting to add/subtract a station/file not found in NDL causes an attribute error. Subtracting all the stations from the remote file does not close the file; however, a READ or WRITE statement while the FAMILYSIZE of the file is zero will cause end of file action.

When a station is subtracted from a file, all references to that station are removed from the files station list, the FAMILYSIZE and all other related attributes are decreased, the controlling MCS is sent a "FILE CLOSE" message for the station, and all knowledge of that station, from the files point of view, is lost.

FILE ATTRIBUTES

When a station is added to a remote file, a new entry is made in the station list, the FAMILYSIZE is increased, and the controlling MCS is sent a "FILE OPEN" message for the station.

FAMILYSIZE (99)

Datacom file attribute. Integer. Returns the number of stations in the files station list. FAMILYSIZE is equal to the number of stations specified by the description of the file in NDL, plus the number of stations added by use of the FAMILY attribute, minus the number of stations deleted by use of the FAMILY attribute. May be read whenever the file is open. Read-only.

FILEKIND (58)

Integer. An attribute that describes the disk file. May be read at any time that a physical file is assigned to a logical file. Default = 192 DATA. Not label equatable. May be accessed by value or mnemonic as follows:

VALUE	MNEMONIC	
0	SYSTEMDIRECTORY	
1	VERSIONDIRECTORY	
2	DIRECTORY	
3	CONTROLDECK	
4	BACKUPPRINTER	
5	RECONSTRUCTIONFILE	
6	SYSTEMDIRFILE	
7	JOBDESCFILE	
8-14	undefined	
15	XDISKFILE	
16	reserved *	backup files
17	BACKUPPUNCH	backup files
18-19	reserved	backup files
20	COMPILERCODEFILE	
21-28	undefined	
29	LIBRARYCODE (including APL Work Spaces)	
30	INTRINSICFILE	
31	MCPCODEFILE	
32	ALGOLCODE	executable code
33	COBOLCODE	executable code
34	FORTRANCODE	executable code
35	XALGOLCODE	executable code
36	PLICODE	executable code
37	JOVIALCODE	executable code
38	undefined	executable code
39	ESPOLCODE	executable code
40	DCALGOLCODE	executable code
41	BASICCODE	executable code
42	XFORTRANCODE	executable code
43	JOBCODE	executable code
43-61	undefined	executable code

VALUE	MNEMONIC	
62	BOUNDCODE	executable code
63	CODEFILE	executable code
64	ALGOLSYMBOL	symbolic files
65	COBOLSYMBOL	symbolic files
66	FORTRANSYMBOL	symbolic files
67	XALGOLSYMBOL	symbolic files
68	PLISYMBOL	symbolic files
69	JOVIALSYMBOL	symbolic files
70	undefined	symbolic files
71	ESPOLSYMBOL	symbolic files
72	DCALGOLSYMBOL	symbolic files
73	BASICSYMBOL	symbolic files
74	XFORTRANSYMBOL	symbolic files
75-93	undefined	symbolic files
94	BINDERSYMBOL	symbolic files
95-191	undefined	symbolic files
192	DATA	
193	SEQDATA	
194	GUARDFILE	
195-255	undefined (user data files)	

*for future BACKUPPRINTER

User programs are able to change compiler code files (e.g., ALGOLCODE, COBOLCODE, etc.) and CODEFILEs to data by setting the FILEKIND attribute. Only compilers are allowed to change a data file into a codefile.

FILETYPE (13)

Integer. Specifies the blocking technique of the file and allows variable record size. Accessible at any time; may be set only when the file is closed.

The values are as follows:

0 Blocked or unblocked fixed-length records.

1 Record length in decimal, in the first four INTMODE characters of the record. If INTMODE=0 (words), then in the first word.

2 Record length in binary, in the first two INTMODE characters of the record. If INTMODE=0 (words), then in the first word.

3 Record length specified externally to the record.

4 Record length in fixed location in the record, specified by the attributes SIZEOFFSET, SIZE2, and SIZEMODE. If SIZEMODE equals words, then SIZE2 is assumed to be 1.

FILE ATTRIBUTES

5 XALGOL linked -- A word between each record specifies the length, in INTMODE units, of the previous record (first half) and the length, in INTMODE units, of the next record (second half of the word). Except in the case of the first backward link and the last forward link which have values of zero, the record length includes the link word.

6 FORTRAN linked -- Binary records.

7 Record structure depends upon the FILETYPE used to create the assigned permanent file.

8 Informs the I/O subsystem that all the attributes other than INTMODE are to be determined by the permanent of physical file.

Only files with FILETYPE=0 may be read in the reverse direction. If FILETYPE is not set or is set to zero and MINRECSIZE is set to a nonzero value less than MAXRECSIZE, then FILETYPE will be set to 3 when the file is opened.

FILETYPEs 1 through 6 refer to variable-length records with variable blocking. If the next record to be written will cause the block to exceed BLOCKSIZE, a physical write is initiated and the record becomes the first record of the next block. If the file is assigned to a peripheral unit (KIND) allowing variable-size physical records (e.g., tape), only the actual block size used will be written. For peripheral units requiring fixed physical record sizes (e.g., disk), the block will be expanded appropriately and a length field of zero will be forced into this expanded area.

FILETYPE 2, 5, and 6 are not permitted for Datacomm (REMOTE) files. FILETYPE 4 is not permitted for REMOTE files if SIZEMODE has the value zero.

FILETYPE 3 must be specified if STATE is to reflect the actual size of the input record rather than the declared buffer size.

When FILETYPE=7, the format of the records in the logical file is determined by the format of the records in the physical or permanent file. Thus, FILETYPE, MINRECSIZE, BLOCKSIZE, SIZEMODE, SIZEOFFSET, SIZE2, UNITS, and INTMODE will be changed to agree with the physical file when the file is opened. When there is no permanent file, as in the case of output to the printer, FILETYPE is set to 0 (fixed-length records) and the other attributes are set to default values depending upon the physical device associated with the file. When a permanent file does not exist, FILETYPE=7 does not change BLOCKSIZE and MAXRECSIZE if their values are nonzero. As FILETYPE=7 sets INTMODE equal the file's actual EXTMODE, software translation is specifically avoided.

FILETYPE=8 is similar to FILETYPE=7 when opening a permanent file except INTMODE is not set. Hence, if the programmer set INTMODE different from the found EXTMODE or if INTMODE was left at default that was different

from EXTMODE, the software translation occurs. FILETYPE=8 thus permits software translation while FILETYPE=7 avoids software translation. When creating a file, if FILETYPE=8, then the use of default values is enforced for all attributes whether or not they had been previously set.

The value of FILETYPE has an effect upon the default value of MINRECSIZE. FILETYPEs which have linkwords or record length fields contained within the record require MINRECSIZE to be at least large enough to hold this information. Files of FILETYPE=0 do not use MINRECSIZE.

The following table displays the default values for attributes when a permanent file does not exist and FILETYPE=7 or 8.

Device	Values
Disk	EXTMODE=INTMODE MAXRECSIZE=BLOCKSIZE=30 words
Supervisory Console	INTMODE=EXTMODE=EBCDIC MAXRECSIZE=BLOCKSIZE=10 words
Remote Terminal	INTMODE=EXTMODE=EBCDIC MAXRECSIZE=BLOCKSIZE=12 words
Paper Tape Reader	INTMODE=EXTMODE=BCL MAXRECSIZE=BLOCKSIZE=10 words
Paper Tape Punch	INTMODE=EXTMODE=BCL MAXRECSIZE=BLOCKSIZE=10 words
Line Printer	INTMODE=EXTMODE MAXRECSIZE=BLOCKSIZE=17 words (BCL) 22 words (non-BCL)
Card Reader or Punch	INTMODE=EXTMODE (INTMODE=EBCDIC if EXTMODE=BINARY) MAXRECSIZE=BLOCKSIZE=20 words (BINARY) 10 words (BCL) 14 words (other)
Magnetic Tape	EXTMODE=INTMODE MAXRECSIZE=BLOCKSIZE=10 words

FLEXIBLE (22)

Boolean. Disk files only. This attribute, when set, permits the automatic creation of a new disk file header when no room exists in the present header for an additional row. All information in the old header is copied into the new header, and all copies of the old header are replaced by the new header. A minimum of 10 rows will be added when expanding a header.

FORMMESSAGE (54)

Pointer attribute. Used to notify the operator that a special form is required for a line printer file. If FORMMESSAGE is set, the string pointed to will be displayed on the operator's console when the file is opened. Printing will not commence until one of the following input messages is provided by the operator:

FILE ATTRIBUTES

```
<mixid> FMLP <unit number>

<mixid> OUDK

<mixid> OU MT <optional unit number>
```

The attribute is applicable to write-only files. It may be set at any time and the string pointed to must terminate in a period. The attribute is label equatable (FORMMESSAGE = <EBCDIC string, 15 characters or less, terminated with a period>). It can be initialized by the file declaration (ALGOL). It must not be set using a string in the file declaration; it must be set to a pointer variable global to the declaration. The pointer variable must be initialized prior to the file declaration.

FORMMESSAGE can be set for files, other than printer files. The only valid response (other than <mixid> DS) is:

```
<mixid> FM <unit mnemonic> <unit number>.
```

INTERCHANGE (66)

Boolean. Disk packs only. When true, this attribute indicates that the file is stored or is to be stored on a Burroughs Interchange pack.

INTMODE (29)

Integer. Specifies the internal character mode of the file. Accessible at any time. May be set only when file is closed. The values for this attribute are as follows:

```
0  SINGLE (word mode)
2  HEX
3  BCL
4  EBCDIC
5  ASCII
```

The value 6 (binary) which is allowed in EXTMODE may not be specified for INTMODE. Default value = value of EXTMODE (input files). Label equatable (INTMODE = <intmode mnemonic>). May be initialized programmatically in ALGOL (INTMODE = <value or mnemonic>) but may not in COBOL or FORTRAN. Can be interrogated in ALGOL or COBOL. If INTMODE and EXTMODE differ, automatic translation may or may not occur. (See EXTMODE.)

INTNAME (72)

Pointer. A pointer expression which points to the internal file name if the file name has been changed; also used to change the internal file name. Accessible at any time. May be changed only when file is closed and the simple identifier must be terminated with a period. Default value is the user-supplied internal file name. Not label equatable. Initialized in file declaration (ALGOL). Must not be initialized to a string; must be initialized to a pointer variable that has been initialized prior to entering the block in which the file is declared.

IOADDRESS (13)

Integer. Returns the absolute memory address of the first word of the specified buffer. Accessible at any time. Values: zero to maximum memory size. Not label equatable and is initialized by MCP only. Direct I/O attribute.

IOCANCEL (3)

Boolean. Indicates that an I/O operation has been canceled for this buffer, because an EOF, EOT, or end-of-reel condition has occurred as a result of an I/O operation for another buffer on the same file. Accessible at any time. Set only by the MCP when an end-of-file condition is detected. Not label equatable. Values 1 (TRUE) or 0 (FALSE) with a default of FALSE. Direct I/O attribute.

IOCHARACTERS (12)

Integer. Returns the number of characters transferred on the last I/O operation for the specified buffer if the operation was complete. Accessible at any time. Set only by MCP. Values of zero to size of buffer. Not label equatable. Direct I/O attribute.

IOCOMPLETE (8)

Boolean. Returns a FALSE if an I/O operation is in progress for the specified buffer or a TRUE if the operation is complete. Accessible at any time, and is set by MCP only. Not label equatable. Direct I/O attribute.

IOCW (2)

Integer. Accepts or returns the I/O control word (IOCW) for the specified buffer. Accessible at any time. May be set at any time. A zero will reset the IOCW. When set, this attribute will stay set until changed by user. Not label equatable. Direct I/O attribute.

IOEOF (6)

Boolean. Returns a one (TRUE) if the end of file was detected as a result of the last I/O operation for the specified buffer. Accessible at any time. Read-only. Values = 0 (FALSE), end of file was not reached; (TRUE), end of file was reached. Not label equatable. Direct I/O attribute.

IOERRORTYPE (4)

Integer. Indicates the error, if any, which occurred as a result of the last I/O operation for the specified buffer. Accessible at any time, and set only by MCP. Direct I/O attribute. Values and meanings are:

FILE ATTRIBUTES

Less than 0	An undefined error occurred
0	No error occurred
1	Not ready
2	Parity
3	Rewinding
4	Descriptor error
5	End of tape or beginning of tape
6	End of file or write lockout
7	An attempt was made to exceed the size specified for a disk row or to read past the end of a disk file
8	I/O was cancelled
9	Short record
10	Break - (Datacom)

IOINERROR (87)

Boolean. Indicates that a read or write error has occurred on a disk file. Used with disk file data recovery. May be read at any time. May not be set or changed programmatically. Values: TRUE if error is associated with file, FALSE otherwise. Not label equatable.

IOMASK (1)

Integer. Provides the ability to set or read the I/O error mask word in the I/O control block (IOCB), which allows the user to specify those errors that will be handled by the MCP. May be read or set at any time. Values: zero argument (RESET) causes the MCP to handle all errors. The bits which may be set correspond in position and in meaning to the error field (bits 16:17) of the result descriptor for the I/O device associated with the file. When set, the attribute will remain set until it is reset by the user. Not label equatable. Direct I/O attribute.

IOPENDING (7)

Boolean. Indicates whether or not an I/O operation for the specified buffer is being processed or is waiting to be processed. Accessible at any time. Read-only, and may not be initialized programmatically. Values: 1 (TRUE) or 0 (FALSE). Not label equatable. Direct I/O attribute.

IORECORDNUM (10)

Integer. Disk files only. Returns the logical record number of the last I/O performed for the specified buffer. Accessible at any time and set only by MCP. Value ranges from zero to the size (in records) of the file. Used in disk file data recovery. Not label equatable. Direct I/O attribute.

IORESULT (5)

Boolean. IORESULT is a direct I/O, read-only attribute which returns the result descriptor for a completed I/O. If a one is returned, then the I/O was cancelled.

IOTIME (9)

Integer. Returns the elapsed time for the I/O operation associated with the specified buffer. Accessible at any time. Read-only. Not label equatable. Direct I/O attribute.

IOWORDS (11)

Integer. Returns the number of words transferred on the last I/O operation for the specified buffer if the I/O operation was complete. Accessible at any time. Read-only. Values range from zero to buffer size (in words). Not label equatable. Direct I/O attribute.

KIND (8)

Integer. Describes the peripheral unit associated with the logical file.

The values, mnemonics and meanings of the Kind attribute are as follows:

Control Card	Compiler	Mnemonics	Device
	0	(default)	Any unit
1	1	DISK *	
2	2	DISPLAY SPO	Single Line Control
3	3	REMOTE DC	Data Communications
4	4	PAPERREADER PAPER PTR	Paper Tape Reader
5	5	PAPERPUNCH PTP	Paper Tape Punch

FILE ATTRIBUTES

Control Card	Compiler	Mnemonics	Device
22	7	PRINTER	Line Printer
18	9	READER	Card Reader
	10		Pseudo Reader
11	11	PUNCH CP	Card Punch
13	13	TAPE7 *	7-Track (NRZ) Tape
14	14	TAPE9 *	9-Track (NRZ) Tape
15	15	PETAPE *	9-Track (phase encoded) Tape
28	45	TAPE *	Any Tape
17	17	DISKPACK PACK	Disk Pack

* Can be modified by "Backup" in Control Card, e.g., KIND=PRINTER BACKUP TAPE7.

LABELTYPE (9)

Integer. Specifies the type of output file label (STANDARD or OMITTED). Valid only for output files. Label is USASI label. Accessible at any time. May be set or changed at any time that file is open.

0 STANDARD

1 OMITTED

3 OMITTEDEOF

Default values are: OMITTED for paper tape punch, remote, and display files; STANDARD for tape, printer, punch, and disk files. Label equatable (LABELTYPE = <labeltype mnemonic>). Initialized by file declaration (ALGOL) (LABELTYPE = <value or mnemonic>), LABEL RECORDS clause (COBOL), FILE card (FORTRAN).

For tape files:

OMITTED - indicates an old-type COBOL unlabeled tape. Reading a tape mark causes an attempted reel switch which can be ULed or FRed.

OMITTEDEOF - indicates an old-type ALGOL unlabeled tape. Reading a tape mark causes EOF action.

LASTRECORD (49)

Integer. Indicates the physical record number of the last record in a file. Valid only for permanent disk files (that is, files which have been closed, entered in the disk file directory, and then reopened) and is read-only. Accessible at any time.

LASTSTATION (106)

Datacom file attribute. Integer. Specifies the station to which output is to be directed. Returns the relative station number (RSN) of the station from which the last input was received by the object program, unless LASTSTATION was explicitly set. May be interrogated or set whenever the file is open. Valid RSN numbers are always greater than zero, and the default value of LASTSTATION when the file is opened is zero. LASTSTATION is implicitly set by a READ or a directed WRITE statement. A write to a file for which LASTSTATION = 0 is interpreted as a broadcast write to every station attached to the file.

LINENUM (52)

Integer. Accessible whenever physical file is assigned to logical file. May be set at any time. Only meaningful for PRINTER files when PAGESIZE attribute has been set. Indicates the current line number for the page between zero and the value of PAGESIZE. Setting LINENUM to a value less than the current line causes an end of page (EOF) result. Not label equatable. Default is initially one, and is incremented each time a record is written on file. Initialized in file declaration or assignment statement (ALGOL), and PROCEDURE DIVISION statements (COBOL).

MAXRECSIZE (15)

Integer. Specifies the maximum record size contained in or permitted on a file. Accessible at any time and may be set only when file is closed. Value ranges from one to 65,535 INTMODE units if UNITS attribute is equal to one. Otherwise, value is given in words. If left unset or set to zero, a default value will be assigned by the I/O subsystem. The default value depends upon the assigned peripheral unit (KIND) and the value of the attribute BLOCKSIZE. Label equatable (MAXRECSIZE = <integer>). Initialized by file declaration (ALGOL), FILE card (RECORD = <integer>) (FORTRAN), size of largest record in Record Description (COBOL).

MINRECSIZE (16)

Integer. Specifies the minimum record size contained in or permitted on a file. Accessible at any time and may be set only when file is closed. Value range from one to 65,535 INTMODE units if UNITS attribute is equal to one. Otherwise, value is given in words. Default = same as MAXRECSIZE. Label equatable (MINRECSIZE = <integer>). Initialized by file declaration (ALGOL), size of smallest record in Record Description (COBOL). May not be initialized (FORTRAN).

FILE ATTRIBUTES

MYUSE (20)

Integer. Specifies how the program will use the file. Accessible at any time and may be set only when file is closed.

0 CLOSED

1 IN

2 OUT

3 IO

Default is 1 (IN), assigned when file is opened. Label equatable (MYUSE = <mnemonic>). If the object program does not explicitly open the file (this is done in ALGOL by setting the OPEN attribute to TRUE: all COBOL files must be explicitly opened) then, unless the MYUSE value is IO, the opening action (Read or Write statement) determines the setting of the MYUSE attribute (IN or OUT respectively). If MYUSE has not been set and the file is opened explicitly, MYUSE is set to IN.

OPEN (30)

Boolean. Indicates whether or not the file is open. Accessible at any time and may be set at any time. Setting the attribute to FALSE when file is open closes the file with retention. Setting the attribute to TRUE when file is closed opens the file. Not label equatable. Set by first READ or WRITE statement (ALGOL, FORTRAN), OPEN statement (COBOL). May be referenced either by value or mnemonic.

OPTIONAL (11)

Boolean. Indicates whether a file which is not present at file open time may be specified as an optional file by the operator. (See OF message in Section 12.) Accessible at any time and may be set only when file is closed.

0 FALSE (the file is required)

1 TRUE (the file is optional)

Default is FALSE. Label equatable (OPTIONAL = <mnemonic>). Initialized in file declaration (ALGOL), (OPTIONAL = <value or mnemonic>), by SELECT statement (COBOL). Not used (FORTRAN).

PACKNAME (39)

Pointer. Returns the name of the disk pack on which the file is located. Read-only, and may be read at any time. Accessible in ALGOL, COBOL, ESPOL, and DCALGOL. May be specified in file declaration.

PAGE (51)

Integer. Indicates the page number of the current logical page in relation to the PAGESIZE. Accessible whenever physical file is assigned to logical file. May be set at any time. Meaningful for print files, and only when PAGESIZE attribute has been set. Value ranges from zero to value of current page. Default is initially 0, and is incremented each time a page is written on the file. Not label equatable and may be initialized in file declaration or assignment statement (ALGOL) or by a PROCEDURE DIVISION statement (COBOL).

PAGESIZE (50)

Integer. Specifies the number of lines on a logical page. May be set or read at any time. Meaningful for print files. May have a value of one to 255. Label equatable (PAGESIZE = <integer>). Initialized by file declaration (ALGOL), LINAGE clause (COBOL). May be interrogated or set by ALGOL statements or COBOL PROCEDURE DIVISION statements.

PAGESIZE (50)

Datacom station attribute. Integer. Returns the number of lines on a logical page as specified in the NDL description of the station. Accessible only when file is open and read-only. Set by the NDL description only.

PARITY (7)

Integer. Specifies whether parity is odd or even. This attribute has meaning for 7-track magnetic tape files only. Accessible at any time and can only be set when file is closed. Set by value or mnemonic in ALGOL.

0 STANDARD (Binary or odd parity)

1 NONSTANDARD (Alpha or even parity)

POPULATION (100)

Datacom and general file attribute. Integer. Returns the number of stations assigned to the file or the number of users assigned to permanent disk file. Accessible only when file is open. Set only by MCP. Maximum population is equal to FAMILYSIZE.

FILE ATTRIBUTES

PRESENT (31)

Boolean. A read-only attribute that returns:

1 TRUE (file is present and assigned to the logical file)

2 FALSE (file is not present)

Not label equatable.

PROTECTION (12)

Integer. This attribute applies only to output disk files. The values of protection are as follows:

0 TEMPORARY

1 SAVE

2 PROTECTED

The default value of PROTECTION is temporary. This means that a new disk file is thrown away when the program goes to end of job, or the block in which the file was declared is exited, unless the file is explicitly closed and locked. If the file is locked, an entry is then made in the Disk Directory and the file becomes a permanent disk file.

If PROTECTION is set to save, an entry in the Disk Directory is made immediately when the file is opened. The file becomes a permanent disk file and remains after the program has finished, unless the file is explicitly closed and purged.

If PROTECTION is set to Protected, an entry in the Disk Directory is made immediately when the file is opened; and as the disk areas are allocated. They are encoded with a pattern which makes it possible to discover the last valid block written on that area in the event of a Halt/Load.

READCHECK (83)

Boolean. If set to TRUE, this attribute causes every disk write to be immediately followed by a disk read for verification of data. If a read parity occurs, the write is reinitiated followed by a read for nine more times or until a successful read. Until this sequence is complete, the incomplete bit in the associated I/O control block (IOCB) is off. If these attempts are unsuccessful, the IOCB will indicate an irrecoverable write error occurred. The default value for READCHECK is FALSE. See disk file data recovery document.

RECEPTIONS (107)

Datacom station attribute or file attribute. Integer. Station attribute returns the number of messages which have been received by the object program from the station. The file attribute returns the number of messages which the object program has received from all of the stations assigned to the file. Accessible only when file is open. Read-only.

RECORD (46)

Integer. Indicates the current position in the file. Values: Minus one to an end of file value. Default: Initialized to minus one by MCP at time file is opened, incremented by one as each record is written or accessed. Not label equatable.

RECORDINERROR (84)

Integer. Specifies the logical record number of the disk file record most recently in error. Used for disk file data recovery. Accessible at any time. A read-only attribute. Value may range from one to the maximum number of logical records contained on the file. Not label equatable.

RECORDKEY (86)

Integer. Returns the logical record number of the disk record most recently in error. Used for disk file data recovery. Read-only. Accessible at any time. Value may range from one to the maximum number of logical records contained on the file. Not label equatable.

REEL (1)

Integer. Specifies the reel number of a tape file. May be read or set at anytime. May have a value of one to 9999. Default value: the first reel has a value of one, the value for each succeeding reel of a file will be incremented by one. Label equatable (REEL = <integer>). This attribute is file relative, and is reset to one at the beginning of each new file on a multifile tape.

RESIDENT (42)

Boolean. Indicates the status of the physical file which is to be assigned to the logical file. Accessible when file is closed. Read-only attribute.

0 FALSE (not present)

1 TRUE (present)

FILE ATTRIBUTES

Interrogating this attribute does not cause the file to be assigned to a logical file or to be opened (see PRESENT). However, if Filetype is seven, or eight, following a test for resident (which returns true), the other attributes which describe the permanent (physical) file will now be accessible. Label equatable.

ROWADDRESS (64)

Integer. Returns the address of the specified row of a disk file. Used in disk file data recovery. Accessible at any time the physical file is assigned to logical file. May be set only by MCP. Not label equatable.

ROWINUSE (60)

Integer. Disk files only. Returns the number of rows in use; that is, the number of rows of a disk file to which data have been written. Accessible at any time that physical file is assigned to logical file. Read-only attribute. Not label equatable.

SAVEFACTOR (5)

Integer. May be set only on output files. May have a value of zero to 999. Default = zero. Label equatable (SAVEFACTOR = <integer>). Initialized by file declaration (ALGOL), SAVE-FACTOR IS clause (COBOL), FILE card (FORTRAN).

SCREEN (116)

Datacom station attribute. Boolean. Returns a value of TRUE if the station has been declared in NDL to be a screen device. Accessible only when file is open. Read-only.

SECURITYGUARD (079)

A pointer valued attribute which names the guard file if security is invoked for the file.

SECURITYTYPE (80)

Integer. Controls or specifies the type of security associated with a disk file. Accessible only when file is open. May be set at any time. The values are as follows:

0 PRIVATE

1 CLASSA

2 CLASSB

PRIVATE - Only the creator of the file (specified by USERCODE) may access the file. Default = 0 (PRIVATE) if file is created by a job for which a usercode was specified. Default = 1 (CLASS A) for files created by all other jobs. Label equatable (SECURITYTYPE = <mnemonic>). May be interrogated in ALGOL or COBOL. May be set in ALGOL program or COBOL program which creates the file. May be changed by a SECURITY COMMAND control statement.

CLASS A - Access to CLASS A files is controlled by the MCP at file-open time. The factors governing access to a file are the user class and security class of the user desiring access, the type of access desired by the user, and the file type of the file which the user desires to access. Each system user will have a USER ID and his user class and security class will be associated with this ID by means of a special file. From the point of view of file security, there are four types of files access which a user may request: READ-ONLY, READ/WRITE, LIBRARY MAINTENANCE, and SECURITY MAINTENANCE. The file type indicates whether access to the file is restricted and if so how. If access is restricted, then there will be a class of privileged users which may be determined on the basis of user class, security class, or both of these. A non-privileged user is not necessarily denied all access to the file; he may, for example, be allowed read-only access to a file which only privileged users may alter. On the other hand, a nonprivileged user might be denied all access to a file to which privileged users had only read-only access.

CLASS B - Others may use the file subject to the constraints supplied by the guardfile.

The following constraints apply to the use of this attribute and also the use of SECURITYUSE (#81).

1. To read the attribute. If the file is not open then the attribute must be explicitly set prior to the attempt and subsequent to the last close on the file.

2. To set the attribute. For permanent files the attribute may only be set by a job under the usercode associated with the file or by a privileged user. If the file was created without a usercode, then only a privileged user may alter the security attributes. ALGOL programs can set the attribute by either value or mnemonic.

SECURITYUSE (81)

Integer. Controls or specifies the manner in which the disk file may be accessed. Accessible only when file is open. May be set at any time. The values are as follows:

0 SECURED

1 IN

2 OUT

3 IO

SECURED indicates that no one except the creator of the file may access the file with the exception that if it is a CODEFILE it may be executed.

IN, OUT, IO indicate READ-ONLY, WRITE-ONLY and READ/WRITE, respectively.

See SECURITYTYPE attribute for constraints on accessing or changing SECURITYUSE attribute. Default is IO. Label equatable (SECURITYUSE = <mnemonic>).

SERIALNO (62)

Returns the physical serial number of a magnetic tape. Presently an integer in value but will be changed in the near future to a string attribute to meet the original specifications. Read-only attribute and is accessible at any time. May be set only by using the "SN" input message. Value may range from zero to 999999. Not label equatable.

SINGLEPACK (40)

Boolean. Disk packs only. Specifies that the areas of an output file are to be assigned to a single disk pack. The default value is FALSE which provides for distribution over multiple packs, the number of packs being the number present when the file is open. If an additional member of the pack set is mounted subsequent to file open, then it will be inserted into the rotation scheme of allocation.

Areas are uniformly distributed over all packs of a pack set. The file must have a multi-file ID which matches the name of the pack set; e.g., file A/B requires a pack or packset named A.

SIZEMODE (32)

Integer. Specifies the units in which record size is given for FILETYPE 4 (four). Accessible at any time. May be set only when file is closed. The values and mnemonics are the same as INTMODE attribute.

Label equatable (SIZEMODE = <mnemonic>).

SIZEOFFSET (33)

Integer. Points to the file within the record that specifies the size of the record. Used with FILETYPE 4 (four). May be read at any time. May be set only when file is closed and its value may range from zero to 65,535 in SIZEMODE units. Label equatable (SIZEMODE = <mnemonic>).

SIZE2 (34)

Integer. Indicates the length of the field pointed to by SIZEOFFSET used with FILETYPE 4 (four). Accessible at any time. May be set only when file is closed. Value may range from zero to 65,535 in SIZEMODE units.

SPEED (24)

Integer. Specifies the speed of a disk storage unit. It is used when allocating AREAS to a disk file. May be read or set at any time by either value or mnemonic.

0 FAST

1 MEDIUMFAST

2 MEDIUMSLOW

3 SLOW

Default value = 0 (FAST). Label equatable (SPEED = <mnemonic>).

STATE (35)

Integer. Read-only attribute that returns the current condition of a file. Accessible only when the file is open. The value returned is a word which is a copy of the SOFTWARE I/O RESULT DESCRIPTOR.

The fields of this word contain the following information regarding the most recent I/O operation on the file:

FILE ATTRIBUTES

Field	Contents
[0:1]	If this bit is on, an error has occurred and the following field is nonzero.
[16:16]	Error information.
[4:1]	If this bit is on, a data error has occurred (size error for variable-length records).
[7:1]	If this bit is on, a parity error has occurred.
[9:1]	If this bit is on, EOF or end-of-page was encountered.
[10:1]	If this bit is on, a short block has been read. This is reported without [0:1] being turned on.
[13:1]	If this bit is on, this datacom file encountered a break on output.
[15:1]	If this bit is on, the I/O time limit was reached.
[16:1]	If this bit is on, a security violation occurred.
[24:8]	Qualification information. Reason for an EOF encountered on a datacom file.

Value	Meaning
1	File assignment denied.
2	File assignment postponed.
3	Illegal use of file.

[47:20]	Actual data size of the last logical record read or written in INTMODE units or words as specified by the value of the UNITS attribute of the file. FILETYPE must be equal to 3 for this field to reflect the actual size of the record rather than the declared size of the buffer.

TAPEREELRECORD (88)

Integer. Tape files only. This attribute will return the record number of the last record read or written relative to the beginning of the current tape reel. For example, after reading the first record of the second reel, TAPEREELRECORD will return a zero.

Note 1: The "close reel" construct will close the current reel without adjusting the record count of the file.

Note 2: TAPEREELRECORD on a closed file will be -1; using TAPEREELRECORD on a non-labeled tape or non-tape file will give an attribute error # 88.

Note 3: Reel switching during a spacing operation will maintain record and block counts for use by TAPEREELRECORD and block count checking.

TITLE (0)

Pointer. The external file name, a series of identifiers separated by slashes and terminated by a period. May be read at any time. A permanent disk file can have its TITLE change when the logical file assigned to it is opened. Each identifier in the title must be 17 characters or less. Default value is the internal file name. Label equatable (TITLE = <file ID>). Initialized by file declaration in ALGOL, (TITLE = "<file ID>."), by VALUE OF ID IS "<file-title>" clause in COBOL, and FILE card in FORTRAN. Note that the ALGOL file declaration requires that the name be terminated with a period and enclosed in quotes, CONTROLCARD allows but does not require the name to be in quotes and disallows the terminating period. The COBOL clause requires that the name be enclosed in quotes but does not accept the period, and the FORTRAN FILE card permits neither the period nor the quotes.

TITLE (0)

Datacom station attribute. Pointer. Returns the name of the station (as specified in the NDL description of the station). Accessible only when file is open. Read-only. If the relative station number (RSN) passed as a parameter is invalid, a ZERO length identifier will be returned followed by a period. If the RSN equals ZERO the action is the same as for the general file attribute TITLE and returns the title or external name of the file.

TIMELIMIT (105)

The Datacom file (KIND = REMOTE) attribute. TIMELIMIT is a positive real number in units of seconds. It can be both set and read. The attribute can be set via a label equation, in the file declaration, at runtime with the file either opened or closed, or in read or write statements. It can be read only while the file is open. The action when TIMELIMIT is greater than zero, is as follows:

a. On a read statement, if no input is received within TIMELIMIT seconds, the read statement is terminated with a TIMELIMIT error.

b. On a write statement, if no buffer becomes available within TIMELIMIT seconds, the write statement is terminated with a TIMELIMIT error.

FILE ATTRIBUTES

A TIMELIMIT error is reported by the logical I/O result descriptor having the attention bit [0:1] and bit [15:1] turned on.

Syntax for setting TIMELIMIT in an ALGOL I/O statement:

```
ADD TO THE <RECORD NUMBER OR CARRIAGE CONTROL> DEFINITION.

   [TIMELIMIT <ARITHMETIC EXPRESSION>]
```

Example:

```
WRITE(FILEID [TIMELIMIT 25.3],12,ARRAYROW);
```

TRANSMISSIONO (117)

Datacom station attribute. Integer. Returns the transmission number of the transmission most recently received from the station by the object program. Accessible only when the file is open. Read-only. If the NDL description does not request the DCP to assign transmission numbers to the transmissions of the station, the value returned will be minus one.

TRANSMISSIONS (112)

Station or file datacom attribute. Integer. Station attribute TRANSMISSIONS returns the number of output messages sent to the station. File attribute TRANSMISSIONS returns the number of output messages sent to all stations attached to the file. Accessible whenever file is open. Read-only.

UNITNO (78)

Integer. Accepts or returns the hardware unit number of the physical unit to which the file is assigned or to which the file is to be assigned. Accessible only when file is open. May be set whenever file is closed. This unit number is used as an index into the MCP unit table.

UNITS (73)

Integer. Indicates whether BLOCKSIZE, MAXRECSIZE, MINRECSIZE, and UNITSLEFT will be specified in INTMODE units or 48-bit words. Accessible at any time. May be set only when file is closed. Default value = 0 (WORDS). Label equatable (UNITS = <mnemonic>). Values and mnemonics are as follows:

0 WORDS (48-bit units)

1 CHARACTERS (INTMODE defined units)

UPDATED (57)

Boolean. Indicates whether or not a disk file is updated (that is, has been read since it was created, thus resetting the starting date to which SAVEFACTOR is referenced). May be read at any time. Set only by MCP. Not label equatable.

USEDATE (61)

Integer. Returns the date of last usage of a disk file in the form YYDDD (JULIAN). Accessible at any time. Set only by MCP. Not label equatable.

VERSION (4)

Integer. Indicates the version (of a CYCLE) of a file created on a specific DATE. Accessible at any time. May be set only when file is closed. Values range from one to 99, with default of one. Label equatable (VERSION = <integer>). Initialized by file declaration in ALGOL, LABEL clause and USE routines in COBOL and is not specifiable in FORTRAN.

WIDTH (118)

Datacom station attribute. Integer. Returns the logical line length in characters for the station as specified in the NDL description of the station. Accessible when file is open. Read-only.

TRANSLATION COMBINATIONS

BCL/EBCDIC TRANSLATION

When 80 column BCL card-code punch cards are read into EBCDIC buffers, the five BCL characters + ← ≤ ≠ ≥ are translated to HEX CO 4F 5F 6D 7D respectively.

When EBCDIC buffers are written to Burroughs Drum Printers (with PC Encoder lugs 7 to 8 jumpered) the EBCDIC bytes 4F 5F 6D 7D are carried to the 4 unique characters specified for the drum option installed on that printer.

When EBCDIC buffers are written to 7 track even parity tapes, all bytes that map to the "?" character when translate from EBCDIC to BCL are re-translate by the tape control into the ">" character. This is because the BCL "?" character is six zeroes which under even parity recording would result in writing of a tape drop out.

The following table clarifies the 80 column card reader and drum printer translation:

BCL card code	+0	+78	-78	028	78
BCL graphic	+	←	≤	≠	≥
	(PLUS)	(ARROW)	(LEQ)	(NEQ)	(GEQ)
BCL internal 6-bit code (octal)	20	37	57	74	17
EBCDIC code (HEX representation)	CO	4F	5F	6D	7D
Usual EBCDIC graphic	+	|	¬	¯	'
EBCDIC card code	+0	+78	-78	058	58
Line printer graphics printed when 8-bit byte write issued:					
With line printer PC encoder card not jumpered from 7 to 8:					
(all drum options)	+	?	?	?	?
With line printer PC encoder card jumpered from 7 to 8:					
(1) Burroughs EBCDIC drum & EBCDIC encoder	+	|	¬	¯	'
(2) Burroughs USA BCL drum & BCL encoder	+	←	≤	≠	≥
(3) Burroughs Latin American BCL drum & Portugese BCL encoder	+	←	≤	≠	^ □
(4) UK option EBCDIC	+	!	¬	¯	'
(5) Burroughs Latin American BCL drum & Spanish encoder	+	←	≤	≠	≥

DATA TRANSLATION COMBINATIONS FOR MAGNETIC TAPE FILES

OUTPUT TAPES:

IF FILE IS ASSIGNED TO:	AND PARITY IS:	AND RECORD DESCRIPTION IS:	THEN OUTPUT IS:
TAPE	ODD	EBCDIC	EBCDIC 9-TRK BINARY
TAPE	EVEN	EBCDIC	ILLEGAL
TAPE	ODD	BCL	BCL 7-TRK BINARY
TAPE	EVEN	BCL	BCL 7-TRK ALPHA
TAPE 9	ODD	EBCDIC	EBCDIC 9-TRK BINARY
TAPE 9	EVEN	EBCDIC	ILLEGAL
TAPE 9	ODD	BCL	BCL 9-TRK BINARY
TAPE 9	EVEN	BCL	ILLEGAL
B 5700 TAPES: TAPE 7	ODD	EBCDIC	EBCDIC 7-TRK BINARY
B 5700 TAPES: TAPE 7	EVEN	EBCDIC	BCL 7-TRK ALPHA (DATA IS TRANSLATED)
B 5700 TAPES: TAPE 7	ODD	BCL	BCL 7-TRK BINARY
B 5700 TAPES: TAPE 7	EVEN	BCL	BCL 7-TRK ALPHA

INPUT TAPES:

IF TAPE IS:	AND PARITY IS:	AND RECORD DESCRIPTION IS:	THEN INPUT DATA IS:
7- OR 9-TRK	ODD		NOT TRANSLATED REGARDLESS OF INTERNAL DESCRIPTION.
7-TRK	EVEN	EBCDIC	TRANSLATED TO EBCDIC FROM EXTERNAL BCL.
7-TRK	EVEN	BCL	TRANSLATED FROM EXTERNAL BCL TO INTERNAL BCL.

For legal combinations with tape where EXTMODE = INTMODE, or if either is of type SINGLE (VALUE=0), there is no software translation.

MEDIA	ACCESS	BLOCKING	FIXED/VAR. LENGTH RECORDS	TYPE OF OPEN	TRANSPARENT WORK AREA ("PROCESSING BOX")?	PHYSICAL MEDIA TRANSFER (AT TIME OF REQUEST) OF LOGICAL RECORD REFERENCED IN PROGRAM?				COMMENTS
						ONE FILE BUFFER		TWO OR MORE FILE BUFFERS		
						READ	WRITE	READ	WRITE	
MAG TAPE	SEQ	UNBLKD	F	INPUT	NO	YES	-	NO	-	buffer
			F	OUTPUT	NO	-	YES##	-	YES##	switch
			V	INPUT	NO	YES	-	NO	-	on multi-
			V	OUTPUT	NO	-	YES##	-	YES##	buffer files
		BLKD	F	INPUT	NO	VARIES*	-	NO	-	buffer switch after
			F	OUTPUT	NO	-	VARIES###	-	VARIES###	last rec. in block
			V	INPUT	NO	VARIES*	-	NO	-	is processed on
			V	OUTPUT	MAYBE#	-	VARIES###	-	VARIES###	multi-buffer files
DISK		UNBLK	F	INPUT	NO	YES	-	NO	-	
			F	OUTPUT	NO	-	YES##	-	YES##	buffer
			F	I/O	NO	YES	YES##	NO	YES##	switch
			V	INPUT	NO	YES	-	NO	-	on multi-
			V	OUTPUT	NO	-	YES##	-	YES##	buffer files
			V	I/O	NO	YES	YES##	NO	YES##	
		BLKD	F	INPUT	NO	VARIES^	-	NO	-	buffer
			F	OUTPUT	NO	-	VARIES###	-	VARIES###	switch
			F	I/O	YES	VARIES*	VARIES###	NO	VARIES###	after last rec.
			V	INPUT	NO	VARIES^	-	NO	-	in block is
			V	OUTPUT	MAYBE#	-	VARIES###	-	VARIES###	processed on
			V	I/O	YES	VARIES*	VARIES###	NO	VARIES###	multi-buffer files
	RANDOM	UNBLKD	F	INPUT	NO	VARIES*^	-	VARIES*^	-	
			F	OUTPUT	NO	-	NO^^^	-	NO^^^	
			F	I/O	NO	VARIES**	NO^^^	VARIES**	NO^^^	
			V	INPUT	NO	VARIES**	-	VARIES*^	-	
			V	OUTPUT	NO	-	NO^^^	-	NO^^^	
			V	I/O	NO	VARIES**	NO^^^	VARIES**	NO^^^	
		BLKD	F	INPUT	NO	VARIES**	-	VARIES*^	-	
			F	OUTPUT	NO	-	NO^^^	-	NO^^^	
			F	I/O	YES	VARIES**	NO^^^	VARIES**	NO^^^	

LEGEND:

#	if (No. of buffers x (maxrecsize - minrecsize)) > maxrecsize, then maxrecsize else blksz + (maxrecsize - minrecsize)
##	buffer is released for output, but program not suspended till I/O complete. (When the new buffer is brought up, its I/O must be complete, otherwise the program is suspended).
###	buffer is released for output only if last record in block, but program not suspended till its I/O complete; but possibly on new buffer brought up (See##).
*	buffer is released for input only if lst record in new block is referenced in program.
**	on random file reads a physical transfer in from disk takes place only if logical record not present in a file buffer.
***	on random file writes the buffer is marked "to be written" but physical transfer doesn't take place until the buffer is needed for another purpose.

MAINTENANCE OF STANDARD SOFTWARE

The standard software released by Burroughs includes the MCP, the intrinsic files, the compilers, and various utility programs. This software can be copied (from the object file), or can be copied and compiled (from the source file), just as any user program can be copied, or copied and compiled, from a user's object or source file. The accompanying chart shows the name and purpose of most of the standard software items, the source language in which the item is written, and the names of the files containing the source code and the object code.

Generation Procedures

The system tape includes dual-processor capability. To take advantage of this capability, a dual-processor machine at functional level 13 is required. If one does not wish to make use of this capability, a machine at functional level 12 will be satisfactory for the operation of the software. Functional level 13 is defined to be functional level 12 plus the following changes:

EI 30173/REE 54869/
EI 30174/REE 54954/
EI 15908-A/REE 54863/
EI 30305/REE 54955/

Compilation and Patching of Standard Software

Any of the standard software items may be compiled by first copying the source file to disk and then using a standard compilation deck for the language in which the item is written. The item may be patched in the same manner as any program may be patched. The ALGOL compiler and the DCALGOL compiler are both contained on the same source file. The compile-time options associated with MCP compilations are described under the topic of "MCP Compilations" at the end of the present discussion. In these cases, determination of the program to be compiled is made by the setting of user options on the compiler control cards. The following are typical of the compiler control cards to be used.

Item	Compiler Control Card
ESPOL or XALGOL	$ SET MERGE BCL CHECK LEVEL 2
FORTRAN, BINDER, or COBOL	$ SET MERGE CHECK LEVEL 2
ALGOL	$ SET MERGE CHECK ALGOL LEVEL 2 RESET DCALGOL (ALGOL and DCALGOL are user options)
DCALGOL	$ SET MERGE CHECK DCALGOL LEVEL 2 RESET ALGOL
ALGOLFILTER	$ SET MERGE CHECK BCL
COBOLFILTER	$ SET MERGE CHECK
LOADER	$ SET MERGE CHECK DECK
MCP*	$ SET MERGE CHECK

* Note: To compile the MCP, a <I> ESPOL STACK = 1200 card is recommended.

MAINTENANCE OF STANDARD SOFTWARE

The following are sample card decks which display the manner in which standard software items may be compiled.

Compilation of the ALGOL Compiler:

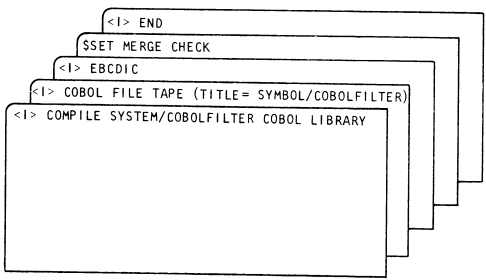
```
<I> END
$SET NEW ALGOL MERGE SEQERR
     (followed by any patch cards)
<I> EBCDIC
<I>STACK = 1000; CORE = 15000, 13000
<I>ALGOL FILE NEWTAPE (TITLE = SYMBOL/ALGOL)
<I>ALGOL FILE TAPE (TITLE = SYMBOL/ALGOL)
<I>COMPILE SYSTEM/ALGOL WITH ALGOL LIBRARY
```

Compilation of the DCALGOL Compiler:

```
<I> END
$SET MERGE CHECK DCALGOL LEVEL 2 RESET ALGOL
<I> EBCDIC
<I> ALGOL FILE TAPE (TITLE = SYMBOL/ALGOL)
<I> COMPILE SYSTEM/DCALGOL ALGOL LIBRARY
```

MAINTENANCE OF STANDARD SOFTWARE

Compilation of the COBOL Filter Program:



(NOTE: <I> represents an invalid character in card column 1.)

If several patches are to be made to a software item, the utility patch merge program (SYSTEM/PATCH) may be used. This program merges individual patches into a master patch deck. For further information, see the detailed description of the patch merge program.

BURROUGHS STANDARD SOFTWARE ITEMS

NAME	PURPOSE	SOURCE LANGUAGE	NAME OF SOURCE FILE	NAME OF OBJECT FILE	COMMENTS
ALGOL	Compiler	ALGOL	SYMBOL/ALGOL	SYSTEM/ALGOL	
ALGOL FILTER	Translator	ALGOL	SYMBOL/ALGOLFILTER	SYSTEM/ALGOLFILTER	Converts B 5500 or B 6700 Compatible ALGOL (XALGOL) to B 6700 Extended ALGOL.
ALGOLINTRINSICS	Intrinsic	ALGOL	SYMBOL/ALGOLINTRINSICS	SYSTEM/INTRINSICS	Bound into Intrinsics File
BACKUP	Utility	ALGOL	SYMBOL/BACKUP	SYSTEM/BACKUP	
BASIC	Compiler	ALGOL	SYMBOL/BASIC	SYSTEM/BASIC	
BINDER	Binding	ALGOL	SYMBOL/BINDER	SYSTEM/BINDER	Allows program units written in the same language or in different languages to be bound into a single program.
CANDE	Message Control System	DCALGOL	SYMBOL/CANDE	SYSTEM/CANDE	Compilation requires 300-word stack and 8000-word core capacity.
CARD LINE	Utility	ALGOL	SYMBOL/CARDLINE	SYSTEM/CARDLINE	
CCTABLEGEN	Table Generator	ALGOL	SYMBOL/CCTABLEGEN	SYSTEM/CCTABLEGEN	Produces tables used by MCP CONTROLCARD procedure
COBOL	Compiler	ALGOL	SYMBOL/COBOL	SYSTEM/COBOL	

BURROUGHS STANDARD SOFTWARE ITEMS (Cont)

NAME	PURPOSE	SOURCE LANGUAGE	NAME OF SOURCE FILE	NAME OF OBJECT FILE	COMMENTS
COBOL FILTER	Translator	COBOL	SYMBOL/COBOLFILTER	SYSTEM/COBOLFILTER	Converts B 5500 COBOL programs to B 6700 COBOL.
COMPARE	Utility	ALGOL	SYMBOL/COMPARE	SYSTEM/COMPARE	Performs a bit-by-bit comparison of one or more pairs of files.
DC1000	Assembler	ALGOL	SYMBOL/DC1000ASSEMBLER	SYSTEM/DC1000ASSEMBLER	
DCALGOL	Compiler	ALGOL	SYMBOL/ALGOL	SYSTEM/ALGOL	Contained in the same source file as ALGOL but is compiled with different user options set.
DCPROGEN	Utility	ALGOL	SYMBOL/DCPPROGEN	SYSTEM/DCPPROGEN	
DUMP ANALYZER	Utility	ALGOL	SYMBOL/DUMPANALYZER	SYSTEM/DUMPANALYZER	
ESPOL	Compiler	ALGOL	SYMBOL/ESPOL	SYSTEM/ESPOL	
ESPOLINTRINSICS	Intrinsics	ESPOL	SYMBOL/ESPOLINTRINSICS	SYSTEM/INTRINSICS	Bound into Intrinsics File
FORTRAN	Compiler	ALGOL	SYMBOL/FORTRAN	SYSTEM/FORTRAN	
GUARDFILE	Utility	ALGOL	SYMBOL/GUARDFILE	SYSTEM/GUARDFILE	Guardfile generation.
HARDCOPY	Utility	DCALGOL	SYMBOL/HARDCOPY	SYSTEM/HARDCOPY	SPO hardcopy.
IADMAPPER	Utility	DCALGOL	SYMBOL/IADMAPPER	SYSTEM/IADMAPPER	Mapping of INSTALLATION-ALLOCATED DISK.

