


Burroughs 

**B 2000/B 3000/  
B 4000 Series  
MCPVI**

SYSTEM SOFTWARE  
PROGRAMMER'S GUIDE

(RELATIVE TO ASR 6.5 RELEASE)

Copyright © 1982, Burroughs Corporation, Detroit, Michigan 48232

PRICED ITEM

“The names used in this publication are not of individuals living or otherwise. Any similarity or likeness of the names used in this publication with the names of any individual, living or otherwise, is purely coincidental and not intentional.”

Burroughs believes that the information described in this document is accurate and reliable, and much care has been taken in its preparation. However, no responsibility, financial or otherwise, is accepted for any consequences arising out of use of this information.

The information contained herein is subject to change without notice. Revisions may be issued to advise of such changes and/or additions.

This edition incorporates the following previously released PCNs for this publication:

1090685-001 (January 3, 1980)  
1090685-002 (September 25, 1980)

Correspondence regarding this publication should be forwarded using the Remarks form at the back of the manual, or may be addressed directly to TIO West Documentation, Burroughs Corporation, 1300 John Reed Court, City of Industry, California 91745, U.S.A.

## LIST OF EFFECTIVE PAGES

Page	Issue
Title	Original
ii	Original
iii	Original
iv	Blank
v thru xiii	Original
xiv	Blank
xv	Original
xvi	Blank
1-1 thru 1-2	Original
2-1 thru 2-94	Original
3-1 thru 3-21	Original
3-22	Blank
4-1 thru 4-28	Original
5-1 thru 5-69	Original
5-70	Blank
6-1 thru 6-7	Original
6-8	Blank
7-1 thru 7-10	Original
8-1 thru 8-18	Original
9-1 thru 9-34	Original
10-1 thru 10-24	Original
A-1 thru A-7	Original
A-8	Blank
B-1 thru B-2	Original
C-1 thru C-3	Original
C-4	Blank

## TABLE OF CONTENTS

Section	Title	Page
	INTRODUCTION . . . . .	xv
1	DEFINITIONS . . . . .	1-1
	DATA DESCRIPTIONS . . . . .	1-1
	CONTROL SYNTAX . . . . .	1-1
	HARDWARE CODES . . . . .	1-2
2	PROGRAM INTERFACE . . . . .	2-1
	PROGRAM BRANCH COMMUNICATE . . . . .	2-1
	ACCEPT (ACCEPT) . . . . .	2-2
	ARM (NO COBOL SYNTAX) . . . . .	2-3
	BREAKOUT CONTROL (NO COBOL OR BPL SYNTAX) . . . . .	2-6
	DATA BASE (NO COBOL OR BPL SYNTAX) . . . . .	2-7
	CLOSE (CLOSE) . . . . .	2-9
	COMPLEX WAIT (NO COBOL OR BPL SYNTAX) . . . . .	2-10
	CORE-SIZE (MEMORY) . . . . .	2-11
	DATE (JDATE) . . . . .	2-12
	DEBUG TRACE (NO COBOL OR BPL SYNTAX) . . . . .	2-13
	DIRECT I/O (NO COBOL OR BPL SYNTAX) . . . . .	2-14
	DISPLAY (DISPLAY) . . . . .	2-15
	DISPLAY-LINES (NO COBOL SYNTAX) . . . . .	2-16
	EXITROUTINE (NO SPECIFIC COBOL SYNTAX) . . . . .	2-17
	FILL FROM (FILL IN) – FILL INTO (FILL OUT) . . . . .	2-18
	FILL FROM BOTTOM (PULLQ) – FILL FROM TOP (POPQ), FILL INTO BOTTOM (PUTQ) – FILL INTO TOP (PUSHQ) . . . . .	2-19
	FILL FROM POLL (POLLQ) . . . . .	2-20
	INTERROGATE FILE (FIND) . . . . .	2-21
	READ WITH LOCK-ONLY (LOCK) . . . . .	2-23
	LOCK WITH SEEK (NO COBOL SYNTAX) . . . . .	2-24
	OPEN (OPEN) . . . . .	2-25
	OVLY (NO SPECIFIC COBOL OR BPL SYNTAX) . . . . .	2-26
	MIXTBL (NO COBOL SYNTAX) . . . . .	2-27
	QUICKTIME (NO COBOL SYNTAX) . . . . .	2-29
	READ (READ) . . . . .	2-30
	SEEK (SEEK) . . . . .	2-31
	SORT (SORT) . . . . .	2-32
	SORT RETURN (NO COBOL OR BPL SYNTAX) . . . . .	2-35
	SPACE (NO SPECIFIC COBOL SYNTAX) . . . . .	2-36
	SPOMESSAGE (NO COBOL SYNTAX) . . . . .	2-37
	STOP (STOP) . . . . .	2-38
	TIME (TIME) . . . . .	2-39
	TIME-60 (TIME60) . . . . .	2-40
	TODAYS-DATE (DATE) . . . . .	2-41
	TODAYS-NAME (NO COBOL OR BPL SYNTAX) . . . . .	2-42
	TRACE (TRACE OR DUMP) . . . . .	2-43
	UNLOCK (UNLOCK) . . . . .	2-44
	USERCHANGE (NO COBOL OR BPL SYNTAX) . . . . .	2-45
	USERCODE (NO COBOL OR BPL SYNTAX) . . . . .	2-46
	WAIT (DOZE) . . . . .	2-47
	WRITE (WRITE) . . . . .	2-48
	WRITE BREAKOUT (BREAKOUT) . . . . .	2-49
	ZIP (ZIP) . . . . .	2-50