BURROUGHS STANDARD SOFTWARE ITEMS (Cont)

NAME	PURPOSE	SOURCE LANGUAGE	NAME OF SOURCE FILE	NAME OF OBJECT FILE	COMMENTS
INTRINSICS	Intrinsics			SYSTEM/INTRINSICS	Bound from ALGOL INTRINSICS and ESPOLINTRINSICS.
IOINTERACT	Testing	DCALGOL	SYMBOL/IOINTERACT	SYSTEM/IOINTERACT	
LIST DIRECTORY	Utility	ALGOL	SYMBOL/LISTDIRECTORY	SYSTEM/LISTDIRECTORY	
LOADER	Utility	ESPOL	SYMBOL/LOADER	SYSTEM/LOADER	Card deck
LOGANALYZER	Accounting	ALGOL	SYMBOL/LOGANALYZER	SYSTEM/LOGANALYZER	
MAINTENANCE	Maintenance	ESPOL	SYMBOL/MAINTENANCE	SYSTEM/MAINTENANCE	Bound to MCP
MAKEUSER		ALGOL	SYMBOL/MAKEUSER	SYSTEM/MAKEUSER	
MCP		ESPOL	SYMBOL/MCP	SYSTEM/MCP	
MCSII		DCALGOL	SYMBOL/MCSII	SYSTEM/MCSII	
NDL		ALGOL	SYMBOL/NDL	SYSTEM/NDL	
PATCH MERGE	Utility	ALGOL	SYMBOL/PATCH	SYSTEM/PATCH	
PRINTCOPY	Utility	ALGOL	SYMBOL/PRINTCOPY	SYSTEM/PRINTCOPY	Prints HARDCOPY output.
REMOTE JOB ENTRY		DCALGOL	SYMBOL/RJE	SYSTEM/RJE	
RLTABLEGEN	Utility	ALGOL	SYMBOL/RLTABLEGEN	SYSTEM/RLTABLEGEN	Generates table for MCP recognition of nonstandard tape labels.
SORT	Sort	ESPOL	SYMBOL/SORT	SYSTEM/SORT	Bound to MCP
SORTSTAT	Accounting	ALGOL	SYMBOL/SORTSTAT	SYSTEM/SORTSTAT	

BURROUGHS STANDARD SOFTWARE ITEMS (Cont)

NAME	PURPOSE	SOURCE LANGUAGE	NAME OF SOURCE FILE	NAME OF OBJECT FILE	COMMENTS
SOURCE DC1000		DC1000	SYMBOL/SOURCEDC1000	NONE	
SOURCE NDL		NDL	SYMBOL/SOURCENDL	NONE	
TAPE DIR		ALGOL	SYMBOL/TAPEDIR	SYSTEM/TAPEDIR	Prints library tape directory on line printer or console.
XALGOL	Compiler	ALGOL	SYMBOL/XALGOL	SYSTEM/XALGOL	

MAINTENANCE OF STANDARD SOFTWARE

MCP COMPILATION

Compile-Time Options

A number of versions of MCP code files may be generated via ESPOL compilations of the Mark 11.4 MCP symbolic file. The form of the generated MCP is governed by the states of ten compile-time options; the combination of option states chosen results in selective compilation of portions of the symbolic file.

The ten MCP compile-time options are:

ACTIVETIME
DIAGNOSTICS
IOTIMES
LINEINFO
MTBF
OPTIMIZER
PRESENCEBITCHARGED
PROCESSORSWAPPING
READLOCK
REVERSEPAPERTAPE

These options may be set or reset via ESPOL compiler control cards merged into the MCP symbolic file at any sequence number in the interval 00000491 to 00000499. The following is an example of such a compiler control card:

```
$SET MERGE CHECK DIAGNOSTICS OPTIMIZER IOTIMES        00000499
```

The default state of each of these options is RESET, except REVERSEPAPERTAPE which has a default state of SET. All options are user options, except READLOCK and LINEINFO which are standard ESPOL compiler options.

The MCP compile-time options are discussed in the following paragraphs in the order just listed. The default state of each option is indicated in parentheses beside the option name.

ACTIVETIME (RESET)

When this option is set, the resultant MCP causes the total amount of time a stack is active to be entered in the system log at EOJ-time. If ACTIVETIME is reset, this information is not entered in the log.

DIAGNOSTICS (RESET)

When this option is set, checking and debugging code is included in the generated MCP. This option is normally reset due to the amount of additional code generated when the option is set.

MAINTENANCE OF STANDARD SOFTWARE

IOTIMES (RESET)

When this option is set, the resultant MCP allows user programs to collect certain timing statistics via the TIME intrinsic. When IOTIMES is reset, these times are not available to user programs.

LINEINFO (RESET)

When this option is set, the resultant MCP code file contains line number information which allows SYSTEM/DUMPANALYZER and PROGRAMDUMP to associate MCP symbolic line numbers with the RCW's in a stack. This option is normally reset since the size of the generated MCP code file is doubled when LINEINFO is set. (This option is a standard ESPOL compiler option.)

MTBF (RESET)

(The name of this option is an acronym for Mean Time Between Failure.) When this option is set, the resultant MCP performs collection and display of peripheral unit error statistics. The run-time system option "DIAGNOSTICS" is meaningful only for a system employing an MCP compiled with MTBF set.

OPTIMIZER (RESET)

When this option is set, the resultant MCP will contain code to handle the Disk File Optimizer. Thus, this option should be reset when generating an MCP code file intended for use at installations which do not have this hardware unit.

PRESENCEBITCHARGED (RESET)

When this option is set, the resultant MCP will cause the processor time required to handle a presence bit interrupt to be charged to the stack which was the cause of the interrupt. In the default case, this stack would not be charged with this processor time. The option is normally reset since it causes the total processor time for a job to be highly mix-dependent.

PROCESSORSWAPPING (RESET)

When this option is set, code is omitted from the resultant MCP which causes internal job priority adjustment allowing jobs with identical declared priorities to compete for a processor with equal chances of obtaining it. This option is normally reset since this action can occasionally lead directly to a "thrashing" contention situation.

READLOCK (RESET)

When this option is set, the resultant MCP contains additional debugging code for READLOCK operations. (This option is a standard ESPOL compiler option.)

MAINTENANCE OF STANDARD SOFTWARE

REVERSEPAPERTAPE (SET)

When this option is set, the resultant MCP allows tape-parity-retry action on paper tape readers. This option must be reset when compiling an MCP which is intended for use at installations using the Facit paper tape reader which cannot be reversed.

B 6700 COMPATIBLE ALGOL FILTER

The purpose of the B 6700 Compatible ALGOL Filter program is to aid in the conversion of B 5500 Compatible ALGOL or B 6700 Compatible ALGOL (XALGOL) programs into a form acceptable as input to the B 6700 Extended ALGOL Compiler. The B 6700 Compatible ALGOL Filter accepts as input BCL-coded programs written in Compatible ALGOL and deletes, generates, consolidates, modifies, and flags constructs for programmer attention. Programs to be filtered should be error-free for useful results since syntactic errors may cause improper filtering.

The B 6700 Compatible ALGOL Filter program is written in a B 6700 Extended ALGOL. The TITLE of the program source file is SYMBOL/ALGOLFILTER, and the TITLE of the corresponding object file is SYSTEM/ALGOLFILTER.

Input Files

This program has two input files: a required primary file (with the internal name CARD) and an optional secondary file (with the internal name TAPE). These files contain the Compatible ALGOL source program to be filtered.

File CARD, a punched card file unless label-equated to another device by a FILE control card, is required regardless of whether or not the file contains card images to be processed by the filter. Data in file CARD must consist of BCL-coded 10-word records; the last record in this file must contain the sequence number 99999999 in columns 73 through 80.

File TAPE, a serial disk file unless label-equated to another device by a FILE control card, is optional and must contain BCL-coded 10-word records in 150-word blocks.

Output Files

Output files which may be created by the ALGOL Filter include a printer listing, a BCL-coded or an EBCDIC-coded disk file, and a BCL-coded or an EBCDIC-coded card punch file.

The printer listing (a 17-word-record line printer file named LINE) contains (depending on the options set) unchanged card images, deleted card images, and generated card images, as well as summary information. The listing provides the following summary information (the summary information is revised to reflect the absence of files as well as their presence):

Card input was:	(file name)	Tape input was:	(file name)
BCLTAPE file created:	( )	EBCDIC NEWTAPE file created:	( )
BCL punch file was:	( )	EBCDIC punch file was:	( )

ALGOL FILTER

Card input was:	(file name)	Tape input was:	(file name)
BCL/EBCDIC output was:	( )	Last error at:	( )
No. of cards read:	( )	No. cards edited:	( )
No. of cards deleted:	( )	No. cards generated:	( )
No. of cards flagged:	( )	No. errors detected:	( )
No. of items scanned:	( )	No. cards counted per minute:	( )
Elapsed Time =	( )	Time processing:	( )

File LINE is generated by the default setting (i.e., SET) of Filter option LIST. If option LIST has been reset and option EDIT has been set, then only those card images that have been edited or generated are listed on file LINE. If option LIST has been reset and option NEWONLY has been set, then the filtered output (edited, generated, or unchanged card images) will be listed.

A serial disk file named NEWTAPE is created for output if option NEW is set. NEWTAPE contains EBCDIC-coded 14-word records in 420-word blocks, has a SAVEFACTOR of 10 days, and may be label-equated to another device via a FILE control card.

A serial disk file anmed BCLTAPE is generated for output if options NEW and BCLDECK are set. BCLTAPE contains BCL-coded 10-word records in 150-word blocks, has a SAVEFACTOR of 10 days, and may be label-equated to another device.

A card punch file named PUNCH is created for output if option PUNCH is set (and BCLDECK is reset). The file PUNCH contains EBCDIC-coded 14-word records and may be label equated to another device.

A card punch file named PUNCHB is generated for output if options BCLDECK and PUNCH are set. PUNCHB contains BCL-coded 10-word records and may be label equated to another device.

The filtered program may be directed to either a card punch file or to a disk file or to both files.

B 6700 COMPATIBLE ALGOL FILTER FILES

Purpose	Kind	File Name	Code	Record Size	Block Size	Comments
Input Card File	Card Reader	CARD	BCL	10 words	Blocked or Unblocked	Primary input (required).
Input Disk File	Disk (serial)	TAPE	BCL	10 words	150 words	Secondary input (optional). Selected when option TAPE is set.
Output Disk File	Disk (serial)	NEWTAPE	EBCDIC	14 words	420 words	Optional output. Selected with option NEW. SAVEFACTOR = 10 days.
Output Disk File	Disk (serial)	BCLTAPE	BCL	10 words	150 words	Optional output. Selected by options NEW and BCLDECK. SAVEFACTOR = 10 days.
Output Card Punch File	Card Punch	PUNCH	EBCDIC	14 words		Optional. Selection by option PUNCH.
Output Card Punch File	Card Punch	PUNCHB	BCL	10 words		Optional. Selected by options PUNCH and BCLDECK.
Output Line Printer Listing	Line Printer	LINE	NA	17 words	NA	Optional. Selected by option LIST.

ALGOL FILTER

Control Cards

Filtering is controlled by option control cards, which are used to control the options of the filter program.

Control card information is contained in the seventy-two character positions of a logical record in file CARD or TAPE. Control information cannot be split across cards or records. On all control cards column one must contain a $ character and columns seventy-three to eighty may contain a sequence number assigned to that particular card or may be blank.

OPTION ACTIONS

Option actions control the void cards and the option cards. Option actions are SET, RESET, POP, or empty.

Associated with each option is a 48-bit stack which reflects the status of the option. The current state of the option is indicated by the bit at the top of the stack. Stored information is retrieved on a last-in first-out basis. An option whose default value is RESET is initially assigned a stack filled with "0's"; an option whose default value is SET is initially assigned a stack with a "1" on top and "0's" elsewhere. Option action SET causes the option stacks (corresponding to the options in the option list) to be pushed down one bit and a "1" to be put on top of these stacks. RESET causes the stacks to be pushed down one bit and a "0" to be put on top of the stack. POP causes the option stacks following it to be popped (pushed up) one bit and the options to take the position corresponding to the popped bit now at the top of the 48-bit option stack.

If the option is empty (not specified), then the options which follow will be set and all other options will be reset (regardless of their previous setting or their default setting).

OPTIONS

The options available to the ALGOL Filter Program are listed below in alphabetical order. The default setting for each option is shown in parentheses following the option.

BCL (RESET)

When set, the BCL option causes the filter program to convert to six-bit strings all strings for which no character size has been specified. If the character size is specified, the string is left unchanged. BCL characters with no EBCDIC equivalent are changed into six-bit strings with the same internal representation as the desired character. The BCL option causes pointer expressions for which no character size has been specified to be assumed to point to six-bit characters. (The BCL option and the SIX option are equivalent and interchangeable.)

BCLDECK (RESET)

When set, the BCLDECK option causes the output card images produced by the Filter to be BCL-coded; otherwise an EBCDIC-coded file is produced.

DOUBLE (SET)

When set, the DOUBLE option allows the filtering of double statements. When reset, this option suppresses such filtering and causes such statements to be flagged.

EDIT (RESET)

When EDIT is set and LIST is reset, then only those card images which have been edited are listed on file LINE.

FIXID (RESET)

FIXID WITH<character string>(RESET)

Certain reserved words in B 6700 ALGOL are not reserved words in Compatible ALGOL and may have been used as identifiers in the program which is to be filtered. When set, FIXID causes these identifiers to be changed whenever they appear in the program by concatenating the <character string> onto the end of every identifier which is also a reserved word in B 6700 ALGOL. If no character string is specified, a "Q" will be used as a character string.

LIST (SET)

When LIST is set, all card images, old and new, whether unchanged, deleted edited, or generated will be listed on file LINE. When LIST is reset, only images with flags for programmer review will be listed.

MERGE (RESET)

When set, MERGE causes source file (file TAPE) input to be merged with patch file (file CARD) input (if TAPE is also set). When MERGE is reset, only input from file CARD is accepted.

NEW (RESET)

When set, option NEW causes the filtered card images to be written on a new source file. This file will be file NEWTAPE if option BCLDECK is reset, otherwise file BCLDECK will be created. When NEW is reset, the new source file is not created.

NEWONLY (RESET)

When set (assuming LIST is reset), causes only the new card images to be listed on output file LINE.

ALGOL FILTER

PATCH (RESET)

When PATCH is set and PUNCH is reset (assuming the source file requires little modification by the filter), a patch deck will be punched which can be merged with the disk file to create a new source file which is acceptable to the B 6700 compiler.

PUNCH (RESET)

When PUNCH is set, the filter program will create a new source file containing the filtered output on punched cards. This file (named PUNCH) contains EBCDIC-coded 14-word records unless BCLDECK is also set. When PUNCH is set and BCLDECK is also set, a new source file named PUNCHB will be created. File PUNCHB will contain BCL-coded 10-word records.

REPLACE<defined identifier>(SET)

When the REPLACE option is set, the identifier following the word REPLACE will be replaced everywhere it occurs by the text with which that identifier is associated in a preceding DEFINE statement. (REPLACE should not be used with defines that have parameters.) In the example the identifier "K" will be replaced by "MAKE" everywhere in the program.

```
DEFINE K = MAKE #;
       :
       .
$ SET NEW
       .
       :
       .
$ REPLACE K
```

REVERSE (SET)

When set, REVERSE causes read reverse statements to be filtered correctly but inefficiently. For example, if REVERSE is set, the read statement READ REVERSE (FID,2,A[*]) becomes:

```
BEGIN
  FID.DIRECTION:=1;
  READ (FID,2,A[*]);
  FID.DIRECTION:=0
END
```

When REVERSE is reset, the word REVERSE is deleted from the read statement and a message is printed telling the programmer that the direction of the "read" should be fixed manually.

SEQ <starting number>+<increment part> (RESET)

When set, the filter resequences filtered output in accordance with the parameter values of the <starting number> and +<increment part>. The

value of the starting number is assigned to the first filtered card image before the card image is output; then for the next card image the value of the starting number is increased by the increment part. An example of this option is: $ SET SEQ 1000 +200.

SIX (RESET)

When set, the SIX option sets in the same manner as the BCL option.

SORT (SET)

The SORT option alters the filtering of SORT and MERGE statements. Hivalue procedures are disallowed in B 6700 ALGOL SORT and MERGE statements. If SORT is set, the filter will generate a call on the hivalue procedure prior to and outside of the SORT or MERGE statements.

TAPE (RESET)

Setting the option TAPE causes file TAPE to be considered as a source of input. When TAPE is reset, input is accepted from file CARD only.

UNDEFINE (SET)

The UNDEFINE option allows the user to control the define-expansion facilities of the filter program. It may be necessary to filter double statements, sort statements, merge statements, and read reverse statements. In order to filter an invocation of a DEFINE the UNDEFINE option must appear as the last option after the DEFINE and before the corresponding invocation.

For example,

```
DEFINE RR = READ REVERSE (FID,2,A[*])#;
  :
  .
$ UNDEFINE
RR;
```

VOID TO SEQUENCE<sequence number><last void number>

VOID TO SEQUENCE causes all input records except $ cards (from TAPE and CARD) with sequence numbers between <sequence number> and <last void number>, inclusively, to be deleted during filtering. (The <sequence number> must appear in columns 73-80; the <last void number> may appear as not more than eight consecutive digits in columns 6-72.)

VOID (RESET)

When VOID is set, all input records, except those with a $ sign in column 1 are ignored until VOID is reset or popped into a reset state.

ALGOL FILTER

VOIDT (RESET)

When option VOIDT is set, all input records from file TAPE are ignored (except those with a $ sign in column 1) until VOIDT is reset or popped into a reset state.

WARN (SET)

If WARN is reset, warning messages will become filtering errors.

PARAMETERS

Parameters are used in conjunction with options on filter control cards to specify limits.

PAGE

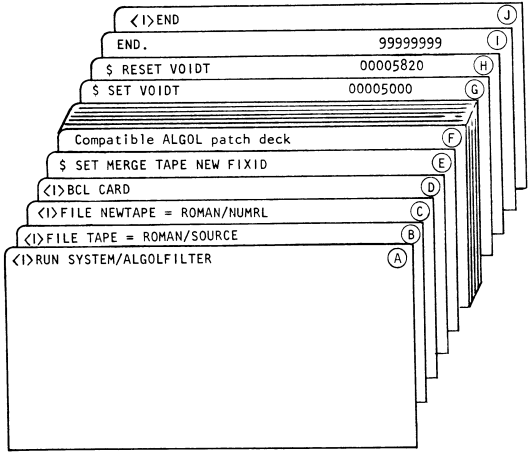
When PAGE is specified on an option card, the line printer skips to a new page before printing the listing.

<starting number> + <increment part>

The parameters are used to specify the starting sequence number and the increment part when option SEQ is set. These numbers are entered as six-digit numbers in columns 2 to 72 on any option card. The increment part is preceded by a "+" sign. If these numbers are not specified, they are zero by default. Resequencing is accomplished when option SEQ is set.

Sample Card Decks

DISK AND CARD INPUT



(A) Causes the B 6700 Compatible ALGOL Filter program to be run.

(B) Compatible ALGOL code for ROMAN/SOURCE is contained in disk file TAPE.

(C) New filtered (B 6700/ALG) code for program ROMAN/NUMRL is saved in disk source file NEWTAPE.

(D) BCL-coded cards in file CARD follow this card.

(E) Patch deck is to be merged (MERGE) with program source deck ROMAN/SOURCE to produce ROMAN/NUMRL.

(F) Patches to file ROMAN/SOURCE.

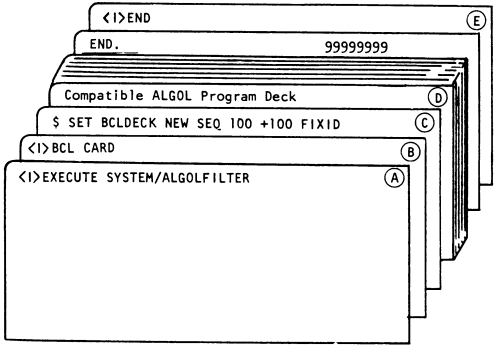
(G) Causes card images from TAPE to be ignored beginning with sequence number 00005000.

(H) Last card to be voided is 00005819.

(I) Last card in deck.

(J) End of filtering.

ALGOL FILTER
CARD INPUT



(A) Causes the B 6700 Compatible ALGOL Filter program to be executed.

(B) Indicates that BCL-coded cards follow in input file CARD.

(C) Causes an output disk file named BCLTAP (NEW option) containing BCL-coded 10-word record (BCLDECK option) to be generated. Resequences filtered output (SEQ 100 +100 option) concatenates a "Q" onto the end of identifiers which are reserved words in B 6700 ALGOL (FIXID option);

(D) Source program cards to be filtered.

(E) Denotes an end of the filter deck.

Error Messages

The filter program is designed to detect possible sources of trouble in the filtering process, whether these be due to major differences between Compatible ALGOL and B 6700 ALGOL, to syntax errors in the input, or to problems that the filter program itself is not able to handle correctly. Such trouble spots are flagged on the line printer output listing and explanatory text is included to describe a possible error. In some cases no error actually exists. However, because possible errors are flagged, the programmer must inspect his program to be sure that the filtering is correct.

ALGOL FILTER

The filter program deletes (DEL) some card images, generates (GEN) new card images, and edits (EDT) card images. Shown below are examples of filtering arranged as they would appear on an output listing.

```
<001>>     ... CHECK FILE DECLARATION.
           .
           .
         * FILE LINE 18(2,15);                          *     DEL
           .
           .
           .

>>>>>      FILE LINE (KIND =103,BUFFERS =2, MAXRECSIZE =15);00005000 GEN
```

Certain card images are edited to reflect B 6700 ALGOL constructs, e.g.,

```
CHANGETOG← FALSE;   becomes

CHANGETOG:=FALSE;                                              EDT
```

COBOL FILTER

B 6700 COBOL FILTER

The B 6700 COBOL Filter is a program used in converting B 200, B 300, B 3500, B 5000 and B 5500 COBOL source programs (coded in BCL or EBCDIC) to a form acceptable to the B 6700 COBOL Compiler. Language constructs to which the filter cannot adjust, i.e., a form acceptable to the compiler, are flagged for programmer review and modification.

The name of the source file for the B 6700 COBOL Filter is SYMBOL/COBOLFILTER; the name of the object file is SYSTEM/COBOLFILTER. The B 6700 COBOL Filter will accept programs in symbolic form from the following types of input media (see table 1).

- Punched Cards
- B 5500 COBOL Compiler "SOLT" blocked magnetic tape (with patch deck)
- BCL alpha unblocked magnetic tape (with patch deck)
- Disk (blocked at 15 records per block)

INTERNAL NAME	EXTERNAL NAME
CARDINPUT	"COBOL"
TAPEINPUT	"SOLT"
TAPE-UN-INPUT	"SOLT"
DISKINPUT	"SOLT"

Note that the filter requires a syntactically correct program as input, i.e., a B 5500 COBOL program with B 5500 syntax errors may be misinterpreted.

A new source language output is provided after filtering has been done and programmer adjustments, if any, have been made. The new output may be resequenced in increments from 1 to 9999 and directed to one of the following types of output media (see table 1).

- Punch card deck
- Magnetic tape (blocked at 15 records per block)
- Disk file (blocked at 15 records per block)

INTERNAL NAME	EXTERNAL NAME
PRINT	"FILTER"
TAPEOUTPUT	"COBOL"/"IMAGE"
PUNCHOUTPUT	UNLABELED
DISKOUTPUT	"COBOL"/"IMAGE"

Table 1. B 6700 COBOL Filter Files

Purpose	Kind	Internal File Name	Default File Title	Code	Record Size	Block Size	Comments
Input Card File	CARD READER	CARDINPUT	COBOL	BCL EBCDIC	10 words 14 words	Unblocked	Primary input of filter control cards and patches. Selected by input option CARD (if input media is not specified, CARD is assumed).
Input Disk File	DISK SERIAL	DISKINPUT	SOLT	BCL	10 words	Blocked 150 words	Secondary input. Selected by input option DISK.
Input Magnetic Tape File (blocked)	MAGNETIC TAPE	TAPEINPUT	SOLT	BCL	56 words	Blocked 448 words	Secondary input. Selected by input option TAPE. A tape labeled SOLT is expected.
Input Magnetic Tape File (unblocked)	MAGNETIC TAPE	TAPE-UN-INPUT	SOLT	BCL Alpha	10 words	Unblocked	Secondary input. Selected by input option TAPE. A tape labeled SOLT is expected.
Library Disk File for Input	DISK RANDOM	CASTA	CASTA/ LIBRARY	BCL	60 words	488 words	Random access. Selected by input option CASTA DISK.
Library Tape File for Input	MAGNETIC TAPE	CASTATAPE	CASTA/ LIBRARY	BCL	56 words	448 words	Selected by input option CASTA TAPE.

Table 1. B 6700 COBOL Filter Files (Cont)

Purpose	Kind	Internal File Name	Default File Title	Code	Record Size	Block Size	Comments
Output Disk File	DISK SERIAL	DISKOUTPUT	COBOL/ IMAGE	BCL	10 words	150 words	Filtered source code output. Sequential access. SAVEFACTOR = 90 days. Selected by output option DISKOUT.
Output Listing	LINE PRINTER	PRINT	FILTER	BCL	15 words	Unblocked	Option LISTALL gives all images. Option SPEC gives images requiring programmer review. Default gives source images with errors and new images.
Output Magnetic Tape File	MAGNETIC TAPE	TAPEOUTPUT	COBOL/ IMAGE	BCL	15 words	225 words	Filtered source code output. Selected by output option COBSYM. SAVEFACTOR = 90 days
Output Punch Deck	CARD PUNCH	PUNCHOUTPUT	PUNCH	BCL	10 words	10 words	Provides a new source deck containing filtered images. Selected by output option PUNCH. Default setting if no output options selected.

Warning messages and indicative numbers that are generally sufficient to determine the action taken or the action necessary to be taken manually are provided. Warning messages of a specific nature are printed in line; those of a general type are, on the other hand, printed to the right of the new generated output. Table 2 lists the disposition of B 5500 COBOL constructs after filtering has been accomplished. Those constructs requiring manual change are identified by the phrase "flag for manual change".

Table 2. Disposition of B 5500 COBOL Constructs

Element	Action
General	
Punctuation	
%	Delete where not acceptable.
;	Delete.
Connective OR	Delete.
Connective AND	Delete.
←	Delete. If any comment follows, create an "*" comment card.
Figurative Constants	
END	Flag for manual change.
New Reserved Words	
Used as Data-Names	Add trailing Q.
B 5500 reserved word abbreviations	Change to COBOL-65 abbreviation or reserved word.
IDENTIFICATION DIVISION	
All Entries	Treated as remarks.
ENVIRONMENT DIVISION	
CONFIGURATION SECTION	
SOURCE-COMPUTER	
<computer-name>	Change to B-6700.
COPY	Include images from CASTA library.
OBJECT-COMPUTER	
<computer-name>	Change to B-6700.
COPY	Include images from CASTA library.

COBOL FILTER

Table 2. Disposition of B 5500 COBOL Constructs (Cont)

Element	Action
DISK SIZE Clause	Change MOD or MODS to modules.
SPECIAL-NAMES	
<hardware-mnemonic-name>	Delete but remember for substitution in WRITE (to channel number).
COPY	Include images from CASTA library.
INPUT-OUTPUT SECTION	
FILE-CONTROL	
COPY	Include images from CASTA library.
SELECT <file-name-1> renaming <file-name-2>	Remember <file-name-1> and copy description of <file-name-2> in DATA DIVISION.
ASSIGN TO <hardware-name-series>	Change names to B 6700 names, flag datacom file for manual change.
ACCESS MODE	Copy from MD in FILE SECTION.
ACTUAL KEY	Copy from MD in FILE SECTION.
FILE LIMITS	Copy from MD in FILE SECTION.
I-O-CONTROL	
COPY	Include images from CASTA library.
MULTIPLE FILE TAPE CONTAINS	Treated as comments.
VALUE OF MFID	Delete and substitute as leftmost <file-ID> in FILE SECTION.
DATA DIVISION	
FILE SECTION	
File Description Entries	
FD FILE-NAME	If subject of ENVIRONMENT DIVISION renaming, add description following complete FD entries.
COPY	Include images from CASTA library.
FILE CONTAINS	Delete and multiply disk areas by records per area and enter product in number of units specification in SELECT statement.

Table 2. Disposition of B 5500 COBOL Constructs (Cont)

Element	Action
DATA DIVISION	
File Description Entries (Cont)	
Record Description Entries	
CLASS IS	Delete, use for PICTURE except when used with variable size item.
COPY <data-name> from library	Text copied. Include images from CASTA library.
Editing Clauses	
Zero Suppress (ZS)	Delete, use for PICTURE.
Check Protect (CP)	Delete, use for PICTURE.
Float Dollar Sign (FS)	Delete, use for PICTURE.
FILLER	If group item at 01 level, add trailing A-P, R-Z, 1-9 depending on the number of occurrences. If group item at other than 01 level, add trailing Q.
JUSTIFIED (JS) LEFT	Flag for manual change.
PICTURE (PC) IS	Verify editing symbols and, if not present, insert.
Point Location	Delete, use for PICTURE.
SIGN IS <data-name>	Flag for manual change.
SIGNED or SN	Delete and use "S" for PICTURE.
SIZE IS <integer-2> (Fixed)	Delete, use for PICTURE.
USAGE IS COMPUTATIONAL (CMP)	Flag for verification, add SYNCHRONIZED RIGHT clause.
Others	
MD Entries	Change to FD, flag for verification and add information to FILE-CONTROL SECTION.

COBOL FILTER

Table 2. Disposition of B 5500 COBOL Constructs (Cont)

Element	Action
Symbolic key	Delete.
File names shown in SELECT...RENAMING Statements in ENVIRONMENT DIVISION	Add to end of FILE SECTION.
Same	Flag for manual change.
ASSIGN TO MEMORY	Flag for manual change.
REDEFINES successive redefining	Change from chaining method to successive redefining of the original entry.
PROCEDURE DIVISION	
Conditional Statement	
UNEQUAL TO (≠)	Change to "NOT =".
EQUALS	Change to "=".
EXCEEDS	Change to ">".
≤	Change to "NOT >".
≥	Change to "NOT <".
ACCEPT FROM <option>	Flag for manual change.
ASSIGN	Flag for manual change.
INCLUDE	Include images from CASTA library.
MOVE <file-name> (Datacom)	Flag for manual change.
PERFORM WITH	Convert to CALL SYSTEM WITH <data-name>.
PERFORM PROGRAM-ID	Convert to "RUN <program-name>"
READ Implicit AT END option	Flag for manual change.
Datacom	Flag for manual change.

Table 2. Disposition of B 5500 COBOL Constructs (Cont)

Element	Action
UNTIL	Flag for manual change.
WRITE <mnemonic-name> for channel	Change to channel number.
ON option	Flag for manual change.
Datacom	Flag for manual change.

Filter Control Card and BCL Switch Card

Following the "<I> EXECUTE COBOL/FILTER" and "<I> DATA COBOL" cards and preceding the program cards and "<I> END" card, a "$" control card may be present. The information on the control card is in a free-field format. If no "$" card is present, "$ CARD PUNCH" is assumed. The control card syntax and semantics are as follows:

Syntax:

<control card> ::= $ <input part> <output part> <list part>
<seq part> <new-ID part> <casta part> <internal mode part>

<input part> ::= <empty> / CARD / TAPE / DISK

<output part> ::= <empty> / PUNCH / COBSYM/ DISKOUT/ DEBUN

<list part> ::= <empty> / LISTALL / SPEC

<seq part> ::= <empty> / NOSEQ / SEQ <integer>

<new-ID part> ::= <empty> / NEWID <character string>

<casta part> ::= <empty> / CASTA TAPE / CASTA DISK

<internal mode part> ::= <empty> / BCL

Semantics:

The $ character must be in column 1.

COBOL FILTER

<input part>

The CARD/TAPE/DISK choice designates the input media. When the tape option is specified, a tape labeled "SOLT" is expected. Patch cards are allowed following the dollar sign control card. The "SOLT" tape can be either unblocked or blocked with five images per block (normal output from a B 5500 COBOL compilation). If this part is empty, "CARD" is assumed. When disk is specified, the disk file must be blocked 15 records per block.

<output part>

The PUNCH/COBSYM/DISKOUT choice designates the output media for the filtered images. "COBSYM" designates an output tape labeled "COBOL/IMAGE" which consists of blocked images (15 records/block) recorded in standard mode. If this part is <empty>, "PUNCH" is assumed.

DISKOUT creates a disk file, internal BCL blocked 15 records per block. "DEBUN" may be specified if no symbolic output is desired.

<list part>

The listing part specifies the following:

If <empty>, source images containing errors and all new images are listed.

If "LISTALL", all source and new images are listed.

If "SPEC", only the images that require programmer inspection are listed.

<seq part>

The sequencing option designates the following:

If <empty>, output images are sequenced by 100.

If "NOSEQ", output images retain the input sequence number except when statement consolidation occurs; in this case the last number is used.

If "SEQ <integer>", output images are sequenced according to the given integer. <integer> may be one through four digits.

Resequencing of the output file is recommended. If "NOSEQ" is specified, it is very likely to produce duplicate sequence numbers and/or numbers out of sequence due to the rearrangement and overflow of input images.

<new-ID part>

The new-ID part designates the following:

If <empty>, the output messages will retain the original ID (COL 73-80).

If "NEWID <character string>", the ID of output images will be replaced by <character string>. The character string may be one to eight characters long.

<casta part>

The casta part designates the following:

If <empty> or "CASTA DISK", library references will be copied from disk.

If "CASTA TAPE", library references will be copied from tape.

<internal mode part>

The internal mode part designates the following:

If <empty> all "DISPLAY" data descriptions will be left unchanged.

If "BCL", all "DISPLAY" data descriptions will be described as six-bit BCL strings (USAGE is DISPLAY-1).

BCL switch cards may be included in the patch deck to set or reset the BCL internal mode option. The format of these cards is as follows:

<seq NR> $ SET BCL/RESET BCL

Where the $ is in column 7.

Error and Advisory Messages

For diagnostic purposes, the output listing may include error and advisory messages, output image code, including sequence numbers, and a concatenated "Q". (See table 3.)

At the end of the listing, the Filter program gives the total count of flagged images for each diagnostic code number, the total number of output images flagged, and the total number of input images scanned.

The filter program codes each input image into one of six filter diagnostic categories. On the line with the filtered image on the output listing, the numeric portion of the diagnostic code is printed in the left margin and the category title is printed in the right margin. The diagnostic categories are listed below with explanations and examples. In various categories the Filter concatenates a "Q" to the end of questionable constructs to aid the programmers.

XXXXX(no title).

Table 3. Message Formats

Advisory Message for Manual Change

Diagnostic Code	Old Seq. No. / New Seq. No.	Count of Flagged Images (by Code)	Program ID	Diagnostic Messages
XXXXX	003900 WORKING-STORAGE SECTION		CBL67	(empty)
1XXXX				FILTER RELOCATED
INCOMPLETE SPECIFICATION				
	2010 77 LINE-CNT PIC 99 VALUE 50 COMP-1		CB56	
X2XXX	004000 77 LINE-CNT PICQ99 VALUE 50 COMP-1Q XXX0007XXX		CB67	NEED MANUAL CHANGE
	080 SOURCE-COMPUTER.B-5500.		CB55	
XX3XX	001200 SOURCE-COMPUTER.B-6500.		CB67	FILTER
XXX4X				FILTER CHANGE
	5620 WRITE HEADINGS AFTER ADVANCING 1 LINE.		CB59	
XXXX5	008800 WRITE HEADINGS AFTER ADVANCING 1 LINEQ.		CB67	VERIFY ENTRY/ CHANGE

COBOL FILTER

Each source code image filtered by the filter program is flagged with five X's in the left margin of the printer listing. If no changes are made by the filter except for a new sequence number and/or a new ID in columns 73-80, then only X's will appear in the left margin to show that the image has been filtered.

Example:

```
        2000 WORKING-STORAGE SECTION.          CBOL5500
XXXXX   003900 WORKING-STORAGE SECTION.        CBOL6700
```

The messages are as follows:

1XXXX FILTER RELOCATED

The input image has been relocated due to copying or disk file specification.

X2XXX NEEDS MANUAL CHANGE

A portion of the input image is not acceptable to the B 6700 COBOL compiler, and the filter program is unable to accomplish a suitable correction. A programmer must review and modify this image. An advisory message is provided on a separate line just above the old image to assist the programmer in making the manual change. (A list of these advisory messages follow these paragraphs.)

```
          2010 77 LINE-CNT PIC 99 VALUE 50 COMP-1.
X2XX5*****004000 77 LINE-CNT PICQ 99 VALUE 50 COMP-1Q.  ***0007***
                                                        NEED MANUAL
                                                        CHANGE
```

XX3XX FILTER

A portion of the input image contained a construct that is not acceptable to the B 6700 COBOL compiler, but the meaning is clear and the filter program has made an adjustment to the contents so that the output image is acceptable. Examples:

```
        080  SOURCE-COMPUTER.  B-5500.
XX3XX 001200  SOURCE-COMPUTER.  B-6500.          FILTER

          162  02 FILLER SZ 4.
XX3XX  002800  02 FILTER PICTURE X(4).           FILTER
```

XXX4X FILTER CHANGE

A more extensive change to the contents of the input source image has been made. Usually, this code indicates that the entire source image has been replaced by a blank image.

COBOL FILTER

```
XXXX5 VERIFY ENTRY/CHANGE
```

The entry itself requires verification for B 6700 use or the filter program has changed the source image in an established and usually acceptable manner. However, the programmer should verify that the modification is acceptable.

For example:

```
        5620  WRITE HEADINGS AFTER ADVANCING 1 LINE.
XXXX5  008800  WRITE HEADINGS AFTER ADVANCING 1 LINEQ. **********VERIFY
                                                        ENTRY/CHANGE
```

ADVISORY MESSAGES FOR MANUAL CHANGES

The filter program generates various advisory messages to aid the programmer in making manual changes. The message appears on the output listing beginning at column 1 on a separate line preceding the item to be changed. Listed below in alphabetical order are these messages with comments on each.

ATTEMPTING TO COPY ITSELF

This message indicates that while performing a copy, the program encountered the image that initiated the "copy." The program continues.

CHANNEL ERROR

This message indicates that the special names association specifies a line printer channel greater than 11. The program continues.

COPY OF COPY LIMIT EXCEEDED

This message indicates that copies were nested over 20 deep. The program continues.

COPY TABLE EXCEEDED

This message indicates that more than 198 copy identifiers have been encountered. The program continues.

END OF JOB ERROR

This message indicates that the last image of the source input did not match END-OF-JOB. The program continues.

INCOMPLETE SPECIFICATION

This message indicates that the preceding source image did not contain sufficient information for an elementary item. The program continues.

INPUT READ ERROR

This message indicates that an irrecoverable read error occurred. Processing continues.

INVALID KEY or AT END

The INVALID KEY or AT END clause was executed during a "read" from library. The program continues.

OBJECT OF CASTA COPY NOT FOUND

This message indicates that the identifier for a library copy was not found in the library directory. The program continues.

OBJECT OF COPY NOT FOUND

This message indicates that the information to be copied was not found. The program continues.

OBJECT OF RENAMING NOT FOUND

This message indicates that the file to be renamed was not found. The program continues.

OUT OF WORK DISK

This message indicates that the work disk file limit was exceeded. The program terminates.

PICTURE STRING EXCEEDS 30 CHARACTERS

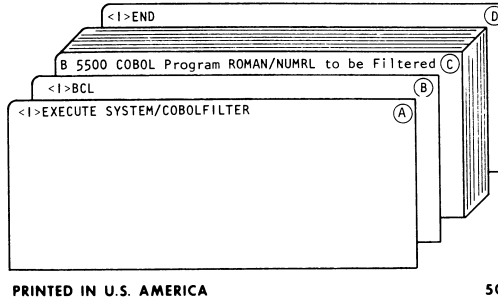
This message indicates that the picture string shown in the source image contains more than 30 characters. The program continues.

SELECT RENAMING LIMIT EXCEEDED

This message indicates that too many files have been selected with the renaming option. The additional file-names are not entered into the table, and their descriptions are not added to the filtered program. The filter program continues.

Sample Program Decks

(card input - no filter control cards)



COBOL FILTER

(A) The B 6700 COBOL Filter program will be executed to filter B 5500 COBOL Program ROMAN/NUMRL.

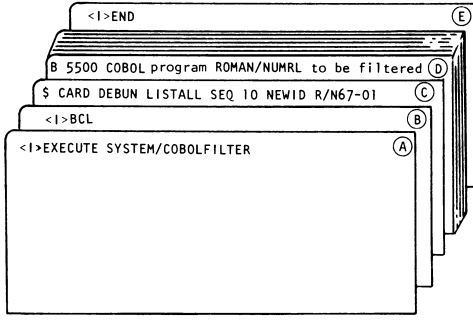
(B) The program deck being filtered is coded in BCL.

(C) The following cards comprise B 5500 COBOL program ROMAN/NUMRL which is to be filtered.

(D) End of filtering.

Note: In the absence of filter control cards, input is assumed to be via cards in a file named CARD and the filtered output will be to a card punch file named PUNCHOUTPUT since filter control card PUNCH is assumed.

(card input)



(A) The B 6700 COBOL Filter Program will be executed.

(B) Indicates that the following cards are encoded in BCL.

(C) Filter control card options indicate: input by card file (CARD); there will be no output symbolic source language file (DEBUN); the output print file will list all source images and new images (LISTALL); the output images will be assigned new sequence numbers starting with 10 and incremented by 10 (SEQ 10); each filtered output image will be identified in columns 73-80 by R/N67-01 (NEWID R/N67-01).

(D) The following cards comprise the B 5500 COBOL program ROMAN/NUMRL which is to be filtered.

(E) End of filtering.

INTERPROGRAM COMMUNICATION

General

Interprogram communication means communication between two or more tasks. A task is any procedure, program, or process which has its own stack. Because a task has a separate stack, one or more tasks may run in parallel with the program (another task) which initiated them. Tasks may communicate with each other even if they are written in different languages or even if they are compiled at different times. Communication between tasks is made possible in three ways: by use of task attributes, which allow characteristics of tasks to be referenced; by the use of events and interrupts, which allow happenings in one task to affect the execution of other tasks; and by the use of locks and readlocks, which prevent common resources (such as an array) from being used by more than one task at a time.

At the present time, task initiation is allowed by B 6700 Extended ALGOL and B 6700 COBOL.

Tasks

There are three types of tasks: coroutines, processes, and independent processes.

a. Coroutines are synchronous, dependent tasks; that is, control is passed back and forth between the coroutine and the initiating task (synchronous operation), but the coroutine must terminate before the initiating task may terminate (dependent operation).

b. Processes are asynchronous, dependent tasks; that is, the process and the initiating task run in parallel (asynchronous operation), but the process must terminate before the initiating task may terminate (dependent operation).

c. Independent processes are asynchronous, independent tasks: that is, the independent process runs in parallel with the initiating task (asynchronous operation); however, the independent process may terminate either before or after the initiating task terminates (independent operation).

A task can be initiated with a CALL, PROCESS, or RUN statement (in COBOL, the EXECUTE verb is synonomous with RUN). A task can be terminated in one of three ways:

a. By exiting from its outer block.

b. By terminating abnormally (e.g., divide by zero, operator DSed).

c. By another task setting the value of the STATUS task attribute for the present task to minus one.

COROUTINES

A coroutine is a synchronous, dependent task, which acts as an interactive task to another task. Coroutines show a primary-secondary relationship; the task which initiates a coroutine may be thought of as a primary task, the initiated coroutine may be thought of as a secondary task. Secondary coroutines may in turn initiate their own coroutines, so that a particular coroutine may at once be secondary to the coroutine which initiated it and primary to the coroutine which it initiated.

A coroutine is declared as an untyped procedure (ALGOL), and parameters may be passed either by name or by value. A coroutine is initiated by a CALL statement in the initiating task. The <task part> (ALGOL), or <control-point identifier> (COBOL) in the CALL statement, causes a unique identifier to be associated with the coroutine, so that the task attributes of the coroutine may be referenced with the aid of this identifier.

Programmatic control can be passed back and forth between an initiating task and its coroutine, by means of CONTINUE statements in ALGOL or CONTINUE (for primary tasks) and EXIT PROGRAM (RETURN HERE) statements (for secondary tasks) in COBOL.

The innermost block in which the coroutine, task identifier, or any call by name parameter is declared is known as the critical block: the coroutine must be terminated before this critical block is exited; otherwise, an error condition will exist and the entire job will be terminated.

PROCESSES

A process is an asynchronous, dependent task, which runs in parallel with the initiating task. Processes may in turn initiate other processes, all of which will run in parallel with the initiating task. A process is declared as an untyped procedure (ALGOL), and parameters may be passed either by name or by value. The process is initiated by a PROCESS statement in the initiating task. The <task part> (ALGOL) or <control-point identifier> (COBOL) in the PROCESS statement causes a unique identifier to be associated with the process, so that the task attributes of the process may be referenced with the aid of this identifier. The innermost block in which the process, task identifier, or any call-by-name parameter is declared is known as the critical block. The process must be terminated before its corresponding critical block is exited; otherwise, an error condition will exist and the entire job will be terminated.

INDEPENDENT PROCESSES

An independent process is an asynchronous, independent task, which runs independently of the initiating task. An independent process may or may not terminate before its critical block is exited. It may be written in a different source language than the initiating task. An

independent process must be compiled separately, and must not reference variables declared globally in the initiating task. All parameters for the process must be call-by-value. Within the initiating task, the independent procedure must be declared as an external procedure.

An independent process is initiated by a RUN statement (for ALGOL only) or an EXECUTE or RUN statement (for COBOL only) in the initiating task. The <task part> (ALGOL) or <control-point identifier> (COBOL) in the RUN statement causes a unique identifier to be associated with the independent process, so that its task attributes may be referenced with the aid of this identifier.

SEPARATELY COMPILED PROGRAMS

Programs which are to be run as independent processes must be compiled separately and at lexicographical level 2, since there is no critical block requirement for independent processes and coroutines may be compiled separately or with the initiating program. If compiled separately, it must be compiled at lexicographical level 2 or compiled at a level higher than 2 and bound before being called or processed.

Initiation in ALGOL

An ALGOL program may initiate a procedure as a task in the following manner:

a. If the task is to be run as an independent process, it must not reference global variables in the initiating program, i.e., separately compiled at lexicographical level 2. In addition, all parameters for the task must be call-by-value parameters.

b. Within the initiating program, a separately compiled program must be declared as an external procedure.

c. A task identifier must be declared in the initiating program.

d. The task attribute NAME for the declared task identifier must be assigned the actual title of the code file for the separately compiled program. The title to be assigned must be in the form of a string which is enclosed in quotes and is terminated with a period.

e. The procedure is invoked as a task with the appropriate CALL, PROCESS, or RUN statement and the declared task identifier is associated with the task.