## TABLE OF CONTENTS (Cont)

Section	Title	Page
2	ZIPSP0 (NO COBOL OR BPL SYNTAX) . . . . .	2-51
	DATA COMMUNICATIONS BRANCH COMMUNICATES . . . . .	2-52
	ACCEPT FROM (ACCEPT) . . . . .	2-54
	DISABLE [ON BREAK] (DATACOMM CANCEL) DISABLE [DISCONNECT] (DATACOMM CANCEL) . . . . .	2-55
	DISABLE ON NO-DATA (DATACOMM CONDCANCEL) . . . . .	2-56
	DISPLAY UPON (DATACOMM DISPLAY) . . . . .	2-57
	ENABLE (DATACOMM ENABLE) . . . . .	2-58
	ENABLE EXTENDED (DATACOMM ENABLE EXTENDED; NO COBOL SYNTAX) . . . . .	2-59
	FILL (DATACOMM FILL) . . . . .	2-60
	FILL EXTENDED (DATACOMM FILL EXTENDED; NO COBOL SYNTAX) . . . . .	2-61
	INTERROGATE (DATACOMM INTERROGATE) . . . . .	2-62
	INTERROGATE-END-TEXT (DATACOMM INTERROGATE ADDRESS) . . . . .	2-63
	READ (DATACOMM READ) . . . . .	2-64
	READ EXTENDED (DATACOMM READ EXTENDED; NO COBOL SYNTAX) . . . . .	2-65
	READY BUFFER (DATACOMM READY BUFFER; NO COBOL SYNTAX) . . . . .	2-66
	TRANSLATE-TABLE (DATACOMM TRANSTBL; NO COBOL SYNTAX) . . . . .	2-67
	WAIT [UNTIL] (DATACOMM WAIT) . . . . .	2-68
	WRITE (DATACOMM WRITE) . . . . .	2-69
	WRITE EXTENDED (DATACOMM WRITE EXTENDED; NO COBOL SYNTAX) . . . . .	2-70
	WRITE-READ (DATACOMM WRITEREAD) . . . . .	2-71
	WRITEREAD EXTENDED (DATACOMM WRITEREAD EXTENDED; NO COBOL SYNTAX) . . . . .	2-72
	WRITEREADTRANS (DATACOMM WRITEREADTRANS) . . . . .	2-73
	WRITEREADTRANS EXTENDED (DATACOMM WRITEREADTRANS EXTENDED; NO COBOL SYNTAX) . . . . .	2-74
	WRITETRANSREAD (DATACOMM WRITETRANSREAD) . . . . .	2-75
	WRITETRANSREAD EXTENDED (DATACOMM WRITETRANSREAD EXTENDED; NO COBOL SYNTAX) . . . . .	2-76
	MICR BRANCH COMMUNICATES . . . . .	2-77
	CONTROL 4 (ACTION 4) . . . . .	2-78
	CONTROL 6 (ACTION 6) . . . . .	2-79
	READ (READ) . . . . .	2-80
	READ FLOW (READ FLOW) . . . . .	2-81
	SELECT (ACTION 0) . . . . .	2-82
	OTHER USER PROGRAM INTERFACES . . . . .	2-83
	USE ROUTINES . . . . .	2-83
	FILE HANDLING . . . . .	2-83
	MCP-UTILITY INTERFACES . . . . .	2-84
	Print/Punch Backup . . . . .	2-84
	Log Programs . . . . .	2-84
	Disk File LOAD/DUMP . . . . .	2-84
	Disk Pack File LOAD/DUMP . . . . .	2-88
	Automatic System Recovery Program . . . . .	2-90
	Sort Intrinsic . . . . .	2-93