Initiation in COBOL

A COBOL program may initiate externally declared procedures as a task in the following manner:

a. The task must be separately compiled.

b. Within the initiating program, the section name to be executed as an external procedure must be defined in the DECLARATIVES portion of the PROCEDURE DIVISION.

c. The title of the code file must be associated with the section name in the DECLARATIVES either by a mnemonic-name in the SPECIAL-NAMES paragraph or by moving the title into the task prior to task initiation. In either case, the title must be a string which is enclosed in quotes and terminated with a period.

TASK ATTRIBUTES

Task attributes allow certain characteristics of tasks to be referenced; thus allowing one task to monitor and control the execution of another task. The value of an attribute can be used to determine the current state of a particular task, and setting this value will, in some cases, change the execution of the associated task. Values and meanings of the task attributes are listed later in this section.

ALGOL Implementation

In order to use task attributes in ALGOL it is necessary to declare a task identifier, and to associate this identifier with a task when the task is invoked. Once this association is made, any attribute of the associated task can be referenced by the construct: <task identifier>. <task attribute>. The reserved word MYSELF is a compiler-supplied task identifier which is associated with any process which uses it. Thus, any attribute of a particular task may be referenced within that task as: MYSELF. <task attribute>.

COBOL Implementation

In order to use task attributes in COBOL it is necessary to declare a control-point identifier in the WORKING-STORAGE section with a USAGE IS CONTROL-POINT (CP) clause. The control-point identifier may be associated with a task in three ways:

a. Between a mnemonic name for a procedure and its code file title in the SPECIAL-NAMES paragraph,

b. Between a mnemonic name and a section name in the DECLARATIVES portion of the PROCEDURE DIVISION, using a USE statement.

c. Between the section name and the control-point identifier in the statement which initiates the task.

Once this association is made, any attribute of the associated task can be referenced by the construct: <control-point identifier> (<task attribute>).

The reserved word MYSELF is a compiler-supplied control-point identifier which is associated with any process which uses it. Thus, any attribute of a particular task may be referenced within that task as MYSELF (<task attribute>).

TASK ATTRIBUTES BY NUMBER

NUMBER	NAME	TYPE	CLASS*
0	NAME	Pointer	0
1	STACKNO	Real	1
2	COREESTIMATE	Real	0
3	DECLAREDPRIORITY	Real	0
4	MAXPROCTIME	Real	0
5	MAXIOTIME	Real	0
6	TARGETTIME	Real	0
7	STACKSIZE	Real	0
8	USERCODE	Pointer	0
9	TASKVALUE	Real	2
10	HISTORY	Real	1
11	TYPE	Real	1
12	STATUS	Real	2
13	PROCESSTIME	Real	1
14	PROCESSIOTIME	Real	1
15	ELAPSEDTIME	Real	1
16	EXCEPTIONTASK	Task	2
17	LOCKED	Boolean	2
18	STOPPOINT	Real	1
19	PARTNER	Task	2
20	INITIATOR	Real	0
21	EXCEPTIONEVENT	Event	4
22	OPTION	Real	2
23	VALIDITYBITS	Boolean	2
24	FILECARDS	Pointer	3
25	TASKATTERR	Real	1
26	PROCESSTYPE	Real	1
28	RESTART	Real	2
29	BACKUPPREFIX	Pointer	0
30	STACKHISTORY	Pointer	1
31	SUBSPACES	Real	2
32	TASKFILE	File	--
33	DECKGROUPNO	Integer	1
34	CLASS	Integer	--
35	COMPILETIME	Integer	--
38	ORGUNIT	Integer	0

TASK ATTRIBUTES BY NUMBER (Cont)

NUMBER	NAME	TYPE	CLASS*
39	MAXCARDS	Real	0
40	MAXLINES	Real	0
41	JOBNUMBER	Integer	0
42	CHARGECODE	Pointer	0

*The indicated attribute classes are defined as follows: 0, read or write, but the value at process initiation is saved by the system for use throughout process execution. 1, read only. 2, read or write. 3, write only. 4, legal to SET, RESET, CAUSE, WAIT, CAUSEANDRESET, and WAITANDRESET.

BACKUPPREFIX (29)

Pointer. This attribute allows a prefix other than the default "BD" to be used as the multi-file ID of the task's backup files. Any valid file TITLE may be specified as this prefix, and this attribute has effect only if bit 6 of the task's option word is on. The construct is useful when a process other than AUTOPRINT is to print the backup files thus preventing the automatic printing and removal of the files.

CHARGECODE (42)

This attribute is a pointer that carries what charge information the individual site management desires. The MCP does not use the CHARGECODE information, it merely logs the information. If a site operations management desires to enforce charging conventions, UDATASTRUCTURE provides a means to enforce site discipline.

CLASS (34)

An integer telling which QUEUE this task is associated with.

COMPILETYPE (35)

Is an integer identifying what type of compilation is desired.

1. COMPILE and GO
2. COMPILE for SYNTAX
3. COMPILE to LIBRARY
4. COMPILE to LIBRARY and GO

DECKGROUPNO (33)

An integer is assigned to each data deck within a WFL JOB deck. The DECKGROUPNO supplies the information of between which two data decks of the JOB the actual COMPILE, EXECUTE or PROCESS verb for this task is found. Since a task may only access a data deck occurring after the occurrence of verb invoking the task, this attribute supplies information on which data decks are appropriate for the task to ask. Set by MCP/WFL.

DECLAREDPRIORITY (3)

Real. Priority used for scheduling by system. May be read or set at any time; however, the value at process initiation is saved by the system throughout process execution. Values range from 0 to 99. Default value = 50. May be initialized by PRIORITY Control Statement. Set by ALGOL assignment statement (TASK.DECLAREDPRIORITY := <real number>;); set by COBOL SET statement (SET TASK (DECLAREDPRIORITY) TO <real number>).

ELAPSEDTIME (15)

Real. The elapsed processor time. This is the total time elapsed since actual process initiation. Read-only attribute; may be read at any time. Values are expressed in microseconds (increments of 2.4 microseconds). For example, a 1 indicates 2.4 microseconds, a 2 indicates 4.8 microseconds, etc.

EXCEPTIONEVENT (21)

Event. The EXCEPTIONEVENT for a task is caused whenever any process having that task as its EXCEPTIONTASK undergoes a change in status. The user task may SET, RESET, CAUSE, or WAIT upon EXCEPTIONEVENT. The composite EVENT functions of CAUSEANDRESET, WAITANDRESET, etc., are also allowable. Values - "Happened" or "Not Happened". (The first word (of two words) placed in the stack for the event declaration contains a "Happened" bit (bit 1). This bit is set when the event is caused; otherwise, this bit is a 0.) Applicable ALGOL intrinsics: AVAILABLE, PROCURE, LIBERATE, HAPPENED, SET, RESET, CAUSEANDRESET, WAITANDRESET, WAIT. Applicable COBOL statements: CAUSE, CLEAR, IF, WAIT, LOCK, UNLOCK, ATTACH, DETACH, ALLOW, DISALLOW, WAIT.

EXCEPTIONTASK (16)

Task. Specifies the process that is to be notified (by causing EXCEPTIONEVENT for the task associated with the process) when a change occurs in the value of the attribute STATUS of the present task. May be read or set at any time. (See also EXCEPTIONEVENT.)

IPC

FILECARDS (24)

Pointer. Allows initiating task to label-equate files of the task which is to be initiated. Write-only attribute. Must be specified before the file is opened. FILECARDS is assigned a string of one or more card images of FILE Control Statements. The card images must not be preceded by, or contain, an illegal character. The last FILE Control Statement must be followed by a null character (4'00''). In ALGOL, the string may be assigned to FILECARDS using a pointer expression. In COBOL, the string may be assigned as follows:

```
SET TSK(FILECARDS) TO A-NAME.
```

Where A-NAME is an array containing the FILE Control Statements followed by an element containing Lower Bounds (i.e. 4''00'' 's).

HISTORY (10)

Real. Specifies the reason for which a process was terminated. Read-only attribute. The contents of this word are primarily used to indicate the reason why the stack is DSed or STed. Values and meanings:

BITS	NAME & MEANING
7:8	HISTYPEF. 0 = NORMAL 1 = DUMPING 2 = QTED 3 = STED 4 = DSED 5 = NORMALEOT
15:8	HISCAUSEF. 1 = OPERATORCAUSE 2 = PROGRAMCAUSE 3 = RESOURCECAUSE 4 = FAULTCAUSE 5 = SYSTEMCAUSE 6 = DCERR 7 = IOERR 8 = SOFTIOERR 9 = NEWIOERR 10 = UNIMPLEMENTED 11 = UNSPECIFIEDCAUSE
23:8	HISREASONF. IF DSED FOR ''FAULTCAUSE'' THEN: 1 = DIVIDEBYZEROV. 2 = EXPOVERFLOWV. 3 = EXPUNDERFLOWV. 4 = INVALIDINDEXV.

BITS	NAME & MEANING
	5 = INTEGEROVERFLOWV.
	6 = INACTIVEOV.
	7 = MEMORYPROTECTV.
	8 = INVALIDOPV.
	9 = LOOPV.
	10 = MEMORYPARITYV.
	11 = SCANPARITYV.
	12 = INVALIDADDRESSV.
	13 = STACKOVERFLOWV.
	14 = STRINGPROTECTV.
	15 = PROGRAMMEDOPV.
	16 = BOTTOMOFSTACKV.
	17 = SEQUENCEERRORV.
	18 = INVALIDPCW.
	19 = STACKUNDERFLOWV.
	20 = ZAPPED.
	40 = DISKPARITYV.
	IF DSED FOR "RESOURCECAUSE" THEN:
	0 = PROCESSEXCEEDV.
	1 = IOEXCEEDV.
	2 = STACKEXCEEDV.
	3 = PRINTEXCEEDV.
	4 = PUNCHEXCEEDV.
	5 = CARDREADEXCEEDV.
	6 = MEMORYEXCEEDV.
	IF DSED FOR "OPERATORCAUSE" THEN:
	0 = RSVPV.
	1 = CLEARUNITV.
	2 = JUSTDSEDV.
	IF DSED FOR "PROGRAMCAUSE" THEN:
	0 = MISSINGCODEFILENAMEV.
	1 = MISSINGCODEFILEV.
	3 = INACTIVETASKV.
	4 = NONEXTERNALRUNV.
	18 = DEATHINFAMILY.
	19 = CRITICALBLOCKV.
	20 = BADGOTOV.
	26 = NOTEXECUTABLEV.
	27 = UNMATCHEDPARAMSV.
	28 = INVCOMPILERV.
	29 = SECURITYERRORV.
	32 = BADRESIZEDEALLOCV.
	37 = MISSINGINTRINSICV.
	38 = INCOMPATIBLELEVELV.
	39 = INFANTICIDE.
	40 = NOTBOUND.
	41 = ILLEGALOWNARRAYV.

IPC

BITS	NAME & MEANING
42	= DIMSIZERRORV.
43	= UPLEVELATTACHV.
44	= ILLEGALSWAPV.
45	= SWAPDOESNTALLOWV.
	IF DSED FOR "SYSTEMCAUSE" THEN:
1	= NOMEMV.
2	= PARITYONPBITV.
3	= ARRAYTOOLARGEV.
	IF STED FOR "SYSTEMCAUSE" THEN:
1	= WSSIZEEXCEEDV.
	IF DSED FOR "UNIMPLEMENTED" THEN:
1	= DYNAMICOWNARRAYV.

INITIATOR (20)

Real. This attribute, used for data communications only, contains the number of the station from which the task was initiated. The attribute may be read or written, but the value at process initiation is saved by the system for use throughout process execution. Values and meanings: Device #.

May be set by STATION Control Statement.

JOBNUM (41)

Integer. This attribute reflects the job number for a queue job. JOBNUM can be read at anytime but can only be conditionally set.

LOCKED (17)

Boolean. This attribute may be set or tested for specific purposes desired by the programmer. May be read or written at any time. Values = TRUE or FALSE. If an attempt is made to set LOCKED to TRUE when it is already TRUE, the process attempting to set LOCKED to TRUE will be suspended until LOCKED is set to FALSE (by some other process). Then, the process attempting to set LOCKED to TRUE will be resumed and will set LOCKED to TRUE again.

NOTE

LOCKED must be tested programmatically. Locking functions which are tested automatically by the MCP are described under "Locks and Readlocks".

MAXCARDS (39)

Real. This attribute provides the limit to the amount of cards to be punched by the task.

MAXIOTIME (5)

Real. Specifies the maximum amount of I/O time that may be used by a process. May be read or set at any time; however, the value at process initiation is saved by the system throughout process execution. Values are expressed in seconds. The specified time can be a fractional amount of seconds. May be initialized by IOTIME Control Statement.

MAXLINES (40)

Real. This attribute provides the limit to the amount of lines to be printed by the task.

MAXPROCTIME (4)

Real. Specifies the maximum amount of processor time that may be used by a process. May be read or set at any time; however, the value at process initiation is saved by the system throughout process execution. Values are expressed in seconds. The specified time can be a fractional amount of seconds. May be initialized by PROCESSTIME Control Statement.

NAME (0)

Pointer. Specifies the file name (TITLE) of the code file to be used. This attribute is used only when initiating an externally-compiled procedure or subprogram. May be read or written, but the value at process initiation is saved throughout process execution. Default value = none. It must be specified. The value may be stored for NAME using an ALGOL REPLACE statement. In COBOL, the value is stored using the SET statement. In both ALGOL and COBOL, the value to be stored (the file TITLE) must be a string containing a period as the last character.

OPTION (22)

Real. Used to set the bits in the option word for the process (in the same manner as the OPTION Control Statement. May be read or set at any time. Values: The bits, when set, have the following values and meanings.

ORGUNIT (38)

This attribute indicates where the JOB originated. It carries a UNIT number or the logical station number of the terminal.

IPC

Bit	Values	Meaning	Comparable OPTION Control Statement
0	1	No segmented arrays allowed.	; OPTION = LONG
1	2	Program dump if any fatal internal error occurs.	; OPTION = FAULT
2	4	Program dump for any external termination condition (such as operator DSed).	; OPTION = DSED
3	8	Control cards will be listed on print file named DIAGNOSTICS	; OPTION = CCLIST
4	16	All print files for the process will be printer backup files.	; OPTION = BACKUP
5	32	Duplicate files for this task will be automatically removed.	; OPTION = AUTORM
6	64		;
7	128	Dump base of stack if and when PROGRAMDUMP	; OPTION = BASE
8	256	Dump all arrays of stack if and when PROGRAMDUMP	; OPTION = ARRAYS
9	512	Dump Segment Dictionary of STed if and when PROGRAMDUMP	; OPTION = CODE
10	1024	Dump Files of stack if and when PROGRAMDUMP	; OPTION = FILES

The option word for the process can then be set in one of the following ways:

```
PROCESS.OPTION := 1"010110";

PROCESS.OPTION := 22;
```

The two statements each achieve the same result. Bits 2, 3, and 5 of the option word for the process will be set, causing programs dumps on all abnormal terminations of the process, and causing all printer files to be printer backup files.

PARTNER (19)

Task. Indicates the task to which control will be passed when a CONTINUE statement is executed in a synchronous process. May be read or set at any time. Value: a task identifier.

i.e., set by: T1.PARTNER := T2;

PROCESSIOTIME (14)

Real. Returns the accumulated I/O time (the total I/O time used to date) for the process. Read-only attribute. May be read at any time.

PROCESSTIME (13)

Real. Returns the accumulated processor time (the total processor time in seconds used to date) for the process. Read-only attribute. May be read at any time.

PROCESSTYPE (26)

Reserved for MCP use.

RESTART (28)

Real. The value of this attribute specifies the number of times the task is to be re-executed following an error termination pending a successful termination. The default value is zero, and RESTART is decremented by one after each execution of the process. An operator-DSed task will not restart.

STACKHISTORY (30)

Pointer. This read only attribute is conditionally stored on a fault and indicates the stack history (nesting of procedure calls) at the time of the fault. STACKHISTORY is unconditionally stored when the task is terminated due to a fault.

The format of the STACKHISTORY is the standard format:

SSS:AAAA:Y,#SSS:AAAA:Y,#....#SSS:AAAA:Y.

or

SSS:AAAA:Y#(DDDDDDDD),#....#SSS:AAAA:Y#(DDDDDDDD).

where SSS is the segment number, AAAA is the address, Y is the syllable, # is a blank space, DDDDDDDD is the line number (only present if LINEINFO was set during program compilation). The period (.) always terminates the last entry.

IPC

STACKNO (1)

Real. Returns the mix number of an active process, or the negative of the mix number of a terminated process. May be read at any time. Values: A negative number indicates a terminated process, a positive number indicates an active process, a zero indicates that the task variable was never associated with an active process.

STACKSIZE (7)

Real. Specifies or returns the stack size requirement used for scheduling by the system. May be read or set at any time; however, the value at process initiation is saved throughout process execution. May be initialized by STACK Control Statement. Value is the number of words required for the stack of the task.

STATUS (12)

Real. Returns the current state of the process; may be set to cause status changes as described below. May be read or set at any time. The values are as follows:

0	Not being used (the task is not associated with a process). Setting the attribute to 0 disassociates the task from the process.
1	The process is scheduled. Setting the attribute causes the process to be scheduled.
2	The process is active. Setting the attribute to 2 has no effect unless the process is suspended.
3	The process is suspended. Setting the attribute to 3 suspends the process.
-1	The process is terminated (DSed or EOJ). Setting the attribute to -1 causes the process to be DSed.
-2	Initiation of the process has failed when attempted by the MCP procedure DOCTOR. Setting this attribute to -2 has no effect. The DOCTOR procedure has stored an error code in task. HISTORY indicates the cause of the failure. (See HISTORY.)

STOPPOINT (18)

Real. Contains the segment and relative address at which the last arithmetic fault occurred in the process, or contains the segment and relative address at which the program was terminated or suspended. May be read at any time. Read-only attribute. Value: STOPPOINT returns the RCW which was created due to the fault.

SUBSPACES (31)

Real. The values (0 thru 3) of this attribute determine the manner in which the Swapper independent runner handles the associated tasks. The four values are:

0: the task's code, stack, and data are not contained in the swap area; 1: the task's code is not contained in the swap area; 2: if the code file of the task is in the user's usercode library, the MCP will place the task's code, stack, and data in the swap area. In all other cases, the code is placed outside of the swap area and the stack and data are placed in the swap area; 3: the task's code, stack, and data are contained in the swap area.

SUBSPACES = 3 for the GO part of COMPILE AND GO jobs initiated thru CANDE, SUBSPACE = 2 for any job executed thru CANDE, and SUBSPACES = 1 for any other jobs (e.g., compiles, execution of LOGANALYZER, DCSTATUS, RESEQBASIC).

TARGETTIME (6)

Real. The time by which the process should be completed. Used for scheduling by the system. May be read or written at any time, but the values at process initiation are saved by the system throughout process execution. The time is expressed in four digits (military format), indicating hours and minutes of a 24-hour clock. May be initialized by TARGET Control Statement.

TASKFILE (32)

TASKFILE is a printer backup file associated with each task. When a PROGRAMDUMP is requested, it is written to the TASKFILE. If the programmer desires to intermix his data between occurrences of his PROGRAMDUMPs, the programmer codes WRITE(MYSELF.TASK, <format and list part>). TASKFILE can also be label equated in a control deck. For example,

```
? FILE TASKFILE (KIND = PRINTER)
```

forces the PROGRAMDUMP directly to a line printer when LPPBDONLY is reset.

TASKVALUE (9)

Real. This attribute may be set or tested for specific purposes desired by the programmer. May be read or written at any time. Values are specified by the programmer. This attribute is generally used as a means of communication between processes.

TYPE (11)

Real. Returns the type of process. May be read at any time. Read-only attribute.

0 = PROCESS, 1 = CALL, 2 = RUN, 3 = JOBSTACK

USERCODE (8)

Pointer. Accepts or returns the usercode for the process. May be read or set at any time, but the value at process initiation is saved by the system throughout process execution. May be initialized by a USER Control Statement. Default value is the usercode of the global process. May be set as follows:

IPC

```
(ALGOL) REPLACE T.USERCODE BY { "ABC."; }
                              { P + N;  }
(COBOL) SET TSK(USERCODE) TO "ABC".
```

Notice that in ALGOL the string must be terminated with a period; whereas in COBOL the string must not be terminated with a period.

VALIDITYBITS (23)

Boolean. This attribute, which is used for internal bookkeeping purposes, is set when the user specifies the setting of any task attribute explicitly by some program using a task attribute statement.

DATA COMMUNICATIONS SOFTWARE

CANDE

Syntax Conventions

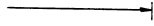
The principal means of displaying CANDE syntax is the syntax diagram. The rules of such diagrams are:

1. Any path traced along the forward direction of the arrows will produce a syntactically valid command.

2. Any "bridge" over a digit may be traversed a maximum number of times specified by the digit. If the digit is followed by an "*", then the path must be crossed at least one time.

3. Upper-case letters in the syntax diagrams indicate keywords which are literally in the commands. Minimum abbreviations are indicated by underscoring.

4. End-of-statement is indicated by:

   a.  which denotes either a semicolon or end of line.

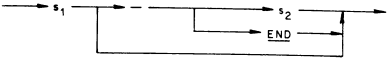
   b.  which denotes end of line.

5. Lower-case letters, words, and phrases are syntactic variables, which represent information to be supplied by the user.

6. Identifiers may vary in length from one through seventeen characters.

7. The variable "delim" represents any arbitrary delimiter which may generally be any non-alphanumeric character except one that occurs in the field being delimited or one that might have special significance in the context of its appearance.

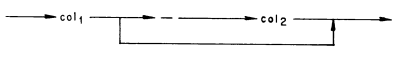
8. A <sequence range> specifies an inclusive range of sequence numbers:



A <sequence range list> consists of one or more sequence ranges separated by commas.

CANDE

9. A <column range> specifies an inclusive range of line columns:



10. A <modifier> is a standard MCP control statement.

    If a modifier is preceded by COMPILER or a <compile type>, it refers to the compilation rather than the execution.

11. A logical station number, <lsn>, is a unique integer assigned to each station in the datacom network.

12. A <station name> is an identifier identifying a station in the datacom network.

LOG-IN PROCEDURE

Connection must be established between the user's terminal and the system in the manner prescribed at the installation. After a connection has been made, the system types a greeting message as follows:

```
B 6700 CANDE <system release level> YOU ARE:
             <station name>(<lsn>)
```

The system then initiates the log-in sequence which consists of the following steps:

a.

System Message	User Response
#ENTER USERCODE PLEASE	enter: <usercode> or <usercode><delimiter> <password>
#ENTER PASSWORD PLEASE	enter: <password>

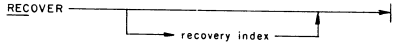
b. The user may then be asked to enter installation-dependent information such as a charge code.

c. If any recovery files exist at log-in time, CANDE will type:

```
#RECOVERY DATA:
   xxxx <workfilename>(<date>)
```

where xxxx is the recovery index, <workfilename> is the name of the recovered workfile, and <date> is the date of the aborted session.

Recovery of the desired workfile is accomplished via the RECOVER command:



d. Completion of log-in is indicated by the message:

LOGGED IN <time><date>

SPECIAL CONTROL CHARACTERS

Special characters to edit messages which are supported by SYSTEM/SOURCENDL are:

end-of-text (ETX)	Control-C
backspace	Control-R
line delete	Rubout
control command	?

FILE NAMES

A filename identifies a file in the B 6700 system. A CANDE filename consists of a list of 1 to 12 identifiers separated by slashes. Each identifier may contain up to 17 characters. In some contexts, the filename may begin with * or (<usercode>) to denote a system or other-user file. A workfilename is limited to 136 characters.

FILE TYPES

Possible types which may be specified for a workfile are:

ALGOL	BASIC
DCALGOL	PLI
XALGOL	ESPOL
FORTRAN	BINDER
XFORTRAN	SEQ
COBOL	DATA

SEQUENCE NUMBERS

BASIC symbol files - 4 digit maximum
COBOL symbol files - 6 digit maximum
other files - 8 digit maximum

CANDE

USER LIBRARY FILES

CANDE files are handled in such a manner as to ensure user file security. All source files are saved unaltered as:

(<usercode>)<filename>

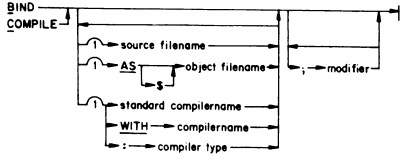
Object files are usually entered into a separate directory as:

(<usercode>)OBJECT/<filename>

"$" may appear (wherever allowed in syntax) to omit "OBJECT/" in disk filename.

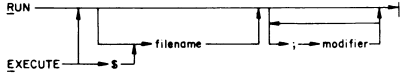
Other files may be generated by CANDE and placed in the user's library.

PROGRAM EXECUTION COMMANDS



Invoke the standard system binder and compilers or any non-standard versions.

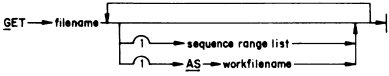
Standard Compilernames:	Compiler types:
ALGOL	ALGOL
XALGOL	XALGOL
DCALGOL	DCALGOL
FORTRAN	FORTRAN
XFORTRAN	XFORTRAN
COBOL	COBOL
BASIC	BASIC
PL/I	PL/I
ESPOL	ESPOL
BINDER	BINDER



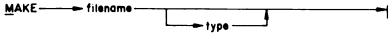
Cause execution of an object program. (Run will provide for compilation if necessary.)

CANDE COMMANDS

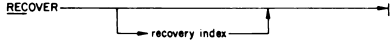
Work File Commands



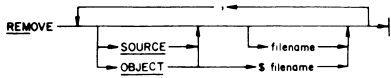
Recalls an existing file as the workfile.



Creates a new workfile.

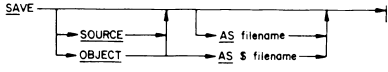


Lists recovery files or recalls a recovery file as the workfile.

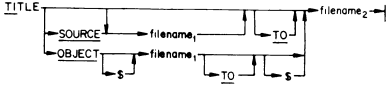


May be used to remove the workfile or any file in the user's library.

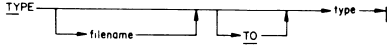
CANDE



Causes the current workfile and/or its associated object file to be saved in the user's library.

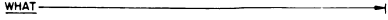


Changes the filename of the workfile or files in the user's library.



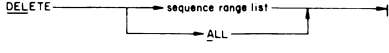
Changes the file attribute FILEKIND of a file.

Forces immediate update action on the workfile.

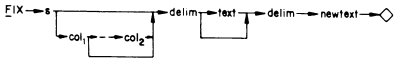


Indicates the state of the workfile.

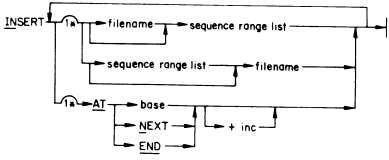
Editing Commands



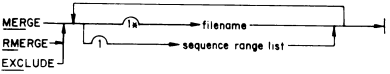
Discards lines from the workfile.



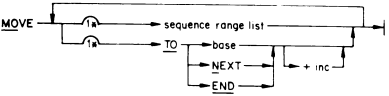
Alters the contents of the workfile by inserting new text or replacing part of a line.



Copies lines from the workfile or other file and places the copies into the workfile with new sequence numbers.

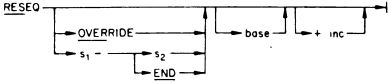


Cause a specified file, or portions thereof, to be merged with the workfile, with the result becoming the workfile.

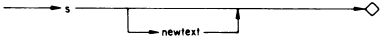


Moves lines from one point to another within the workfile and changes their sequence numbers.

CANDE

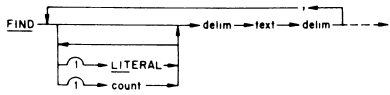


Assigns new sequence numbers to lines in the workfile without changing the order of appearance of any lines.

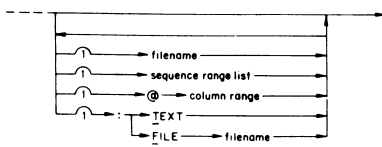


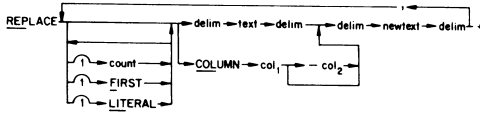
Any line beginning with a digit is a "command" to enter a new line of text at the sequence number specified or to replace or delete the line at that sequence number.

Search Commands

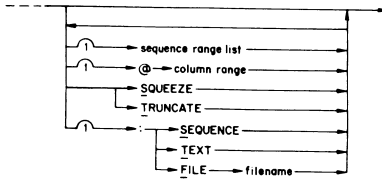


Searches a file for the appearance of specific text.

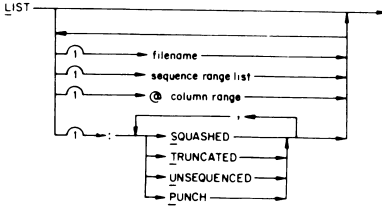




Scans line-by-line through the workfile or selected portions, replacing certain target with new text or inserting new text.

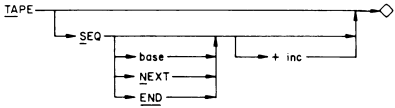


nput/Output Commands



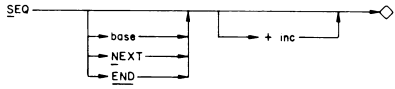
Displays the contents of the workfile or some other file to the user at his terminal.

CANDE

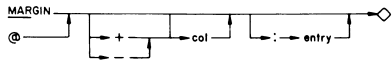


Initiates reading input from paper tape.

Editing Mode Commands



Invokes automatic sequence mode, causing the system to provide the sequence number for each new line.

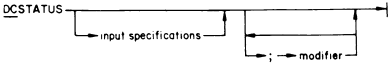


Controls the entry of text at the left margin of a line.

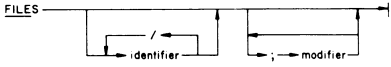
Environment Commands

Terminates the user's current session.

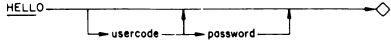
Used to specify a charge code for a session.



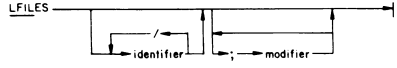
Causes execution of SYSTEM/DCSTATUS which provides general information regarding the status of the datacom network.



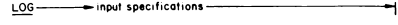
Results in the execution of SYSTEM/LISTFILES which provides a list of the names and types of files in the user's library.



Initiates a new user session without disconnecting the terminal.

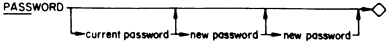


Results in the execution of SYSTEM/LISTFILES which provides a list of the names and B 6700 file attributes of files in the user's library.

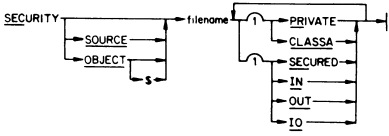


Results in the execution of SYSTEM/LOGOUT which provides a log of job performance.

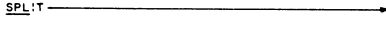
CANDE



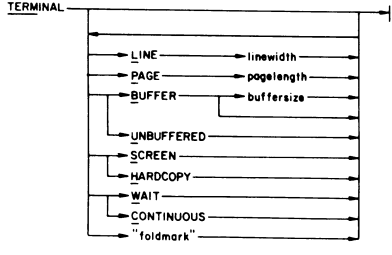
Allows a user to change his password.



Allows specification of the security attributes of a user's file.

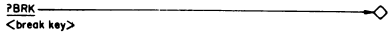


Divides session to print accumulated listings.

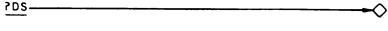


Specifies attributes of the user's terminal.

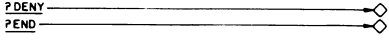
CONTROL COMMANDS



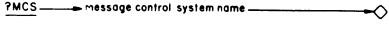
Terminates the current output to the terminal.



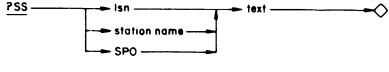
Causes the termination of any program scheduled or initiated by a user or a CANDE editing or output process.



Provides for terminating the usage of a terminal as a remote file.



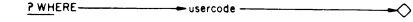
Provides a user the ability to transfer to a MCS other than SYSTEM/CANDE.



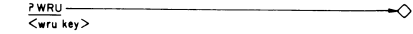
Provides a user the ability to send a message to another station.

Provides the current status of user's program.

CANDE



Provides the station name and lsn of a user.



Provides identification of the operating datacom system and the user's terminal.

NOTE

For information concerning the CANDE control functions, refer to CANDE OPERATIONS MANUAL, Form No. 5000615, Dated 18 October 1972.

REMOTE JOB ENTRY SYSTEM

The concept of remote job entry (RJE) involves a remote satellite system consisting of a small terminal computer connected to a central system via a communications line to which one 80-column card reader, one line printer (120 characters minimum), and one 72 character supervisory console can be connected. This particular type of system allows the following:

a. Introduction of programs from a remote reader.

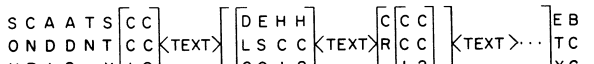
b. Introduction of data from a remote reader.

c. Output data printed on the remote line printer.

d. Monitoring and controlling of programs via a remote supervisory console.

MESSAGE FORMAT

The message format is designed to maximize line thruput and detection and recovery of errors. The format of the message is assumed to be in ASCII-67 and as follows:



A description of each part of the message format is as follows:

a. SOH - Start of header indicates that data is being transmitted.

b. CNR - This character may be NUL, NAK or ACK and is used for error detection and recovery as follows:

When a message is to be sent, then CNR is set to:

(1) NUL if the previous input has already been acknowledged.

(2) NAK if the previous input was received with errors.

(3) ACK if the previous input was received correctly.

When a message is received as the acknowledgment of a previous output, then:

(1) If CNR is NULL the previous output was not seen and must be re-sent.

(2) If CNR is NAK the previous output was received incorrectly and should be re-sent.

(3) If CNR is ACK then the previous output was received correctly; therefore, it can be discarded and the next output, if any, can be transmitted.

c. The address designate characters (AD1, AD2) relate a message to a particular device on a remote satellite as follows:

A D 1 2	
0 0	= System control
0 1	= To/from RSC
0 2	= To card punch/from card reader
0 3	= To line printer/from (is an error)
0 4	= To paper tape punch/from paper tape reader
0 5	= To/from magnetic tape
0 6 through 09	= (Reserved for expansion)
1 0 through 99	= AD1 and AD2 of terminals connected to the remote satellite.

d. The transmission number (TN) is counted up one modulo 10 (0-9) with each transmission. It is used to detect duplicate or missing transmissions.

e. The STX character serves as a delimiter between the header portion and the text protion of the message.

f. The CR character serves as a separator between records within a message.

g. If the message carries a unit designate of line printer, then the first two characters after the STX or each CR are considered as the carriage control characters (CC1, CC2) which deliver carriage control to the remote line printer as follows:

C C 1 2	
0 0	= Print no space
0 1	= Print and skip to top of form
1 0	= Print and single space
1 1	= Print and double space

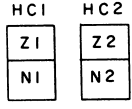
h. The sequence "DLC ESC HC1 HC2" may be encountered anywhere within the message and represents a field of identical, adjacent characters as follows:

DLC - The character being supressed.

ESC - Field compression indicator.

HC1, HC2 - The number of times DLC is repeated.

HC1 and HC2 represent a hex count and only the numeric portions of the fields should be used and the zone should be discarded. For example:



The number of characters in the field is

(N1*16+N2) = 255 maximum

There is a slight complication due to the B 6700 operating in EBCDIC, the message being in ASCII-67 and the HC1, HC2 being in hex. Care must be taken in what zones are to be put into Z1 and Z2 to ensure that a hex value is maintained in N1 and N2 after HC1 and HC2 undergo ASCII to EBCDIC translation. It is recommended that the remote satellite use the following conversions when generating the HC1 and HC2 characters.

ASCII-67	HEX	To give N1 or N2 in EBCDIC
0	30	0
1	31	1
2	32	2
3	33	3
4	34	4
5	35	5
6	36	6
7	37	7
8	38	8
9	39	9
:	3A	A
#	23	B
@	40	C
<vertical bar>	27	D
=	3D	E
"	22	F

NOTE

The above does not consider the proper setting of the eighth bit (parity bit).

i. The ETX character is used to denote the end of a message.

j. The BCC is the block check character, i.e., longitudinal parity character. It is the exclusive or of all characters starting with the ADI through and including the STX character. The purpose of the BCC is to ensure proper receipt of messages.

NAMING CONVENTIONS

In order to allow remote users to be isolated from each other, in so far as file name selection and at the same time maintain a system of unique names to the central system, a user identification can be assigned by the data processing manager to each user of the system. The RJE MCS obtains this user/identification in either of two ways, by station log-in or by a special control card.

The RJE MCS will ensure that the user identification is valid before either allowing connection of the RJE satellite to the system or running a program. In order to reference system files, it is necessary to override this automatic prefixing of files by the MCP. This can be accomplished by preceding the first identifier of a file name by an asterisk (*), either internally or externally. For example, to reference a site file called A/B, a label equation card of the following form would have to be used:

```
<I> FILE <internal-name> (TITLE = * A/B)
```

REMOTE DECKS

The RJE MCS expects to see its input from a remote site in the form of decks for which there are four types; COMPILE, EXECUTE, RUN, and DECK. Decks are comprised of a number of "card image" messages, each message consisting of 80 characters. The first character of a control card is the control character (<I>) and is ignored by the RJE MCS.

COMPILATION DECK

The appearance of a compile deck indicates a request for compilations. The format of a typical compilation request deck is as follows (Note: zero, one or more "user cards" are allowed):

```
<I> COMPILE <program name> WITH <compiler or binder name>

(Note: "BIND <program name> WITH <binder or compiler name>" is
also allowed.

<I> COMPILER CONTROL CARDS

<I> COMPILER LABEL EQUATION CARDS
```

<I> OBJECT CONTROL CARDS

<I> OBJECT LABEL EQUATION CARDS

<I> EBCDIC

(compiler $ and input cards)

<I> END

The object program name from the compile card is handled by the MCP.

* A/B would go unchanged but
A/B would become USERCODE/<user-ID>/A/B

The compiler and object control cards (STACK, PRIORITY, etc.) if present are used by the MCP as is.

EXECUTE DECK

The appearance of an execute deck indicates a request to execute a program residing at the central system. The deck may be in two formats (Note: zero, one or more "USER CARDS" are allowed):

Format 1:

<I> USER = <user-ID>

<I> EXECUTE <PN>

(Note: RUN <PN> is also allowed.)

<I> <optional label equation>

<I> END

Format 2:

<I> USER = <user-ID>

<I> EXECUTE <PN>

(Note: RUN <PN> is also allowed.)

<I> <optional label equation>

<I> EBCDIC <DN1>

(input data)

<I> END

Note: The "<I> END" must always be on a separate card in an execute deck.

RJE

DATA DECKS

Data can be collected on the central system via a data deck. The following is a format of a data deck (Note: zero, one or more "USER CARDS" are allowed):

```
<I> DECK { BCL    } <DN>
         { DATA   }
         { EBCDIC }

     (input data)

<I> END
```

REMOTE SUPERVISORY CONSOLE (RSC)

A remote-operator/system interface is required in order to make decisions which cannot be pre-programmed. This feature is provided through output and input messages via the remote supervisory console (RSC).

OUTPUT MESSAGES

There are three types of output messages:

1. Local
2. System status
3. Log-on and log-off

Local Messages

Local messages are generated by the remote satellite having to do with its own control, such as reader hopper empty, printer out of paper, etc.

System Status Message

The format of a system status message is:

```
<mix index> : <status> : <priority> : <program-name>
```

where

```
<mix index> ::= {program reference number}
<status> ::= BOJ/          (beginning of job)
             SNTX/         (syntax in compilation)
             EOJ/          (end of job)
             STOP/         (program suspended)
             DS-ED/        (program terminated)
             SCHD/         (program scheduled)
             EXEC          (program in execution)
<priority> ::= {a digit with a range of 00 to 99 indicating program
                priority}
```

LOG-ON and LOG-OFF Messages

If a connection to the central site exists, the RJE MCS will respond with the following message:

```
B6700 SYSTEM/RJE 2.4.00 MIX=nnnn.
#ENTER USERCODE PLEASE.
```

The remote operator then must enter the user-ID assigned by the central system and then the RJE MCS will respond with

```
#AND YOUR PASSWORD.
```

The remote operator then must enter his password. If the password or usercode is incorrect, then the RJE MCS will respond with

```
#SECURITY ERROR, ENTER USERCODE.
```

If the password is recognized, the RJE MCS will respond with

```
<station-name> LOGGED ON AT <time> <date>
```

To terminate the connection, the remote operator enters

```
BYE
```

If the site is essentially "idle" the RJE MCS will respond with the following message and then terminate itself:

```
<station-name> LOGGED OFF AT <time> <date>
```

ERROR MESSAGES

Control deck errors will be recognized and appropriate messages shall be given as follows:

```
COMPILER NAME NOT FOUND.
CONTROL CARD ERROR.
DATA DECK ERROR.
FILE CARD ERROR.
FILE ID TOO LONG.
FILE SECURITY ERROR.
IDENTIFIER TOO LONG.
INVALID FILE IDENTIFIER.
NO FILE.
NO FINAL QUOTE IN A STRING.
NO INV. CHARACTER IN COL. 1.
NOT IMPLEMENTED.
PASSWORD ERROR.
RUN/COMPILE DECK NOT FOLLOWED BY END.
UNRECOGNIZED CONSTRUCT.
USER CARD ERROR.
USER CARD REQUIRED.
MESSAGE LOST - RESUBMIT DECK.
```

RJE

INPUT MESSAGES

System Control Messages

System control messages include the following:

a. WT (return current time)

b. WD (return current date)

c. WM (return RJE level and MIXID)

d. SS {message of up to 25 characters}

e. The RJE options include the following:

where <option> ::= AUTOPRINT/PBDONLY/AUTORM

SO <option> - sets a particular option

RO <option> - resets a particular option

TO - lists the options and indicates whether the particular option is set or reset. Each option is as follows:

(1) AUTOPRINT: if true this option will print automatically, on the RJE printer, all printer backup disk files as they are produced.

(2) PBDONLY: if true this option will cause the system to automatically force all printer files to be equated to printer backup disk.

(3) AUTORM: sets AUTORM option on all jobs.

NOTE:

If both PBDONLY and AUTOPRINT are set then all printed output of programs fired up from the RJE satellite will be automatically printed on the remote printer.

f. PB <mix index>. Prints on the remote printer all printer backup produced by the program whose reference number is <mix index>.

g. SB <mix index>. Prints on the central system printer all printer backup produced by the program whose reference number is <mix index>.

h. <mix index>QT and <mix index>DS. Stop printing of a printer backup file. DS purges the backup file. QT does not. <mix index> is the reference number of the print backup routine.

i. SF <integer>. This option changes the blocksize on transmission between systems. The default is 407 characters/block which is the maximum. The minimum is 83 for transmission from the DC1000 and 135 for transmission from RJE. This factor should be set lower for systems with high error rates.

j. RS <message up to 25 characters>. Sends a message to the system SPO to which the system operator may respond.

k. TF. Responds with the factor set.

l. US. Responds with all RJE users and their SN's. The requesting status is denoted by an *.

Program Control Messages

Program control messages include the following information:

a. CONTROL DECK. Compile and execute deck may be typed in from the RSC.

b. <mix index>DS. Suspends a program.

c. MIX. Returns a list of all running and scheduled programs introduced from the RJE satellite.

d. <mix index>DS. Terminates a program.

e. <mix index>TI. Returns the elapsed and processor time of a program.

f. PD <file-ID>. Returns a message indicating the existence of <file-ID>.

g. PD <file-ID>/ Returns a list of the files in the directory file <file-ID>.

h. <mix index> OK. Causes a suspended program to resume running.

SM Control Messages

SM messages are accepted from the central site console. The basic form of an SM message is <MCS mix number>SM:<text>. <text> must be in one of the two basic forms, <key word> <optional text> or <lsn> <key word> <optional text>. <key word>s are two letter mnemonics. The following discussion refers only to SM messages which must be entered at the central site console.

<key word>s Requiring LSN

a. PH. If an automatic callback was attempted but unsuccessful, the PH message will signal RJE to try dialing again. Note that

an attempt to dial must have already occurred. This restriction avoids erroneously or capriciously connecting switched lines.

b. SO <option>. SO sets the <option> specified. <option> can be either LOGON or USER. If USER is set, all decks originating from that station must be preceded by a USER card. If LOGON is set, a log-on sequence as described in the RJE manual is required. If LOGON is set and USER is reset, all decks originating from that station which do not have a USER card will be run under the usercode as given by the log-on sequence. If LOGON is reset, no log-on sequence is required, and no default usercode will be used. If a station is active and LOGON is not set for that station when the central site operator sets LOGON, the station will be logged off and required to log-on.

   The settings of these two options will be remembered across different runs of RJE. However, if RJELINKED file is removed, settings will return to the default of LOGON = set and USER = reset.

   Note that when running under a usercode, library maintenance cannot be performed on files not under that usercode directory unless it is a privileged user. To find files under a usercode through RJE, use PD USERCODE/<user>/<file name>, where <file name> can be an equals sign. PD <file name> will search the system directory.

c. RO <option>. The RO message resets the above options.

d. SV. The SV message saves the specified station. The station is made not ready, and error recovery is discontinued.

e. RY. The RY message makes the station, and line associated with that station, ready. This would appropriately be performed on a previously saved station. (See SV message discussion.)

f. RE <optional to> <MCS name>. The RE message allows the central site operator to release an RJE station as given by the <lsn> to another MCS. An example would be:

```
31 RE TO SYSTEM/CANDE
```

   Note that only the station specified is released. If, for example, LSN 31 were a remote supervisory console, the previous statement would release only the remote supervisory console; the printer, card reader and DC1000 would function normally. If the LSN released is a printer or remote supervisory console, no further output to that LSN will be attempted.

<key word> Allowing Optional LSN

a. WH. The WH message without LSN provides the LSN state (i.e., active, inactive, saved) and station name of each station known to RJE. If an LSN is provided, the station name, usercode (if applicable), log-on requirements, USER card requirements or default usercode use, switched status, state and phone number (if applicable) are given for the specified station. The infor-

mation is provided via sequential diaplays on an asynchronously running task.

<key word>s Not Allowing LSN

a. DP <dump options>. Provides a RJELINKED file printout and a program dump. It is only applicable when RJE is compiled with DEBUG set. <dump options> are base, arrays, files, code or an <integer>. Options may be separated by commas or blanks. If no options are specified, the options used will be as appeared on the option card when RJE was compiled.

b. HI. This message causes the current debug listing to be printed. Consecutive HI messages will print only the DEBUG listing not previously printed.

c. QT. This message will cause RJE to go to EOJ in an orderly fashion. In particular, stations will be logged off so that correct accounting can be provided. This technique should be used in preference to DSing RJE.

d. SS <string>. This is a broadcast of the <string> to all active RJE stations. The message will be printed on the RSC as:

#SS ALL:<string>

ERROR RECOVERY

During system operation, connection between the central system and the RJE satellite may be lost. If all recovery attempts fail, the RJE MCS suspends all programs that were introduced from the RJE site, and then notifies the system operator and waits for a response. The system operator can enter either of the following two messages:

a. OK. This will cause the RJE MCS to re-establish connection to the line. It will resume everything it suspended if the connection can be made. If it cannot re-establish a connection, the system operator will be informed.

b. DISCONTINUE. This entry will cause the RJE MCS to terminate everything it controls then terminate itself.

DEBUG COMPILE OPTION

If the option DEBUG is set when SYSTEM/RJE is compiled, certain features are added.

a. Monitor output will be produced.

b. DP message - The DP message causes a program dump and dumps the linked list disk file containing control cards and data.

c. In response to the WM message, "(DEBUG)" will be printed after the RJE level.

OPERATING THE DC 1000

The following procedure, when executed properly, should enable the DC 1000 to operate as a remote controller.

a. Power up - Accomplished by pressing the power on/off switch.

NOTE

A problem in the power up sequence of some DC 1000's (correction in progress) may result in failure to load the object program. If this occurs, let the DC 1000 set with power on for 2 to 5 minutes, then quickly power down and power back up again.

b. Load papertape bootstrap loader.

(1) Place the papertape binary loader into the teletype reader positioned at the first character.

(2) Actuate the reset switch.

(3) Actuate the auto bootstrap button.

(4) The binary loader will read into the system.

(5) Wait for all paper tape to be read.

(6) The system is now set to load object program.

c. Loading of object program.

(1) Insert card deck in card reader.

(2) Ready card reader.

(3) Actuate the run switch.

(4) Wait for all of deck to be read.

d. Actuate the step switch.

e. Actuate the reset switch.

f. Push clear display button.

g. Push P-register switch down, all others up.

h. Load 1000 @ 16 in display.

i. Push enter button.

j. Actuate the run switch.

MSCII

COMPILATION INSTRUCTIONS

MCSII may be compiled using the following control cards, where <I> is understood to mean an invalid character in column one:

```
<I> REMOVE SYSTEM/MCSII; END.
<I> COMPILE SYSTEM/MCSII DCALGOL LIBRARY.
<I> DCALGOL FILE TAPE (TITLE = SYMBOL/MCSII)
<I> EBCDIC
 $  MERGE
      (patch cards, if any)
<I> END
```

EXECUTION INSTRUCTIONS

Prior to any execution of MCSII, or any MCS for that matter, it is required that the local data communications environment has been generated via the use of NDL and DCPROGEN, such that the two files SYSTEM/NIF and DC/CODE appear in the disk directory. It is also required that the appropriate DCP(S) is (are) initialized from the console SPO via the "DC" input message.

When the above prerequisites are met, the MCS may be initiated by any one of the following three methods:

a. Introduction of a control card to the B 6700 system (via the console SPO, site card reader, etc.) containing:

```
<I> EXECUTE SYSTEM/MCSII; END.
```

b. Opening of a file by an object job containing one or more stations controlled by MCSII.

c. Any input from a station which is controlled by MCSII and for which "ENABLEINPUT=TRUE" has been specified in the NDL description.

CONTROL STATEMENTS (CS)

Syntax:

```
<CS block> ::= <CS> | <CS block>; <CS>
<CS> ::= <attach statement> |
         <broadcast statement> | <status request statement> |
         <line status statement> | <alter statement> |
         <whoami statement> | <global monitor statement> |
         <move statement> | <swap statement>
```

MSCII

Semantics:

Entities may not be split across cards (if input is from the site card reader). <CS>s may be continued across record boundaries, but all of a <station name> or syntactic entity must appear, complete, on the card on which it was begun. Control statements can be input from the console SPO, site card reader, etc.

The control statements are described as follows:

ATTACH STATEMENT

Syntax:

```
<attach statement> ::= ATTACH <station name> <attach status list> |
                       ATTACH <station DLS> <attach status list>
<station name> ::= {an NDL-defined station name}
<attach status list> ::= <empty> | , <status item> |
                         <attach status list>, <status item>
<station DLS> ::= DLS (<DCP number>, <line number>,
                  <station number>)
<status item> ::= ENABLED | READ | MONALL | MONERR
<DCP number> ::= <unsigned integer>
<line number> ::= <unsigned integer>
<station number> ::= <unsigned integer>
```

Examples:

```
ATTACH TWX/ONE, ENABLED, READY
ATTACH DLS(0,6,0), ENABLED, MONALL, READY
```

Semantics:

The <attach statement> is the means by which MCSII may be directed to perform attachment to one of its stations.

The <attach statement> may specify the initial status of the station through the use of an optional <attach status list>. If "ENABLED" is included as an <attach status item> and attachment is successful, MCSII will perform an explicit "ENABLE INPUT" DCWRITE call for the station. If "ENABLED" is omitted as an <attach status item> and attachment is successful, MCSII will perform an explicit "DISABLE INPUT" DCWRITE call for the station, and then issue a "READ-ONCE ONLY" DCWRITE call for the station. If "READY" is included as an <attach status item> and attachment is successful, MCSII will perform an explicit "MAKE STATION READY/NOT READY" DCWRITE call using a variant of zero (0) (ready), for the station. If "READY" is omitted as an <attach status item> and attachment is successful, MCSII will perform an explicit "MAKE STATION READY/NOT READY" DCWRITE call, using a variant of one (1) (not ready) for the station. If "MONALL" is included as an <attach status item> and attachment is successful, all station activity, MCS-initiated, file open-initiated, DCP-initiated, errors, etc., will be monitored on a site line printer. If "MONALL" is omitted as an <attach status item>, then no monitoring of the activities of the station will take place unless global monitoring of all station activity has been invoked. If "MONERR" is included as an <attach status item> and attachment is successful, all

error messages pertaining to the station will be monitored. If "MONERR" is omitted as an <attach status item>, then no monitoring of the station errors will occur unless "MONALL" (for the station) or global monitoring (for all stations) exists as monitoring options.

BROADCAST STATEMENT

Syntax:

```
<broadcast statement> ::= TO <destination list> <text>
<destination list> ::= <destination> | <destination list>,
                       <destination>
<text> ::= {any desired sequence of characters not beginning
           with a comma or a slash and not containing a semicolon}
<destination> ::= ALL | <station name> |
                  <station DLS> | <station LSN>
<station LSN> ::= <unsigned integer>
```

Examples:

```
TO ALL SEND THIS TO ALL STATIONS
TO DLS(0,6,3), TTY/ONE, 2, 4, TWX0
CLOSING UP SHOP IN 5 MINUTES
```

Semantics:

The <broadcast statement> affords the facility of "broadcasting" messages to any and all currently attached stations.

All broadcast output is sent to the destination stations in the form:

```
FROM <station name> (<station LSN>): <text>
```

If the <broadcast statement> was introduced to MCSII via the site file "OPTIONS", then the broadcast output to the destination stations will appear in the following form:

```
FROM SITE: <text>
```

STATUS REQUEST STATEMENT

Syntax:

```
<status request statement> ::= STATUS <station ID> |
                               STATUS ALL
```

Examples:

```
STATUS DLS(0,1,3)
STATUS ALL
```

Semantics:

The <status request statement> provides the means for the site or a remote station to enquire of MCSII the current status of a station or all stations.

MSCII

LINE STATUS STATEMENT

Syntax:

```
<line status statement> ::= LINES
```

Example:

```
DCP<DCP no.>, CLUSTER <cluster no.>, LINES 1, 4, 15 ATTACHED
```

Semantics:

The <line status statement> will return a table giving the readiness and "attached" nature of all lines known to the MCS.

ALTER STATEMENT

Syntax:

```
<alter statement> ::= ALTER <station ID> <status change list>
<status change list> ::= <status change> |
                         <status change list>, <status change>
<status change> ::= <status replacement list> = <state part>
<status replacement list> ::= <status item> |
                              <status replacement list> =
                              <status item>
<state part> ::= TRUE | FALSE
```

Examples:

```
ALTER EGO ENABLED=TRUE
ALTER SUPER/EGO ENABLED=READY=FALSE, MONALL=TRUE
```

Semantics:

The <alter statement> allows the introduction of control information which will modify the current status of a currently-attached station.

WHOAMI STATEMENT

Syntax:

```
<whoami statement> ::= WHOAMI
```

Semantics:

The <whoami statement> causes MCSII to produce the identification of the source of the <whoami statement> at that source.

GLOBAL MONITOR STATEMENT

Syntax:

```
<global monitor statement> ::= MONITOR | STOP | DEBUG
```

Semantics:

The <global monitor statement> allows the control of monitoring of all data communications activity. "MONITOR" will cause monitoring of all activity for all stations to commence. The introduction of "STOP" will cause monitoring of all activity for all stations to cease. "DEBUG" invokes more detailed monitor interpretation of station activity. (Except stations which are operating under "MONALL" and/or "MONERR").

MOVE STATEMENT

Syntax:

```
<move statement> ::= <move DCP statement> |
                     <move cluster statement> |
                     <move station statement>
<move DCP statement> ::= MOVE DCP <DCP number> <ready part>
<move cluster statement> ::= MOVE CLUSTER (<DCP number>,
                             <cluster number>) <ready part>
<move station statement> ::= MOVE STATION <station ID> <to part>
                             <ready part>
<ready part> ::= READY | NOTREADY
<cluster number>::= <unsigned integer>
<station ID> ::= <station name> | <station DLS> | <station LSN>
<to part> ::= TO (<DCP number>,<cluster number>,<adapter number>) |
              NULL
<adapter number> ::= <unsigned integer>
```

Semantics:

The <move DCP statement> will move all the clusters on the specified DCP to the NDL specified exchange DCP. The <move cluster statement> will move the specified cluster from the specified DCP to the NDL specified exchange DCP. The <move station statement> may be used to add, subtract, or move a station from one line to another as specified in the MOVE/ADD/SUBTRACT DCWRITE function (<to part> = NULL is equivalent to subtract).

SWAP STATEMENT

Syntax:

```
<swap statement> ::= <swap line statement>
<swap line statement> ::= SWAP (<DCP number>,<cluster number>,
                          <adapter number>) (<DCP number>.<cluster
                          number>,<adapter number>) <ready part>
```

Semantics:

The <swap statement> will cause the MCS to initiate a swap line request as per the specifications of DCALGOL.

SECTION 4
SYSTEM MESSAGES
AND
DISPLAY OF STATUS

SECTION 4 - CONTENTS

SECTION 4
SYSTEM MESSAGES AND DISPLAY OF STATUS

INDEX OF SYSTEM MESSAGES

SYSTEM INPUT MESSAGES

A "+" indicates that the message contains syntactic items beside the mnemonic. A "-" indicates that the mnemonic alone is a valid message.

INDEX OF SYSTEM MESSAGES

INDEX OF SYSTEM MESSAGES

SYSTEM OUTPUT MESSAGES

RSVP MESSAGES

SYSTEM MESSAGES

INDEX OF SYSTEM MESSAGES

INDEX OF SYSTEM MESSAGES

INDEX OF SYSTEM MESSAGES

INDEX OF SYSTEM MESSAGES

JOB ORIENTED MESSAGES

INDEX OF SYSTEM MESSAGES

INDEX OF SYSTEM MESSAGES

SYSTEM INPUT MESSAGES

SYNTAX CONVENTIONS

The syntax of each input message described in this section is displayed via the following notation. The device employed here is the syntax diagram constructed of words formed of upper- and lower-case letters, arrows, special characters, and digits. The basic rule is that any path traced along the forward directions of the arrows will produce a syntactically valid statement. All words formed of upper-case letters (except EBCDIC) and all special characters (i.e., commas, colons, hyphens, slashes, etc.) in the diagram must appear in the message as shown; all words formed of lower-case letters are syntactic variables representing user-supplied constructs. Any "bridge" over a digit, such as:

may be traversed a maximum number of times specified by the digit (e.g., 14 times in this example).

The syntactic variable "number" denotes the appearance of an appropriate unsigned integer constant. The variable "delim" represents a blank or any special characters.

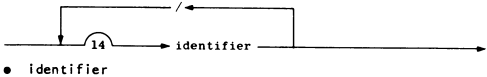
NOTE

For a detailed explanation of the input messages, refer to WORK FLOW MANAGEMENT USER'S GUIDE, Form No. 5000714, dated 16 April 1973.

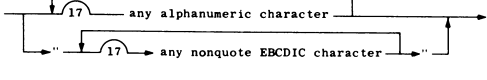
MISCELLANEOUS DEFINITIONS

The following items appear as syntactic variables in the message syntax diagrams:

- filename

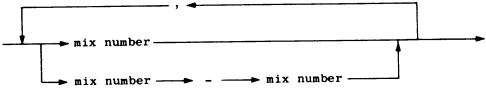


- identifier

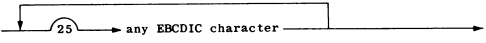


- mix number

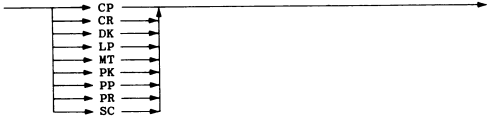
- mix number list



- text



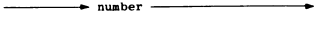
- device



The following synonyms are permitted:

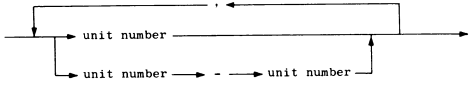
```
TAPE    = MT
DISK    = DK
SPO     = SC
PRINTER = LP
READER  = CR
PACK    = PK
```

- unit number



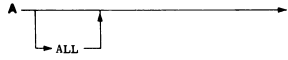
SYSTEM INPUT MESSAGES

- unit number list

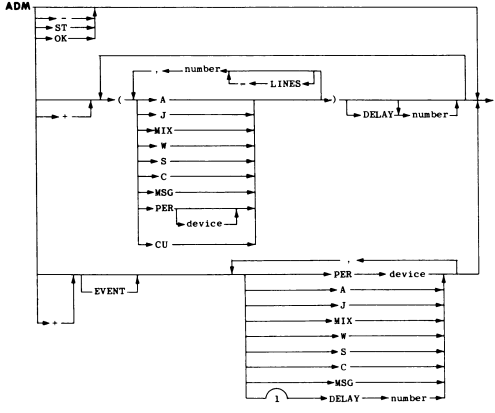


MESSAGE SYNTAX

The following paragraphs discuss the available B 6700 SYSTEM INPUT messages. A syntax diagram describing the form of each message precedes the discussion of that message.

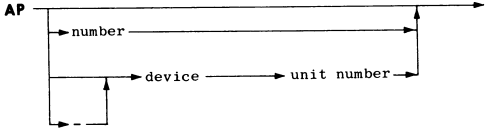


Lists all active jobs and tasks. Displays any active suppressed jobs or tasks when ALL is used.

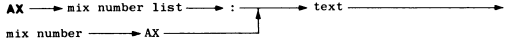


Allows a supervisory console to be placed in an automatic display mode of mix and system status displays.

SYSTEM INPUT MESSAGES



Allows the number of printers and punches to be used for the automatic output of backup files to be specified.



Used to input text for a programmatic ACCEPT operation.

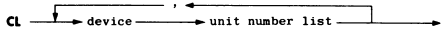
Lists the most recently completed jobs and tasks.

Causes execution of SYSTEM/COUNTANALYZER program.

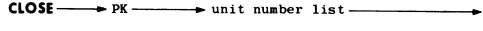
Specifies disk directory copy frequency.

Used to load a new intrinsics stack.

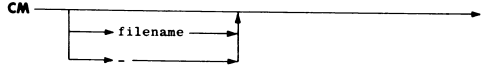
SYSTEM INPUT MESSAGES



Allows a peripheral unit to be cleared.

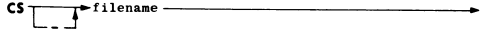


On a split system, CLOSE performs the same operations as the PO message except that the pack becomes unavailable for that system.

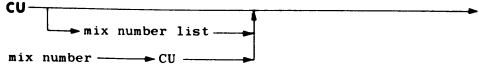


Allows a change to a new MCP.

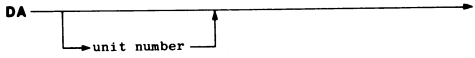
Designates a code file as a control program.



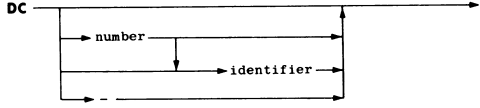
Designates a code file as a supervisor program.



Provides information regarding job or system core usage.

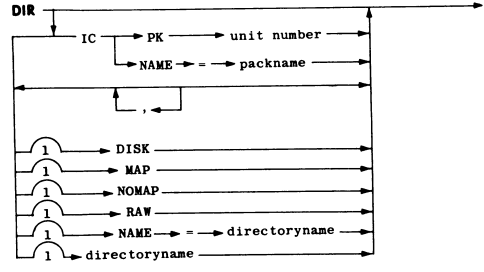


Causes execution of SYSTEM/DUMPANALYZER program.



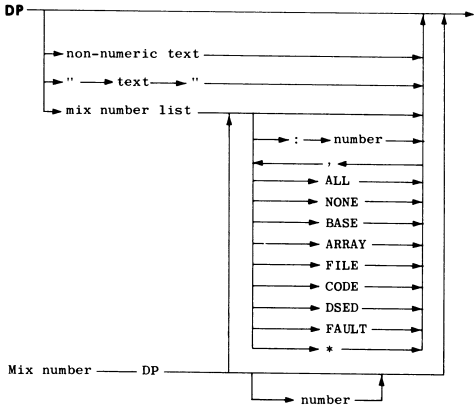
Provides for initiation of a DCP.

Initiates a directory copy operation.

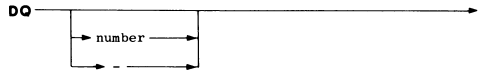


Yields a directory of head-per-track or disk pack files.

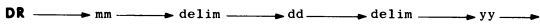
SYSTEM INPUT MESSAGES



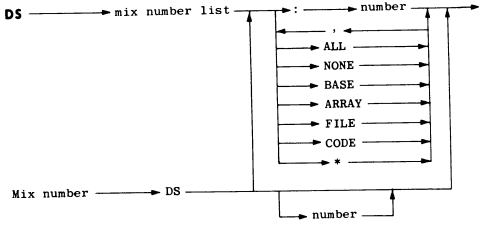
Initiates a program or memory dump.



Assigns a queue as the system default queue.

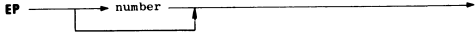


Resets the current date used by the MCP.

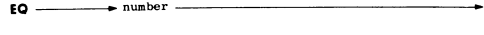


Terminates the execution of a job.

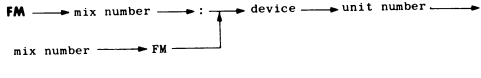
Allows suspension of normal selection of jobs for execution.



Causes any job files waiting to be printed to be discarded if AUTOPRINT is not active. In effect, job summary will not be printed.

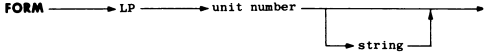


Causes the job queue identified by the number to be eliminated from system.

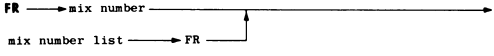


Directs program output to a device readied in response to a FORMMESSAGE request.

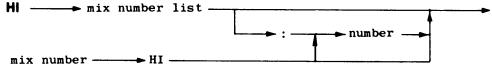
SYSTEM INPUT MESSAGES



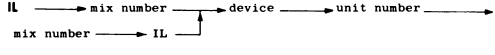
Names a printer with the specified string to be used in conjunction with the FORMMESSAGE file attribute.



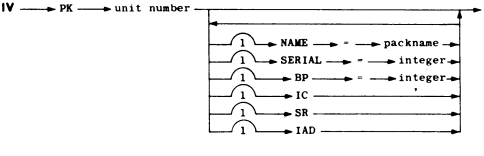
Indicates to the system the final reel of an unlabeled tape file or terminates a COPY/COMPARE on that tape unit.



Allows the EXCEPTIONEVENT of a running stack to be caused.

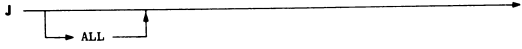


Assigns an input file to a program with the label ignored.

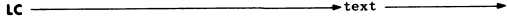


Allows the initialization of a disk pack.

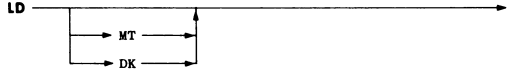
SYSTEM INPUT MESSAGES



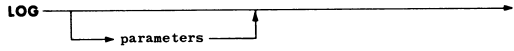
Causes the display of tasks (active, waiting, and scheduled) by job structure.



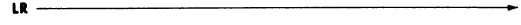
Enters a comment into a job or summary log.



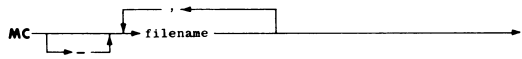
Allows a group of job control decks to be copied to a tape, or read from a tape.



Causes SYSTEM/LOGANALYZER program to be entered into mix and executed.



Generates an empty summary and/or maintenance log.

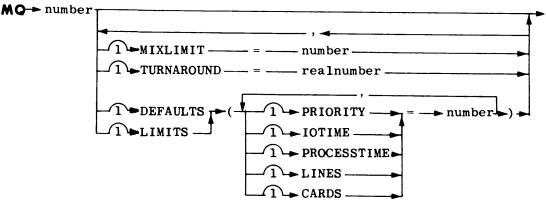


Designates a code file as a compiler code file.

SYSTEM INPUT MESSAGES

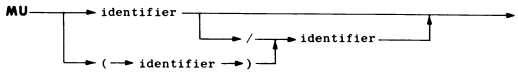
Yields a display of the current job mix.

Yields a list of ready memory modules.



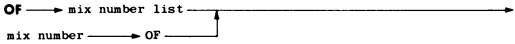
Creates a job queue and allows the attributes of that queue to be specified.

Yields a display of the most recently produced system output messages.

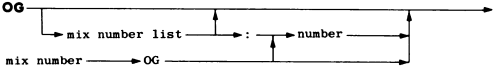


Allows an identifier to be recognized as a valid usercode.

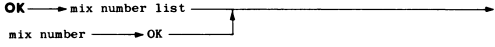
SYSTEM INPUT MESSAGES



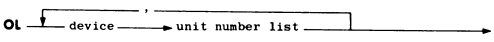
Indicates that an optional input file sought by a program is not present.



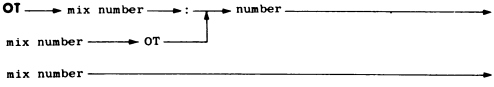
Allows the OVERLAYGOAL for a specific job to be altered.



Results in the reactivation of a suspended job.

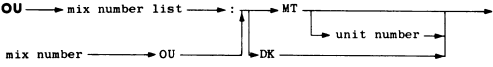


Yields a display of the label table for the indicated peripheral devices.

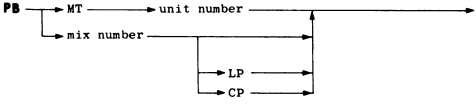


Yields a display of the current contents of the indicated cell of the specified running stack.

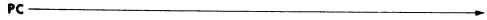
SYSTEM INPUT MESSAGES



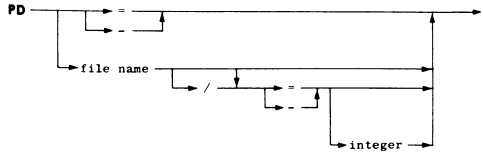
Dispatches output to a backup tape or disk file when an LP REQD, CP REQD, or FM REQD condition occurs; also may assign output to a locked or scratch tape when an MT REQD condition occurs.



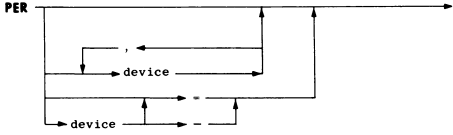
Causes a backup tape or disk file to be printed or punched.



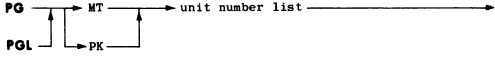
Yields a display of current system configuration and software status.



Yields a display or listing of specific files on head-per-track disk.

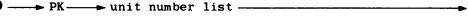


Yields information regarding the current status of the indicated peripheral devices.

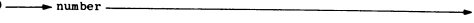


Purges (and optionally locks) specified tape units.

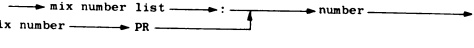
Yields a display of the approximate value of ¶.



Requests permission from the system to power-off the indicated disk pack.

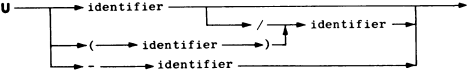


Removes all jobs from a queue while leaving the queue intact.



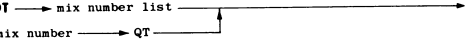
Allows the priority of the indicated job to be altered.

SYSTEM INPUT MESSAGES

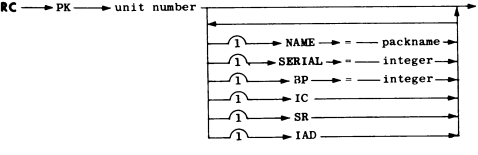


Causes a usercode to be recognized as a privileged user.

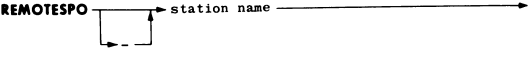
Yields a display of the current state of all attributes associated with the indicated job queue.



Terminates the output of a backup file without removing that file from the directory.

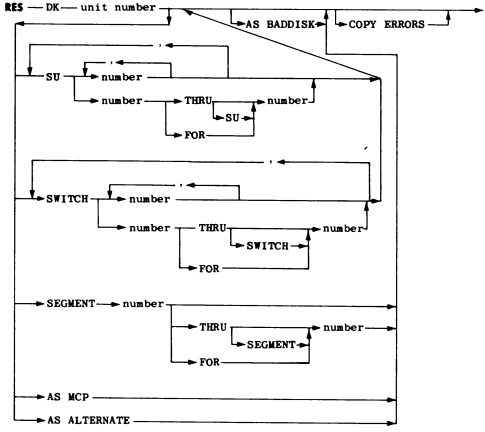


Used to label or relabel a disk pack.



Names a datacom station to be used for operator control of the system.

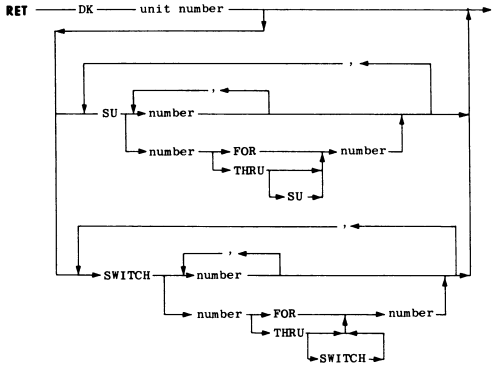
SYSTEM INPUT MESSAGES



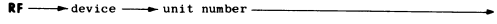
Allows the removal of a portion of the head-per-track disk subsystem from the system, marking it as an IAD or BADDISK file.

Causes the indicated suppressed active job to be included in the job mix display.

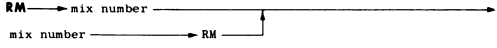
SYSTEM INPUT MESSAGES



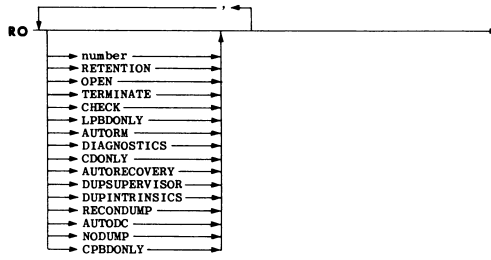
Allows a reserved portion of the head-per-track disk subsystem to be returned to the system.



Causes the current reliability factor of a peripheral unit to be displayed.

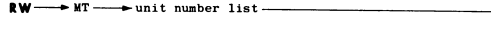


Removes the older of the two disk files when a DUP LIBRARY condition occurs.

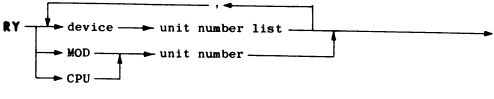


Allows run-time system options to be reset or yields a display of all options that are currently reset.

Removes security restrictions from the indicated card reader.

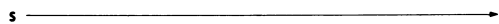


Rewinds and locks the indicated tape unit.

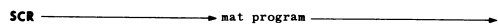


Causes the indicated devices to be made ready for use by the system.

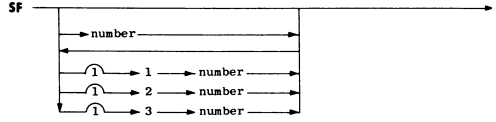
SYSTEM INPUT MESSAGES



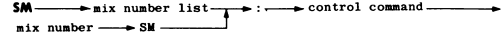
Causes scheduled tasks, which are not in job queues, to be displayed.



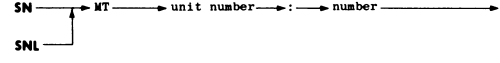
Allows a MAT program to be input from the supervisory console.



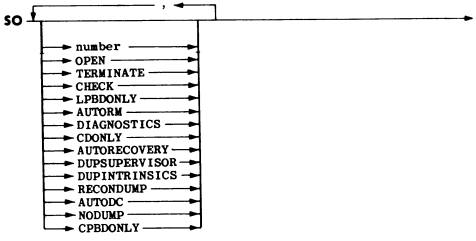
Allows the setting and testing of the memory management parameters.



Allows a control command to be directed at an executing MCS.

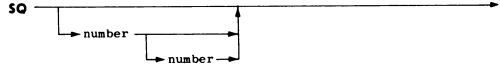


Assigns a serial number to (and optionally locks) the indicated tape unit.

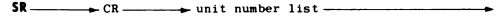


Allows run-time system options to be set or yields a display of all options that are currently set.

Shows which job files are waiting to be printed.

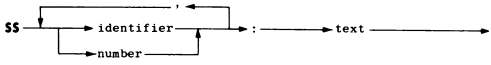


Yields information regarding the contents of one or all of the system job queues.

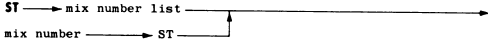


Imposes security restrictions upon the indicated card reader, causing it to reject all decks not preceded by a USER control card.

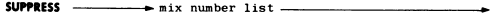
SYSTEM INPUT MESSAGES



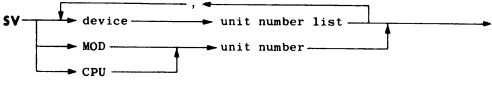
Allows a message to be sent from the supervisory console to a remote terminal operating under control of an MCS.



Causes the temporary suspension of the execution of the indicated job.

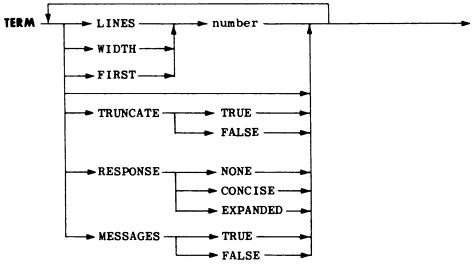


Causes the indicated active job to be suppressed from the job mix display.



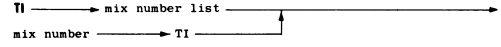
Causes a unit to be made inaccessible to the system.

Initiates code swapping between core memory and swap-area disk by the SWAPPER independent runner.



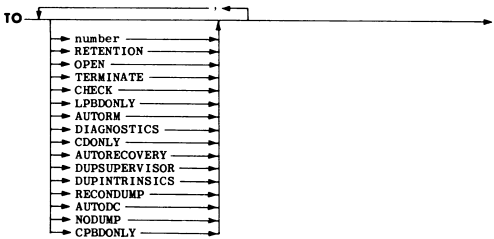
Controls supervisory console display format.

Yields display of the current setting of the three memory management parameters.



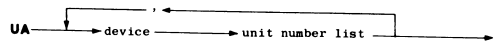
Yields display of elapsed system use times associated with the indicated job.

SYSTEM INPUT MESSAGES



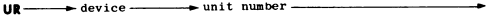
Allows the current state of any or all of the fifteen run-time system options to be checked.

Resets the current time-of-day in use by the MCP. The desired time is given as a four-digit integer, the first two digits of which specify the hour (from 00 to 23) and the last two digits of which specify the minute (from 00 to 59).

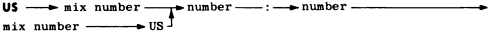


Resets the reliability factor of the indicated peripheral unit to 100%.

Assigns a file as an input file to a program and assumes that the file is unlabeled.

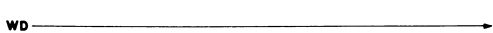


Removes a unit from the system for maintenance; the unit may be restored to the system via the UA message.



Is used in the event of a genealogy mismatch between an internal program file and a tape file; the tape is to be accepted if the values of the CYCLE and VERSION attributes of the tape agree with the first and second numbers, respectively, in the US message.

Yields a display of all suspended tasks waiting for operator intervention.



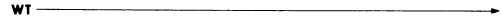
Causes the current date to be displayed in the following form: mm/dd/yy.

Causes the title and status of the current system intrinsics file to be displayed.

SYSTEM INPUT MESSAGES

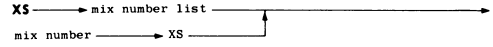
Causes the title and level of the current MCP to be displayed with a list of compile-time options which were set when the MCP was compiled.

Causes the title of the current supervisor program to be displayed.



Causes the current time of day to be displayed.

Is used to eliminate bad spots on head-per-track disk from the disk available table.



Initiates the execution of the indicated scheduled job.

SYSTEM INPUT MESSAGES

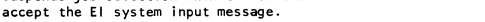
NOTE

The following four messages are called "primitives" and are only to be used for a system failure that requires overriding of the Controller.

Causes a change in the running MCP to the MCP specified by the filename to be immediately initiated without waiting for a zero mix count.

Invokes a non-fatal memory dump.

Suspends job selection when the Controller will not accept the EI system input message.



Facilitates the removal of the JOBDESC file when that job directory is irrecoverably damaged.

DISPLAY OF STATUS

DISPLAY OF STATUS

Disk Directory Table

The disk directory table is displayed in response to the PD keyboard input message. This table contains all file names in the disk directory that are in the set specified by the input message.

A typical directory table in response to the message PD SYMBOL/= follows:

```
PD SYMBOL/=
SYMBOL (DIRECTORY)
. Y (DIRECTORY)
. . MCP (SYMBOL:ESPOL)
. . SORT (SYMBOL:ESPOL)
. . CONTROLLER (SYMBOL:DCALGOL)
. . WFL (SYMBOL:DCALGOL)
. . JOBFORMATTER (SYMBOL:DCALGOL)
. . MAINTENANCE (SYMBOL:ESPOL)
. CCTABLEGEN (SYMBOL:ALGOL)
. DUMPANALYZER (SYMBOL:ALGOL)
. MAKE (DIRECTORY)
. . TAPE (SYMBOL:ALGOL)
. LOADER (SYMBOL:ESPOL)
. PACKDIR (SYMBOL:DCALGOL)
. SCTABLEGEN (SYMBOL:ALGOL)
. BLOCKCHAR (SYMBOL:ALGOL)
. COUNTANALYZER (SYMBOL:ALGOL)
```

If there are no names in the specified set, the message NULL FILE is displayed.

For a complete listing on a line printer of all files stored on disk, the DIR input message is used.

Label Table

The label table is displayed in response to the OL keyboard input message. This table contains an entry for each peripheral unit of the designated type that is on line.

An entry in the label table may consist of the following parts in the order given: the designation of a unit, a file identification, the status of the unit, and a job identification.

A typical entry in the label table in response to the message OL MT97 follows:

```
MT97 P [000003] #1 (1256) 6710Y/FILE000
```

If there are no tape units on line, the reply to the message OL MT is NULL MT TABLE.

Mix Table

The mix table is displayed in response to the MIX keyboard input message. This table contains an entry for each job being executed or recently terminated and for each job that has been suspended. The table is displayed continuously except for brief periods when it is replaced by other tables in response to an input message.

The contents of a job's entry in the mix table depends on whether the job is active (being executed normally or recently terminated) or suspended.

An entry for an active job may consist of the following parts in the order given: a mix index number, a compiler name (if the job is a compilation), a job name, a priority number, and status code (BOJ, EOJ, or DS-ED). The absence of a status code in an entry means that the job referred to is running.

A typical mix table entry for a recently-begun active compilation job follows:

```
0230 JOB  55
..0231  55  COBOL  TASK/A
```

Peripheral Unit Table

The peripheral unit table is displayed in response to the PER keyboard input message. This table contains entries giving the status of each peripheral unit in the system.

A typical display of the peripheral unit table in response to the message PERMT follows:

```
PERMT
----- MT STATUS -----

97 P [000083] #1 6710Y/FILE000
98*P [000000]    S C R A T C H
50*9 [000018] #1 RMC/FILE000
51 9 [000065] #1 CONTROLDECK
52*9 [000052] #1 MEMORY/DUMP
```

SYSTEM OUTPUT MESSAGES

SYSTEM OUTPUT MESSAGES

System output messages are classified in three categories: RSVP, system, and job oriented.

RSVP messages usually refer to a job, and they require operator attention.

System messages are informative in nature and require no special operator response.

Job oriented messages are "no response" informative messages related to a job.

<terminal reference> - This portion of a message gives information concerning the location of job termination.

Syntax:

```
     terminal reference  ::= @ <SEG><PIR><PSR>
where
     <SEG> ::= {segment}
     <PIR> ::= {program instruction register}
     <PSR> ::= {program syllable register}
```

RSVP Messages

RSVP messages are all preceded by <mix index> except for the following:

```
LP, PBT MT RQD <file label><rdc>:<job name>
LP RQD <file label><rdc>:<job name>
MAG TAPE REQUIRED
PBT MT <file label><rdc>:<job name>
PP RQD <file label><rdc>:<job name>
```

ACCEPT {accept data}

The occurrence of this message means that an object program has executed an ACCEPT statement. The operator must respond with an AX input message. The accept data is usually a question such as "how many copies".

<nnnn> DISK SEGMENTS REQUIRED

This message occurs when the amount of disk storage needed by a program exceeds the amount of disk storage that is available for use. Creating space by removing a file or files and entering an OK keyboard input message cause the job to continue.

DUP FIL<file label>

The occurrence of this message means that an object program attempted to gain access to an input file but the MCP found more than one file with the specified <file label>. Processing of the program is suspended until the operator takes action.

The condition can be corrected by making only one of the files available or by entering an IL message. The unit designated in the IL message should be the unit on which the needed input file is located.

DUP LIBRARY<file label>

The occurrence of this message means that an attempt was made to enter a file in the disk library but the label of the file is identical with a label already in the disk directory. The library maintenance process is temporarily suspended until the operator takes corrective action.

The condition may be corrected by using a CHANGE or REMOVE control statement followed by an OK message or by entering an RM or OF message. By the use of the RM message, the disk file with the label specified in the DUP LIBRARY output message is removed.

The same action also occurs for disk pack files.

*IC 20(ZERO) 30(nnnnn);IIHF START

This message is displayed when trying to ready a processor. The values must be stored in these IC Registers, the IIHF flip-flop is set, and START must be pressed to get the Processor on line and to remove the RSVP message. One can also DS the stack "CPURECIPIENT" in response to this message.

INTRINSICS FILE REQUIRED

When a program attempts to call an intrinsic and fails to find an intrinsic file, the program displays the above message and waits for operator intervention.

The operator may then:

a. DS the program or

b. Do a CI on an existing disk file or

c. Copy an intrinsic file to disk and do a CI on it.

LP,PBT MT RQD<file label><rdc>:<job name>

The occurrence of this message means that a program needs a line printer or a printer backup tape and neither is available. The program is

RSVP MESSAGES

suspended until a line printer, backup tape, or scratch tape becomes available or until an OU message is entered to designate an alternative unit, such as disk.

LP RQD <file label><rdc>:<job name>

The occurrence of this message means that a program needs a line printer and none is available. The program is suspended until a line printer becomes available or until an OU message is entered to designate an alternative unit, such as disk.

MAG TAPE REQUIRED

This message indicates that the memory dump procedure of the MCP could not find a tape to dump to and is asking the operator to specify one. The operator responds with the keyboard message MT <nn>.

NO FILE<file label>

The occurrence of this message means that a program needs an input file which is apparently unavailable. If the file is mislabeled it can be made available by, for example, the use of an IL message. If the file is not labeled, a UL message can be used. If it is an optional file, an OF message is needed. If a program has read the final reel of an unlabeled file, an FR message is needed.

NO MEM

The occurrence of this message means that the MCP was unable to obtain needed primary memory for a program. If sufficient memory is not obtained, the operator must enter the DS message for the program.

NO SCRATCH PRINTERS

The occurrence of this message means that line printers are unavailable for system use.

NO USER DISK

The occurrence of this message means that a disk area was needed for a program but sufficient disk space was not available. Under these circumstances, the operator should try to free enough disk by removing some unneeded files from disk. (By entering the message PD/ = it is possible to learn what files are actually on disk.) Then, if enough disk has been freed, the entry of an OK message will cause processing of the program to begin again. If not enough space can be made available, a DS message should be entered for the program.

NULL LP TABLE

The occurrence of this message means that line printers are unavailable for system use.

OPERATOR STOPPED

The occurrence of this message means that the specified job was suspended in response to an ST input message. An OK message is needed if processing is to continue. A DS message can also be entered.

PBT MT RQD <file label><rdc>:<job name>

The occurrence of this message means that a program needs a scratch tape for a printer backup file and none is available. The program is suspended until a scratch tape is made available or until an OU message is entered to designate an alternative unit, such as disk.

PP RQD <file label><rdc>:<job name>

The occurrence of this message means a program needs a paper tape punch and none is available.

<unit mnemonic> READ CHECK

The occurrence of this message means that a read check error occurred on a card reader. The last card read must be reread, or if the error is a check feed error type, the last two cards must be reread. If a second card also fails, the card hole punches may be off or the card reader may need servicing.

RECOPY REQD :<file id>

The responses for this message are OK, DS, OF, and FR for more than one destination tape.

<file label> REQUIRES <unit mnemonic>

The occurrence of this message means that a job required the peripheral device and it is not available.

REQUIRES FM {user message of 25 characters or less}

The occurrence of this message means that a program is ready to open a file for which FILE.FORMMESSAGE has been set. An FM input message is needed if processing is to continue, or a DS may be entered, or an OU input message may be entered for line printer.

REQUIRES SC

The REQUIRES SC (scratch console) message occurs if there are no consoles available. The operator may use an OU message to designate a console.

*<n> SECT REQ ON PK <nnn>

This message indicates sectors are required on a particular disk pack. Acceptable replies to this message are OK, DS or OU input responses.

RSVP MESSAGES

*<nnnnnn> [CLASS <nnn>] SEGMENTS REQUIRED

This message indicates a job has been suspended waiting system disk availability. Disk can be made available by removing files or the job can be OKed to search the available tables again or the job may be DSed. If classed disk is required, disk must be made available from the given class EU.

*WAITING FOR HEADER SPACE

This message indicates the disk file header limit in core memory has been exceeded and the stack will wait for headers to be forgotten and resume or the operator can DS the job.

<nnnn> WORDS REQUIRED

When an attempt is made to make an array present and not enough words are available, the program is suspended with this message. The job can be subsequently OK-ed or DS-ed.

System Messages

Those marked with an asterisk are preceded by <mix index>.

*<file id> ALREADY OPEN<terminal reference>

The occurrence of this message means that a user's program attempted to open a file that was already open.

AN INTEGER WAS EXPECTED

The occurrence of this message means that an integer was expected on a control card (for example, a core-required card or a stack-size card) but was not found.

*ASSIGN OP REQD<terminal reference>

This message occurs on a NAMELIST input, when a character other than an equals sign is encountered following a simple variable name or array name or a subscripted array name. The program is terminated, unless a branch label for a data error is present.

*BAD LIBRARY TAPE

The occurrence of this message means that a library tape has irrecoverable parity errors and that its contents cannot be placed on disk. The library maintenance process will be discontinued, and a DSed message to that effect will be displayed.

*BAD RESIZE/DEALLOCATE<terminal reference>

This message indicates an attempt to resize a segmented or value array and the program is discontinued.

*<unit mnemonic> BEGIN OR END OF TAPE

This message indicates an abnormal beginning or ending of paper tape reader operation occurred.

*<file id> BREAK ON OUTPUT<terminal reference>

This message indicates a break occurred while doing output to a remote device.

*<file id> BLOCK CNT ERR<terminal reference>

The occurrence of this message means that the number of blocks processed by the program did not agree with the count of blocks maintained in the trailer label.

SYSTEM MESSAGES

*<job name> @ <priority> BOJ

The occurrence of this message is an indication that the execution of the named object program has begun.

*<compiler name><job name> @ <priority> BOJ

The occurrence of this message is an indication that a compilation has begun.

<unit mnemonic> BUSY

The occurrence of this message means that an I/O operation was attempted on the specified unit and the unit was apparently busy. This condition indicates a hardware or software failure.

<file label> CHANGED TO <file label>

This message appears after the MCP has performed an operation specified in a CHANGE control statement.

*CMPLX>MAXREAL<terminal reference>

This message occurs on a NAMELIST input, if either the real or the imaginary part of a complex number exceeds the maximum value for a single precision operand. The program is terminated, unless a branch label for a data error is present.

COMPILER NAME NOT FOUND

The name specified (on the compiler card) as a compiler is not in the directory.

CONTROL CARD ERROR <unit mnemonic> {information from control card}

The occurrence of this message means that the MCP had expected to read control information from the designated I/O unit but has found the information to be in error. (For list of control card error messages, see Section 5.)

*<file label> COPIED

This message appears after the MCP has performed the operation specified in a COPY control statement.

*CORE NOT AVAILABLE

This message indicates swapper was not able to get a chunk of contiguous save memory the size of swap core size in SYSTEM/SWAPDISK.

CP RQD <file label><rdc>:<job name>

The occurrence of this message means that a program needs a card punch and none is available. The program is suspended until a card punch becomes available.

*<file id> CREATED

This message is in response to a successful XD input message.

*CRITICAL BLOCK EXIT<terminal reference>

The occurrence of this message means that a program attempted to exit from a block that was in use by a subtask of that program. (IPC message.)

DA

```
<unit mnemonic> <retry data>
  where
  <retry data> ::=
     <I/O operation> DA = <integer>
      ; # SEG = <integer>;
      # RTRY = <integer>;
      # TRNS = <integer>
```

The occurrence of this message means that retries had to be made on the disk file. The <I/O operation> is an R if it was read, W if a write. The <integer> appearing after DA is the disk address; the <integer> appearing after SEG is the number of segments read or written; the <integer> after RTRY is the number of retries necessary; and the <integer> after TRNS is the number of disk transactions since the last Halt-Load operation.

*<file id> DATA ER NO LABL<terminal reference>

The occurrence of this message means that a data error has occurred and that there is no branch label.

DATACOM REQUESTED

The occurrence of this message means that an attempt was made to open files when the DCP was not initialized. The program is terminated.

*DATA NAMELIST<terminal reference>

This message occurs when a character other than a comma separating fields or "& END" terminating the NAMELIST is encountered. The program is terminated unless a branch label for a data error is present.

SYSTEM MESSAGES

DCP <nnn> INITIALIZED

This message is the response to the DC input message which causes the DCP numbered <nnn> to be initialized.

DCP <nnn> NOT ON LINE

The occurrence of this message means that an attempt was made to initialize a DCP that is in local. The DCP being addressed could not be addressed.

DCP <nnn> RUNNING

This message is displayed when an attempt is made to initialize a DCP that has already been initialized and is running.

*DEATH IN THE FAMILY <terminal reference>

When a task is terminated because of an error, the termination of each of its subtasks is signaled by the occurrence of this message. (IPC message.)

DECAY = <unsigned integer> %

This message is the response to a TF (type factor) working set message. <unsigned integer> represents the percent amount of memory that can be forgotten.

DECK <integer> REMOVED

This message appears after the pseudo card deck specified has been removed from disk. The removal of a pseudo card deck from disk can be the result of the completion of a job, the entry of an RD input message, or the entry of a REMOVE DECK control statement.

DECREMENT = <unsigned integer> %

This message is the response to a TF (type factor) working set message. <unsigned integer> represents the percent amount of working set decrement.

*<unit mnemonic> DESCR.ERR

The occurrence of this message means that an error in the I/O descriptor has been encountered by the kernel I/O routines.

DESIRED = <unsigned integer>/MINUTE

This message is the response to a TF (type factor) input message. <unsigned integer> represents number of overlays needed.

*DIMENSION SIZE ERROR I = J <terminal reference>

The occurrence of this message means that a program has requested a nonoverlayable area that is larger than the available space. I is the dimension number and J is the size of the attempted dimension.

DIRECTORY DAMAGED

This message is displayed upon detection of conflicting row addresses at "directory complement" time. It indicates that some directory destruction has occurred. The user may then execute SYSTEM/LISTDIRECTORY and dump any conflicting files/rows. Removal of conflicting files and a subsequent Halt-Load should clear up the situation. If an internal failure is detected, a memory dump is initiated.

DISK ADDRESS ERROR

This is a DFO (disk file optimizer) information message.

DISK PACK INPUT COMMAND MESSAGES

The following messages can occur as a consequence of disk pack input commands (such as PO, RC, IV, etc.):

a. BAD INTERNAL PARAMETER (FROM CONTROLLER)
b. BAD PACKNAME
c. BASE PACK REQUIRED
d. EXISTING SERIAL NUMBER
e. I/O ERROR, IV ABORTED
f. I/O ERROR, PO ABORTED
g. I/O ERROR, RC ABORTED
h. MISSING OR INVALID SERIAL NUMBER
i. NAME REQUIRED
j. PACK CLOSED
k. PACK ERROR
l. PACK IS IN USE
m. PACK SCRATCHED
n. POWERED OFF
o. TOO MANY BAD SECTORS, IV ABORTED
p. UNIT NOT AVAILABLE

DISK PARITY ON LIBRARY MAINTENANCE

The occurrence of this message means that a library maintenance operation was not successfully completed.