## TABLE OF CONTENTS (Cont)

Section	Title	Page
2	DMPALL . . . . .	2-94
	MCP-COMPILER INTERFACES . . . . .	2-94
3	INTERPROGRAM COMMUNICATION . . . . .	3-1
	BASIC CONCEPTS . . . . .	3-1
	INTERPROCESS CONTROL TECHNIQUES . . . . .	3-2
	ASYNCHRONOUS PROCESSES . . . . .	3-5
	COMPARISON OF MECHANISMS . . . . .	3-7
	EXAMPLES OF CRCR AND STOQUE PROCESSING . . . . .	3-8
	INTERPROCESS CONTROL MECHANISMS . . . . .	3-13
	CORE-TO-CORE TRANSFER . . . . .	3-13
	Data Length and Types . . . . .	3-13
	Program Names . . . . .	3-14
	Program Synchronization . . . . .	3-14
	STORAGE QUEUE . . . . .	3-15
	STOQUE Parameter Block . . . . .	3-15
	Buffer . . . . .	3-16
	Queue Names . . . . .	3-16
	Queue Organization . . . . .	3-16
	Storage Queue Entries . . . . .	3-17
	Available Space List . . . . .	3-18
	LANGUAGE CONSTRUCTS FOR CRCR AND STOQUE . . . . .	3-18
	CORE-TO-CORE CONSTRUCTS . . . . .	3-18
	STOQUE CONSTRUCTS . . . . .	3-20
4	DEVICE ALTERNATES . . . . .	4-1
	GENERAL CONCEPTS . . . . .	4-1
	CARD READER ALTERNATES . . . . .	4-2
	BASIC MECHANISM OF CARD READER ALTERNATES . . . . .	4-2
	LDCNTL (LOAD CONTROL) UTILITY . . . . .	4-4
	PSEUDO-READERS . . . . .	4-6
	User Creation of Pseudo-Decks . . . . .	4-7
	Pseudo-Reader Processing . . . . .	4-7
	Pseudo-Reader Activation . . . . .	4-9
	Control Card Processing . . . . .	4-10
	Programmatic File Access . . . . .	4-10
	Reader Deallocation . . . . .	4-10
	Removing Pseudo-Readers . . . . .	4-11
	Recovery of Pseudo-Decks . . . . .	4-11
	Operator Interface To Pseudo-Readers . . . . .	4-11
	Stacked Pseudo-Decks . . . . .	4-13
	CREATION OF PSEUDO-DECKS FROM BACKUP PUNCH FILES . . . . .	4-13
	SUGGESTIONS AND SPECIAL METHODS . . . . .	4-14
	Deferral of Decks . . . . .	4-14
	Cautions and Capacities . . . . .	4-15
	Control Card Linkage . . . . .	4-15
	When Not to Use Pseudo-Readers . . . . .	4-15
	PRINTER ALTERNATES . . . . .	4-16
	ESTABLISHING A PRINTER MEDIUM . . . . .	4-17
	LABEL PROCESSING . . . . .	4-19
	CLOSING PRINT FILES . . . . .	4-20
	REOPENING PRINT FILES . . . . .	4-20