DIV BY ZERO BRANCH <job name>, <terminal reference>

The occurrence of this message means that a divide operation using a divisor of 0 was attempted by an object program but a programmatic recovery feature was used.

SYSTEM MESSAGES

DK ERR RD = <nnnnnn>, UNIT = <nnn>, SEG = <nnnnnn>

This message is displayed when a disk failure occurs. The result descriptor (RD), the unit number (UNIT) of the EU, and the segment address (SEG) are provided.

*<program id> DSED <terminal reference>

This message appears after the specified job has been discontinued in response to a DS input message.

DUMP WAS BY : <cause of dump>

This message occurs if a non-fatal memory dump was attempted and the option "NODUMP" is set (option 13).

EOT NO LABEL <file label> : <job name>,<terminal reference>

The occurrence of this message means that an object program has reached the end of the declared area of the designated file on disk. Consequently, the processing of the program has been discontinued.

*<unit mnemonic> ERR RD =<result descriptor>

This message occurs when a non-recoverable error occured on either disk or disk pack. This is an informative message that may be used for further recovery efforts.

<unit mnemonic> ERROR — RW/L

This message occurs when an error occurred in one of several places while trying to write beginning or ending labels.

*<file id> EXCEEDED TIME LIMIT <terminal reference>

This message indicates the timelimit has been exceeded for remote input and the program is discontinued.

<file label> EXPIRED

This message, which occurs at Halt-Load time, refers to files on disk for which the save period has expired.

EXPON OVRFLW BRANCH <job name>,<terminal reference>

The occurrence of this message means that an exponent overflow occurred but that a programmatic recovery feature was used.

FACTOR = <unsigned integer> %

This message is the response to a TF (type the factor) working set input message. <unsigned integer> represents the factor itself, the present value of the ratio of virtual core to actual core.

For non-working set MCP, the response to the TF message is as follows:

```
FACTOR = <unsigned integer> %
```

<unit mnemonic><I/O operation> FAILURE — D <integer>

The occurrence of this message means that one of the following error conditions persisted after ten retries:

<integer> = 19. Parity error between I/O control and core or disk file control.

<integer> = 20. Parity error on transfer from disk.

*FAULT OF BAD TASK ATTRIBUTE <terminal reference>

This message occurs when a fault occurs while attempting to set a task attribute.

*GOING AWAY <terminal reference>

This message occurs when SWAPPER has been DSed and is awaiting all jobs in swapspace to go to end of job.

INACTIVE QUEUE <terminal reference>

This message occurs when an attempt is made to remove a message from a queue in which no message has ever been stored.

INCOMPATIBLE LEVEL

This message indicates that the MCP and a compiler are incompatible, specifically in the area of changes made to the MCP when corresponding changes have not yet been made to the compiler. (The job is DS-ed.)

*ILLEGAL COMPILER <terminal reference>

This message indicates an attempt was made to compile a program with a program that has not been made a compiler via the MC message.

*ILLEGAL VISIT <terminal reference>

This message occurs if the lexographic structure of the task to be visited (continued) does not conform to the visitor's structure.

SYSTEM MESSAGES

INCREMENT = <unsigned integer> %

This message is the response to a TF (type factor) message. <unsigned integer> represents the percentage increment of the working set.

*<unit mnemonic> INCOMPLETE RECORD

This message indicates the paper tape device stopped before the word count has been counted down to 0.

*INCORRECT SYNTAX <terminal reference>

This message results when the NAME or USERCODE task attribute is set to a string that does not conform to the standard naming convention.

*INITIATE ACTIVE TASK <terminal reference>

This message will occur when a program attempts to initiate a task that already has a stack assigned to it.

*INSTR TIMEOUT <terminal reference>

This message occurs when, after 2 seconds, the processor has failed to provide a Syllable Execute Complete Level (SECL).

INTERPROGRAM COMMUNICATION (IPC) MESSAGES (Each IPC message is explained at its own place in the proper section.)

```
MISSING CODE FILE NAME
MISSING CODE FILE
MUST BE CODE FILE
INITIATE ACTIVE TASK
NON-EXTERNAL RUN
VISIT NONACTIVE TASK
ILLEGAL VISIT
NON ANCESTRAL EXCEPTIONEVENT
INCORRECT SYNTAX
STACKNO IS READONLY
TASKTYPE IS READONLY
PROCESSTIME IS READONLY
IOTIME IS READONLY
STARTTIME IS READONLY
STOPPOINT IS STORED
EXCEPTIONEVENT IS READONLY
INVALID COBOL CALL
INVALID COBOL PARAMETER
DEATH IN THE FAMILY
CRITICAL BLOCK EXIT
ILLEGAL BAD GO TO
EXC I/O TIME
EXC PROC TIME
INVALID PARAMETER
TASKERROR IS READONLY
```

PROCESSTYPE IS READONLY
NONEXECUTABLE CODE FILE
PARAMETER MISMATCH
ILLEGAL COMPILER
SECURITY VIOLATION

INTGR OVERFLW BRANCH <job name>,<terminal reference>

The occurrence of this message means that an integer overflow occurred; but that a programmatic recovery feature was used.

<unit mnemonic> IN USE <job id>

This message occurs in response to a PER or an OL input message and indicates that the unit specified is being used by a job.

INTRINSIC FILE NOT LOADED

This message occurs when the MCP attempts to initialize the intrinsic file in response to a CI message, and the file is either not specified or not on disk.

*<unit mnemonic> INVALID CHR. IN COL. <nn>

This message indicates an illegal character has been detected in a column other than 1 on the card reader.

INVALID EOJ <job name>,<terminal reference>

The occurrence of this message means that a COBOL or FORTRAN program attempted to execute the END statement or that a sequence error occurred. Consequently, processing of the program was discontinued.

INVALID INDEX BRANCH <job name>,<terminal reference>

The occurrence of this message means that an invalid index occurred but that a programmatic recovery feature was used.

*INVALID SYSTEM/USERDATAFILE

This message indicates SYSTEM/USERDATAFILE does not have the following attributes:

a. FIXED LENGTH RECORD
b. UNITS ARE WORDS
c. EXTERNAL MODE OF SINGLE DATA
d. NOT CRUNCHED
e. BLOCKSIZE IS 30
f. MAXRECSIZE IS 30
g. ROWSIZE $\geq$ 8
h. FILETYPE OF DATA
i. END OF FILE > 0

SYSTEM MESSAGES

<file label> INVALID UNIT

This error message indicates that an attempt has been made to open output an invalid unit (i.e., pseudoreader).

INV KBD {typed-in information}

The occurrence of this message means that the MCP was not able to recognize a message entered from the keyboard.

<unit mnemonic> INV MEM ADR

The occurrence of this message means that an invalid address occurred when data was to be transferred between an I/O channel and primary memory. The MCP recognizes the error condition and, if possible, rectifies the errors. The primary purpose of this message is to draw attention to a condition which could denote a hardware failure.

*IO ERROR READING TAPE DIRECTORY

The occurrence of this message means that an error was encountered while the directory of library files was being read from tape.

*IO ERR POSITIONING TAPE SOURCE

This message occurs during library maintenance, when spacing to the next input file on tape is going on.

*I/O ERROR ON SYSTEM/USERDATAFILE

This message indicates an I/O error occurred while manipulating SYSTEM/USERDATAFILE.

<unit mnemonic> I/O MEM PAR

The occurrence of this message means that a parity error occurred when data was to be transferred between an I/O channel and primary memory. The MCP recognizes the error condition and, if possible, rectifies the errors. The primary purpose of this message is to draw attention to a condition which could denote a hardware failure.

<unit mnemonic> IRRECOVERABLE WR PARITY

The occurrence of this message means that an irrecoverable parity error occurred on the designated unit.

*<disk unit> IS ON-LINE
NOT READY
READY

These are the three responses to the RYDK <nnn> input message.

LABEL EQUATION ERRORS

0 - Peripheral unit or variant type.

1 - Disk information: speed, access, packed.

2 - Label type.

LIBRARY MAINTENANCE ERROR MESSAGES

The following are messages that result from library maintenance errors:

a. BAD PACK UNIT NUMBER <unit id>
b. TAPE DIRECTORY WRITE ERROR <unit id>
c. TAPE DIRECTORY READ ERROR <unit id>
d. TAPE DIRECTORY NOT FOUND <unit id>
e. RECORD SEQUENCE ERROR <unit id>
f. TAPE NOT LIBRARY TAPE <unit id>
g. TAPE POSITIONING ERROR <unit id>
h. TAPE DIRECTORY COMPARE ERROR <unit id>
i. TAPE DISKHEADER READ ERROR <unit id>
j. UNEXPECTED TAPE DIRECTORY <unit id>
k. FILE TITLE ERROR <unit id>
l. TAPE DISKHEADER WRITE ERROR <unit id>
m. BAD COPY ONTO <unit id>
n. IO ERROR DURING COMPARE <unit id>
o. COMPARE ERROR <unit id>
p. BAD DISK HEADER ON TAPE <unit id>
q. IO ERROR DURING COPY <unit id>
r. <file id> IS NOT A DIRECTORY
s. <file id> NOT ON DISK
t. <file id> NOT ON DISK PACK
u. <file id> IS A SYSTEM FILE
v. <file id> COPIED
w. <file id> ADDED

<file label> LIBRARY MAINTENANCE IGNORED

This occurrence of this message means that the MCP was not, because of some invalid condition, able to perform the library maintenance operation specified on a control card. An example of such a condition is an attempt to remove a nonexistent file.

<file name> (LOADED)

This message is one of the responses to the CI input message. It indicates that the intrinsic file has been loaded.

LOG<integer> %FULL

This message informs the operator how full the log is. The <integer> is some multiple of 5.

SYSTEM MESSAGES

LOG 95% FULL-AUTOMATIC LR

This message tells the operator that the current log is 95 percent full, that the system has automatically renamed and saved it, and that a new system log file has been created.

%<unit mnemonic>LOW ON PAPER TAPE

This message occurs when a paper tape punch has run low on paper tape.

LP,PBT MT RQD <file label><rdc>:<job name>

The occurrence of this message means that a program needs a line printer or a printer backup tape and neither is available. The program is suspended until a line printer, backup tape, or scratch tape becomes available or until an OU message is entered to designate an alternative unit, such as disk.

LP RQD <file label><rdc>:<job name>

The occurrence of this message means that a program needs a line printer and none is available. The program is suspended until a line printer becomes available or until an OU message is entered to designate an alternative unit, such as disk.

MAXIMUM =<unsigned integer> %

This message is the response to a TF (type factor) message. <unsigned integer> represents the maximum percentage of core memory needed in the working set.

MAXIMUM PSEUDOREADERS =<integer>

This message occurs in response to the RN input message.

MAX COPIES OF AUTOPRINT =<integer>

This message occurs in response to the AP input message.

*MAX SLICE NUMBER NOT VALID

This message indicates the sixth word (maximum slice number) is outside the range 0-255.

MCP CHANGE ABORTED (NOT ON DISK)

This message is displayed in response to the CM input message if the needed MCP file is not on disk.

MCP CHANGE ABORTED (NOT ESPOL CODE)

This message is displayed in response to the CM input message if the MCP file is not an ESPOL code file.

MCP CHANGE PENDING

This message is one of the responses of the system to a CM input message, if the needed MCP file is on disk.

*<unit mnemonic> MEMORY ACCESS ERROR

This message indicates a memory access error occurred on paper tape.

MEMORY MODULES <ready memory list> READY

This message is displayed in response to the MM input message.

Example:

MEMORY MODULES 0, 2-5 READY

<unit mnemonic> MEM PROTECT ERR

This message indicates an attempt at storing in a memory protected word (bit 48) while performing an I/O operation. This usually indicates a failure in the hardware.

MINIMUM = <unsigned integer> %

This message is the response to a TF (type factor) message. <unsigned integer> represents the minimum percentage of core memory needed in the working set.

*MIN TIME SLICE NOT VALID

This message occurs when the fifth word (minimum time slice) of system/swapdisk is less than 0.

*MISSING ROW

This message occurs when a row address in the file SYSTEM/SWAPDISK is missing.

SYSTEM MESSAGES

*MOD#<nn> NOT READIED :<msg>

<msg> = 1. NOT ON LINE
2. ADDRESS LINE FAILURE
3. ZERO-TEST FAILURE
4. DROPPING OPERAND BITS
5. DROPPING TAG BIT (49)
6. DROPPING TAG BIT (50)
7. DROPPING TAG BIT (51)
8. NOT VISIBLE BY ALL PROCS
9. NOT VISIBLE BY ALL MPXS

MPX PARITY ERR

This message occurs when the system detects the transfer of an even number of bits between the processor and multiplexor.

MT RQD <file label><rdc>:<job name>

The occurrence of this message means that a program needs a scratch tape for a tape file.

NAMELIST ERRORS (Each NAMELIST error message is explained at its own place in the proper section.)

INVLD VARIABLE	INVLD INDEX
NUM>INTEGER	INVLD SUBSCRIPT
DATA-NAMELIST	INVLD LT PAREN
ID EXPECTED	INVLD NUMTYPE
NUMBER REQD	ASSIGN OP REQD
RT PAREN REQD	OUTPUT LIST REQD
INVLD SIGN	(UNFORMATTED I/O)
INVLD ASTERISK	NVLD CW-UNFORMATTED IO
INVLD RPT FLD	LIST EXCEEDS REC. SZ
INVLD EXPONENT	OUTPUT LIST TOO LONG
COMPLX>MAXREAL	

*NAME READONLY ON ACTIVE TASK<terminal reference>

This message indicates an attempt was made to set the task attribute NAME to an active task.

NEW MCP PB ON<unit mnemonic>

This message indicates that a new printer backup file has been opened on the unit indicated. (Produced when the OPEN option is set.)

<unit mnemonic> NEW PBT

The occurrence of this message means that a new printer backup tape was opened. (Produced when the OPEN option is set.)

NO ACCESS TO EXCHANGE

This is a disk file optimizer (DFO) error message. DFO operation is cut off.

NO DCP CODE

This message occurs when an NDL description has been created but DCPROGEN has not been run against it to create DCP code.

NO FILE <vol id> / FILE000

The occurrence of this message means that an attempt was made to copy files from a library tape that was not available to the system.

NO FILES ADDED

This message is displayed when all user-specified files are already present on disk.

NO FILES COPIED

This message is displayed when no user-specified files can be located on disk or tape.

NO VALID CHARACTER

This message indicates that the system expected to read a control card through the card reader, but the card that was read did not have an invalid character punched in column 1.

NO NDL DESCR

This message occurs when the DCP that one is trying to initialize was not described in the NDL compilation.

*NON ANCESTRAL EXCEPTIONEVENT <terminal reference>

This message will occur if one tries to use an EXCEPTIONEVENT that is not within your ancestral addressing range.

*NON EXECUTABLE CODE FILE <terminal reference>

This message means an attempt was made to execute a file that is not a code file.

*NON-EXTERNAL RUN <terminal reference>

This message will occur if a program attempts to RUN an internal procedure.

SYSTEM MESSAGES

*NOT ALLOWED—SWAPSPACE <terminal reference>

This message occurs when a user task is attempted to be initiated in SWAPSPACE and the program is discontinued.

<file label> NOT CHANGED (NOT ON DISK)

This message occurs in response to a change control statement, if the <file label> to be changed is not in the disk directory (that is, if the file is not on disk).

<file label> NOT CHANGED (SYSTEM FILE)

This message occurs in response to an attempt to change the name of a system file (such as SYSTEMDIRECTORY). Such a change is not permitted.

<file label> NOT COPIED -- DIRECTORY

This message indicates that a file was not copied to a library tape, because it was a directory.

<file label> NOT COPIED -- NOT ON <tape or disk>

The occurrence of this message means that the library maintenance process could not locate a file it was to copy.

<file label> NOT IN DIRECTORY

The occurrence of this message means that a control statement referenced a file that was not in the disk directory.

<program id> NOT IN DIRECTORY

The occurrence of this message means that an EXECUTE, RUN, or COMPILE statement referenced a program that was not in the disk directory.

<file name> (NOT LOADED)

This message is one of the responses to the WI input message. It indicates that the intrinsic file has not been loaded.

*<dcp number> NOT ON LINE

This message indicates the DCP has not responded to a system attention scanout and is not on line.

*<unit mnemonic> NOT READY

The occurrence of this message means that an I/O operation was attempted on a unit that was not ready. This condition could be an indication of a hardware or software failure or simply an indication of the fact that the unit is in local.

<unit mnemonic> NOT READY EU

The occurrence of this message means that an I/O operation was attempted on a unit of disk but that the disk file electronics unit was not ready. This condition would indicate a hardware or software failure.

<file label> NOT REMOVED (NOT ON DISK)

This message indicates that an attempt was made to remove a file that is not on disk (that is, whose name is not in the disk directory).

<file label> NOT REMOVED (SECURITY ERROR)

This message occurs in response to an illegal attempt to remove a file under security.

<file label> NOT REMOVED (SYSTEM FILE)

This message occurs in response to an illegal attempt to remove a system file (for example, WORKAREA or SYSTEMDIRECTORY).

*NUMBER REQD <terminal reference>

This message occurs when NAMELIST encounters on input a variable name (ID) and equals sign but encounters a separator, terminator, or another variable name before seeing the value. The program is terminated, unless a branch label for a data error is present.

*<unit mnemonic> OUT OF PAPER

This message indicates a line printer has sensed a break in the paper indicating it has run out of paper.

*OUT OF SWAP DISK

This message indicates SWAPPER is out of swap disk for the current mix situation and will generally require SWAPPER being DSed.

*PARAMETER MISMATCH <terminal reference>

This message indicates an incorrect number or kind of parameters were attempted to be passed to an external procedure.

*<unit mnemonic> PARITY

This message occurs on a paper tape reader only when the MCP compile time option REVERSE PAPERTAPE is reset and paper tape parity retry cannot be tried. With a paper tape punch and magnetic tape, a parity occurred while writing.

SYSTEM MESSAGES

PARITY ON <unit mnemonic>

The occurrence of this message means that the MCP received an irrecoverable parity condition while reading the label or scanning down a multi-file volume.

*<file id> PAR ON POSITION <terminal reference>

The occurrence of this message means that an error was encountered during an attempt to position a file in order to open it.

*PK <nnn> DIRECTORY DAMAGED

This message indicates there are overlapping files on the given disk pack.

*PROCESSOR <n> (ON/OFF) LINE

This message results when a processor is successfully readied or saved.

*<unit mnemonic> PUNCH CHECK

The occurrence of this message means that an irrecoverable punch check error occurred during the punching of a card. Processing continues, but the card punch may need servicing.

*<unit mnemonic> PURGED

This message indicates the unit has been successfully purged.

*<file id> QTED : BAD DATA

This message results when AUTOPRINT gets a fault while trying to print a backup file and automatically QT's the file.

RANGE = <unsigned integer> %

This message is the response to a TF (type factor) working set message. <unsigned integer> is the percentage overlay range needed.

READ ERROR FOR {control card information}

The occurrence of this message means that a read error, probably irrecoverable parity, occurred during the reading of a control deck for the disk. The control card which is printed denotes the deck which will be deleted. The decks that follow will be loaded normally.

<unit mnemonic> READ PARITY

This message indicates that a parity error occurred while an attempt was being made to read a tape label.

RECONSTRUCTION DATA ERROR

This message indicates a data error occurred while RECONSTRUCTING to a backup EU and the resulting reconstructed file structure may be corrupt.

RECORD SEQUENCE ERROR

This is a library maintenance error message. It occurs when a sequence error is discovered when sequence checking for source tape copies.

<file label> REMOVED

This message appears after an operation specified in a REMOVE control statement has been completed.

RESERVE ERROR MESSAGES

The following is a list of error conditions and messages displayed by the XD, RESERVE, and RETURN input messages:

a. SEG ADR GTR MAX EU ADR
b. ILLEGAL SU/SW NUMBER
c. DELIMITER EXPECTED
d. NUMBER EXPECTED
e. INVALID UNIT
f. ONLY APPLICABLE TO DISK UNITS
g. UNIT NOT READY
h. SECOND # MUST BE GTR THAN FIRST #
i. INVALID ALTERNATE EU SYNTAX
j. ONLY ONE AREA ALLOWED
k. CANNOT RESERVE MCP CODE FILE AREA
l. MAXIMUM BACK UP EUs EXIST
m. INCOMPATIBLE EU CLASSES
n. PREVIOUS RESERVE INCOMPLETE
o. ATTEMPT TO RESERVE IAD AREA
p. ONLY IAD AREAS MAY BE RETURNED
q. CANNOT BE XD-ED
r. WAITING ON: JOB <mix #>
s. DIR PAR
t. IN IAD AREA
u. INSUFFICIENT RESERVE SPACE

<tape label> RET

This message indicates that a tape that is being saved has an expired save factor. If the tape were not being saved, it would have been purged automatically. (Produced when the RET option is set.)

SYSTEM MESSAGES

*<dcp number> RUNNING

This message indicates the said DCP has been successfully initialized.

<unit mnemonic> RW/L

The occurrence of this message means that a tape has been rewound and locked.

<unit mnemonic> SAVED

This message is a response of the system to a SV input message.

*SCAN PARITY <terminal reference>

This message occurs if during a SCAN-IN or SCAN-OUT operation to the scan bus, a parity error is detected. Processing of the program is discontinued.

<unit mnemonic> SCRATCH

The occurrence of this message means that a tape was purged by an input message or a program.

*SEQUENCE ERROR <terminal reference>

This message indicates that there is a stack linkage problem.

*SET CONDITIONAL HALT SWITCH FOR CPU <n>

This message results when an attempt is made to save a CPU <n> and the CONDITIONAL HALT switch on the MDL for this processor is OFF.

*SLICING RATIO NOT VALID

This message indicates word seven (slice ratio) of SYSTEM/SWAPDISK is outside of the range 0 or $\geq$ 1.

*<unit mnemonic> SN REQUIRED

This message indicates an attempt was made to purge a tape that did not have a serial number and the operator must scratch the tape with the SN message.

*STACK BOTTOM ERR <terminal reference>

This message occurs when an attempt is made to cut back the stack below BOSR.

STACK PARITY

This is a disk file optimizer (DFO) error message. DFO operation is cut off.

SU NOT AVAILABLE

This is a disk file optimizer (DFO) error informative message.

*SWAPCORE NOT VALID

This message means the fourth word SWAP CORE SIZE is not a multiple of the core slot size (i.e., 990 words).

*SWAP DISK NOT VALID

This message results when the rowsize of SYSTEM/SWAPDISK is not a multiple of the disk slot size (i.e., 44 segments).

*SYSTEM/USERDATAFILE IS FULL

This message occurs while trying to create or change user files, and the system has run out of disk for SYSTEM/USERDATAFILE.

*SYSTEM/SWAPDISK NOT THERE

This message results when a disk file SYSTEM/SWAPDISK is not in the directory and an SW message is entered.

TAPE DIRECTORY NOT FOUND

This is a library maintenance message. It occurs when sequence checking for source tape copies.

<unit mnemonic> TO BE SAVED

This message indicates that the unit specified in the SV input message is currently in use.

UNEXPECTED TAPE DIRECTORY

This is a library maintenance error message. It occurs when sequence checking for source tape copies.

UNIMPLEMENTED CONSTRUCT

This message occurs when a program attempts to use a construct which is not implemented, such as a dynamic own array.

*<unit mnemonic> UNIT NOT READY

This message indicates the unit went NOT READY while trying to write the beginning label.

SYSTEM MESSAGES

*UP LEVEL TASK ASSIGNMENT

This message results when an attempt is made to assign to one's EXCEPTIONTASK a task of incompatible lexocographic levels.

*VALIDITY CHECK FAILURE

This message indicates the first three words of the file SYSTEM/SWAPDISK do not contain the string 8"SYSTEM/SWAPDISK. ".

*VISIT NONACTIVE TASK

This message occurs when a program attempts to visit (continue) a non-active task.

*<unit mnemonic> WRITE ERROR

This message indicates some sort of non-parity write error (e.g., memory protect, memory access) occurred while trying to write the beginning label.

*<unit mnemonic> WRITE PARITY ERROR

This message indicates a write parity occurred while trying to write the beginning label.

Job Oriented Messages

Those marked with an asterisk are preceded by <mix index>.

<file name> ADDED

This message is displayed when a file has been added from tape to disk.

*ARRAY TOO LARGE <terminal reference>

This message indicates that an attempt has been made to make present an array that is larger than the overlay row size of the MCP. This can be avoided by segmenting the array. Processing of the program is discontinued.

DISK PACK ERROR MESSAGES

The following list shows the disk pack error messages:

a. <file id> CONTAINS ABNORMALLY CLOSED FILES

b. <file id> PK MISSING ROW

c. <file id> REQUIRES PK

d. <file id> REQUIRES PK WITH SN# <n>

*DISPLAY : {25 characters of text}

The word DISPLAY preceding a string of text indicates that the string of text has been generated by a user program, not by the MCP.

*DIVIDE BY ZERO <terminal reference>

The occurrence of this message means that a divide operation using a divisor of 0 was attempted by an object program. Consequently, the processing of that program was discontinued.

*<file id> EOF NO LABEL <terminal reference>

The occurrence of this message means that the end of an input file to an object program was reached and that there was no specification as to the action to be taken. Consequently, the processing of the program was discontinued.

*<job name> @ <priority> * EOJ

The occurrence of this message is an indication that the execution of the named object program has come to a normal end.

JOB ORIENTED MESSAGES

*<compiler name><job name> @ <priority> * EOJ

The occurrence of this message is an indication that a compilation has come to a normal end, with no syntax errors.

*EXCEPTIONEVENT IS READONLY <terminal reference>

The occurrence of this message means that an attempt was made to store a value in the EXCEPTIONEVENT task attribute of a task variable. (IPC message.)

*EXCHANGE ABORTED—UNEQUAL ROW SIZES

This message results when attempting to exchange disk rows and the rowsizes are unequal.

*EXC I/O TIME<terminal reference>

This message occurs when the I/O time of a job exceeds the limit for I/O time specified for the job. (IPC message.)

*EXC PROC TIME<terminal reference>

This message occurs when the processor time of a job exceeds the limit for processor time specified for the job. (IPC message.)

*EXPON OVERFLOW <terminal reference>

The occurrence of this message means that an object program performed an operation that caused an exponent overflow to occur. Consequently, the processing of the program was discontinued.

*EXPON UNDERFLO <terminal reference>

This message occurs when a value less than the minimum value for the exponent part of a real number is encountered.

FILE CLOSE ERRORS

0 - File not open.

1 - Irrecoverable I/O error during close processing.

2 - Record count error.

3 - Block count error.

4 - Logic error.

*<file id> FILE NOT OPEN <terminal reference>

The occurrence of this message means that an attempt was made to close a file that had not been opened.

FILE OPEN ERRORS

0 - File already open (OPEN called when file open).

1 - Parity on label.

2 - Parity while positioning file.

3 - Illegal translation.

4 - Incompatible blocking specification.

5 - Output reverse is not allowed.

6 - Illegal input reverse.

7 - Minimum tape block size is 3 words. (Even parity 9 track tape.)

8 - Bad label type.

9 - Invalid OPEN code: MYUSE = 0.

10 - Invalid input code.

11 - Invalid output code.

12 - Invalid I/O code.

13 - COBOL random file with no actual key.

14 - Open output on illegal output file. (Directory, code, crunched, duplicated, conflict.)

15 - Bad record type.

16 - No memory for buffers.

17 - Header too small for GUARDFILE name.

18 - Error while setting up duplicated file list.

19 - Invalid direct I/O file.

20 - Row size exceeds cylinder size (disk pack).

JOB ORIENTED MESSAGES

*ID EXPECTED <terminal reference>

This message occurs when NAMELIST encounters a value or input for initialization of a variable without having seen the variable in which to store that value. The program is terminated, unless a branch label for a data error is present.

*ILLEGAL BAD GO TO <terminal reference>

This message occurs when an attempt is made to go to a label which is in another stack. (IPC message.)

*ILLEGAL OWN ARRAY <terminal reference>

This message is the result of an attempt to "process" or "call" a procedure which contains OWN array declarations to be initiated as a dependent process.

<job id> ILLEGAL SWAP <terminal reference>

This message is a process fatal error message and is displayed if the following conditions are not met:

1. Both arrays must have same dimensions,
2. Both descriptors must describe dopevectors of the same size,
3. Both descriptors must describe whole arrays, not subarrays,
4. Both descriptors must reside in the same stack.

*ILLEGAL VISIT <terminal reference>

This message occurs when an illegal attempt has been made to "continue" to another task. (IPC message.)

*<file id> INCOMPAT BLKING <terminal reference>

The occurrence of this message means that the actual blocking factor of the file does not coincide with that specified in the program.

*INCORRECT SYNTAX <terminal reference>

The occurrence of this message means that an attempt was made to replace a pointer-valued task attribute by a string having incorrect syntax. (IPC message.)

*INITIATE ACTIVE TASK <terminal reference>

The occurrence of this message means that an attempt was made to reuse a task when the task was still active.

*INTGR OVERFLOW <terminal reference>

The occurrence of this message means that an object program performed an operation that caused an integer overflow to occur. Consequently, the processing of the program was discontinued.

*INTRINSIC <intrinsic number> MISSING

This message occurs when a program attempts to call an intrinsic whose <intrinsic number> is not found in the intrinsic file. This is a fatal error; job terminates.

*INV ADDRESS <terminal reference>

The occurrence of this message means that an object program performed an operation which addressed a nonexistent memory location. Consequently, the processing of the program was discontinued.

INVALID ADDRESS

The occurrence of this message means that an attempt to gain access to an invalid address occurred. Processing of the job is discontinued.

INVALID ARGUMENT TO A MATHEMATICAL INTRINSIC FUNCTION

<mix index> <function name> <invalid arg> <parameter value> <terminal reference>

The occurrence of this message means that an invalid argument to a mathematical intrinsic function has been encountered. The program is terminated and the message is displayed to the operator. In addition, the message is recorded in the system log. If a printer file is open at the time the error occurs, the message is also written on the printer file. The following intrinsic functions may generate an invalid argument message:

ALGAMA	COS	DSIN
ALOG	COSH	DSQRT
ALOG10	COTAN	ERF
ARCOS	CSIN	EXP
ARSIN	CSQRT	GAMMA
ATAN	DATAN	SIN
ATAN2	DATAN2	SINH
CABS	DCOS	SQRT
CCOS	DEXP	TAN
CEXP	DLOG	TANH
CLOG	DLOG10	XTOI

JOB ORIENTED MESSAGES

*<unit mnemonic> INVALID CHR. IN COL. <integer>

The occurrence of this message means that an invalid character has appeared in a position other than character position 1 of a record. The <integer> denotes the column number.

*INV INDEX <terminal reference>

The occurrence of this message means that an object program attempted to index out of the range of an array being referenced. Consequently, processing of the program was discontinued.

*INV PROG SYL <terminal reference>

This message means either an invalid tag configuration in the P Register or the Program Controller detected an invalid instruction (i.e., two consecutive VARI operators (OP = 95) or an attempt to strobe families A, B, C, D, E, J or K while in edit mode). Processing of the program is discontinued.

*INVLD ASTERISK <terminal reference>

This message occurs on NAMELIST input when an asterisk has been found out of context for a repeat field. The program is terminated, unless a branch label for a data error is present.

*INVLD EXPONENT <terminal reference>

This message occurs on NAMELIST input if the exponent specified is explicitly greater than 32,767. The program is terminated, unless a branch label for a data error is present.

*INVLD INDEX <terminal reference>

This message occurs on NAMELIST input when too many values are specified for initializing an array. The program is terminated, unless a branch label for a data error is present.

*INVLD LT PAREN <terminal reference>

This message occurs on NAMELIST input, if a left parenthesis follows a simple variable. The program is terminated, unless a branch label for a data error is present.

*INVLD NUMTYPE <terminal reference>

This message occurs on NAMELIST input in the following three cases:

1. Storing logical to other than logical.
2. Storing double precision to other than double precision.
3. Storing complex to other than complex.

The program is terminated, unless a branch label for a data error is present.

*INVLD RPT FLD <terminal reference>

This message occurs on NAMELIST input if a zero or negative repeat field has been specified. The program is terminated, unless a branch label for a data error is present.

*INVLD SIGN <terminal reference>

This message occurs on NAMELIST input if signed data is assigned to a logical variable or string constant. The program is terminated, unless a branch label for a data error is present.

*INVLD SUBSCRPT<terminal reference>

This message occurs on NAMELIST input if the number of subscripts for an array name, when an attempt is being made to store a value, does not agree with the number of subscripts in the declaration. The program is terminated, unless a branch label for a data error is present.

*INVLD VARIABLE<terminal reference>

This message occurs when NAMELIST finds an identifier not in the NAMELIST declaration. The program is terminated, unless a branch label for a data error is present.

*INV OPERATOR<terminal reference>

The occurrence of this message means that a processor instruction has attempted to use the wrong type of control word or data.

*<file id> INV OPN OUT REV<terminal reference>

The occurrence of this message means that an attempt has been made to open an output file reverse.

*<file id> INV PER LBLEQTN<terminal reference>

The occurrence of this message means that an attempt has been made to perform a label equation to an incompatible unit type.

*<file id> INV TRANSLATION<terminal reference>

This message occurs when an attempt is made to open a file for which translation from external to internal, or vice versa, is not provided by the system hardware.

*<file id> IO ERROR IN CLOSE<terminal reference>

The occurrence of this message means that an error was encountered while an attempt was being made to write a trailer label or to position a file.

JOB ORIENTED MESSAGES

*IO ERROR READING DISKHDR FROM TAPE

The occurrence of this message means that an error was encountered while a disk file header was being read from a source (library maintenance) tape.

I/O ERRORS

0 - Read on a file after end-of-file action.

1 - Read on a file that has not been properly opened.

2 - Read parity (no use routine).

3 - Write on a file that has not been properly opened.

4 - No label for end-of-file (or end-of-table).

5 - No label for parity.

6 - No label for size error.

7 - Bad unitfeature word.

8 - Illegal read reverse.

9 - General principles.

10 - Seek for FORMATU U.

11 - Seek on unopened COBOL file.

12 - Seek on invalid file type.

13 - Space forward on output file.

14 - Illegal space backward.

15 - Parity while positioning during seek.

16 - COBOL disk seek on one buffer file.

17 - Direct I/O using segmented array is illegal.

18 - Unassigned.

19 - Unassigned.

20 - Bad control block (CB) number.

21 - Wait on free CB.

22 - Disk file security error.

I/O ERRORS (Cont)

23 - I/O on CB in use.

24 - I/O on free CB (MCP).

25 - I/O on locked event.

*IO ERROR WRITING DISKHDR TO DEST [n]

This message indicates that an error occurred while a disk file header was being written to output unit n.

*IOTIME IS READONLY<terminal reference>

The occurrence of this message means that a program attempted to store a value in the IOTIME task attribute of a task variable. (IPC message.)

*IRRECOVERABLE PARITY ERR DURING COPY ON DEST [n]

The occurrence of this message means that an irrecoverable parity error occurred during library maintenance on output copy n.

*IRRECOVERABLE PARITY ERR DURING COPY ON SOURCE

The occurrence of this message means that an irrecoverable parity error occurred during library maintenance on the source file.

*IRRECOVERABLE PARITY ERROR ON DIRECTORY DEST[n]

The occurrence of this message means that a parity error was encountered in writing a tape directory on the destination specified.

{item on a control card} IS APPARENTLY MISSPELLED

This message indicates that the item apparently misspelled is not recognized as a valid word in a control card.

{reserved word} IS USED IMPROPERLY

The occurrence of this message means that a syntactically incorrect use of a reserved word was encountered on a control card.

*LIST EXCEEDS REC SZ<terminal reference>

This message occurs on input, when the size of the list is greater than the record size and the file is a FORTRAN linked file. The program is terminated unconditionally.

JOB ORIENTED MESSAGES

LOGICAL I/O ERRORS

a. *ILLEGAL BACKWARD SEEK <file id>

b. *ILLEGAL READ REVERSE <file id>

c. *ILLEGAL SEEK <file id>

d. *LOGIC ERROR <file id>

e. *PARITY SEEK <file id>

f. *READ ON OUTPUT FILE <file id>

g. *READ ON UNOPENED FILE <file id> (COBOL only)

h. *READ REVERSE ON UNOPENED FILE <file id> (COBOL only)

i. *SEEK ON UNOPENED FILE <file id> (COBOL only)

j. *SPACE FORWARD ON OUTPUT FILE <file id>

k. *UNEXP IO ERR

l. *WRITE ON INPUT FILE <file id>

m. *WRITE ON NON-DATA FILE <file id>

n. *WRITE ON UNOPENED FILE <file id> (COBOL only)

*MEMORY EXCEEDED <terminal reference>

This message indicates a task in swapspace attempted to use more memory than was available in SWAPSPACE and the task is discontinued.

*MEMORY PARITY <terminal reference>

This message is displayed when the system detects the transmission of an even number of bits between the processor and memory. Processing of the program is discontinued.

*MEMORY PROTECT <terminal reference>

The occurrence of this message means a program tried to do a store operation into a protected word (bit 48 ON). Processing of the program is discontinued.

*MISSING CODE FILE NAME <terminal reference>

The occurrence of this message means that an attempt was made to initiate an external task without naming the code file for the task. (IPC message.)

*MSG SIZE ERROR <terminal reference>

This message indicates an attempt was made to allocate a message larger than the rowsize of the message areas (512 default rowsize).

*MUST BE CODE FILE<terminal reference>

The occurrence of this message means that in an attempt to initiate an external task the file was found but it was not a code file. (IPC message.)

NAMELIST ERRORS (Each NAMELIST error message is explained at its own place in the proper section.)

INVLD VARIABLE	INVLD INDEX
NUM>INTEGER	INVLD SUBSCRIPT
DATA-NAMELIST	INVLD LT PAREN
ID EXPECTED	INVLD NUMTYPE
NUMBER REQD	ASSIGN OP REQD
RT PAREN REQD	OUTPUT LIST REQD
INVLD SIGN	(UNFORMATTED I/O)
INVLD ASTERISK	NVLD CW-UNFORMATTED IO
INVLD RPT FLD	LIST EXCEEDS REC. SZ
INVLD EXPONENT	OUTPUT LIST TOO LONG
COMPLX>MAXREAL	

*NEW INPUT ON <unit specifier>

This message indicates that an input file has been opened for a job. The message occurs only if the OPEN option is set.

*NEW OUTPUT ON <unit specifier>

This message indicates that an output file has been opened for a job. The message occurs only if the OPEN option is set.

*NO FILE<file label>

The occurrence of this message means that a program needs an input file which is apparently unavailable. If the file is labeled, it must be made available by, for example, the use of an IL message. If the file is not labeled, a UL message can be used. If it is an optional file, an OF message is needed. If a program has read the final reel of a multivolume unlabeled file, an FR message is needed.

*NON ANCESTRAL EXCEPTIONEVENT<terminal reference>

The occurrence of this message means that a task attempted to reference the EXCEPTIONEVENT of a task variable for some task other than itself or a direct ancestor. (IPC message.)

JOB ORIENTED MESSAGES

*NON-EXTERNAL RUN<terminal reference>

The occurrence of this message means that an attempt was made to run something other than an external procedure. (IPC message.)

*<procedure name> NOT BOUND<terminal reference>

This message indicates an attempt was made to call a procedure that has not been bound into the program and the job is terminated.

*NUM>INTEGER<terminal reference>

This message occurs on a NAMELIST read, if the variable type is INTEGER and the number type to be stored in the variable exceeds the maximum value for an INTEGER. The program is terminated, unless a branch label for a data error is present.

*NVLD CW-UNFORMATTED IO<terminal reference>

This message occurs when an attempt is made to read unformatted data that has been written without specifying the FORTRAN option LINKWORD or when the file has been mispositioned. The program is terminated unconditionally.

*OPERATOR DSED<terminal reference>

This message indicates the site operator has DSed the job thru SPO input, and the program is discontinued.

*OUTPUT LIST REQD<terminal reference>

This message occurs when an unformatted write statement contains no list. The program is terminated unconditionally.

*OUTPUT LIST TOO LONG<terminal reference>

This message occurs on output when the list is greater than the record size and the file is not a FORTRAN linked file. The program is terminated unconditionally.

*<file ID> PARITY ON LABEL<terminal reference>

The occurrence of this message means that a parity error was encountered when an attempt was being made to write a label.

*PARITY ON PRESENCE BIT<terminal reference>

This message indicates a disk parity error occurred while trying to make present a program code or overlayable array area, and the job is discontinued.

JOB ORIENTED MESSAGES

*<file ID> PAR NO LABEL<terminal reference>

The occurrence of this message means that an irrecoverable parity occurred on the designated file and no programmatic recovery facility was specified. Consequently, processing of the program in question was discontinued.

*PRGMD OP ERR<terminal reference>

This message occurs if an attempt is made to execute an invalid primary, family F or family G operator code. Processing of the program is discontinued.

*PRINT LIMIT EXCEEDED<terminal reference>

This message occurs when the print limit of a job is specified and has been exceeded and the job is discontinued.

*PROCESSTIME IS READONLY<terminal reference>

The occurrence of this message means that an attempt was made to store a value in the PROCESSTIME task attribute of a task variable. (IPC message.)

*PUNCH LIMIT EXCEEDED<terminal reference>

This message indicates the punch limit of a job is specified and has been exceeded. The job is discontinued.

*QUEUE ATTRIBUTE ERROR:<n>

Where n is defined as:

0 - ACTIVE
1 - SECURE
2 - MEMORYLIMIT
3 - MEMORYSIZE
4 - SIZE
5 - MESSAGECOUNT
6 - USERCOUNT
7 - TANK
8 - INSERTEVENT
9 - HEADSIZE

*<file ID> READ AFTER EOF<terminal reference>

This message indicates that a user's COBOL program attempted to read a file after the end-of-file had been sensed.

JOB ORIENTED MESSAGES

*<file ID> READ BEFORE OPEN<terminal reference>

This message indicates that a user's COBOL program attempted to read a file before it had been opened.

READONLY TASK ATTRIBUTE

The following messages will occur when an attempt is made to set a READONLY task attribute:

a. BDNAME READONLY ON ACTIVE TASK

b. EXCEPTIONEVENT IS READONLY

c. FILECARDS READONLY ON ACTIVE TASK

d. HISTORY READONLY

e. IOTIME IS READONLY

f. PROCESSTIME IS READONLY

g. PROCESSTYPE IS READONLY

h. STACKHISTORY IS READONLY

i. STACKNO IS READONLY

j. STACKSIZE READONLY ON ACTIVE TASK

k. STARTTIME IS READONLY

l. STOPPOINT STORED

m. TASKERROR IS READONLY

n. TASKFILE IS READONLY

o. TASKTYPE IS READONLY

*RT PAREN REQD<terminal reference>

This message occurs on NAMELIST input when a right parenthesis is missing following a complex value or subscript list for an array. The program is terminated, unless a branch label for a data error is present.

*<job ID> @<priority><core requirement> SCHD

This message indicates that a job has been scheduled for execution but has not yet gotten a BOJ. The <core requirement> is the percentage (in tenths of a percent) of the total memory configuration needed to run the job.

*SEG ARRAY ERR<terminal reference>

The segmented array error message is displayed when a program attempts to make an access to data outside the range of an area descriptor by the use of the string operators (SCAN, TRANSFER, COMPARE, or SKIP).

This condition indicates a programming error either in the initial specification of the array size or in the calculation of the limits to be used in a SCAN, REPLACE, or COMPARE statement.

SELECT ERROR<file label>:<job name>,<terminal reference>

The occurrence of this message means that an object program attempted to perform an invalid operation on the designated file (for example, the rewinding of a card reader file). Consequently, processing of the program was discontinued.

<job ID> SORT ERROR #<integer> DSED @<terminal reference>

The sort errors currently implemented are as follows:

1 - Record size passed to sort is <1.

2 - Count of sort input and output records does not agree.

3 - User's supplied parameters, disk size and number of tapes, are both 0.

4 - Amount of disk specified is insufficient to do a disk-only sort--stringing phase.

5 - Amount of disk specified is insufficient to do a disk-only sort--merging phase.

6 - Input file already open.

7 - Irrecoverable parity error on user's input or output files or block # of work file retrieval out of sequence (memdump also occurs).

8 - Insufficient size of output file.

9 - Output file already open.

10 - Irrecoverable parity error while writing sort work file.

11 - Irrecoverable parity error while reading sort work file.

12 - Irrecoverable parity error while reading or writing sort control file.

JOB ORIENTED MESSAGES

<compiler name><job name>@<priority> SNTX

The occurrence of this message is an indication that a compilation has been terminated and one or more syntax errors have been detected.

*<job ID> SNTX <nn> @<sequence number>

The SNTX message is displayed when a syntax error is detected by the ALGOL compiler. <nn> ranges from 1 to 10.

*STACKNO IS READONLY<terminal reference>

This message indicates that an attempt was made to store a value in the STACKNO task attribute of a task variable. (IPC message.)

*STACK OVERFLOW<terminal reference>

The occurrence of this message means that the operation performed by an object program caused its stack to overflow its limit. Consequently, processing of the program was discontinued. The ?STACK = <nn> can be used to extend the stack.

*STACK UNDERFLOW<terminal reference>

This message indicates that a program attempted to reference below BOSR or S Register less than F Register.

*STARTTIME IS READONLY<terminal reference>

This message indicates that an attempt was made to store a value in the STARTTIME task attribute of a task variable. (IPC message.)

*STOPPOINT STORED<terminal reference>

This message indicates that an attempt was made to store a value in the STOPPOINT task attribute of a task variable. (IPC message.)

*SUSPENDED BY SYSTEM

This message occurs when running under WORKINGSET, a task is suspended. A PK or DS response may be entered.

*TASKTYPE IS READONLY<terminal reference>

This message indicates that an attempt was made to store a value in the TASKTYPE task attribute of a task variable. (IPC message.)

*<file ID> UNMATCHED GENEALOGY<terminal reference>

This message indicates that the genealogy encountered on the file that was opened does not match that described by the file declaration.

•UP LEVEL ATTACH<terminal reference>

This message indicates the program attempted to attach itself to an UPLEVEL event, and the program is discontinued at this point.

•VISIT NONACTIVE TASK<terminal reference>

This message indicates that an attempt was made to execute a CONTINUE statement for a task variable that is inactive. (IPC message.)

•<file ID> WRIT BEFOR OPEN<terminal reference>

This message indicates that a user's COBOL program attempted to write on an output file before it had been opened.

•<unit mnemonic> WRITE LOCKOUT

The occurrence of this message means that a program attempted to write on a magnetic tape with no write ring or on a disk pack which had been locked out with hardware lockout switches.

SECTION 5
WORK FLOW LANGUAGE

SECTION 5 — CONTENTS

SECTION 5

WORK FLOW LANGUAGE

INTRODUCTION

The Work Flow Management system includes enhanced facilities for user control of task initiation and resource allocation. These enhancements are a natural expansion of the existing system control statements into a complete Work Flow Language. This language allows the user to describe each job as a network of specific interrelated tasks.

As operating systems have evolved and become more sophisticated, so have the demands upon efficient job control. To enable the control statement interpreter to handle all modes of operation, the LOADCONTROL/CONTROLCARD implementation has been replaced with a Work Flow Language (WFL) compiler. The WFL compiler is a true compiler; it produces B 6700 machine code to control the tasks within a job as the user prescribes. Thus, all user jobs are "programs" written in the Work Flow Language. The WFL compiler is invoked any time a user job is entered at a system card reader or supervisory console, or "zipped" from a user program. The WFL compiler performs the following functions:

a. Syntactically checks the accepted Work Flow statement input.

b. Generates machine code to handle the tasks which constitute the job as specified by the Work Flow statements.

c. Generates the jobfile disk file for the job.

Control information may be entered into the system in either of two ways: as control statements entered at the input keyboard or on punched cards called control cards entered at a card reader.

INPUT KEYBOARD

From the input keyboard, single control statements or lists of control statements separated by semicolons may be entered in the form shown below.

```
<invalid character> <control statement list>   <ETX>

<invalid character> ::= ?

<control statement list> ::= <control statement> |

                             <control statement list> ; <control

                              statement>
```

WORK FLOW LANGUAGE

System control statements input via a supervisory console do not require an initial "?" character unless the statement begins with one of the character pairs "DI", "DA", or "PB". Thus, the following is a valid example of a system control statement input from the SPO:

```
RUN LINK/TEST(4); END
```

PUNCHED CARDS

Control cards are distinguished from other punched cards by the occurrence of an invalid character in Column 1. The invalid character in Column 1 is required on the first card of a control statement deck and may appear on subsequent cards to serve as a control statement delimiter.