## TABLE OF CONTENTS (Cont)

Section	Title	Page
4	BACKUP PRINT FILES . . . . .	4-21
	BACKUP TAPE FILES . . . . .	4-21
	BACKUP DISK FILE . . . . .	4-22
	Printer Control Data . . . . .	4-23
	ACCESS AND CONTROL OF PRINTER BACKUP FILES . . . . .	4-26
	Accessing Printer Backup Files Programmatically . . . . .	4-26
	Operator Interface to Printer Backup Files . . . . .	4-26
	PRINTER BACKUP UTILITIES . . . . .	4-28
	Backup Tape (PBTOUT) . . . . .	4-28
	Backup Disk/Disk Pack (PBDOUT) . . . . .	4-29
	USER SUPPLIED PRINT UTILITIES . . . . .	4-31
	General Considerations . . . . .	4-31
	General File Handling . . . . .	4-31
	User-Coded PBTOUT . . . . .	4-31
	User-Coded PBDOUT . . . . .	4-33
	SUGGESTIONS AND SPECIAL METHODS . . . . .	4-36
	When Not To Use Printer/Punch Alternates . . . . .	4-37
	Special Source Program Methods . . . . .	4-37
	User-Coded Utilities . . . . .	4-37
	Other Uses . . . . .	4-38
	Special Uses of Printer and Punch Alternates . . . . .	4-38
	CARD PUNCH ALTERNATE . . . . .	4-39
	ESTABLISHING A PUNCH MEDIUM . . . . .	4-40
	File Open Time Control . . . . .	4-40
	LABEL PROCESSING . . . . .	4-41
	CLOSING PUNCH FILES . . . . .	4-42
	PUNCH BACKUP FILES . . . . .	4-42
	ACCESS AND CONTROL OF PUNCH BACKUP FILES . . . . .	4-44
	Accessing Punch Backup File Programmatically . . . . .	4-44
	Removal of Punch Backup Files . . . . .	4-44
	Operator Interface to Punch Backup Files . . . . .	4-44
	PUNCH BACKUP UTILITY (PCHOUT) . . . . .	4-45
	USER SUPPLIED PUNCH UTILITY . . . . .	4-46
	General Considerations . . . . .	4-46
	User-Coded PCHOUT . . . . .	4-47
5	MCPVI TABLES . . . . .	5-1
	DISK FILE HEADERS . . . . .	5-2
	DISK FILE HEADER (ON DISK) . . . . .	5-3
	DISK PACK FILE HEADER (ON DISK PACK) . . . . .	5-5
	FILE HEADER IN MEMORY . . . . .	5-8
	FILE INFORMATION BLOCK (FIB) . . . . .	5-10
	FILE INFORMATION BLOCK LAYOUT . . . . .	5-10
	FIB REDEFINITIONS . . . . .	5-28
	FILE BUFFER DESCRIPTORS . . . . .	5-33
	BUFFER STATUS BLOCK LAYOUT . . . . .	5-33
	INPUT/OUTPUT ASSIGNMENT TABLE . . . . .	5-37
	IOAT LAYOUT . . . . .	5-37
	JOB REFERENCE TABLE . . . . .	5-47
	JRT LAYOUT . . . . .	5-47
	LABELS . . . . .	5-51