On the input keyboard, the invalid character is the ? symbol. On control cards, the invalid character is not the ? but some invalid combination of punches, as for example 1-2-3. Throughout this section the invalid character is represented by the symbol <I>.

If two or more control statements are punched on a single card, they must be separated by semicolons. The invalid character is neither needed nor allowed following a semicolon.

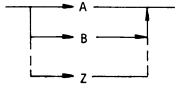
BASIC ELEMENTS AND CONSTRUCTS

The Work Flow Language uses the EBCDIC character set. Any invalid EBCDIC character is illegal with the following exception: an invalid EBCDIC character in column 1 of a control card (denoted by <I>) will be treated as a semicolon. Moreover, an <I> is required (1) on the first card of any job, (2) on the first card following any data deck, and (3) on the last card of any job.

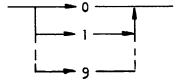
Identifiers and numbers are terminated by any non-alphanumeric character (including a blank). Cards can be terminated by a percent sign (%), or by a minus (-) or a period (.), if such a character is syntactically invalid. The remainder of the card is not scanned.

The following diagrams describe the syntax of the basic constructs of the Work Flow Language.

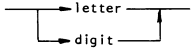
letter:



digit:

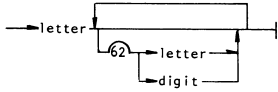


alphanumeric:

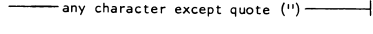


character:

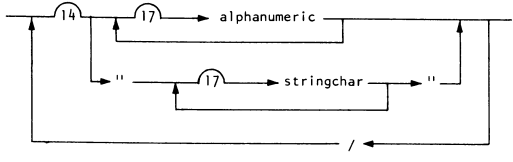
Identifier (contextually referred to as real, Boolean, label, subroutine, file, or task identifier):



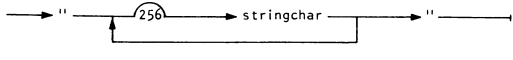
stringchar:



filename:



string:

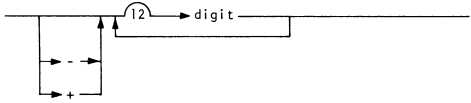


WORK FLOW LANGUAGE

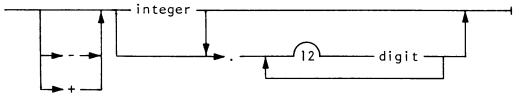
comment:

---any sequence of characters not containing a semicolon (;), period (.), percent sign (%), hyphen (-), (or, in some contexts, a left bracket).

integer:



real:



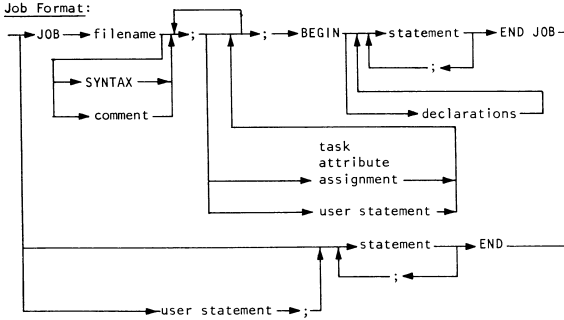
Examples:

Identifiers	A Z123 ABC123
Filenames	A A/B 1 1/C/3 "A?-1B"/C
Strings	"ABC" "?%->"
Integer	-12 +750 12345
Reals	-3.1416 .2 +1.

JOBS

A job is a "program" written in WFL. During execution, a job will run out of its own "job stack". A task is any process initiated by a job. Examples of tasks are: user program execution, compilations, library maintenance, etc. The job stack is used to control the execution of tasks; it may contain simple variables, task variables, and files.

Job Format:



Example:

```
<1> JOB A/JOB; USER C/D; CORE=5000;
        PRINTLIMIT=100;
    BEGIN
       COMPILE X WITH ALGOL TO LIBRARY;
       DATA CARD
          .
          .
          .
<1> RUN X;
<1> END JOB
```

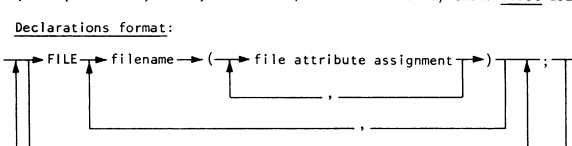
DECLARATIONS

All files and subroutines must be declared. A file declaration may specify attributes of the file. A subroutine declaration specifies the statements contained in the subroutine. Subroutines may run in the job stack (see subroutine statement) or in a separate stack (see process statement). A file declared in the job is called a "global" file. It may be passed to tasks by file equation (see task attribute assignment).

WORK FLOW LANGUAGE

With the exception of files and subroutines, all job stack variables (reals, Booleans, tasks, and labels) are determined by their first usage.

Declarations format:



Example:

```
<1> JOB EXAMPLE SYNTAX;
BEGIN
    FILE GLOBAL (KIND=DISK,MAXRECSIZE=30); % FILE DECLARATION
    SUBROUTINE SUB; % START OF SUBROUTINE DECLARATION
    BEGIN
       RUN X; FILE F1:=GLOBAL; % INITIATES PROGRAM X, THE
       RUN Y; FILE F2:=GLOBAL; % FILE WITH INTERNAL NAME "F1" WILL
                               % BE THE GLOBAL FILE "GLOBAL"
    END; % END OF SUBROUTINE DECLARATION
    RUN Z; % THIS IS THE FIRST EXECUTABLE STATEMENT
    PROCESS SUB; % INITIATES THE SUBROUTINE AS A DEPENDENT
                 % PROCESS IN ITS OWN STACK
    SUB; % INITIATES SUBROUTINE IN JOB STACK
L:  I:= 1; % I IS A REAL VARIABLE, L IS A LABEL
    B:= TRUE; % B IS A BOOLEAN VARIABLE
    RUN X [T]; % T IS A TASK VARIABLE
    GO L1; % L1 IS A LABEL
L1:
<1> END JOB
```

TASK ATTRIBUTES

Task attributes allow the job to monitor and control the execution of tasks. Misusing an attribute (e.g., setting a read-only attribute) will result in a run-time error. Task attributes can be specified at two times:

a. When the code file is compiled (see compile statement).

b. When the task is initiated (see task initiation statements and task statements).

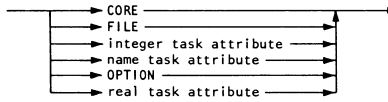
If these sources conflict, the attributes specified at initiation will override.

Example:

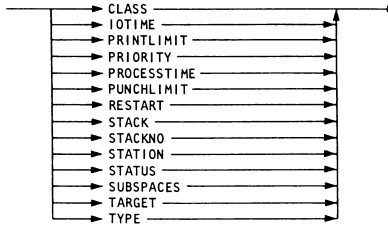
```
COMPILE X ALGOL LIBRARY;
    ALGOL PRIORITY=50; % SETS ALGOL's PRIORITY
    PRIORITY=60; % SETS PRIORITY IN CODE FILE X
RUN X; % RUNS AT PRIORITY 60
RUN X;
    PRIORITY=70; % OVERRIDES COMPILED-IN PRIORITY
```

Task Attributes Formats:

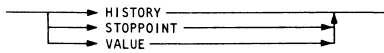
Task Attribute:



Integer Task Attribute:

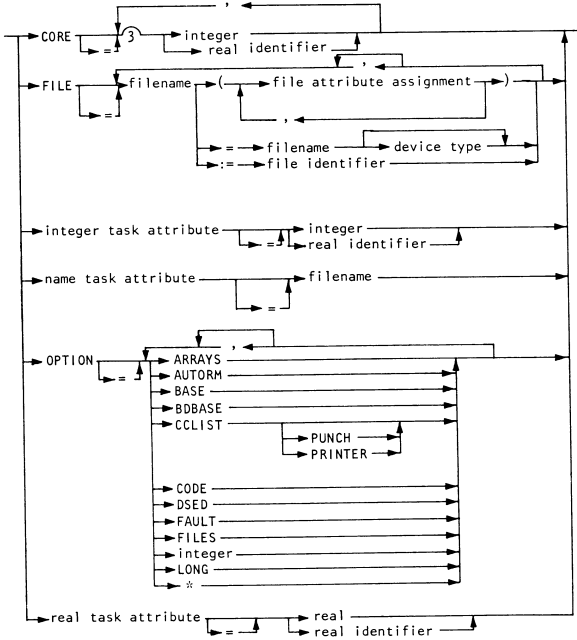


Real Task Attribute:



WORK FLOW LANGUAGE

The task attribute assignments have the following format:



FILE ATTRIBUTES (FILE EQUATION)

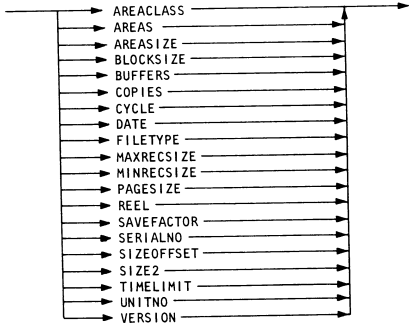
File attributes for files internal to tasks may also be set. The files are identified by their internal names. When a file is opened, file attributes are taken from the file declaration and merged with the file attributes contained in the task (those in the task will override).

Example:

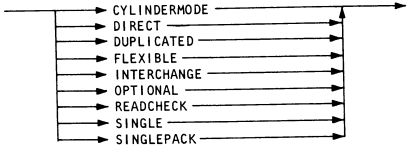
```
RUN X;
   FILE F1 = X/Y DISK;
   FILE F2 (TITLE=X/Y, KIND=DISK, INTMODE=BCL);
   FILE F3 :=G;
```

File Attribute formats:

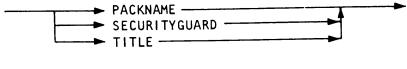
Numeric File Attribute:



Boolean File Attribute:

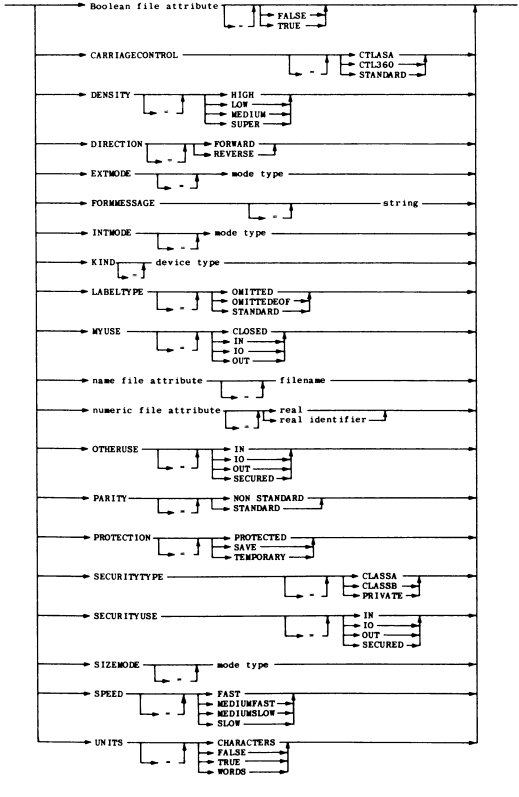


Name File Attribute:

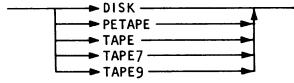


WORK FLOW LANGUAGE

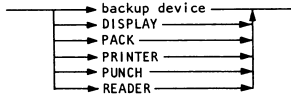
The file attribute assignments have the following format:



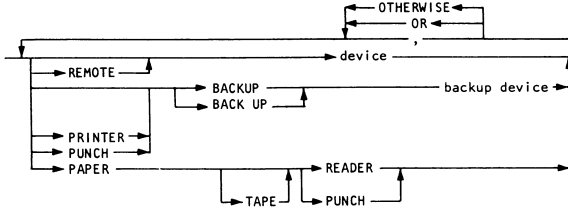
Backup Device:



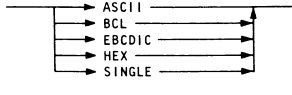
Device:



Device Type:



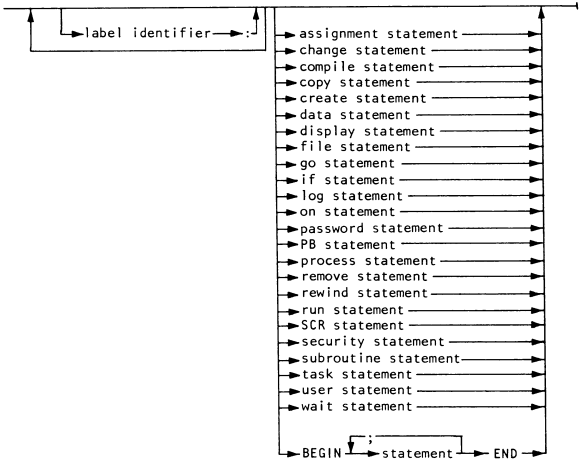
Mode Type:



WORK FLOW LANGUAGE

STATEMENTS

The statements of the Work Flow Language govern task initiations, data labeling, and control of job execution. The statements format is as follows:



TASK INITIATION

Task initiation statements (compile, copy, log, PB, process, run, and SCR) are used to start user programs and system functions in separate stacks. The process statement will initiate the task asynchronously, that is, the job stack will execute concurrently with the task stack. All other task initiation statements will run synchronously, i.e., the execution of the job is suspended until the task is completed.

Example:

```
        RUN X;
        RUN Y;
% Program X will run.  When it is finished, program Y will run.
```

A task variable may be attached to the task in order to monitor and control the execution of the task. (See task attributes.) A task variable may be attached to only one task at a time. Thus,

```
RUN X [T];      COPY A TO B[T]
```

is permissible but

```
PROCESS COMPILE X WITH ALGOL [T];
RUN  Y [T];
```

would result in a run-time error.

DATA DECKS

Tasks may reference two types of card decks. Internal decks are entered as part of the job and put in the job file by the WFL compiler. External decks are entered separately from the job and are read directly from the card reader. The CDONLY option, which is controlled from the operator console, determines whether or not external decks may be referenced. If the option is reset, then (1) any task which cannot find a card deck in its job file will hang on a no-file until the deck appears in a physical reader, and (2) if a data statement is read which is external to any job, the card reader will be labeled and made available to user programs.

If the option is set, then (1) any task which cannot find a card deck in its job file will be DSed, and (2) if a data statement is read which is external to any job, a syntax error occurs and the deck will be flushed from the reader.

A task may read internal decks either (1) located between its task invocation statement and the next task invocation statement (referred to as a local deck), or (2) located before the first task invocation statement of the job (referred to as a global deck).

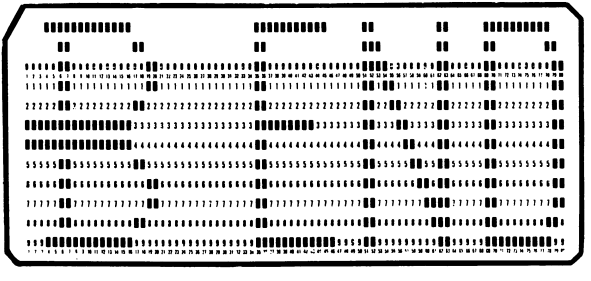
A global deck must have a name, but local decks may be unnamed. When a task tries to open a card file, it will look for the first unread local deck with no name or the correct name. If it cannot find a local deck, it will look for the first global deck with the correct name. (Note that local decks are read only once, but global decks may be referenced many times. Each program opening a global deck will start reading at the beginning of the deck.)

The statement following a BCL or EBCDIC data deck must start with an invalid character in column 1. Binary decks must be terminated by a binary end (BEND) card. The BEND card is only a data terminator; it does not act as an END card for the job.

WORK FLOW LANGUAGE

NOTE

Binary data decks are terminated with a binary end (BEND) card (shown below) rather than with an ordinary END card. The BEND card can be keypunched, but the easiest way to produce one is with a short program in which a binary card punch file is opened and immediately closed. The BEND card is punched as the ending record of the binary card punch file. Only the first 48 columns of the BEND card are checked by the system in order that EOF on 51-column binary decks will be recognized.



JOB EXECUTION

Job execution control statements govern the flow of execution of a job and provide parameters used by the job stack. These statements are as follows:

a. Assignment
b. Change
c. Create
d. Display
e. File
f. GO
g. IF
h. On
i. Password
j. Rewind
k. Security
l. Subroutine
m. Task
n. User
o. Wait

WFL STATEMENT SYNTAX DIAGRAMS

The following diagrams describe the syntax of the WFL statements. A brief description is provided for each of these statements which are presented in alphabetical order.

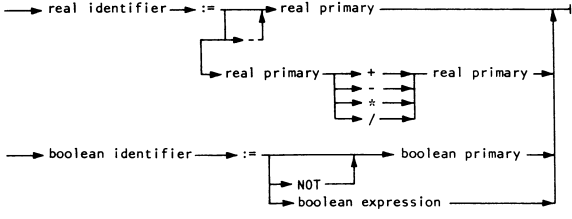
ASSIGNMENT STATEMENTS

The assignment statement is used to change the value of real and Boolean variables.

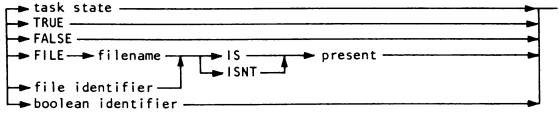
Examples:

```
     N := 1;
     I := -3.1;
     I := X + 2;
     A := X * Y;
RESULT := T (VALUE) + F (MAXRECSIZE)
     B := TRUE;
    BY := NOT FALSE;
     B := I <5;
READY  := T IS ABORTED;
     B := F ISNT PRESENT;
     B := FILE X/Y IS PRESENT;
  AORB := A OR B;
```

Assignment Statement:

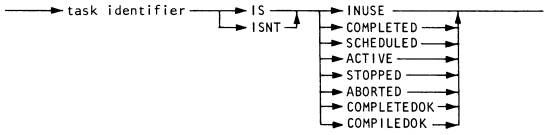


Boolean Primary:

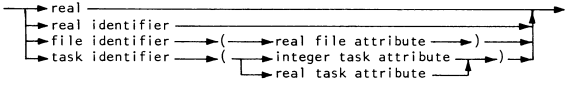


WORK FLOW LANGUAGE

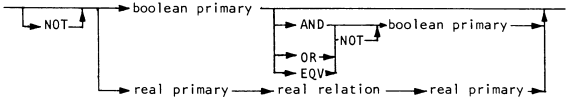
Task State:



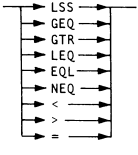
Real Primary:



Boolean Expression:



Real Relation:



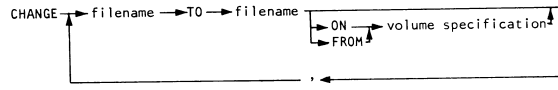
CHANGE STATEMENT

The change statement changes the titles of files on disk or disk pack.

Examples:

```
CHANGE X TO Y;
CHANGE X TO Y, X/X TO Y/Y ON MYPACK,
    XX TO YY ON PACK;
```

Change Statement:



COMPILE STATEMENT

The compile statement is used to initiate compilers. The first filename is the name of the code file. The compiler is either specified by using a recognized system compiler or by preceding the filename with the word WITH. The compiler name is prefixed by directory "SYSTEM". If the resulting file is not a compiler code file, the job will be DSed. The disposition of the code file is given following the compiler name:

a. GO: execute the code file if it compiles.
b. LIBRARY: lock the code file in the disk directory if it compiles.
c. SYNTAX: compile for syntax only.
(SYNTAX cannot be specified in conjunction with other dispositions.)

If a task identifier is specified following the code file name, and the GO disposition has been specified, the task variable is attached to the execution of the code file. The task variable specified following the compiler name is attached to the compiler. Task attributes may be assigned to the compiler by prefacing them by any system recognized compiler name.

Example:

```
ALGOL STACK=100
```

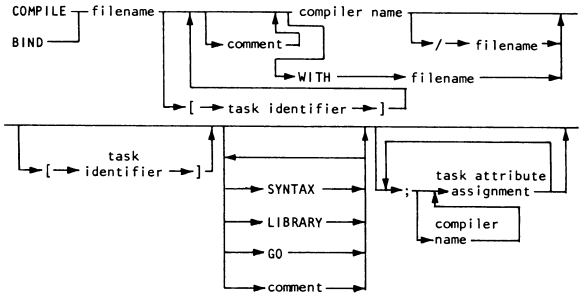
Attributes not preceded by a compiler name are permanently attached to the code file. These can only be constants, not variables. These values are put in the stack whenever compiled program is executed unless the attribute values are overridden by run-time task attribute values. (See run statement.)

Example:

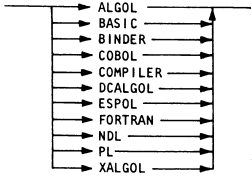
```
COMPILE X ALGOL;
COMPILE X/Y GT WITH A/B CT LIBRARY AND GO;
    COMPILER FILE CARD = C/D DISK;
    PL PRIORITY = 55;
    FILE F(TITLE=Y/Z); STACK=100;
COMPILE A PL/I SYNTAX;
BIND X/Y BINDER LIBRARY;
```

WORK FLOW LANGUAGE

Compile Statement:



Compiler Name:



COPY STATEMENT

The copy statement is used to copy files to and from disk, tapes, and disk packs.

Examples:

a. COPY X TO Y [T];

   Copies a file X from disk to a tape Y, attaching task variable T to the copying task.

b. COPY X FROM Y;

   Copies file X from tape Y to disk.

If the verb ADD is used, the files are copied to each destination where they are not already present.

Examples:

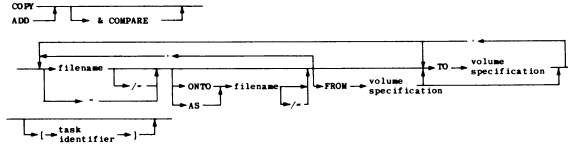
ADD X/Y FROM T;

Copies file X/Y from tape T to disk, if there is not currently a disk file named X/Y.

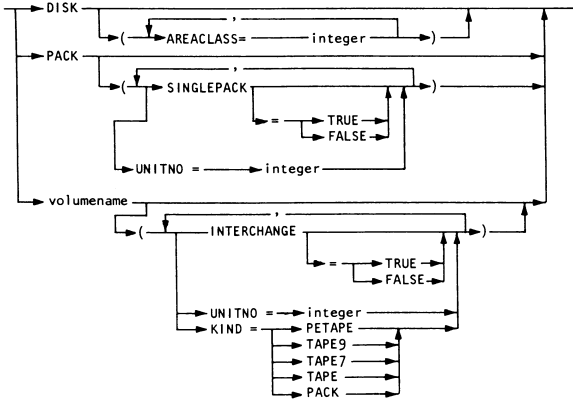
ADD Z/= FROM TO TO R (KIND=PACK), TO DISK.

May load different files to R and to disk, depending on what is present on each device.

Copy Statement:

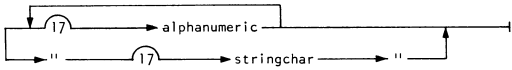


Volume Specification:



WORK FLOW LANGUAGE

Volumename:



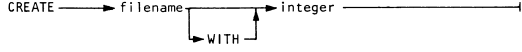
CREATE STATEMENT

The create statement is used to create a directory with a given scramble modulus.

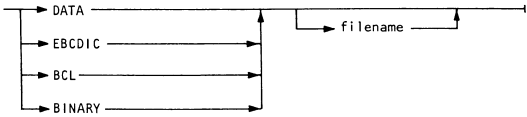
Example:

```
CREATE X WITH 10  ;
```

Create Statement:



DATA STATEMENT



Example:

```
<1> JOB X;
    BEGIN
        DATA GLOBAL/DECK1;
          .
          .
<1> BINARY GLOBAL/DECK2;
          .
          .
    (BEND CARD)
<1> RUN X;
    BCL
     .
     .
<1> COMPILE A FORTRAN;
    DATA
     .
     .
<1> DATA
     .
     .
<1> END JOB
```

DISPLAY STATEMENT

The display statement displays a string on the operator console.

Examples:

```
DISPLAY "HI THERE";
DISPLAY "*##?";
```

Display Statement:

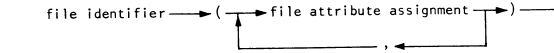
FILE STATEMENT

The file statement allows the job to set file attributes in global files.

Examples:

```
G(KIND=DISK, MAXRECSIZE=J);
RUN X; FILE F:=G;
```

File Statement:



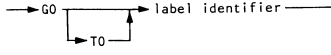
GO STATEMENT

The go statement transfers control to the label specified. If the label is not specified, the syntax error will be given.

Example:

```
GO TO L;
  .
  .
  .
L: RUN X;
L1 : L2: RUN Y;
GO L2:
```

Go Statement:



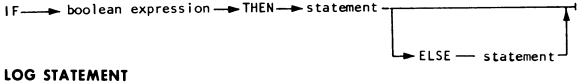
WORK FLOW LANGUAGE

IF STATEMENT

The if statement is used to make decisions within the job.

Examples:

```
IF T IS EOJ THEN GO TO L
    ELSE I := 1;
IF T(VALUE) = 5 THEN RUN X;
IF FILE X/Y ISNT PRESENT THEN
    DISPLAY "NO FILE X/Y"
   ELSE RUN X/Y;
```



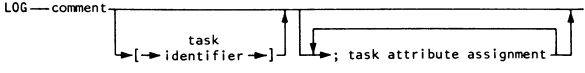
LOG STATEMENT

The log statement initiates SYSTEM/LOGANALYZER passing the following text up to a semicolon or left bracket as parameter.

Example:

```
LOG MIX=123;
IOTIME=50;
```

Log Statement:



ON STATEMENT

On statements allow the job stack to take user-specified action when (1) a task is abnormally terminated (ON FAULT), or (2) the system is Halt/Loaded and the job must be restarted (ON RESTART).

The statement "ON FAULT, statement" enables the condition FAULT. If any task subsequently is terminated abnormally, the statement will be executed. This includes operator or programmatic DS as well as arithmetic faults.

The statement "ON FAULT;" disables the condition. An abnormal task termination will not have any effect on the job.

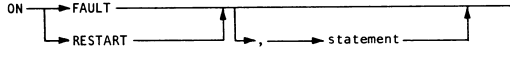
Similarly, the RESTART condition can be enabled and disabled by the statements:

```
ON RESTART, statement;
ON RESTART;
```

Only one statement for each condition can be enabled at one time. Thus, any ON statement disables any previous ON statements for the same condition. If a subroutine executes an ON statement while running in the job stack, it will disable any previous ON statements until the subroutine is finished or the ON condition is disabled. Subroutine exit or an ON RESTART; will return the restart condition to what it was before the subroutine was called.

If the enabled statement does not execute a GO statement, the job will resume executing at the point where it would have been if the statement had been disabled.

On statement:



Example:

```
<1> JOB X;
    BEGIN
      SUBROUTINE SUB;
      BEGIN
        ON FAULT, % CALL THIS "SUB FAULT"
          BEGIN
            DISPLAY "SUB FAULT TAKEN";
            GO ERR;
          END; % END OF ON STATEMENT
        RUN X; % IF X ABORTS, "SUB FAULT TAKEN" IS DISPLAYED
        ON FAULT;
        RUN Y; % IF Y ABORTS, "JOB FAULT" IS DISPLAYED
        END; % END OF SUB
        RUN A; % IF A ABORTS, NO FAULT IS EXECUTED
        ON FAULT, DISPLAY "JOB FAULT"; % CALL THIS "JOB FAULT"
        RUN B; % IF B ABORTS, "JOB FAULT" IS DISPLAYED
        SUB;
      ERR:
<1> END JOB
```

If the MCP is Halt/Loaded during execution of a job, the job will be restarted at the most recent point where no tasks were running. Typically, this is just prior to the last initiation. The values of all real and Boolean variables are saved during Halt/Load. All tasks and files are reinitialized to their original values. Thus, the information in tasks and files is not available after the job is restarted. If the job uses this information, it may be necessary to include an ON RESTART statement which executes a "go" to a point where the tasks and files are set up.

WORK FLOW LANGUAGE

Example:

```
     <1> JOB X;
         BEGIN
          FILE G;
          ON RESTART, GO TO L;
L:        RUN P1: FILE F1:=G;
          RUN P2; FILE F2:=G;
     <1> END JOB
```

If a Halt/Load occurs while P2 is running, the job would normally not rerun P1 since the restart point would be just prior to the initiation of P2. However, P2 nees the information written to file G by P1. Therefore, an ON statement is executed to ensure that the restart point is prior to the initiation of P2.

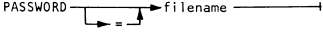
PASSWORD STATEMENT

The password statement is used to change the password of the job's current user code.

Example:

```
     PASSWORD = XYZ;
```

Password Statement:



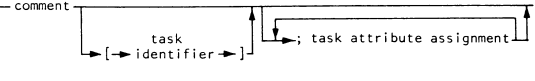
PB STATEMENT

The PB statement initiates SYSTEM/BACKUP passing the following text up to a semicolon or left bracket as a parameter.

Example:

```
     PB D123/000 LINE SAVE[T];
     PRINTLIMIT=100;
```

PB Statement:



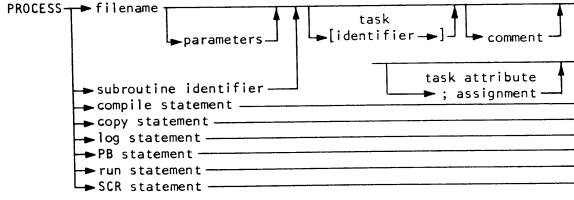
PROCESS STATEMENT

The process statement initiates tasks asynchronously. If a task variable is attached, the job stack can subsequently wait for the process to terminate (see wait statement), monitor its execution (see if statement), or alter its execution (see task statement). The job does not terminate until all asynchronous tasks have terminated. The name displayed on the screen for a compile, run, etc., is the program name; the name of a processed subroutine is the job name.

Examples:

```
PROCESS X/Y (3,1) [T];
PROCESS SUB; CORE=100;
PROCESS COPY A TO B [T];
```

Process Statement:



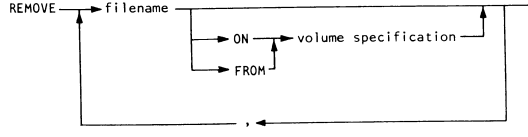
REMOVE STATEMENT

The remove statement removes files from disk or disk pack.

Example:

```
REMOVE X,Y ON MYPACK, X/Y ON PACK, Z
```

Remove Statement:



WORK FLOW LANGUAGE

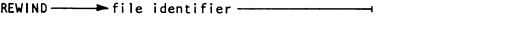
REWIND STATEMENT

The rewind statement rewinds a global file.

Example:

```
RUN X; FILE F1:=G;
```

Rewind Statement:



RUN STATEMENT

The run statement is used to initiate a previously compiled code file. If the file named is not a code file, the job stack will be discontinued (DSed). If the file is not present on the disk, the job stack will display a "no file:" message on the operator console and wait for the file. The types of parameters that may be passed to a task are as follows:

a. Numbers: integers, reals, real variables, and strings preceded by a character size.

b. Booleans: the values TRUE and FLASE, and Boolean variables.

c. Arrays: strings without character sizes.

Parameter checking against the codefile will be done, and any mismatches will DS the job. All parameters are passed by value (i.e., the value of a variable in the job stack cannot be changed by passing it to a task). Binary, octal, and hex strings will be passed as reals (i.e., RUN(1"10", 3"21", 4"1F") is the same as RUN(2,17,31)) and must be less than one word in length. BCL, ASCII, and EBCDIC strings are left-adjusted, blank-filled and passed as reals. They must also be less than one word in length.

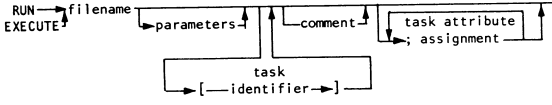
Untyped strings are assumed to be EBCDIC and passed as arrays. The last word is filled with EBCDIC blanks if necessary.

A list of task attribute assignments can be given which will set attributes in the task's stack. These values will override any attributes set when the code file was compiled. (See task attributes and the compile statement.)

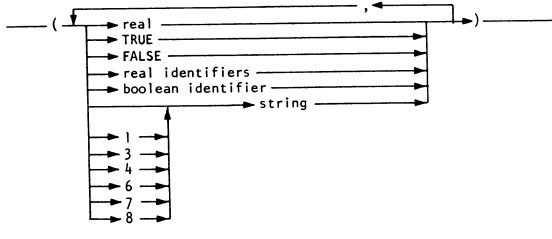
Example:

```
RUN X;
RUN A/B [T]; STACK = 500;
RUN A/B (1,1,FALSE,3.14, 3"123", "HI THERE")
    [T1];  FILE C := G; BDNAME=X;
     FILE F(BLOCKSIZE = 30, KIND=PETAPE),
          G(KIND=DISK);
     FILE H= ABC/D  DISK;
RUN A/B THIS IS A COMMENT; VALUE =1;
```

Run Statement:



Parameters:



SCR STATEMENT

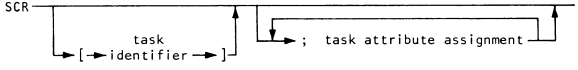
The SCR statement initiates the MCP on-line maintenance intrinsic. The intrinsic requires a data deck when initiated from the card reader. See the On-Line Maintenance and Test (MAT) Language Information Manual, Form No. 5000169.

WORK FLOW LANGUAGE

Example:

```
SCR [T];
```

SCR Statement:



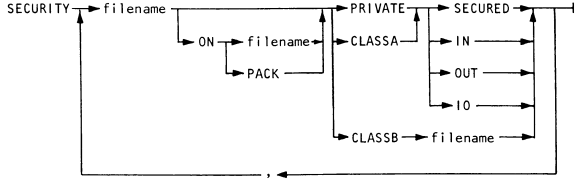
SECURITY STATEMENT

The security statement is used to change the security of a file on disk of disk pack.

Examples:

```
SECURITY AB/XY PRIVATE IO;
SECURITY A/B ON MYPACK CLASSB XYZ;
```

Security Statement:



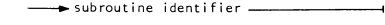
SUBROUTINE STATEMENT

The subroutine statement executes a subroutine in the job stack.

Example:

```
SUB;
```

Subroutine Statement:



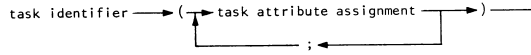
TASK STATEMENT

The task statement allows the job to set task attributes in task variables.

Example:

```
T(PRIORITY J; FILE F(KIND=DISK));
RUN X T ;
```

Task Statement:



USER STATEMENT

The user statement changes the user code of the job. If this statement appears in a subroutine running in a separate stack, it only changes the user code of the subroutine; the user code of the job stack is not changed.

Examples:

```
USER A/B;
USER X;
```

The correct password must be given if there is one connected with the user code.

User Statement:

WAIT STATEMENT

The wait statement allows the job stack to suspend execution until specified conditions are met. These conditions are as follows:

a. WAIT (OK): will suspend the job stack until the operator enters an "OK" from the console.

b. WAIT (integer): will wait the specified number of seconds and then resume execution.

   Example: WAIT(5); will suspend the job stack for 5 seconds.

c. WAIT (task identifier): will wait until the task has completed;

Example: WAIT(T)

d. WAIT (task identifier (numeric task attribute) real relation, real primary): will wait until either the task is completed or the task attribute satisfies the relation. The job stack is waiting for its own "EXCEPTIONEVENT". It will not resume execution until this event is caused and one of the above conditions is met. The job's EXCEPTIONEVENT can be caused by the following:

1. A task changing its task state.

2. The operator entering a "HI" from the console.

3. Programmatically from the task.

Example: in an ALGOL task

```
CAUSE (MYSELF.EXCEPTIONTASK.EXCEPTIONEVENT)
```

Example:

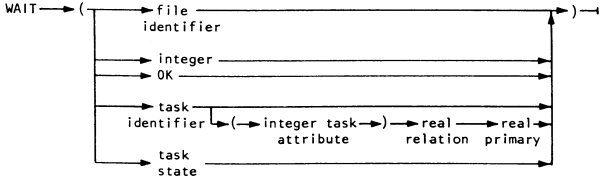
```
WAIT (T(VALUE) >0);
```

e. WAIT (task state): will wait until either the task is completed or achieves the given state.

Example: WAIT (T IS STOPPED);

f. WAIT (file identifier): waits until the file is present. This statement opens the file and will display a no file on the operator console if the file is absent.

Wait Statement:



SECTION 6

UTILITIES

SECTION 6 — CONTENTS

SECTION 6

UTILITIES

Utilities of the B 6700 SYSTEM

Utility	Name of Source File	Name of Object File
Card Line	SYMBOL/CARDLINE	SYSTEM/CARDLINE
Compare	SYMBOL/COMPARE	SYSTEM/COMPARE
Dump Analyzer	SYMBOL/DUMPANALYZER	SYSTEM/DUMPANALYZER
Guardfile Generation	SYMBOL/GUARDFILE	SYSTEM/GUARDFILE
IAD Mapper	SYMBOL/IADMAPPER	SYSTEM/IADMAPPER
Intrinsic Functions	SYMBOL/ESPOLINTRINSICS SYMBOL/ALGOLINTRINSICS	SYSTEM/INTRINSICS SYSTEM/INTRINSICS
Library Maintenance	SYMBOL/MAINTENANCE	SYSTEM/MAINTENANCE
List Directory	SYMBOL/LISTDIRECTORY	SYSTEM/LISTDIRECTORY
Log Analyzer	SYMBOL/LOGANALYZER	SYSTEM/LOGANALYZER
Makeuser	SYMBOL/MAKEUSER	SYSTEM/MAKEUSER
Patch	SYMBOL/PATCH	SYSTEM/PATCH
Printer Backup	SYMBOL/BACKUP	SYSTEM/BACKUP
SPO Hardcopy	SYMBOL/HARDCOPY	SYSTEM/HARDCOPY
SPO Hardcopy Printing	SYMBOL/PRINTCOPY	SYSTEM/PRINTCOPY
Tape Label Recognition	SYMBOL/RLTABLEGEN	SYSTEM/RLTABLEGEN

NOTE

See also "MAINTENANCE OF STANDARD SOFTWARE" in Section 3.

CARD LINE

Use:

The SYSTEM/CARDLINE utility program is used to list a BCL or EBCDIC data deck on the line printer. In addition to a listing of the card images, a card count and sequence check is included. The field checked for sequence errors is columns 73 - 80.

UTILITIES

In addition to the card-to-print function, other utility functions may be accomplished by label equating the program's input and/or output files. The input file is named CARD, and the output file is named LINE.

For example,

To punch a BCL, EBCDIC, or BINARY card deck, the LINE file should be equated to a card punch file as follows:

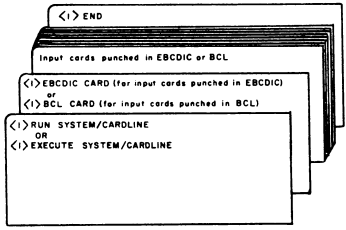
```
<1> RUN SYSTEM/CARDLINE
  ; FILE LINE(KIND=PUNCH,EXTMODE=BCL)
```

When SYSTEM/CARDLINE is run with a taskvalue of 1, the output is single spaced. The spacing is identical to the taskvalue. (Default is double-spacing.)

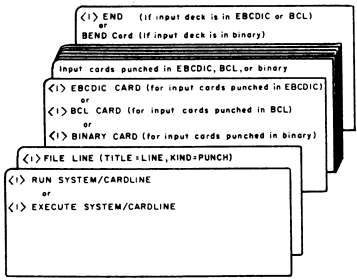
For example:

```
<1> RUN SYSTEM/CARDLINE; VALUE = 1
```

Card Deck for Listing EBCDIC or BCL Data:



Card Deck for Punching EBCDIC, BCL, or Binary Data:



Refer to Binary Statement for information on the BEND (binary end) card.

COMPARE

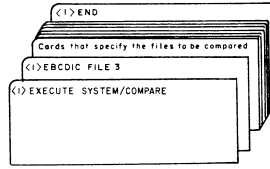
Use:

To compare (by use of the program SYSTEM/COMPARE) one or more pairs of files. The program performs a bit-by-bit comparison on each record of each pair of files. If the records are not identical or if one of the specified files is not present, an appropriate error message is printed. The comparison of a pair of files is terminated after five unsuccessful record comparisons have been made, and the program skips to the next pair of files.

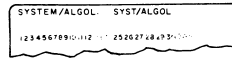
The symbolic code for the compare program is supplied on the symbol tape in a file named SYMBOL/COMPARE; the object code is supplied on the system tape in a file named SYSTEM/COMPARE.

UTILITIES

After the program SYSTEM/COMPARE has been copied from the system tape, it may be executed by using the following card deck:



The names of the files to be compared are specified as follows: the name of the first file is punched on a card, starting in card column 1. The file name must be immediately followed by a period. The name of the second file is punched on the same card, starting in column 25. This file name must also be followed by a period. For example, the card illustrated below specifies that the files "SYSTEM/ALGOL" and "SYST/ALGOL" are to be compared.



Messages:

When the program starts to compare the two files, the following message is printed out:

NOW COMPARING FILES <first fileid>. AND <second fileid>.

If the files are identical, no further action is taken; the program will compare the next pair of files. If the files are not identical or if one file is not present, one of the following messages will be printed.

**********FILES PRESENT BUT DIFFER IN BLOCKING SPECS

**********FILES DIFFER IN RECORD NUMBER <RECORD NO. BEING COMPARED>

**********FIRST FILE NOT IN DIRECTORY, NOT COMPARED.

**********SECOND FILE NOT IN DIRECTORY, NOT COMPARED.

**********EOF ERROR

**********PARITY ERROR IN FIRST FILE

**********PARITY ERROR IN SECOND FILE

**********CARD READER PARITY ERROR

DUMP ANALYZER

Use:

To produce (by use of the program SYSTEM/DUMPANALYZER) a concise, readable memory dump from the memory dump tape (labeled MEMORY/DUMP) generated by the MCP.

Dump Tape (MEMORY/DUMP):

The memory dump tape may be either a seven-track, nonreturn-zero (NRZ) tape; a nine-track NRZ tape; or a phase-encoded (PE) tape. This tape may be produced in any of the following ways:

a. By entering the DP system input message.

b. By forcing a memory dump from the memory tester and the MDL panels.

c. If the initialization function TERMINATE is reset, by the abnormal termination of a job.

NOTE

If a memory dump is generated before a Halt-Load operation has been completed, the memory dump will go directly to a line printer and SYSTEM/DUMPANALYZER cannot be used.

Compilation of SYSTEM/DUMPANALYZER:

The card deck shown below may be used in compiling the program SYSTEM/DUMPANALYZER.

```
<I> END
$ MERGE
<I> DATA
<I> ALGOL FILE TAPE (TITLE=SYMBOL/DUMPANALYZER)
<I> COMPILE SYSTEM/DUMPANALYZER ALGOL LIBRARY
```

In addition to the normal options of the ALGOL compiler (section 8), there is a single user option that can be specified in the compilation deck for SYSTEM/DUMPANALYZER. This option, which is unique to SYSTEM/DUMPANALYZER and which affects the running of the program SYSTEM/DUMPANALYZER, governs the printing out of the contents of memory areas. If the analysis is to be printed out on a line printer that is 132 characters wide, the control card $ SET LONGPRINTER should be included in the compilation deck. Then seven memory words will be printed out on each line. If, on the other hand, the analysis is to be printed out on a line printer that is only 120 characters wide, this control card should be omitted from the compilation deck, and only six memory words will be printed on a line.

Running of SYSTEM/DUMPANALYZER:

The run-time options for SYSTEM/DUMPANALYZER are as follows:

RUN-TIME OPTIONS

Option	Default if File Options is Not Present	Default if File Options is Present	Action if Set
SINGLE	Set (True)	Set (True)	To produce a single-spaced dump of memory areas.
AREADUMP	Set (True)	Set (True)	To break up a dump into areas according to memory links.
AVAILDUMP	Set (True)	Reset (False)	To cause the contents of all available areas of memory to be dumped. Automatically sets AREADUMP.
CODEDUMP	Set (True)	Reset (False)	To cause the contents of code segments to be dumped in full. Automatically sets AREADUMP.
LINKDUMP	Set (True)	Reset (False)	To cause the links for each area of memory to be printed after the area heading. Automatically sets AREADUMP.
MIX = ALL	Set (True)	Reset (False)	To cause the dumping and analysis of all stacks that have an entry in the stack vector.
MIX = ACTIVE	Reset (False)	Reset (False)	To cause the dumping and analysis of the stacks of only those jobs that are active at the time of the dump.
MIX=<mix index list>	Reset (False)	Reset (False)	To cause the dumping and analysis of the stack or stacks of the job or jobs which are specified (as, for example, MIX = 149, or MIX = 149 326 257).
DOUBLE	Reset (False)	Reset (False)	To produce a double-spaced dump of memory areas.
FULLDUMP	Reset (False)	Reset (False)	To produce an analyzed dump that is equivalent to that produced by setting the following options: MIX = ALL, AREADUMP, LINKDUMP, CODEDUMP, and AVAILDUMP.
RAWDUMP	Reset (False)	Reset (False)	To produce an uninterpreted dump of all of core memory.
STACKSONLY	Reset (False)	Reset (False)	To produce stack dumps only (all other options are set to false).

The following execution deck produces an analysis that is satisfactory for most purposes. Naturally, if a deeper analysis is desired, other options may be specified and added to the file labeled OPTIONS; if a less detailed analysis is desired, however, some options may be omitted.

```
<!> END
LINKDUMP
AREADUMP
MIX = ALL
<!> DATA OPTIONS (for option cards punched in EBCDIC)
        or
<!> BCL OPTIONS (for option cards punched in BCL)
        or
<!> EBCDIC OPTIONS (for option cards punched in EBCDIC)
<!> EXECUTE SYSTEM/DUMPANALYZER
```

The program SYSTEM/DUMPANALYZER can also be executed by the entry of the DA system input message at the input keyboard. In this case, the settings of the run-time options are as follows:

Option	Setting
SINGLE	Set (True)
AREADUMP	Set (True)
AVAILDUMP	Set (True)
CODEDUMP	Set (True)
LINKDUMP	Set (True)
RAWDUMP	Reset (False)
STACKSONLY	Reset (False)
MIX=ALL	Set (True)

UTILITIES

Handling of Errors Encountered During the Execution of SYSTEM/DUMPANALYZER:

Parity Errors on Dump Tape: If a parity error occurs on the dump tape, a line of slashes and a message that specifies the last valid memory address found on the tape will be printed on the first page of dump analyzer output. SYSTEM/DUMPANALYZER proceeds, using the last valid address as the maximum address.

Bad Memory Links: If a memory link that is bad or apparently bad is encountered by SYSTEM/DUMPANALYZER, a line of slashes and an explanatory message is printed out. The area having the bad link is dumped in raw form, and the analysis continues with the next area.

Invalid index: If an invalid index occurs during the dumping of a stack, a "croak" message appears and the dump analyzer goes on to the next stack.

Error Messages:

BAD LINK TAGS

The occurrence of this message indicates that the tag fields of the memory links are bad.

INCONSISTENT LINKS

The occurrence of this message indicates that there is a discrepancy between the memory links surrounding an area of memory.

LINK Z OF NEXT AREA NOT ACCESSABLE (sic)

This message is produced when the sum of the contents of the size field within the first memory link and the address of the first memory link is greater than the maximum memory address known to SYSTEM/DUMPANALYZER for the dump being analyzed.

GUARDFILE

Use:

This program builds guard files, consisting of two basic entry types (PROGRAM NAME and USERCODE), that are acceptable in structure and format to the MCP procedure that checks a user's access rights when he opens a guarded (CLASSB) file. It is planned that future releases will add updating capabilities, $-card options, more entry types, sequencing, etc. Those planning to make patches that would alter the characteristics of the GUARDFILE produced are advised to refer to the MCP procedure "CHECKGUARDFILE" at 46280000.

Characteristics of a Guard File

The format of a guard file is as follows:

Record #0

Word 0 - "GUARD" - To help identify this as a legitimate guard file.

Word 1 - Type of guard file.

Currently this must have a value of 1 and is provided to allow use of guard files having different characteristics.

Word 2 - Update Count

To serve as an aid in determining the necessity for garbage collection.

Word 4 - Default Access Rights.

Words 5-59 - Reserved for future use.

Records 1-n

These records contain the access control entries with as many entries per record as is possible without record boundary overflow. The format for an individual entry is as follows:

Word 0 - Entry information word
Next entry word index = 47:6
Next entry record number = 41:14

Sub-entry word index = 27:6
Sub-entry record number = 21:14
Access rights = 7:3
Entry type = 4:5

UTILITIES

Words 1-n Access qualification information:
For entry types 1 and 2, which are usercode and program name respectively, this will be a standard-form name. Other entry types are undefined.