## TABLE OF CONTENTS (Cont)

Section	Title	Page
5	EXTERNAL LABEL FORMATS . . . . .	5-51
	Burroughs Standard Label . . . . .	5-51
	USASI Standard Label . . . . .	5-52
	Installation Labels . . . . .	5-55
	Unlabeled Files . . . . .	5-56
	Unreadable Labels . . . . .	5-56
	Scratch Tape Files . . . . .	5-56
	PROGRAM LABEL DEFINITIONS . . . . .	5-57
	Unlabeled Files . . . . .	5-57
	Standard Labels . . . . .	5-58
	USASI Labels . . . . .	5-58
	Installation Labels . . . . .	5-59
	Special Cases . . . . .	5-60
	MIX TABLE . . . . .	5-61
	MIX TABLE LAYOUT . . . . .	5-61
	SECURITY ATTRIBUTES STORAGE AREA . . . . .	5-68
	SECURITY ATTRIBUTES STORAGE AREA LAYOUT . . . . .	5-68
	COMPLEX WAIT TABLE . . . . .	5-69
6	OBJECT PROGRAMS ON DISK . . . . .	6-1
	PROGRAM PARAMETER BLOCK . . . . .	6-1
	PROGRAM PARAMETER BLOCK LAYOUT . . . . .	6-1
	PROGRAM LOADING . . . . .	6-3
	MISCELLANEOUS . . . . .	6-4
	FILE PARAMETER BLOCK . . . . .	6-4
	PROGRAM CODE . . . . .	6-5
	PROGRAM SEGMENT DICTIONARY . . . . .	6-6
	PROGRAM OVERLAY MECHANISM . . . . .	6-7
7	HOW TO READ A DUMP . . . . .	7-1
	OBTAINING A PROGRAM DUMP . . . . .	7-1
	READING THE DUMP . . . . .	7-1
	STACK OPERATION . . . . .	7-4
	MEMORY DUMP FILE STRUCTURE . . . . .	7-9
	FILE NAMING . . . . .	7-9
	FILE LAYOUT . . . . .	7-9
	CONTROL RECORD FORMAT . . . . .	7-10
8	RELATIVE AND INDEXED I/O . . . . .	8-1
	RELATIVE I/O OVERVIEW . . . . .	8-1
	INDEXED I/O OVERVIEW . . . . .	8-1
	RELATIVE FILE IMPLEMENTATION CONSIDERATIONS . . . . .	8-2
	RELATIVE FILE FORMATS . . . . .	8-3
	RELATIVE FILE FIB . . . . .	8-4
	RELATIVE FILE DATA BLOCK (RF BLOCK) . . . . .	8-4
	RELATIVE FILE ACCESS ROUTINE INTERFACE . . . . .	8-6
	CLOSE . . . . .	8-6
	DELETE . . . . .	8-6
	OPEN . . . . .	8-7
	READ . . . . .	8-7
	REWRITE . . . . .	8-7
	START . . . . .	8-8
	WRITE . . . . .	8-8



## TABLE OF CONTENTS (Cont)

Section	Title	Page
8	INDEXED FILE IMPLEMENTATION CONSIDERATIONS . . . . .	8-8
	INDEXED FILE FORMATS . . . . .	8-10
	INDEXED DATA FILE FORMAT . . . . .	8-10
	INDEXED KEY FILE FORMAT . . . . .	8-11
	INDEXED DATA FILE FIB . . . . .	8-12
	INDEXED KEY FILE FIBS . . . . .	8-13
	INDEXED FILE DATA BLOCK (IF BLOCK) . . . . .	8-14
	INDEXED FILE BUFFERS . . . . .	8-15
	INDEXED FILES ACCESS ROUTINE INTERFACE . . . . .	8-16
	CLOSE . . . . .	8-16
	DELETE . . . . .	8-16
	OPEN . . . . .	8-16
	READ . . . . .	8-17
	REWRITE . . . . .	8-17
	START . . . . .	8-17
	WRITE . . . . .	8-18
9	READER-SORTER DLP INTERFACE . . . . .	9-1
	DLP OVERVIEW . . . . .	9-1
	WRITE-FLIP-READ OPERATION . . . . .	9-1
	CONTROL STATE USER ROUTINE . . . . .	9-2
	NORMAL STATE ROUTINE . . . . .	9-4
	Normal State Processing Routine . . . . .	9-4
	Flow Stop Routine . . . . .	9-4
	SOFT TANK . . . . .	9-4
	FLOW AND DEMAND MODES OF PROCESSING . . . . .	9-5
	MEMORY MAP . . . . .	9-5
	MCP READER-SORTER EXTENSIONS . . . . .	9-11
	OPEN BCT . . . . .	9-11
	CLOSE BCT . . . . .	9-12
	MCP MICR MODULE . . . . .	9-12
	Reader-Sorter BCTs . . . . .	9-12
	FLOW MODE PROCESSING SUMMARY AND USER PROGRAM	
	OUTLINE . . . . .	9-19
	DOCUMENT FLOW DESCRIPTION . . . . .	9-19
	NOTES ON THE RESULT STATUS . . . . .	9-22
	NOTES ON THE SOFT RESULT DESCRIPTOR . . . . .	9-25
	I/O INVALID TO THE DLP (BIT 1) . . . . .	9-27
	BCT INVALID TO THE MCP (BIT 2) . . . . .	9-27
	FLOW CONDITION ERROR (BIT 3) . . . . .	9-27
	SYSTEM INTERFACE PARITY ERROR (BIT 4) . . . . .	9-27
	MICROFILM OPERATION NOT COMPLETED (BIT 5) . . . . .	9-28
	NON-PRESENT OPTION (BIT 6) . . . . .	9-28
	POCKET SELECT ERROR (BIT 8) . . . . .	9-28
	BAD INTERFACE INFORMATION (BIT 9) . . . . .	9-28
	TIMEOUT (BIT 10) . . . . .	9-28
	CAMERA NOT READY (BIT 11) . . . . .	9-28
	PARITY ERROR (R-S → DLP) (BIT 12) . . . . .	9-28
	READER-SORTER POWER FAILURE (BIT 13) . . . . .	9-29
	MEMORY OVERFLOW (BIT 14) . . . . .	9-29
	DLP ERROR (BIT 15) . . . . .	9-29

## TABLE OF CONTENTS (Cont)

Section	Title	Page
9	NON-IMPACT ENDORSEMENT . . . . .	9-29
	MICROFILMING . . . . .	9-30
	MISCELLANEOUS . . . . .	9-32
	B 9138 MERGE ON TEXT OPTION . . . . .	9-32
	DLP DELIMITER CHARACTER SET . . . . .	9-33
	ASSEMBLER CONSTRUCTS . . . . .	9-33
	USER Statement . . . . .	9-33
	FILE Statement . . . . .	9-34
	UNIT CARD . . . . .	9-34
10	4A CONTROL APPLICATION PROGRAM INTERFACE . . . . .	10-1
	INTRODUCTION . . . . .	10-1
	COLDSTART/WARMSTART UNIT CARD . . . . .	10-1
	USER FILE STATEMENT . . . . .	10-1
	MEMORY MAP . . . . .	10-1
	MCP INTERFACE (MICR 4A CONTROL BCT) . . . . .	10-10
	STANDARD NON-POCKET SELECT/READ BCT VERIFICATION . . . . .	10-10
	STANDARD POCKET SELECT/READ BCT VERIFICATION . . . . .	10-11
	OPEN BCT . . . . .	10-11
	CLOSE BCT . . . . .	10-11
	START FLOW BCT . . . . .	10-11
	DEMAND FEED AND READ BCT . . . . .	10-12
	POCKET LIGHT/GENERATE IMAGE COUNT MARKS BCT . . . . .	10-13
	MICROFILM SLEW BCT . . . . .	10-13
	STATUS BCT . . . . .	10-14
	CHARACTERISTICS BCT . . . . .	10-14
	LOGICAL READ BCT . . . . .	10-15
	POCKET SELECT/READ BCT . . . . .	10-16
	B 9138 MERGE ON TEXT OPTION . . . . .	10-19
	CONTROL DELIMITERS . . . . .	10-20
	NON-IMPACT ENDORSEMENT . . . . .	10-20
	NOTES ON RESULT STATUS . . . . .	10-21
	MICROFILMING . . . . .	10-23
A	EXAMPLE PROGRAM AND MEMORY DUMP . . . . .	A-1
B	BRANCH COMMUNICATE CHART . . . . .	B-1
C	PROGRAM CONSIDERATIONS FOR SHARED FILES . . . . .	C-1
	SHARED FILE OPERATIONS . . . . .	C-2
	Additional Shared File Programming Considerations . . . . .	C-1