A sample guard file structure (type 1) is as follows:

```
DEFAULT = NO

USER A (NO)

            PROGRAM P1 (RO)

            PROGRAM P2 (RO)

            PROGRAM P3 (RW)

USER B (RW)

USER C (RO)

USER D (RW)

PROGRAM P (RO)

            USER U1 (RW)

            USER U2 (RW)

            USER UN (RW)
```

Semantics:

The items in parentheses associated with each condition indicate the access rights to be allowed if that condition is satisfied. Vertical branches are the path to be followed if a condition is not satisfied and the horizontal branches cause additional checking to be done if it is satisfied. The search will halt if a condition is not satisfied and there is no vertical branch, or if the condition is satisfied and there is no horizontal branch. Access rights will be granted based on the last satisfied condition.

Example:

Referring to the type 1 guard file structure, user A would have no access to a file guarded by this guard file unless he was using programs P1, P2 or P3. If he was using P2, he would have read-only access. User B would have unrestricted read-write access. Any user running program P would be allowed to read the file, while those listed could also write it.

A guard file may be associated with a file via the pointer-valued file attribute SECURITYGUARD, i.e.,

```
REPLACE F.SECURITYGUARD BY "MY/GUARDFILE."
```

or through the use of the SECURITY control statement, i.e.:

```
<I>SECURITY MYFILENAME CLASSB MY/GUARDFILE
```

The guard file may also appear in label equations and declarations. Restrictions on these actions are the same as those on the other security-related attributes. A guard file may have any securitytype, but this will only be checked when the guard file is accessed as a regular file. When a guard file is associated with a file, its name will be placed in that file's header. When this is encountered at file open time, the system will first search the directory of the guarded file's owner and then the general system directory, unless the guard file name was specified using the "*" or "(X)" constructs. Failure to find a named guard file or inconsistency in its format will result in denial of access to the guarded file. Guard files will not be invoked for the "owner" of a guarded file; class A security will be used instead.

Syntax for SYSTEM/GUARDFILE

```
<guard file entry> ::= <primary entry> | DEFAULT <access rights>

<primary entry> ::= <entry type> <access list>;

<entry type> ::= USERCODE | PROGRAM

<access list> ::= <access entry> | <access entry>, <access list>

<access entry> ::= <qualification list> | [<qualification list>]
                   USING <primary entry> | <qualification entry>
                   USING <primary entry>

<qualification list> ::= <qualification entry> |
                         <qualification entry>,<qualification list>

<qualification entry> ::= <qualification info> <access rights>

<qualification info> ::= <usercode> | <program name>

<access rights> ::= <empty> | =RW | =RO | =WO | =XO | =NO
```

Semantics:

The mnemonics for <access rights> indicate read-write, read-only, write-only, execute-only, and no access respectively. If <access rights> is <empty>, then the next <access rights> specified in the <access list> will be assigned. If none is found, then a default value of "NO" is used. <qualification info> will be interpreted according to the entry type for the access list in which it appears. The "DEFAULT" entry specifies the access to be granted if no conditions are satisfied in

UTILITIES

the search of the guard file. If used, it must be alone on the first card. If it is not used, the default value will be "NO" access. On succeeding cards, all 80 columns may be used; hence, sequence numbers are not allowed. All program names encountered will be assumed to have the usercode of the person creating the guard file unless otherwise specified.

Example:

To create the guard file illustrated in paragraph entitled "Characteristics of a Guard File", the following card deck might be used:

```
<1> USER ME

<1> RUN SYSTEM/GUARDFILE

<1> FILE GUARD (TITLE = MYGUARDFILE)

<1> EBCDIC CARD

        DEFAULT = NO

        USERCODE A=NO

             USING PROGRAM P1, P2=RO, P3=RW;

        , B=RW, C=RO, D=RW;

        PROGRAM

        P=RO USING

             USERCODE U1, U2, UN=RW;;

<1> END
```

HARDCOPY

The programs SYSTEM/HARDCOPY and SYSTEM/PRINTCOPY provide a means of capturing supervisory input and output messages within a disk file and printing the contents of that file.

SYSTEM/HARDCOPY utilizes the DCALGOL intrinsic ATTACHSPOQ to establish itself as the recipient of copies of all supervisory console traffic. It reformats all received messages replacing control characters by blanks, condensing multiple blanks to a single blank, and dividing messages into 132 character print lines. It writes the reformatted messages into a disk file titled HARDCOPY. All I/O to HARDCOPY is done with direct I/O, and HARDCOPY is a protected file.

Whenever SYSTEM/HARDCOPY is initiated, it looks for an existing file called HARDCOPY. If HARDCOPY already exists, the last recorded message

is found, and a message indicating a restart is added at the end. A message giving the time and date is inserted every fifteen minutes.

Immediately following the writing of the first message that extends into the last area of HARDCOPY, the title of HARDCOPY is changed to HARDCOPIES/<integer>. (The <integer> is incremented by one each time a new HARDCOPIES is created.) At this time a run of SYSTEM/PRINTCOPY is requested to print the full file, and remove it from the directory.

In addition to the automatic printing of full files, it is possible to print the HARDCOPY in sequential pieces. Each time a HI system input message is directed at SYSTEM/HARDCOPY, a run of SYSTEM/PRINTCOPY is requested that will print all messages since the last HI initiated printing.

SYSTEM/PRINTCOPY is essentially a special disk-to-printer utility program. It reads records from a HARDCOPY disk file and writes them to a direct line printer file. It takes one parameter, an integer value that determines what file and how much of the file will be printed. If the parameter equals zero, the file selected is HARDCOPY. If the parameter is non-zero, then HARDCOPIES/<integer> (where <integer> is the minimum length representation of the absolute value of the parameter) is selected.

In addition, if the sign bit of the parameter is equal to 0, the entire file will be printed; but if it is equal to 1, then only unprinted portions will be printed (used only for HI-initiated printing).

HARDCOPY Disk File Format

Record zero of a HARDCOPY file contains information used for naming copies and controlling SYSTEM/PRINTCOPY. The contents of HARDCOPY record zero are outlined as follows:

WORD	CONTENTS
Word 0	the integer that will be/was used for renaming the file.
Word 1	the record number containing the word following the last recorded message.
Word 2	the number of valid words within the record specified by Word 1.
Word 3	the record number containing the first word of the first message that has not been printed by means of a HI input command.
Word 4	the number of words within the record specified by Word 3 that precede the first unprinted record.
Word 6-29	not used.

UTILITIES

The remaining records contain supervisory console messages. The first word of each message has bits 47:8 = 1 and bits 39:40 equal to the length of the message (not including the first word) as five EBCDIC numeric characters. The last message in the file is always followed by a word with all bits equal to zero.

Use of the Programs

In order to run in an environment which will contain/maintain a hard copy of SPO traffic, it is necessary to:

a. Ensure that SYSTEM/HARDCOPY and SYSTEM/PRINTCOPY are loaded into disk.

b. Ensure that SYSTEM/HARDCOPY has been specified as a "SUPERVISOR" program. This can be done by typing in the control message:

```
CS SYSTEM/HARDCOPY
```

Specifying SYSTEM/HARDCOPY as a "SUPERVISOR" program will cause the program to be automatically initiated after a Halt/Load.

NOTE

The proper format of the HI system input message is:

<mix no> HI

The effect of this message is to cause the EXCEPTIONEVENT of the running stack denoted by <mix no>.

IADMAPPER

To assist the user in utilizing the Installation Allocated Disk (IAD) portions of the MCP, a utility program called SYSTEM/IADMAPPER has been provided. IADMAPPER allows the user to create headers on disk, modify headers on disk, and save to tape selected headers of files on disk. If desired, the header attributes can be listed from disk headers or headers saved to tape by using this program. SYMBOL/IADMAPPER is written in DCALGOL.

Input to IADMAPPER is an EBCDIC card deck consisting of one or more IADMAPPER statements. There are only six different statements indicated by the keywords CREATE, UPDATE, SAVE, RESTORE, LIST, and STOP. SAVE requires a scratch tape and RESTORE requires a previously created tape in the format of the tape which SAVE creates. This format is a series of two or more records grouped in pairs. The first word of every record contains the size of the record (binary value). The tape is created and read using FILETYPE 2 (fixed length block - variable length record) with

the blocksize = 2048 words. The first record of each pair contains a sequence counter in word [1] for consistency checking followed by the file identifier. The second record contains the actual header.

Card input is free form with blanks being delimiters. Columns 1-72 are scanned and sequence numbers if appearing will be ignored. Comments may appear on any card if preceded by a percent sign. An unconditional listing of any headers which have been modified or in any way accessed is provided.

No attempt has been made to outguess the user or the MCP as to the correct combination of header attributes or their acceptable values. Syntax checking of the IADMAPPER statements has been provided. If a syntax error occurs, the last item scanned, type of statement, and the card image (with an underlying asterisk to indicate columns scanned) will be printed. The job will terminate. If, when reading or writing a header, an error result is returned, a one line notation to that effect will be printed, and, following that, the same information will appear as when a syntax error occurs; the job is then terminated.

Syntax for SYSTEM/IADMAPPER

```
<IAD input> ::= <IAD statement>; | <IAD input> <IAD statement>;

<IAD statement> ::= <create statement> | <update statement> |
                    <save statement> | <restore statement> |
                    <list statement> | <stop statement>

<create statement> ::= CREATE: <file header attributes list>

<update statement> ::= UPDATE: <file header attributes list>

<file header attributes list> ::= <file header attributes> |
                                  <file header attributes list>,
                                  <file header attributes>

<file header attributes> ::= <file ID> (<header attribute list>)

<header attribute list> ::= <header attribute> | <header attribute
                            list>, <header attribute>

<header attribute> ::= <header attribute mnemonic> = <value>

<save statement> ::= SAVE TO <external tape ID> ; <file header list>

<restore statement> ::= RESTORE FROM <external tape ID>:<file header
                        list> | RESTORE FROM <external tape ID>

<list statement> ::= LIST: <file ID list> | LIST FROM <external
                     tape ID>: <file ID list> | LIST FROM
                     <external tape ID>

<stop statement> ::= STOP
```

UTILITIES

```
<file ID list> ::= <file ID> | <file ID list>, <file ID>

<file header list> ::= <file header designation> | <file header
                       list>, <file header designation>

<file header designation> ::= <file ID> | <file header attributes>

<header attribute mnemonic> ::=

   ROWADDRESS (<integer>) |
   ROWSIZE |
   MAXRECSIZE |
   BLOCKSIZE |
   MINRECSIZE |
   EOFSEGMENT |
   EOFBITS |
   SAVEFACTOR |
   MODE |
   FILETYPE |
   UNITS |
   SIZEMODE |
   SIZEOFFSET |
   SIZE2 |
   DATE |
   LASTACCESSDATE |
   ROWS |
   IAD |          (Boolean valued)
   FILEKIND |
   DUPLICATED |  (Boolean valued)
   ROWCLASS ( integer )

<value> ::= <integer> | <boolean constant> | <address specification>

<boolean constant> ::= TRUE | FALSE

<address specification> ::= EU <integer> ADDRESS <integer>
```

Semantics:

The CREATE statement creates a header on disk for each file indicated in the list. No default header attributes are assumed, and any file of the same name will be overridden and the previous header attributes lost.

The UPDATE statement assumes a file header on disk and the only header attributes changed will be the attributes explicitly appearing in the header attribute list.

The SAVE statement reads the indicated file headers from disk, updates header attributes as appropriate, and writes the updated header to a tape with the name specified.

The RESTORE statement looks on the tape specified for each file header in turn as indicated in the <file header list>. When the file header is found, the header is modified by any attributes listed and the header is written on disk. If a header for this file already existed on disk, it will be overridden. If the file is not found, an error occurs. This search on tape is sequential and requires that the <file header list> be ordered such that the tape search will be sequential and forward. If the colon is replaced by a semicolon and no <file header list> occurs, every header on tape is written to disk with no change in the header attributes.

The LIST statement has two functions: to list the headers from a tape as created by the SAVE statement and to list headers of files appearing on disk. When listing from tape all the headers may be listed when the file list does not appear, or if the file list is used, only those headers for the files indicated will be listed. If FROM <external tape ID> does not appear in the LIST statement, disk is assumed and the file list must appear. No attribute list is allowed in the LIST statement.

The optional STOP statement terminates scanning of any remaining input cards.

Valid IADMAPPER Statements

```
CREATE:  A/B (ROWS=11, MAXRECSIZE=30, ROWSIZE=18166,
         ROWADDRESS(9)=EU 2 ADDRESS 0);
UPDATE:  C(DATE=72001);
SAVE TO HEADER/TAPE:  A/B, C(ROWS=25);
RESTORE FROM HEADER/TAPE:  A/B(DATE=72005);
RESTORE FROM HEADER/TAPE;
LIST:  A/B, C;
LIST FROM HEADER/TAPE;
LIST FROM HEADER/TAPE:  C;
STOP;
```

Invalid IADMAPPER Statements

```
CREATE:  A/B;                    % NO ATTRIBUTE LIST
UPDATE;                          % NO FILE HEADERS SPECIFIED
SAVE: C;                         % NO TAPE NAME GIVEN
SAVE TO HEADER/TAPE;             % NO FILE HEADERS SPECIFIED
RESTORE TO HEADER/TAPE: C;       % MISSING FROM
LIST;                            % NO FILE LIST FOR DISK
LIST: C(ROWS=5);                 % ATTRIBUTES NOT ALLOWED IN LIST
```

Example 1.

In order to gain a better understanding of how these functions may be used, assume the existence of installation "A" which does all of its critical data processing against two "large" data files, a NAMEAND-ADDRESS file and a HISTORY file. Since "A" is a B 6700 installation

UTILITIES

currently running under our standard software, these files are stored on head-per-track disk files with backup on magnetic tape.

Conversion

To take advantage of Installation Allocated Disk, "A" must first decide which files he wishes to convert to IAD, how many SU's he requires for storing each file and which SU's he wishes to use. Let's assume that "A" has a disk configuration as follows:

```
EU 1  SU 1 - 5
EU 2  SU 1, 2, 3
EU 3  SU 1, 2, 3
```

"A" has determined through running LISTDIRECTORY (or IADMAPPER) that the NAMEANDADDRESS file and the HISTORY file each require two SU's for storage. "A" decides to structure his files so that the NAMEANDADDRESS file will reside on EU 2, SU's 1 and 2. SU 3 will be reserved for backup. Similarly, the HISTORY file will reside on the first two SU's of EU 3 with the third SU reserved for backup. "A" can accomplish this by:

1. Doing a COLD START with the following parameter card

```
RESERVE EU 2, 3
```

   After the COLD START EU's 2 and 3 are set aside as IAD and EU 1 is used for MCP, system files, programs, overlay space, etc.

2. Loading SYSTEM/IADMAPPER from the system tape and inputting the following deck.

```
<!> RUN SYSTEM/IADMAPPER; EBCDIC
CREATE: NAMEANDADDRESS (ROWS = 2, ROWSIZE = 218000,
        MAXRECSIZE = 30, ROWADDRESS(0) = EU 2 ADDRESS 0,
        ROWADDRESS(1) = EU 2 ADDRESS 218000),
HISTORY (ROWS = 2, ROWSIZE = 218000, MAXRECSIZE = 30,
        ROWADDRESS(0) = EU 3 ADDRESS 0,
        ROWADDRESS(1) = EU 3 ADDRESS 218000),
NAMEANDADDRESSROW1 (ROWS = 1, ROWSIZE = 218000, MAXRECSIZE = 30
            ROWADDRESS(0) = EU 2 ADDRESS 0),
NAMEANDADDRESSROW2 (ROWS = 1, ROWSIZE = 218000, MAXRECSIZE = 30,
            ROWADDRESS(0) = EU 2 ADDRESS 218000),
HISTORYROW1 (ROWS = 1, ROWSIZE = 218000, MAXRECSIZE = 30,
            ROWADDRESS(0) = EU 3 ADDRESS 0),
HISTORYROW2 (ROWS = 1, ROWSIZE = 218000, MAXRECSIZE = 30,
            ROWADDRESS(0) = EU 3 ADDRESS 218000);
<!> END
```

The first two file specifications cause directory entries to be built for the NAMEANDADDRESS and HISTORY files. Each SU contains 218000 segments so the files have been set up as two rows each where each row is an appropriate SU of IAD. The remaining four file specifications redefine these two files so that there is a separate directory entry for each row of each file.

Installation "A" then loads his files from tape via the control statements:

```
?COPY NAMEANDADDRESS ONTO NAMEANDADDRESS, HISTORY ONTO HISTORY
FROM MASTERTAPE; END
```

Operation

At this point "A" is on the air. The critical files for "A" are now located on IAD, and all programs which process these files function the same as when the files were maintained on system disk, no reprogramming is required.

If "A" decides to take advantage of his backup SU's, he may now copy his database to tape as 1 single row file by inputting:

```
?COPY NAMEANDADDRESSROW1 TO BT1, NAMEANDADDRESSROW2 TO BT2,
HISTORYROW1 TO BT3, HISTORYROW2 TO BT4; END
```

Recovery

Suppose that some time later SU 1 of EU 3 were to fail. The system would continue running unhampered except for particular programs which were accessing data off this SU. To recover this data "A" would input:

```
<I>RUN SYSTEM/IADMAPPER; EBCDIC
 UPDATE: HISTORY
       (ROWADDRESS(1) = EU 3 ADDRESS 436000),
       HISTORYROW2(ROWADDRESS(0) = EU 3 ADDRESS 436000);
<I>END
```

followed by

```
  ?COPY HISTORYROW2 ONTO HISTORYROW2 FROM BT4; END
```

At this point "A" has recovered all of his data on his backup SU and is free to run on-line diagnostics against the failing SU or take it off line for maintenance.

Notice that various other forms of recovery are possible. For instance two copies of certain files could be maintained on IAD. Recovery then could be effected without reloading files from tape.

UTILITIES

Example 2.

If installation "A" had been satisfied (temporarily at least) with the physical location of his critical files on disk, he could have avoided reloading files at conversion time by doing the following:

1. Load SYSTEM/IADMAPPER

2. Inputting

```
<I>RUN SYSTEM/IADMAPPER; EBCDIC
 SAVE TO IADSPECS:NAMEANDADDRESS,HISTORY;
<I>END
```

SYSTEM/IADMAPPER would create an output tape file called IADSPECS which contained copies of the disk headers for the NAMEANDADDRESS and HISTORY files.

3. Do a COLD START reserving the disk on which these two files reside. This can be determined from the output listing produced by running IADMAPPER in step 2. COLD START to the new MCP which supports IAD.

4. Input the following control deck:

```
<I>RUN SYSTEM/IADMAPPER; EBCDIC
 RESTORE FROM IADSPECS;
<I>END
```

INTRINSICS FUNCTIONS

Use:

To make effective use of main memory by providing a single copy of the object code file for each of a number of commonly used routines. The locations of the object code files for these reentrant routines are known to the MCP and indirectly to the compilers in which they are used, and thus these code files are available to user programs as intrinsic functions.

LIBRARY MAINTENANCE

Use:

To perform (by means of a process of the MCP) library utility operations. The library maintenance process may be activated either by the entry of control statements at the input keyboard or by the use of control cards.

Refer to CHANGE, COPY, and REMOVE control statements.

LIST DIRECTORY

Use:

To produce a listing of the names of the files that are stored on disk.

Control Statement Used in Calling SYSTEM/LISTDIRECTORY:

```
<1> RUN SYSTEM/LISTDIRECTORY; END
```

Keyboard Messages Used in Calling SYSTEM/LISTDIRECTORY:

DIR (used in producing a listing of the names of all files on disk)

PD (used in producing a selective display of names of files on disk; refer to Disk Directory Table)

MAP Option:

If the program SYSTEM/LISTDIRECTORY is compiled with the MAP option set, then, in addition to the list of files, the program will also provide the disk addresses and sizes of disk areas in use by each file, a list of areas (sorted by address) which can be made available by the removal of one file, and a mapping of disk checkerboarding.

SYSTEM/LISTDIRECTORY is released with the MAP option set, but it can be recompiled with the option reset if desired. The program will run more rapidly if the MAP option is reset.

UTILITIES

If SYSTEM/LISTDIRECTORY is run with a taskvalue of 1, the effect is as if it had been compiled with MAP reset.

For example:

```
<1>RUN SYSTEM/LISTDIRECTORY; VALUE=1; END.
```

Label-equation of the internal file D to a directory title causes only the files in that directory to be listed.

For example:

```
<1>FILE D(TITLE = SYMBOL)
```

LOG ANALYZER

The system input message LOG will invoke LOGANALYZER, with the default file SYSTEM/SUMLOG. It accepts the syntax in figure 6-1. There are two independent criteria on which log printout is selected. First, a particular log file and time interval are chosen. Within that range of records, the second criterion is applied: which log entry type to select. If no time interval is given, the default is the entire time interval of the specified log. If no log entry type is specified, the default is all record types.

The options BOJ, EOJ, HL and MSG print all records of those types from the log. EOT includes both EOT and EOJ records; BOT includes both BOT and BOJ records. ABORT is equivalent to EOT. IO includes open and close records. ALL prints the entire log. The options DCP and MCS will print all DCP and MCS records, respectively.

The MIX command will print BOJ, BOT, IOERROR, EOT, EOJ and message records, either for all task numbers or for the task or tasks listed. Note that the mix number refers to a specific task and not to a job. If the mix number refers to a job, only BOJ, EOJ, LOGON, and LOGOFF records will be printed.

The TASK, JOB, and SESSION (which is synonymous with JOB) commands are expansions upon the MIX command. The TASK command yields all entries associated with the indicated task (if one corresponds to the description). This task may be specified by mix number or task name. JOB and SESSION commands yield all entries associated with the indicated job and its tasks. This job (session) may be specified by mix number or job name. When a task or job name of the form filename/= is specified in a TASK, JOB, or SESSION command, LOGANALYZER references tasks or jobs with names the left-most parts of which are identical to the filename.

For example:

```
LOG JOB "A/B/="
```

will reference all jobs whose names begin with "A/B/".

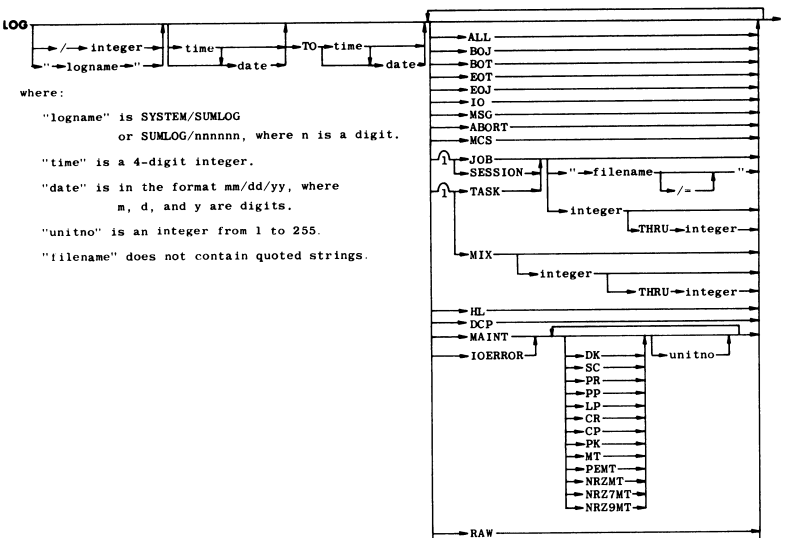


Figure 6-1. Syntax of LOG System Input Message

UTILITIES

IOERROR is similar to MAINT, but the latter prints additional information such as memory access errors. These are accumulated internally and written out to disk only after a certain number occurs, since logging each occurrence would aggravate the memory access problem.

RAW produces an unanalyzed dump of the log file.

NOTE

The OPEN log entry contains a new form of "unit type" specification for line printer files as follows:

(UE)LP = unbuffered EBCDIC LP
(BE)LP = buffered EBCDIC LP
(UB)LP = unbuffered BCL LP
(BB)LP = buffered BCL LP

Input Examples:

LOG: Retrieves all job entries followed by all miscellaneous entries.

LOG BOJ: Retrieves all beginning-of-job entries.

LOG JOB 546: Retrieves log for job number 546.

LOG "SYSTEM/SUMLOG" ALL: Retrieves all entries in the current system log.

LOG "SUMLOG/000005": Retrieves all entries in file SUMLOG/000005.

LOG 1100 MIX 123: Retrieves all entries for mix number 123 from 1100 hours to present.

LOG 5/11/70: Retrieves all entries for date 5/11/70.

LOG 1200 5/11/70 TO 1700 5/11/70 MIX 345: Retrieves all entries for mix number 345 from 1200 on 5/11/70 to 1700 on 5/11/70.

LOG IO: Retrieves all I/O (i.e., OPEN and CLOSE) entries for all jobs in the log. (Same as LOG FILE.)

LOG MAINT: Retrieves all present maintenance log entries.

LOG MAINT DK: Retrieves all maintenance log entries for head-per-track disk units.

LOG MAINT 99: Retrieves all maintenance log entries for all peripheral units with a unit number of 99.

LOG MAINT 60 MT97: Retrieves all maintenance log entries for all devices with a unit number of 60 and for magnetic tape unit 97.

LOG MAINT MT97 60: Retrieves all maintenance log entries for magnetic tape units 97 and 60.

LOG DCP: Retrieves Datacom error records.

LOG MCS: Retrieves all MCS type entries (e.g., LOG-ON, LOG-OFF, etc.).

LOG JOB "MYJOB": Retrieves all entries for jobs named MYJOB and their tasks.

NOTE

For detailed information concerning LOG-ANALYZER, refer to WORK FLOW MANAGEMENT REFERENCE MANUAL, Form No. 5000706, dated 16 April 1973.

Log Entries

The following discussion describes the present implementation of job log entries. The general format of each log entry is described first, followed by a discussion of the various defined types of log entries.

It is desirable to organize the log in such a way that a disk error or bad information in one record does not hamper retrieval of any other records. For this reason, all logical log records are cut into contiguous fixed-size physical records, with the first word in every record providing a record group description. MCP procedure WRITELOG performs the blocking of any size array row into log records and writes them into the correct jobfile. See figure 6-2.

All log entries contain, in the first word of every record, a Record Group Description (see figures 6-2 and 6-3). This word contains four fields:

1. The cardinality of this record within its group ("this is record m of n"), beginning with 1.
2. The total number of records in the group.
3. The job number of the associated job.
4. The task number of the task causing the log entry.

In the first record of each and every log entry, the next three words contain the following information:

UTILITIES

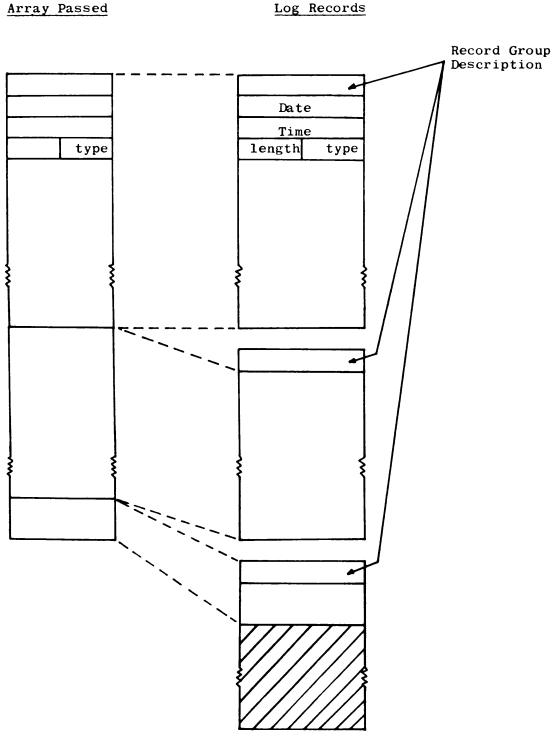
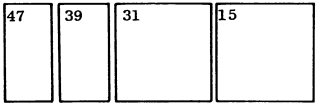


Figure 6-2. Blocking of Log Entry Into Records

Word 0: Record Group Description

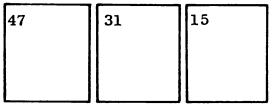


[47:8] Cardinality
[39:8] Number of records this entry
[31:16] Job number
[15:16] Task number

Word 1: Date, Binary(Julian)

Word 2: Time, 2.4 μs

Word 3: Type



[47:16] Length of entry in words
[31:16] Major type
[15:16] Minor type

Figure 6-3. Format of First Four Log Entry Words

UTILITIES

1. Word 1 contains the current date in binary Julian format.

2. Word 2 contains the current time in units of 2.4 usec.

3. Word 3 contains the log entry type. This word is divided into a length field, plus major and minor classifications fields. The length field contains the number of words in the log entry. The major field gives a rough classification according to originator; the minor field completes the classification of the entry.

When analyzing the log, a log program should reassemble the record before using the word numbers. These numbers all refer to the words as contained in an array, not as contained in the file. See figure 6-2.

The physical records of each log entry will always be contiguous in the SUMLOG. These entries will be in chronological order. The records of a single log entry will not be split across two different logs by an LR, but entries relating to a single job may be split.

Log Entry Types

There are seven major classes of log entry records (identified by major type numbers), each having subclasses (identified by minor type numbers). The log entry classes are as listed in table 6-1.

NOTE

For a detailed description of log entry formats, refer to WORKFLOW MANAGEMENT USER'S GUIDE, Form No. 5000714, dated 16 April 1973.

MAKEUSER

Use:

Creates a file that contains information concerning remote users for the Message Control System (MCS). (For detailed information concerning the MAKEUSER program, refer to Section 15 of B 6700 SYSTEM MISCELLANEA, Form No. 5000367, dated 16 April 1973.)

Table 6-1. Log Entry Classes

MAJOR TYPE	MINOR TYPE	RECORD CLASS
1		JOB RECORD
	1	Beginning-of-Job (BOJ) Record
	2	End-of-Job (EOJ) Record
	3	Beginning-of-Task (BOT) Record
	4	End-of-Task (EOT) Record
	5	File Open Record
	6	File Close Record
	7	Job Rejection Record
	8	Aborted History Record
2		MAINTENANCE RECORD
	1	Mainframe Configuration Record
	2	Peripheral Configuration Record
	3	Maintenance Comment Record
	4	Memory Access Error Record
	5-9	Reserved for later implementation
	10	I/O Error Record
3		STRING RECORD
		To be specified
4		MCS RECORD
	1	Session Log-On Record
	2	Session Log-Off Record
	3	RJE Control Card Record
	4	MCS Message Record
	5	Session Time-Accumulation Record
5		DCP MAINTENANCE RECORD
	1	DCP Initialization Record
	2	MCS Initialization Record
	3	DCP Fault Error Record
	4	MCS Result Message Record
6		MISCELLANEOUS RECORD
	1	Halt/Load Record
	2	Log Release Record
	3	SETSTATUS Record
	4	Security Violation Record
7		INSTALLATION RECORD

UTILITIES

SYSTEM/PATCH Utility Program

The patch merge program (SYSTEM/PATCH) is an ALGOL utility program used to merge one or more patch decks into a single patch deck (on disk) which may be used as the input CARD file for an ALGOL, ESPOL, DCALGOL, COBOL, or FORTRAN compilation. The present program is specifically designed to generate a patch deck to be used when compiling a standard Burroughs software item.

The patch decks to be merged by the program may be EBCDIC- or BCL-coded. The merged patch deck may be either BCL- or EBCDIC-coded if the input patches are BCL-coded; the merged patch deck must be EBCDIC-coded if the input patches are EBCDIC-coded.

The patch program merges all input patch records by sequence number. Only numeric or blank sequence numbers are accepted. The program allows resequencing and patching into resequenced areas of a patch.

Files Used by SYSTEM/PATCH

The following files are visible to the user:

a. PRNT - The output printer file.

b. CARD - The input file containing patches to be merged by the program.

c. PATCH - The output disk file containing the merged patches.

d. TAPE - The old symbolic disk file against which the input patches are to be compared.

NOTE

The TAPE file is necessary for resequencing. If it is needed and has not been label-equated via a WFL FILE card, then the program will ask for its TITLE via an ACCEPT message. The TITLE may not contain quoted strings in this case.

The mode of the CARD file determines the output mode of the PATCH file. The old symbolic and any patches to be accessed from disk are translated if necessary. If the CARD file is EBCDIC-coded, then BCL-coded cards not containing multiply signs can be input and will be translated. (See discussion of "Control Cards" in the following paragraphs.)

Card Input:

a. Dollar Cards

The following compiler options are recognized by SYSTEM/PATCH:

SEQ
VOID
VOIDT
MERGE
GO TO
BUMP TO

All other options will be passed to the compiler via the PATCH file but ignored by SYSTEM/PATCH. Such compiler options as AREACLASS, INSTALLATION, LEVEL, LIMIT, and VERSION are checked for format since their associated parameters could cause invalid actions if improperly specified.

A dollar card is defined to be a card with a "$" in column 1 (or column 7 for COBOL). A "$" in any other card column is not recognized.

b. Patch Delimiter Cards

Each input patch must be immediately preceded by a card with "$#" in columns 1 and 2 (columns 7 and 8 for COBOL - i.e., when the "COBOL" patch control option is set). The remainder of the card may be used for comments. (See discussion of "LABEL" and "MARK" patch control options in later paragraphs.)

c. Control Cards

Control cards to SYSTEM/PATCH are signified by a "$." in columns 1 and 2 (columns 7 and 8 for COBOL - i.e., when the "COBOL" patch control option is set). Such control cards may only appear before all input patches or between input patches, except for $.DISK.

The following patch control options may be set or reset on these control cards:

1. Output Control Options

Output on PRNT can be generated by the following options if there were no fatal errors during the patch merging process.

(a) LISTP - List the input patches by patch deck. (The default state of LISTP is SET.)

(b) LIST - List the merged patch.

(c) COMPARE - Output a comparison of the merged patch with the old symbolic, giving all patch cards and pertinent cards from the old symbolic.

UTILITIES

(d) GUARD - Output a list of all sequence numbers of patches merged into a guarded area of the old symbolic. (See following NOTE.)

(e) SINGLE - Single-space the listings.

NOTE

The GUARD option allows specific portions of the old symbolic to be "guarded" in such a manner as to produce a list of patches merged into those areas. Guarded symbolic code should be specified as follows:

```
$.GUARD m - n  comment
```

On this card "m - n" indicates the range of sequence numbers of cards to be guarded (n must be greater than m), and "comment" may be any character string. When a patch card is merged into the old symbolic between sequence numbers m and n, then the sequence number of the patch card, the number of the patch, and the comment from the $.GUARD card is written to the PRNT file.

Up to 100 areas may be guarded in this manner.

The GUARD option must be the last control option named on a "$." card.

2. Other Options

(a) COMPILE - Zip the compilation of the old symbolic with the merged patches if no fatal errors were discovered. Control cards for the compilation are given on "$*" cards appearing after COMPILE is set (but not within an input patch). The CARD and TAPE files are label-equated automatically by SYSTEM/PATCH.

For example:

```
$.SET COMPILE
$*COMPILE A/B WITH ALGOL LIBRARY
$*ALGOL FILE NEWTAPE(TITLE = S/A/B)
```

These $* cards may not contain comments. The "$*" must appear in columns 1 and 2; thus, "COBOL" may not be set when such cards appear among the input patches.

(b) LABEL - (for use with ALGOL, ESPOL, and DCALGOL symbolics). Put label information right justified on each card. This enables the user to identify the

source of cards in the symbolic at any future time. A typical use would be to place the implementor's initials in a fixed field on the patch delimiter cards and have this placed on each card in the patch.

The information is taken from the patch delimiter ($#) cards. The LABEL control card gives the starting column of this information. The label information is terminated by the first blank, e.g., $.LABEL 3 will cause SYSTEM/PATCH to take the string of non-blank characters starting in column 3 of the patch delimiter, to preface it with a "%", and try to place it right justified on each card in the patch. If any card has non-blank characters in the pertinent right-most columns, the label will be omitted and a non-fatal warning issued. Users are cautioned against continuing strings across card boundaries when using this option.

(c) DISK - Place the information on the given disk file into the input at this point. This information should not contain patch delimiter cards or dollar cards with "VOID" or "SEQ" options (i.e., it should look like the output (PATCH file) of SYSTEM/PATCH).

```
Example:  $.DISK X/Y/Z
```

(d) BCL - Treat the following patches as BCL input. If the mode of the input file is EBCDIC, a soft translate is performed. This option is not needed if the input mode is BCL.

(e) COBOL - Treat the input in COBOL format: Sequence numbers in columns 0 through 6, $ signs in column 7. After this option is set, all control cards and patch delimiters must start in column 7.

(f) DUMP - If a fatal error occurs in patch N, merge the first N-1 patches and lock them on a disk file. If the patch file would have had the title "X/Y/Z", then this file will have the title "DUMP/X/Y/Z". The DISK control card can be used to restart the merge without rereading the first N-1 patches.

(g) MARK - (for use with ALGOL, ESPOL, and DCALGOL symbolics). Place Mark level information in columns 81 through 88 of the merged patch. This information is taken from each patch delimiter ($#) card for which MARK is set immediately after the first non-blank ("noise") character string on the card.
For example:

```
$.MARK LABEL 3
$#XYZ 2.03.045
```

In this example, all cards from this patch will contain "2.03.045" in columns 81-88 in the merged patch. These cards will also be labeled "%XYZ" (as discussed under "LABEL").

The MARK option may also be used in conjunction with the ALGOL, ESPOL, and DCALGOL compiler option "VERSION" in the following manner.

The format of the VERSION compiler $ card is:

```
$VERSION  VV.CCC,
```

where VV represents the Version,
the first V represents the Mark number,
the second V represents the Level number,
and CCC represents the Cycle.

When compiling from SYSTEM/PATCH with MARK set, then

If on the $# card the first string after the noise string is a string of three of fewer digits and VERSION is set, then

a. The ALGOL compiler will replace the three-digit patch number with the string of 10 characters as follows: version (2 characters), period cycle (3 characters), period patch (3 characters); i.e., VV.CCC.PPP.

b. The ESPOL compiler will replace the three-digit patch number with the string of eight characters as follows: version (2 characters), cycle (3 characters), patch (3 characters). When printed on a listing, the ESPOL compiler will insert periods between the Version and Cycle, Cycle and Patch.

(h) EXECUTE - Zip a specified program execution if no fatal errors were discovered. Control cards for the execution are specified on "$*" cards as for COMPILE. The terminal "END JOB" control statement is supplied by SYSTEM/PATCH.

EXECUTE and COMPILE may not both be set when SYSTEM/PATCH finishes processing the last patch deck. A compilation of the symbolic or an execution may be zipped, but not both.

Example:

```
$.EXECUTE
$*JOB MYJOB;BEGIN
$*RUN E/H; VALUE = 1
$*FILE CARD(FILETYPE=7,KIND=DISK,TITLE=MY/PATCH)
```

All of the above options may be SET, RESET, and POPped similarly to the compiler options. If no action is specified, SET is assumed, but, unlike the compilers, no action is taken to options not specifically mentioned.

Sample Input Deck:

```
<I> JOB PATCHIT; BEGIN
<I> RUN SYSTEM/PATCH
<I> FILE TAPE (TITLE=SYMBOL/FILE)
<I> FILE PATCH (TITLE=MY/PATCH); EBCDIC CARD
 $.LIST SINGLE COMPARE
 $.MARK GUARD 300-1000 GUARD AREA ONE
 $.GUARD 2050-31000 GUARD AREA TWO
 $.SET COMPILE
 $*COMPILE SYSTEM/FILE WITH ALGOL LIBRARY
 $#ABC 2.03.001 FIRST ABC PATCH
 $MERGE CHECK LIMIT 6 SINGLE LIST
 $SET STACK
    .
    .
    .
 $#ABC 2.03.002 SECOND ABC PATCH
 $.DISK S/PATCHFILE
 $#ABC 2.03.003 THIRD ABC PATCH
    .
    .
    .
<I> END JOB
```

To compile with the merged patches, the compiler file CARD must be label equated to the patch file.

Example: `<I> COMPILER FILE CARD (TITLE=MY/PATCH,KIND=DISK, EXTMODE=BCL)`

Explicitly setting extmode to BCL or EBCDIC is necessary, since the external mode of a disk file is not currently set at file open time.

Brief Description of Algorithm

The program uses two passes to produce the merged patch. The first pass reads the cards from CARD and writes them to a scratch disk file. If no errors have been detected, the patches are merged onto the disk file "PATCH". This file is then locked, the EXCEPTIONEVENT of the parent

UTILITIES

(if any) is caused, and the message "OK TO COMPILE" is displayed on the SPO (unless the COMPILE option is set--in which case the compile is fired off). Any post-merge output (list or compare) is then generated in parallel with the compile.

The following actions can occur during the first pass:

a. ERROR CHECKING - Sequence errors and failure to POP recognized (SEQ, VOIDT, and VOID) options which have been set will cause fatal errors.

b. RESEQUENCING - All occurrences of the "SEQ" compiler option are eliminated by merging the input cards with the appropriate sections of the symbolic file and preceding patches. They are then written to the scratch file preceded by a "$SET VOIDT" and followed by a "$POP VOIDT".

   Note 1: This procedure allows subsequent patches to patch into the resequenced area.

   Note 2: Binary searching is done to locate the pertinent areas, hence sequence errors in the old symbolic could give unpredictable results.

   Note 3: Any resequencing which would cause a sequence error in the new symbolic (e.g., the increment was too large) will cause a fatal error.

c. VOIDING - All occurrences of the VOID compiler option are replaced by VOIDT's with intervening cards deleted. This prevents a VOID from eliminating cards in a subsequent patch.

For IPC, if no errors are encountered, then a TASKVALUE of 1 is returned; otherwise, a negative TASKVALUE with a magnitude equal to the number of errors is returned.

PRINTCOPY

The program SYSTEM/PRINTCOPY is used to print the supervisory console traffic from the disk file generated by SYSTEM/HARDCOPY. (Refer to the discussion of the HARDCOPY program in this section.)

PRINTER BACKUP

Use:

SYSTEM/BACKUP is used to print or punch backup files. SYSTEM/BACKUP is run by entering a ?PB command at the SPO or by using a PB control statement. The syntax for the PB control statement is as follows:

```
<PB task attribute> ::= PB <input designator> <output unit>
                        <option part>

<input designator> ::= MT<unit number> | D<mix number> |
                       D<PB file ID>

<unit number> ::= <integer>

<mix number> ::= <integer>

<PB file ID> ::= <mix number> <slash> <file title>

<slash> ::= /

<file title> ::= {title part of BD entry}

<output unit> ::= <empty> | LP<unit number>

<option part> ::= <empty> | <associative option> | <general option> |
                  <general option> <associative option>

<associative option> ::= <key declaration> <area specifier> |
                         RECORD <start rec> |
                         RECORD <start rec> <stop rec>

<key declaration> ::= KEY <key specifier>

<key specifier> ::= ALGOL | COBOL | FORTRAN | REPORT |
                    <column> <key length>

<column> ::= <integer>

<key length> ::= <integer>

<area specifier> ::= RANGE <start> <stop> | EQUAL <sequence> |

<start> ::= <integer> | "<EBCDIC string>"

<stop> ::= <integer> | "<EBCDIC string>"

<sequence> ::= <integer> | "<EBCDIC string>"

<start rec> ::= <integer>

<stop rec> ::= <integer>

<general option> ::= <general option part> |
                     <general option> <general option part>

<general option part> ::= SAVE | COPIES=<integer> |
                          SINGLE | DOUBLE
```

UTILITIES

Backup Files:

The MCP will build a backup file when one is specified by MCP system options, by program file attributes, or by the system operator (by use of the OU keyboard message).

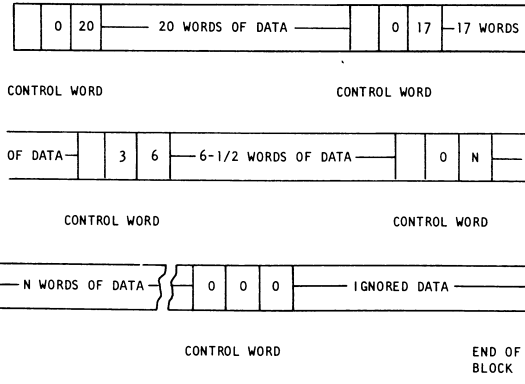
A backup file on tape is named BACKUP/<file name>. Backup tapes may be written as multireel files and as multifile reels. Backup files on disk are named BD/<mix index>/<file name>. The <file name> is the name used by the program which created the file.

System BACKUP files are variable-length record, fixed-length block files. Each block is 300 words long. Within a block, each logical record is composed of one control word followed by 0 or more words of data. A terminal control word of all zeros indicates that there are no more records in the present block.

Each control word is divided into specific fields. These fields are as follows:

Field	Contents
47:28	Identical with the corresponding portion of an I/O CONTROL WORD.
19:3	Character count residue for the data record (if the record to be printed consists of complete words, the value of this field will be zero).
16:17	Word count for the following data in the record in full words, not counting the control word.

Records in a BACKUP file are not split across blocks. Thus, if the last record in a given block ended in word 297 and the next record were 10 words long, then word 298 would be a control word containing all zeros. This control word indicates an end of the present block, and the next record will begin at the start of the next block. The figure below illustrates a typical block of BACKUP records.



Format of System Backup Files

UTILITIES

RLTABLEGEN

The SYSTEM/RLTABLEGEN program constructs value arrays which it binds to the MCP, providing for system acceptance of nonstandard tape labels. The following paragraphs describe the ability of the II.2 MCP to accept installation-defined tape labels. A description of the operation of the RL TABLE GEN program follows this discussion.

Installation-Defined Tape Labels

The MCP's label recognition routine (READALABEL) is able to recognize a large class of installation-defined labels in addition to standard tape labels.

To invoke this ability, it is necessary to write a description of the desired nonstandard labels, pass this description through a table building program, (i.e., SYSTEM/RLTABLEGEN) bind the resulting VALUE ARRAY to the MCP, (accomplished by the program), and then CM to the resulting MCP.

Tapes with installation-defined labels may then be used in the usual manner, with the following restrictions:

a. The label records must be a contiguous group of BCL or EBCDIC records at the beginning of the tape, separated from the data by one tapemark.

b. The label type must be determinable by examination of the first record only.

c. Label record size must be less than 230 characters.

d. The MCP will not recognize any user labels (e.g., UTL, UHL, etc.) on these label types.

e. No "up tape" labels will be recognized. That is, once the file is opened and the tape is positioned to the beginning of the data, the tape will become treated as an unlabelled tape.

f. Because installation labels are checked for before standard labels, the recognition sequence must be complete enough not to interfere with standard label recognitions.

Use of SYSTEM/RLTABLEGEN

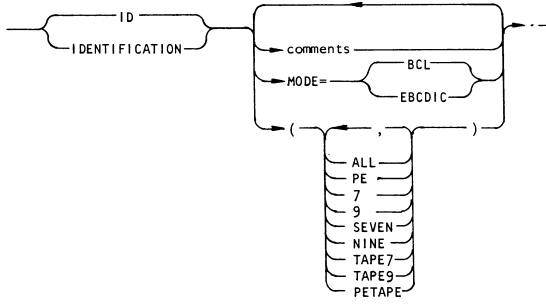
The following is a description of the input to the program SYSTEM/RLTABLEGEN, which is used to create the value array installation labels which are used by the MCP to identify nonstandard labels.

The input to SYSTEM/RLTABLEGEN consists of any number of label descriptions. The elements of such label descriptions are discussed here.

Label Description Format

A label description must consist of two divisions: the ID (or IDENTIFICATION) division and the FIELD (or FIELDS) division. A label description is constructed of the items discussed here in order.

The ID division is demarcated by the ID statement, the syntax for which is as follows:



Examples:

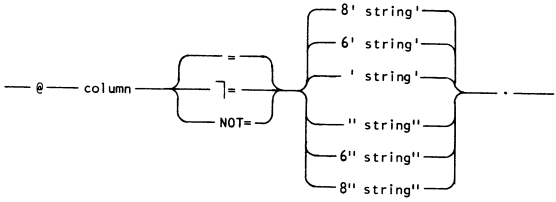
```
IDMODE = EBCDIC FOR (PE, NINE) TRACK TAPES.

IDENTIFICATION FOR BRAND-X LABELS (7) MODE = BCL.
```

NOTE

To use the "railroad" notation for syntax, simply start on the left edge and proceed to the right. A choice may be made between ID or IDENTIFICATION, followed by any mixture of MODE clauses, comments, and a list of the tape types on which these labels may be found. The list of tapes is any combination of the identifiers listed, separated by commas and enclosed in parentheses. The ID statement must end with a period.

The recognition sequence follows the ID statement. The syntax of a recognition statement is as follows:

UTILITIES



NOTES:

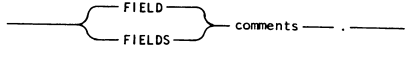
a. column is an integer.

b. string is any string excluding the quote character within the quotes, and excluding the apostrophe in the upper three strings.

As many recognition statements as needed may appear.

The FIELD division statement, which denotes the beginning of the description of the label, is described:

FIELD STATEMENT:



Following the FIELD statement is the description of the various fields found in labels. There are basically three types:

STRINGS, NUMBERS, and BIT FIELDS.

Each field is used to store file attribute information.

These label fields are described via the RECORD and field-location statements. The syntax of these statements is as follows:

NOTES:

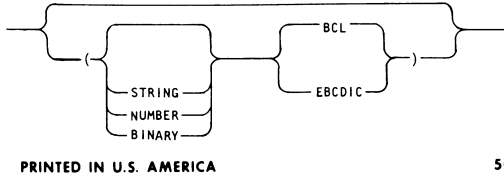
a. recordno. is an unsigned integer.

b. column is an unsigned integer.

c. length is an unsigned integer.

d. requalificatidn is expanded below.

Requalification:



UTILITIES

In the preceding syntax "column" and length" are integers representing the one-relative location and length of the field being described.

There are two basic statements: the RECORD number statement and the field-location statement. The RECORD statement is used to group field-location statements by the records in which they appear. Counting for both records and columns within records starts at one. Each record statement must reference a higher record than the previous one.

Field-location statements consist of a reserved word describing the file attribute with which the value in the field is associated followed by starting column location within the current record and length of the field.

Each attribute value is of an assumed type. The character type (BCL or EBCDIC) is assumed to be that expressed by the MODE clause in the ID statement. Attributes of STRING type may be requalified to override the expressed MODE, but may not be changed to type BINARY or NUMBER.

NUMBER or BINARY attributes may be requalified to any mode or type except STRING.

- MFID, FID

  Type STRING. Max length 17 characters. Either is optional. If both are missing or if both reference fields that turn out to be all blank, the tape will be labelled "UNTITLED.". All leading and trailing blanks are deleted from the two fields.

- RCDFRMT

  Type BINARY. The low order 8 bits of this field are assumed to be a BCL or EBCDIC letter. The following is a conversion table from letter to file type:

LETTER	FILE TYPE
F	0
D	1
V	2
I	4
L	5
Z	6
anything else	3

- PARITY

  Type BINARY. This field is automatically set to the parity of the label records, but if specified, must be in accordance with the way in which this label attribute works on B 6700 standard labels. (The low order bits = 1 imply standard parity).

- DENSITY

  Type BINARY. This field is automatically set based on the density of the label records. If it is specified, it must reflect the density in its low order two bits.

- UNITS

  Type BINARY.

- REEL, CYCLE, VERSION, CREATIONDATE, MAXRECSIZE, BLOCKSIZE, MINRECSIZE

  These attributes are all assumed to be integers; i.e., digit strings of no more than 11 digits. They all correspond to the file attributes of the same name. They are of significance only if file type 7 is set.

Program Specifications

Source input records are accepted by the program via the reader file with the internal name CARD. The program output file consists of the printer listing named LINE.

The line printer listing contains a pseudo-reproduction of the input statements which the program generates from the table it produces from the input statements. Successful operation has occurred when the essential information in the input records is contained in the output listing. The end of this listing contains the value array generated by the program.

Binding of the generated value array to the MCP is automatically accomplished by the RLTABLEGEN program by zipping the program binder. This program requires that the present MCP have the TITLE of SYSTEM/MCP, and the code file generated by the binder has the TITLE SYSTEM/NEWMCP.

Example.

Problem:

Recognize BCL labels on 7-track and 9-track drives that can be identified by ANSI VOL header. The only difference between this and B 6700 tape labels is that the SYSTEM-ID field is blank.

Solution:

The following deck will produce an MCP which will accept these labels.

UTILITIES

```
<!>RUN SYSTEM/RLTABLEGEN; EBCDIC CARD
 IDENTIFICATION FOR LOCAL LABEL SEQUENCE MODE=BCL (SEVEN, NINE).
   @ 1 = "VOL 1".
   @ 80 = '0'.
   @ 11 not = " ".
   @ 29 = " ".
 FIELDS.
   RECORD # 1.
      MFID    @ 12 for 17.
   RECORD # 2.
      FID              @  5 FOR 17
      REEL             @ 32 FOR 4
      CYCLE            @ 36 FOR 4
      VERSION          @ 40 FOR 2
      CREATION DATA    @ 43 FOR 5
   RECORD # 3.                  ,     .
      RCDFRMT          @  5 FOR 1
      BLOCKSIZE        @  6 FOR 5  (NUMBER EBCDIC).
      MAXRECSIZE       @ 11 FOR 5
      DENSITY          @ 16 FOR 1  (BINARY EBCDIC).
      PARITY           @ 18 FOR 1
<!>END
```

DM Utilities

Utility	Name of Source File	Name of Object File
DM clear	SYMBOL/DMCLEAR	SYSTEM/DMCLEAR
DM dump program	SYMBOL/DMPRINTIT	SYSTEM/DMPRINTIT
DDL formatter	SYMBOL/DDLDECOMPILER	SYSTEM/DDLDECOMPILER
DM converter	SYMBOL/DMUPDATE	SYSTEM/DMUPDATE
DM recover	SYMBOL/DMRECOVER	SYSTEM/DMRECOVER
DM restarter	SYMBOL/GETDMRESTARTFILE	SYSTEM/GETDMRESTARTFILE
DM reconstruction/ restoration	SYMBOL/DMROWRECOVERY	SYSTEM/DMROWRECOVERY

SYSTEM/DMCLEAR

DMCLEAR sets the database user count to zero. The database monitor would not go to EOJ unless this count were zero. If either SDL/STRUCTURE or DDL goes to an abnormal EOJ, either run DMCLEAR or halt/load. If DMCLEAR is to be run, the affected database directory must first be removed from disk (e.g., <1> REMOVE DM / <data-base-name>).

Then run SYSTEM/DMCLEAR. With no taskvalue, the resulting SPO message ACCEPT:<DATA-BASE-NAME> should be answered with <mix-number> AX <data-base-name>. With a taskvalue of 1, all data base user counts are cleared.

SYSTEM/DMPRINTIT

DMPRINTIT is a utility dump program which formats and prints DM files, allowing the scanning of the contents of the data files and indices. The input file CARD is free form with spaces as delimiters and provides the following syntactic possibilities:

```
<DMPRINTIT data-card> ::= <data-base-card>|<mode-card>

<data-base-card> ::= <file-name> <limits 1> <limits 2> |
                     <all-card> | <errors-file-card> | <auditarchive-
                      card>

<all-card> ::= <data-base-name> <slash> ALL

<errors-file-card> ::= <data-base-name> <slash> ERRORS

<auditarchive-card> ::= <data-base-name> <slash> AUDITARCHIVE
```

UTILITIES

```
<file-name> ::= <data-base-name> <slash> <structure-name>
                <optional period>

<structure-name> ::= <unquoted form> | <quoted form>

<unquoted form> ::= {SDL-assigned-structure-number} | SDL

<quoted form> ::= "<unquoted form> <copy-number>"

<copy-number> ::= #1 | #2

<limits 1> ::= <empty> | <integer> | FROM <integer>

<limits 2> ::= <empty> | <integer> | TO <integer>

<mode-card> ::= FORMAT | NO FORMAT | ALPHA | NO ALPHA |
                ALPHA1 | NO ALPHA1 | HEX | NO HEX |
                USERPACK <packname> | NO USERPACK

<slash> ::= {the character "/"}

<optional period> ::= <empty> |.
```

Examples:

a. TEST/0027 13 (prints from 13 on)

b. TEST/ALL

c. TEST/0008. FROM 16 TO 32

d. TEST/"SDL#2"

Mode cards provide the following options:

1. FORMAT — formats the data before printing (default);
2. NO FORMAT — prints the data unformatted in hexadecimal characters;
3. ALPHA — causes the alphanumeric equivalent of the data to be printed on the right side of the page;
4. NO ALPHA — resets the above option (default);
5. ALPHA1 — causes the alphanumerics to be printed below each hex line;
6. NO ALPHA1 — resets the above option (default);
7. NO HEX — eliminates the hex line and prints only the alphanumerics;
8. HEX — resets the above option (default);

9. USERPACK <packname> allows access to files stored on named pack where the packname is not equal to the data-base name (e.g., audit files);

10. NO USERPACK resets the above option (default:packname=data-base name).

The capacity to produce a hex dump of any file on disk or diskpack (with at least a two-level title) is available through DMPRINTIT. This is particularly useful in printing the contents of the audit files.

SYSTEM/DDLDECOMPILER

DDLDECOMPILER formats and prints (and optionally, punches) the DDL specification of any or all sets of a data-base, given the presence of the DDL tables. The input syntax is as follows:

```
<data-cards> ::= <data-base-card> | <$-option-card>

<data-base-card> ::= DATA-BASE = <data-base-name> <set option>

<set-option> ::= <set> = <set-list> | <empty>

<set> ::= SET | SETS

<set-list> ::= ALL | <set-name-list>

<set-name-list> ::= <set-name> | <set-name-list>, <set-name>

<$-option card> ::= $ <option-list>

<option-list> ::= <option-word> | <option-list> <blank> <option-word>
                  | <empty>

<option-word> ::= <action-control-word> | <option-parameter>

<action-control-word> ::= SET | RESET | POP

<option-parameters> ::= HEADER | HEADERS | PUNCH | SINGLE
                        | SEQ <seq-option> | PAGE

<seq-option> ::= <integer> | <integer> + <integer>
```

Examples:

```
a. DATA-BASE = TEST   SETS = ALL

b. DATA-BASE = TEST   SET = S1, S2, S3

c. $SET PAGE SINGLE PUNCH SEQ 500 + 10
```

UTILITIES

A more complete semantic description is contained in Section 6 of the B 6700/B 7700 DATA MANAGEMENT MANUAL, Form No. 5000235, dated 11 January 1973.

SYSTEM/DMUPDATE

DMUPDATE is a one-time-only program that converts existing 11.3 databases to 11.4 databases. This saves the user from regeneration and recompilation tasks.

COMPILE INFORMATION: The file tape should be label-equated to "SYMBOL/DMUPDATE" and a $ MERGE card should be included in the deck.

RUN INFORMATION: The object job will look for a card file named CARD which is free-form as follows:

```
DATA-BASE = <data-base-name>;
```

The steps for running are:

1. Load the data-base to be updated;
2. Run SYSTEM/DMUPDATE;
3. If no errors are noted, the updated database is usable or the user may dump it to tape for later use (not forgetting to dump the errors file and the altered "overflow" file).

SYSTEM/DMRECOVER

Recovery of a data-base is accomplished by running SYSTEM/DMRECOVER. This procedure is invoked automatically by the data-base monitor when necessary. It may also be run by the user. Its parameter is a string consisting of the name of the SDL file (or the first copy, if duplicated) followed by a period. DMRECOVER recovers the data-base, writes all necessary restart files to disk, and indicates its successful completion by creating an empty file named DM | <data-base-name> | RECOVERED. It also keeps a log of any I/O errors it encounters in a duplicated file named DM | <data-base-name> | "RECOVERRS#1" and DM | <data-base-name> | "RECOVERRS#2". I/O errors may result in an unsuccessful recovery.

The recovery errors file has 3 word records with the following format:

Word 0	47:20	Structure number of file in error
	27:28	Record in error
Word 1	16:03	Type of file in error (0-data file or index table; 1-SDL file; 2-audit trail; 3-unused; 4-recovery errors file itself)
	13:01	Read or write error (0-read; 1-write)
	12:01	Copy # in error (0-first; 1-second)
	11:12	Blockfactor of file in error
Word 2	47:48	I/O result descriptor

SYSTEM/GETDMRESTARTFILE

All necessary restart files are produced automatically by DMRECOVER when it is run. However, if a DM job is DS-ed but the monitor goes to normal EOJ, recovery of the data-base is not appropriate. If the job is resubmitted using the delayed restart conventions, the user obtains his restart file by running SYSTEM/GETDMRESTARTFILE. The name of the restart file is:

```
DMRESTART | <data-base-name> | <internal-file-name> |
<11 decimal digits>
```

where the 11 digit field contains 3 subfields:

```
<4 DIGIT ID FIELD> <4 DIGIT JOB #> <3 DIGIT LOG.TASK #>.
```

The input file CARD is a list of keyword and value pairs (<KEYWORD>=<VALUE>) delimited by blanks. Keywords may be shortened and inexact but value must be exact (leading zeros, if applicable, must be punched). The parameters may appear in any order.

PARAMETER NAME	VALUE	REQUIRED
DATABASE NAME	=<database name>	YES
INTERNAL FILE NAME	=<intname of restartfile>	YES
MIX NUMBER	=<4 DIGITS>	YES
JOB NUMBER	=<4 DIGITS>	YES
DATE	= MM/DD/YY	*
JULIAN DATE	= YYDDD	*
TIME	= HH:MM:SS	*
AUDIT SERIAL NUMBER	= 4 DIGITS	*
ID	= 4 DIGITS	NO

* Either the audit serial number is specified or the date and time are, but not both.

Examples:

1. DATA BASE NAME = MYDB MIX-NUMBER = 0032
   JOB-NUMBER = 0031
   TIME = 00:01:00 DATE = 01/02/73
   INTERNAL FILE NAME = MYRESTARTFILE

UTILITIES

2. DATA = TESTDB AUDIT = 0003 JOB = 1234
   MIX = 1236 ID = 7777 INTNAME = RSF

Most of this input may be obtained from the run-time system-produced listing. The internal file name and the data-base name are obtainable from the program listing.

SYSTEM/DMROWRECOVERY

The user interface to reconstruction and restoration is through the program SYSTEM/DMROWRECOVERY. The basic intent is that the user may instruct the monitor to rebuild a data-base structure, but that the monitor via the errors file determines which rows shall be restored and which need to be reconstructed to completely rebuild all bad rows for that structure. Three basic commands are:

1. REMOVE which gives the user the ability to remove from the errors file the indication that a row or rows of a DM file had incurred a read error;

2. INSERT which enables the user to set read-error bits in the errors file for any row or rows;

3. REBUILD which signals the monitor to rebuild all bad rows for a particular database and structure.

Example:

```
DATA-BASE = T0047
   SDL
       REMOVE FROM ERROR FILE
            COPY ONE ROWS 2, 5, 7 THRU 11
            COPY TWO ALL ROWS
       AND REBUILD
            BACKUP FILE FOR COPY ONE IS
                          (DMS/T0047/"SDL#1"),
                       FOR COPY TWO IS
                         (DMS/T0047/"SDL#2")
   ;
   STRUCTURE 2
       INSERT INTO ERROR FILE
            COPY ONE ROW 13 AND COPY TWO ROWS 9-18
       REBUILD ALL ROWS
            BACKUP FILE
                FOR COPY ONE IS (DMS/T0047/"0002#1")
                    IAD CORRESPONDENCE IS 1=5, 2=6, 6=0
                FOR COPY TWO IS (DMS/T0047/"0002#2")
                    IAD CORRESPONDENCE IS 1=1, 3=3, 4=5
            AUDIT SERIAL NUMBER = 2;
```

A more complete description of the syntactic possibilities of DMROW-RECOVERY is offered in the 11.4 System Notes, pages 105 through 113.

DATACOM Utilities

SYSTEM/DCSTATUS

Use:

The SYSTEM/DCSTATUS program allows one to perform an analysis of the current run-time state of the B 6700 DATACOM subsystem. The program makes use of the DCSYSTEMTABLES intrinsic to gain access to tables maintained by the MCP (and DCP), and to perform a run-time snapshot analysis of them.

Selection of Output Options

The program must be supplied with an option list which specifies those elements of the DATACOM subsystem which are to be analyzed. The options are specified on a hierarchial basis, i.e.,

a. Analysis for a station only.

b. Analysis for all stations on a line.

c. Analysis for all lines on a cluster.

d. Analysis for all clusters on a DCP.

e. Analysis for all DCP's.

The hierarchial item is selected by using the key words: ALL, DCP, CLUSTER, LINE, or STATION; each higher order item is inclusive of all lower order items. Thus, if a cluster is specified, then the analysis is performed for all stations on all lines on that cluster.

Analysis output from the program will normally be sent to a site line printer. However, the program will detect if the printer file is label-equated to a remote file. For this case, the output format is modified so that the output will fit a 72-character line width.

Syntax of Options

```
<options> ::= <option list>

<option list> ::= <subsystem specification> |
                  <subsystem specification>; <option list>

<subsystem specification> ::= ALL | <DCP designate> | TABLES |
                              <cluster designate> | <line designate> |
                              <station designate> | <terminal designate>

<DCP designate> ::= DCP <DCP number>

<DCP number> ::= <unsigned integer>

<cluster designate> ::= CLUSTER <DCP number>, <cluster number>
```

UTILITIES

```
<cluster number> ::= <unsigned integer>

<line designate> ::= LINE <DCP number>, <line number> |
          LINE <DCP number>, <cluster number>, <line number>

<line number> ::= <unsigned integer>

<station designate> ::= STATION <lsn> <NDL option>

<lsn> ::= <empty> | <logical station number>

<NDL option> ::= <empty> | NDL

<terminal designate> ::= TERMINAL <remote type index>

<remote type index> ::= <empty> | <unsigned integer>
```

Semantics:

The TABLES option produces a raw hexadecimal dump of the DATACOM controller and DCP line and station tables. Tag bits are omitted from the dump.

The ALL option produces a complete analysis of the DATACOM network. The tables are dumped and an analysis of the line and station tables together with an analysis of each remote type is performed. All other options are subsets of this option.

The <DCP designate> option produces a full analysis of the designated DCP's lines and stations.

The <cluster designate> option produces a full analysis of the designated cluster's lines and stations.

The <line designate> option produces a full analysis of the designated line and its stations.

The <station designate> option produces the following output. If <NDL option> is empty, then an analysis of the tables maintained by the DATACOM controller is produced. If <lsn> is empty, then the analysis is produced for all stations. This will include those stations which are not currently assigned to a line, and will thus not be analyzed under the DCP, LINE or CLUSTER options. If NDL is specified, then the analysis is produced from the Network Information File, rather than from in-core tables and the DCPCODE file, and contains the NDL declarations for that station. These may not be the current attributes of the station, because modifications may be made at run-time via DCWRITES.

The <terminal designate> option produces the NDL specifications of terminals. This information is retrieved from the Network Information File. The <remote type index> is the index used by the DATACOM controller into a table which describes each terminal specified in NDL. In

physical terms, terminals are numbered in the sequence in which they appear in the NDL specification of the network.

Running Instructions

The SYSTEM/DCSTATUS program is compiled as a procedure with an array as a parameter. The option list is passed to the program via this array.

To compile the program, the following cards are required:

```
<!> COMPILE SYSTEM/DCSTATUS DCALGOL LIBRARY
<!> DCALGOL FILE TAPE (TITLE = SYMBOL/DCSTATUS)
<!> EBCDIC
  $ MERGE INSTALLATION
  $ SET LEVEL 2
  (the $ cards)
<!> END
```

To execute the program, the following cards are required:

```
<!> EXECUTE SYSTEM/DCSTATUS ("<options>")
<!> END
```

SYSTEM/DCSTATUS may also be executed via CANDE with all output being directed to the remote terminal initiating the execution. The proper CANDE command is:

DCSTATUS <options>

Examples:

a. `<!> EXECUTE SYSTEM/DCSTATUS ("LINE 0, 2; CLUSTER 0, 4");END`

   This will cause output to be sent to a site line printer.

b.
```
<!> EXECUTE SYSTEM/DCSTATUS ("ALL")
<!> FILE LINE (TITLE = M332, KIND = REMOTE)
<!> END
```

   This will cause the output to be sent to a remote station named M332.

c. `DC STATION 32`

   When this command is entered through CANDE, SYSTEM/DCSTATUS will be entered into the mix and executed, with information regarding station number 32 being returned to the originating remote terminal.

UTILITIES

Miscellaneous Information

SYSTEM/DCSTATUS may be called directly from SYSTEM/MCSII using the DP control statement. Output may be directed to a site line printer or to the remote device on which the command was entered.

If site is used, any number of commands may be entered simultaneously, output being directed to a site line printer. If remote is used, then output will be directed to the remote device on which the DP command was entered. It must be noted that in remote mode, only one copy of DCSTATUS may be run at a time. Care should be taken when using the remote option, as SYSTEM/DCSTATUS will necessarily produce voluminous output for the ALL, DCP and CLUSTER options, and it is not possible to terminate a program from a remote terminal when using SYSTEM/MCSII.

The DATACOM status intrinsic does not lock the various MCP tables that it accesses. It is therefore possible that the contents of the tables may change while the intrinsic is running. In particular, some flags maintained in the tables are set in a transient manner. It will therefore be a coincidence only that a run of DCSTATUS catches them set.

SYSTEM/DCSTATUS will only return meaningful information if DATACOM is initialized. If DATACOM is not initialized, then DCSTATUS output will indicate such a situation.

If, however, run-time system option number 12, AUTODC, is set, then a run of SYSTEM/DCSTATUS will cause an immediate initialization of the DATACOM tables, and information returned will be meaningful. Using this mode of operation, no DCP's will be fired up, and to start DATACOM activity a further "DC <DCP number>" must be entered on the SPO. The DATACOM tables will remain initialized after the DCSTATUS run. To return this memory area to the system, a DCP must be fired up, and then DS-ed.

SECTION 7
OPERATING PROCEDURES

SECTION 7 - CONTENTS

SECTION 7

OPERATING PROCEDURES

SYSTEM INITIALIZATION

There are two system initialization procedures used in bringing a new MCP from tape or user disk to system disk, starting at disk address 0: cold start and cool start. The cold start is a drastic measure undertaken with power on but with nothing on disk or undertaken when it is necessary to wipe out the disk directory and to rebuild it. The cool start is a less drastic measure than the cold start; in a cool start the disk directory is not destroyed.

Cold start and cool start operations are performed from a card reader by use of a loader deck and a set of system initialization function cards.

LOADER DECK

The loader deck is an EBCDIC deck that contains the object code produced by the compilation of the program SYMBOL/LOADER.

SYSTEM INITIALIZATION FUNCTIONS

The system initialization function cards are punched in EBCDIC and in free format. Selected system initialization function cards are placed at the end of the loader deck to form either a cold start deck or a cool start deck.

The syntax for system initialization function is as follows:

```
<system initialization function> ::= <date function> |
                                     <overlay function> |
                                     <directory function> |
                                     <load function> |
                                     <option function> |
                                     <terminal function> |
                                     <cold start function> |
                                     <list function> |
                                     <system disk function> |
                                     <peripheral configuration function> |
                                     <partition function>
```

The cold start, list, system disk, peripheral configuration, and partition functions are used only by the B 7700 system.

SYSTEM INITIALIZATION

DATE FUNCTION

Syntax:

<date function> ::= DATE <month> <slash> <day> <slash> <year>

<month> ::= <digit> <digit> | <digit>

<slash> ::= /

<day> ::= <digit> <digit> | <digit>

<year> ::= <digit> <digit> | <digit>

Example:

DATE 5/15/71

OVERLAY FUNCTION

Syntax:

<overlay function> ::= OLAYROW <row size>

<row size> ::= <integer>

Semantics:

1. Row size specifies the number of segments to be placed in the initial row of the MCP overlay file, as well as the size of each row of a stack's overlay file.

2. The inclusion of an OLAYROW card will force a cold start. A cold start removes all disk directory information. (Note, however, that the resident MCP still has access to itself.) This card must be omitted from the cool start deck.

Example:

```
OLAYROW 700
```

DIRECTORY FUNCTION

Syntax:

<directory function> ::= <directory location> | <directory row>

<directory location> ::= DIRECTORYLOC <electronic unit no.>

<segment no.>

<directory row> ::= DIRECTORYROW <row size>

Semantics:

1. The DIRECTORYLOC card will force a cold start; thus, this card must be omitted from a cool start deck.

2. On the B 7700, the <electronics unit no> may be empty. In this case, the system disk EU is assumed.

3. The DIRECTORYROW card will force a cold start; thus, this card must be omitted from a cool start deck.

SYSTEM INITIALIZATION

LOAD FUNCTION

Syntax:

```
<load function> ::= <load tape> | <load disk>

<load tape> ::= LOAD <file name> FROM <tape label>

<load disk> ::= LOAD <file name> DISK
```

Examples:

```
LOAD SYSTEM/MCP FROM SYSTEM
LOAD SYSTEM/MCP DISK
```

OPTION FUNCTION

Syntax:

```
<option function> ::= <setting> <option word>

<setting> ::=  SET | RESET

<option word> ::= <option> | <option pair>

<option> ::= OPEN | TERMINATE | CHECK | LPBDONLY | AUTORM |
             DIAGNOSTICS | CDONLY | AUTORECOVERY | DUPSUPERVISOR |
             DUPINTRINSICS | RECONDUMP | AUTODC | NODUMP | CPBDONLY

<option pair> ::= <option> <digit> <digit> | <option> <digit>
```

Semantics:

The 14 run-time system options are as follows:

- OPEN

  When this option is set, a file-open message is displayed for each job whenever a file is opened.

- TERMINATE

  When this option is set, abnormal job terminations result in an attempted program dump rather than a full memory dump. If this option is reset, such abnormal terminations do result in a full memory dump.

- CHECK

  When this option is set, memory dumps under both abnormal termination and FORGETCHECK conditions are inhibited. These dumps are automatic when CHECK is reset.

- LPBDONLY

  When this option is set, all printer output files are assigned to printer backup disk. These files can then be printed by the AUTOBACKUP routine.

- AUTORM

  When this option is set, the MCP automatically removes the old file when a duplicate-file condition occurs. When AUTORM is reset, an RM or OF input message is required when such a condition occurs.

- DIAGNOSTICS

  When this option is set, an RSVP message ("RF DEGRADATION", etc.) is displayed at the operator's console any time the reliability of a hardware unit is degraded by a set amount. This option is meaningless when the MCP has been compiled without setting the MTBF option.

- CDONLY

  The state of this option is ignored as pseudo readers are no longer used.

- AUTORECOVERY

  When this option is set, a Halt/Load is attempted following all system fatal memory dumps (except hung processor). DCPs which were running prior to the Halt/Load are subsequently reinitialized, and the AP number is restored to its value previous to the Halt.

- DUPSUPERVISOR

  This option is provided for use with Directory Reconstruction. If filename has been designated as the supervisor program by a CS input message and this option is set, then at Halt/Load time the MCP will attempt to execute a code file titled filename/EU#nnn, where nnn is the number of the EU from which the Halt/Load occurs. If this option is reset, the code file with the TITLE of filename will be executed after a Halt/Load regardless of the EU involved.

SYSTEM INITIALIZATION

- DUPINTRINSICS

  This option is provided for use with Directory Reconstruction. If filename has been designated as the intrinsics file by a CI input message and this option is set, then at Halt/Load time, the system will attempt to use as the intrinsics file the code file with the TITLE of filename/EU#nnn, where nnn is the number of the EU from which the Halt/Load occurs. This provides an intrinsics file for the supervisor program, if any. If this option is reset, the code file with the TITLE of filename will be used as the intrinsics file at Halt/Load time regardless of the EU involved.

- RECONDUMP

  When this option is set, memory dumps during Directory Reconstruction are allowed and made nonfatal. When the option is reset, memory dumps are inhibited during reconstruction.

- AUTODC

  When this option is set, the automatic generation of a datacom control stack is provided for upon request from an executing job.

- NODUMP

  When this option is set, the MCP is prevented from attempting dumps to tape. Potential nonfatal dumps are denoted by a message at the supervisory console and logged. The source of a fatal dump is displayed in a system message at Halt/Load time. When NODUMP is reset, dumps are taken in the normal fashion.

- CPBDONLY

  When this option is set, all card punch output files are assigned to punch backup disk. These files can then be punched by the AUTOBACKUP routine.

The appearance of an integer from 0 to 47 with an option associates that number with the option.

Options can be set and reset from the input keyboard by use of the RO and SO messages, as well as by the use of a card in the cold start or cool start deck.

Example:

```
SET TERMINATE
```

TERMINATE FUNCTION

Syntax:

```
<terminal function> ::= STOP | *
```

Semantics:

1. The STOP card causes any cards following it to be flushed and ignored until an end card is encountered. Cards with the functions needed are placed before the STOP card and the remainder after it. In this way the integrity of the deck of cards can be maintained.

2. The asterisk (*) function transfers control to another peripheral device (card reader or keyboard). When control is returned, scanning will be resumed where it left off (just beyond the *).

LIST FUNCTION (B 7700)

Syntax:

<list function> ::= LIST

Semantics:

The cards following the list card up to the "stop" card are listed. If errors occur, error messages are listed on the printer.

COLD START FUNCTION (B 7700)

Syntax:

<cold start function> ::= COLD

Semantics:

This card may be used to cause a cold start request when it is not desired to specify a directory or overlay function, but to use system default values.

SYSTEM DISK FUNCTION (B 7700)

Syntax:

system disk function ::= DISK <dlist>
dlist ::= <unit no.> | <dlist> <unit no.>

Semantics:

The system disk card is used to specify the disk electronics unit to be used as system disk. Up to six unit numbers may be specified in the <dlist>. Each EU will be tried in turn from left to right until one is found that is ready, not write-lockout, and not reserved.

If no system disk card is used, disk units are tried in order from lowest unit number to highest.

SYSTEM INITIALIZATION

PERIPHERAL CONFIGURATION FUNCTION (B 7700)

Syntax:

```
<peripheral configuration function> ::= <unit card> |
                                        <exchange card> |
                                        <DFO card> |
                                        <DCP card>

<unit card> ::= <utag> <numlist> <info>

<utag> ::= UNIT | UNITS

<numlist> ::= <num> | <num> - <num> | <num> <numlist>

<num> ::= <integer>

<info> ::= <map> | EXCH <id> <ot>

<map> ::= IOM <numlist> CH <numlist> <type>

<type> ::= PRINTER | LP | READER | CR | PUNCH | CP | DISK1 | DISK11 |
           DISKPACK | DISPLAY | SPO | PETAPE | TAPE7 | TAPE9 | PTR |
           PTP | KIND <num>

<ot> ::= <type> | <empty>

<id> ::= (identifier for an exchange)

<exchange card> ::= EXCH <id> <map> <rwalk>

<rwalk> ::= RING <numlist> | <empty>

<DFO card> ::= DFO <num> <spec>

<spec> ::= <dlist> <indlist>

<dlist> ::= <empty> | DIRECT <id> | DIRECT <id> DIRECT <id>

<indlist> ::= <empty> | INDIRECT <id> | INDIRECT <id> INDIRECT <id>

<DCP card> ::= DCP <d> IOM <d> ADDRESS <d>

<d> ::= (Integer GEQ 0 and LEQ 7)
```

Semantics:

1. Commas may appear at reasonable places. Unit numbers range from 1 thru 255. Each unit number must appear on only one card, except for <rwalk>.

2. IOM numbers range from 0 thru 7. Channel numbers (CH) range from 1 thru 20, and from 24 thru 31 (disk only). The "KIND <num> " form of TYPE must specify a known type.

3. If <ot> specifies type, the type must agree with the type on the EXCH card.

4. The <num> - <num> form of <numlist> specifies a range (32-34 is equivalent to 32, 33, 34).

5. A <numlist> is not necessarily sequential or ascending.

6. An <id> must start with a letter.

7. An <id> must be unique from other <id> s in the first 5 characters plus length ("DISK27" and "DISK29" are equivalent, whereas "DISK27" and "DISK291" are not).

8. A unit is marked as being on an exchange if one (or more) other units are specified for the same <map>.

9. When <rwalk> is empty, the units on an exchange are linked for ringwalk in ascending order, regardless of order of appearance in parameter deck.

10. A ringwalk other than ascending may be specified with <rwalk>; in this case, units are linked in the order in which they occur in the <numlist> following "RING"; all units in a <rwalk> must be on the exchange, and all units on the exchange must be in the <rwalk>.

11. For <type>, "DISKI" is IC4, IC3, etc., and DISKII is IIB-4, etc.

12. On a DFO card one of <dlist> and <indlist> must not be empty.

13. Unit number and path assignments are by no means arbitrarily selected by an operator. The cards must reflect exactly the way in which the peripherals were installed; consequently the deck ought to be built in cooperation with the site engineer.

14. When disks are to be used with a disk file optimizer, the DFO card is used:

    Example:

```
UNITS 96-105, EXCH DKA
EXCH DKA IOM 0 CH 27 DISKI
UNITS 160-169, 176-185, EXCH DKC
EXCH DKC IOM 0, 1 CH 29, DISKI
DFO 28, DIRECT DKA INDIRECT DKC
DFO 29, DIRECT DKC INDIRECT DKA
```

SYSTEM INITIALIZATION

The unit number for the DFO and connections for direct and indirect ports are again determined by installation procedures and not arbitrarily selected.

Note that in the above example DKC is a 2 x 20 disk exchange and consequently occupies both direct ports on DFO 29 and both indirect ports on DFO 28. Also, note that the <id> is used to link the DFO to a set of EU's.

15. The DCP card is used to describe how the DCP's (if any) were installed:

    Example:

    ```
    DCP 1  IOM 1  ADDRESS 1
    DCP 2  IOM 1  ADDRESS 2
    DCP 3  IOM 0  ADDRESS 1
    ```

    On this card, the DCP number is a logical number and has no relation to installation; the IOM address combination reflects installation.

EXAMPLES:

Some examples of simple peripherals are:

```
UNIT 10, IOM 1, CH 13, LP
UNIT 11, IOM 0, CH 14, PRINTER
UNIT 26, IOM 4, CH 4, CR.
```

The unit number is determined by the device's position in the status vector, and the IOM-channel combinations reflect how the devices are installed.

Exchanges may be described in several ways:

```
UNIT 32-41 IOM 0-1 CH 24,25 DISKI
UNIT 17 EXCH TP2 TAPE7
UNIT 18 EXCH TP2 TAPE7
UNIT 20-24 EXCH TP2
EXCH TP2 IOM 1 CH 2-4 TAPE7
```

Note that the <id> is used to link one or more unit cards to an EXCH card which describes the access path(s) via <map>.

An example with <rwalk>:

```
UNIT 64, EXCH DK4
UNIT 66,65, 70 EXCH DK4
UNIT 67-69,71, EXCH DK4
EXCH DK4, IOM 0-4, CH 30, DISKII RING 71-64
```

That exchange would be linked in the unit table from 71 thru 64 (if the <rwalk> was empty, they would be linked 64-71).

PARTITION FUNCTION (B 7700)

Syntax:

<partition function> ::= PART <slist>

<slist> ::= <section> | <slist> <section>

<section> ::= <cpmx> | <mcmx> | <iomx> | <dfox> | <dcpx> | <sharedx>

<cpmx> ::= CPM <numlist>

<iomx> ::= IOM <numlist>

<mcmx> ::= MCM <numlist>

<dcpx> ::= DCP <dto>

<dto> ::= NONE | ALL | <numlist>

<dfox> ::= DFO <dto>

<sharedx> ::= SHARED <slt>

<slt> ::= NONE | ALL | ODD | EVEN | <numlist>

<numlist> ::= <num> | <num> - <num> | <num> | <numlist>

<num> ::= <integer>

Semantics:

1. Partition information is not preserved across cool or cold start; in order to keep a partition in force, the partition parameter cards must remain in the parameter deck.

2. If no partition cards appear, no partition is assumed; all the units specified by peripheral cards and all main frame modules will be used by the MCP.

3. If no partition card appears for a particular <section>, all items in that <section> are included in the partition.

4. More than one partition card may appear in the deck; the various cards are "OR"ed inclusively for each <section>.

5. Partition cards may appear in any order and anywhere in the deck prior to "STOP". No cross checking between partitions is done; specifying overlapping partitions may cause catastrophic errors.

SYSTEM INITIALIZATION

6. Specifying which IOM's are in the partition will cause all non-exchange devices on IOM's not in the partition to be deleted, including DCP's. In addition, all exchange devices which are not connected to any IOM in the partition will be deleted, including DFO's. Exchange devices which are connected only to IOM's in the partition, and non-exchange devices on IOM's in the partition, are retained in the unit table; all DCP's and DFO's connected wholly within the partition are retained subject to modification by a <dfox> or <dcpx> section. All exchange devices or DFO's connected to IOM's both in and out of the partition are retained, subject to modification by a <sharedx> or <dfo's> section.

7. A list is acquired by logical DCP number in the case of DCP's, and by physical unit number in the case of DFO's and peripheral devices. If a DCP, DFO, or SHARED section does not appear, then "ALL" is assumed for that section. More than one occurrence of a section results in "OR"ing of all devices specified in all occurrences of that section. "ODD" and "EVEN" refer to odd and even numbered units, respectively.

   The list is then applied to all exchange devices (not DFO's) which are shared by (connected to) IOM's both in and out of the partition. References made to devices which are connected wholly within or wholly without the partition are ignored. Units not in the list are deleted from the unit table.

Cold Start

By use of the cold start operation a new MCP is loaded to system disk. After the MCP has been loaded and a brand new disk directory listing the MCP has been built, the system does a Halt-Load and the first 8192 words of the MCP are read into core memory. The MCP is then in control of the system, and the system software can then be placed on user disk by the entry of the following control statement at the input keyboard.

```
? COPY SYSTEM/= FROM SYSTEM; END
```

To load a new MCP from tape, use the following procedure.

a. Turn system power on.

b. Mount tape SYSTEM and make it available to the system.

c. Press and release CARD LOAD SELECT push button on operator's console.

d. Place cold start deck (loader deck followed by selected system initialization function cards) in cold-start reader, make card reader ready, and press and release START push button on card reader.

e. Press and release LOAD push button on operator's console; first card of loader deck is read.

Cool Start

By use of the cool start operation, it is possible to replace the object code for the MCP without destroying the disk directory. As with a cold start, the new MCP is loaded to system disk, the system does a Halt-Load, and the new MCP is in control of the system.

The operating procedure for doing a cool start (by loading a new MCP from tape) is the same as for doing a cold start, except that a cool start deck is used instead of a cold start deck. The only difference between a cool start deck and a cold start deck is that system initialization function cards, OLAYROW, DIRECTORYLOC, and DIRECTORYROW used in a cold start deck must be omitted from the cool start deck.

Also, if there is a version of the MCP on user disk, a cool start can be done from the input keyboard by use of the CM input keyboard message.

On the B 7700, a cool start is often done merely to build unit tables without reloading the MCP code file.

Halt-Load

The Halt-Load operation is a disk-load operation used in bringing a fresh version of the MCP from system disk to main memory. The first 8192 words of the MCP, starting at disk address 0, are read into main memory.

The effects of the Halt-Load operation are as follows:

a. All processor registers are cleared and new values are inserted in some.

b. All I/O activities cease.

c. Loading from disk is initiated.

Thus, the processing of all jobs is terminated by a Halt-Load and must be started again after the Halt-Load. Note, however, that the disk directory and, hence, all files on disk are not disturbed by a Halt-Load operation.

To perform a Halt-Load operation from the operator's console, proceed as follows.

a. Simultaneously press HALT and ENABLE push buttons; then release them.

b. Press and release LOAD push button.

DISPLAY OF PROCESSOR REGISTERS (B 6700)

DISPLAY OF PROCESSOR REGISTERS (B 6700)

To display in the A register the contents of the processor registers listed below, use the following procedure.

Name of Register	Hexadecimal Address
D0 through D31	00 through 1F
PIR	20
SIR	21
DIR	22
TIR (BUF3)	23
LOSR	24
BOSR	25
F	26
BUF	27
PBR	30
SBR	31
DBR	32
TBR (BUF2)	33
S	34
SNR	35
PDR	36
TEMP	37

a. Set SECL switch (on processor maintenance control panel) to position "up."

b. Set LOCAL-REMOTE switch (on processor maintenance control panel) to position LOCAL.

c. Press and release ADJ-0.0 push button (on processor maintenance control panel).

d. Place in the B register the hexadecimal address of the register the contents of which are to be displayed. (For example, to display the contents of PDR, place the hexadecimal value 36 in the B register.)

e. Press and release BROF push-button indicator (on processor display panel).

f. Press and release READ-IC push button (on processor maintenance control panel).

g. Read the value displayed in the A register (bits 19:20).

DISPLAY OF PROCESSOR REGISTERS (B 6700)

To display in the SM register the contents of the processor registers S, F, PBR, PIR, BOSR, and LOSR, use the following procedure. (All switches referred to in this procedure are located on the processor maintenance control panel.)

a. Set LOCAL-REMOTE switch to position LOCAL.

b. Press and release ADJ-0,0 push button.

c. Press and hold one of the following READ PROC REG push buttons (depending on the register to be displayed): S, F, PBR, PIR, BOSR, or LOSR.

d. Read the value displayed in the SM register (SM00 through SM20), located on the processor display panel.

FORCING A MEMORY DUMP (B 6700)

To force a dump of main memory, use the following procedure.

a. Enter the DP message at the input keyboard. If this attempt to force a memory dump is unsuccessful, proceed with step b below.

b. On the memory tester panel,

1. Place the hexadecimal value 14 in MEMORY WRITE register. Set tag to 1.

2. Place the hexadecimal value 33 in WORD ADDRESS register.

3. Press and release WRITE REQUEST push button.

4. Simultaneously press O'RIDE PROTECT and START push buttons. Release them.

The actions described above cause an IRW to the MEMDUMP PCW to be written in address D[0]+3, which is the address to which all processors go to answer any interrupt. Therefore, the IRW having been written into location D[0]+3, the next processor that responds to any interrupt enters MEMDUMP and begins the dump cycle.

If at this point, however, the processor or processors are apparently not answering external or internal interrupts and no memory dump occurs, an alarm interrupt must be forced into a processor. (This type of interrupt also causes entry into

FORCING A MEMORY DUMP (B 6700)

an interrupt-handling procedure by way of location D[0]+3.) The safest alarm interrupt to force into a processor is the stack-underflow interrupt. Proceed with steps c through h below to force a stack-underflow interrupt into a processor and to, again, attempt a memory dump.

c. On processor maintenance control panel, set SECL switch to position "up."

d. On general controls panel, set SYST clock control to position "up."

e. In INTERRUPT CONTROL section of processor display panel, set SUFL (stack-underflow interrupt) flip-flop and reset all other flip-flops in INTERRUPT control.

f. Set SYST clock control to position "down."

g. Set SECL switch to position "down."

h. On processor maintenance control panel, press and release START push button. If no dump occurs, proceed with steps below.

i. Set SECL switch to position "up."

j. Place LOCAL/REMOTE switch to position LOCAL.

k. Press and release ADJ 0,0 push button.

l. Press and release UNIT CLEAR push button.

m. Enter a hexadecimal 30 in the two least significant characters of the A register.

n. Set AROF and BROF.

o. Press and release WRITE-IC push button.

p. Enter the following syllables into the P register of the processor: 3 AE 40 14 B0 B0 AB (MKST; NAMC: MEMDUMP; LIT:0; LIT:0; ENTR).

q. Set PROF, IIHF, and NCSF.

r. Place the SECL and the LOCAL/REMOTE switch in the position "down."

s. Press and release START push button.

FORCING A MEMORY DUMP (B 7700)

1. Place the processor(s) in SINGLE INSTRUCT. Also place them in TEST mode to avoid a timeout. Note: At least one processor must be in control mode 0 or 1; otherwise, the dump cannot be forced by this method.

2. Purge the stack buffer on each processor, if required. (Depress and hold PBI and toggle the PLS switch).

3. Write an IRW to (0, 14) into the D [0] + 3 location in memory.

   a. Place the MCM that contains address zero in TEST and LOCAL mode.

   b. Load the IRW (1 000000 000014) into the input register (Row 7).

   c. Load a control word specifying a single word overwrite operation to address hexadecimal 33 into the control word register (Row 6). The control word should have bits 47, 45, and 0 on and should have 00033 in the address field.

   d. Ensure that the single/continuous cycle switch is in the single position.

   e. Depress START to accomplish the write operation.

   f. Place the MCM in REMOTE.

4. Take the processor that is to perform the dump out of SINGLE INSTRUCT and press the START pushbutton. If the processor is responding to external interrupts, the dump will commence. If not, proceed to step 5.

5. Place the processor in SINGLE INSTRUCT. If the processor is in control mode 1, press the START pushbutton until it returns to control mode 0. If the SNP flip-flop (Panel 2, Row 7, bit 33) is on, press the pulse pushbutton until it goes off.

6. Set the stack underflow interrupt (Panel 2, Row 14, Bit 12). Note: Dial the correct setting into the thumbwheel switches, place panel 2 toggle switch 12 up, all others down, and press LOAD. Do not press CLEAR.

7. Take the processor out of SINGLE INSTRUCT and press the START pushbutton. The dump should commence.

CLEARING MEMORY (B 6700)

To clear main memory, use the following procedure. (All switches and indicators referred to in this procedure are located on the memory tester panel.)

a. Set TEST-NONTEST switch to position TEST; red TEST ON indicator lights.

b. Set INHIBIT-HALT-ERR/CHGE switch to position INHIBIT.

c. Set WRITE-READ switch to position "center."

d. Set SINGLE-CONT switch to position "center."

e. Set MEM CLEAR switch to position "up."

f. Press and release CLEAR push button.

g. Press and release START push button.

h. Press and release CLEAR push button.

i. Set MEM CLEAR switch to position "down."

j. Set TEST-NONTEST switch to position NONTEST; TEST ON indicator goes off.

APPENDIX A
B 6700 PUBLICATIONS

SYSTEM MANUALS

Publication	Form No.
Facility Installation Manual	5000326
Field Engineering Installation Manual	5000177
B 6700 Information Processing System Reference	1058633
B 6700 System Hardware Handbook	5000276
B 6700/B 7700 System Software Handbook	5000722
System Miscellanea	5000367

SOFTWARE INFORMATION MANUALS

Publication	Form No.
BASIC Language	5000383
CANDE Operations	5000615
COBOL Filter	5000011
COBOL Reference	5000656
Command and Edit (CANDE) Language	5000318
Compatible ALGOL Filter	5000003
Data Communications Extended ALGOL (DC ALGOL)	5000052
Data Communications Functional Description	5000060
Data Management	5000235
ESPOL Language	5000094
Extended ALGOL Compiler	5000136
Extended ALGOL Language	5000128
FORTRAN Reference	5000458
Input/Output Subsystem	5000185
MAKEUSER Program	5000227
Mathematical Intrinsics	5000151
MCSII User's Guide	5000219
Network Definition Language	5000078
On-Line Maintenance and Test (MAT) Language	5000169
PL/I Language	5000201
Program Binder	5000045
Remote Job Entry (RJE) System	5000300
Sort Program	5000144
SYSTEM/DUMPANALYZER	5000334
System Status Intrinsic	5000425
Workflow Management Reference	5000706
Workflow Management User's Guide	5000714

HARDWARE FIELD ENGINEERING TECHNICAL MANUALS

Publication	Form No.
Card Reader Control	1030640
Console Display Control	1045721
Core Memory	1041613
Data Communications Processor	1041639
Disk File Control IV	1045028
Display and Maintenance Diagnostic Logic	1041621
Dynamic Reconfiguration	5000607
I/O Processor	1041647
Peripheral Controller	1045713
Power Supply	1041654
Processor	1040326
9-Track Magnetic Tape Control	1034204
65K Mass Memory	5000680
Controlled Reconfiguration	5000599

REFERENCE CARDS

Publication	Form No.
Cluster Adapter Overlay Template	5000581-001
CANDE	5000581-002
Result Descriptors	5000581-003
Data Communication Processor	1051638

CUT ALONG THIS LINE

Postage Will Be Paid by Addressee

No Postage Stamp Necessary If Mailed in the United States

BUSINESS REPLY MAIL

First Class Permit No. 381; City of Industry, Ca. 91749

Burroughs Corporation
P. O. Box #1223
City of Industry, Calif. 91749

Attn: Publications Department
Technical Information Organization

Burroughs Corporation Remarks Form
Title: B 6700 Handbook Form: 5000722

Date: _____

CHECK TYPE OF SUGGESTION

☐ Addition ☐ Deletion ☐ Revision ☐ Error

GENERAL COMMENTS AND/OR SUGGESTIONS FOR IMPROVEMENT OF PUBLICATION

From: Name _____ Date _____

Title _____

Company _____

Address _____