## LIST OF ILLUSTRATIONS

Figure	Title	Page
3-1	Storage Queue Organization . . . . .	3-17
4-1	Example Pseudo-Deck Creation Program . . . . .	4-8
5-1	Examples of Tapes Created with Burroughs Standard Label Format . . . . .	5-53
7-1	A Stack Containing No Entries . . . . .	7-4
7-2	A Stack Containing One Entry . . . . .	7-5
7-3	A Stack Containing Two Entries . . . . .	7-5

## LIST OF ILLUSTRATIONS (Cont)

Figure	Title	Page
7-4	Typical Stack Entry . . . . .	7-5
7-5	Stack After NTR A . . . . .	7-7
7-6	Stack After NTR B . . . . .	7-7
7-7	Stack After EXT, Routine B . . . . .	7-8
9-1	Processor/R-S DLP Communications Paths . . . . .	9-2
9-2	Memory Map Functions . . . . .	9-5
9-3	Document Flow in Control State User and Normal State Routines . . . . .	9-21
9-4	Single Jet Endorser (Usually the Front of the Document) . . . . .	9-29
9-5	Three-Jet Endorser (Usually the Back of the Document) . . . . .	9-30
10-1	Memory Map Functions . . . . .	10-2
A-1	Program Used to Produce a Dump . . . . .	A-1
A-2	Dump Produced From Program in Figure A-1 . . . . .	A-2
C-1	Recommended LOCK Sequences - Two Examples . . . . .	C-2

## LIST OF TABLES

Table	Title	Page
1-1	Hardware Codes . . . . .	1-2
2-1	Pre-terminate Error Codes . . . . .	2-4
2-2	Data Communications Descriptor Variants . . . . .	2-52
2-3	Data Communications BCT Quick Reference . . . . .	2-53
2-4	Disk File LOAD/DUMP Parameters . . . . .	2-84
2-5	LOADMP Pass File . . . . .	2-85
2-6	LOADMP Type 1 Record . . . . .	2-86
2-7	LOADMP Type 21 Record . . . . .	2-86
2-8	LOADMP Type 22 Record . . . . .	2-86
2-9	LOADMP Data Block . . . . .	2-87
2-10	PACKUP Parameters . . . . .	2-88
2-11	PACKUP Pass File . . . . .	2-89
2-12	PACKUP Type 1 Record . . . . .	2-89
2-13	Control Block 1 . . . . .	2-90
2-14	Control Block 2 . . . . .	2-90
2-15	Data Block . . . . .	2-90
2-16	Sort Parameter File . . . . .	2-94
4-1	Analogies Between Physical and Pseudo Components . . . . .	4-3
4-2	Pseudo-Reader Card Format . . . . .	4-6
4-3	Printer Backup Combinations . . . . .	4-19
4-4	Backup Print File Label Format . . . . .	4-22
4-5	Punch Backup Options . . . . .	4-41
4-6	Backup Print File Label Format . . . . .	4-43
5-1	Result Descriptor Codes . . . . .	5-36
5-2	I/O Descriptor Errors . . . . .	5-36
5-3	Burroughs Standard Label Format . . . . .	5-51
5-4	USASI Label Record 1 . . . . .	5-52
5-5	USASI File Header Label Record . . . . .	5-54
5-6	Installation Label Card Format . . . . .	5-55
5-7	Basic Label Area . . . . .	5-57

**LIST OF TABLES (Cont)**

<b>Table</b>	<b>Title</b>	<b>Page</b>
5-8	Standard Label Format . . . . .	5-58
5-9	USASI Label in Program Area . . . . .	5-58
8-1	RF Block Format . . . . .	8-5
8-2	Indexed File Data Block Format . . . . .	8-14
9-1	Memory Map Description . . . . .	9-6
9-2	Reader-Sorter BCT Types and Operations . . . . .	9-12
9-3	Result Status Exception Conditions for Write-Flip-Read . . . . .	9-22
9-4	Reporting Result Status Exception Conditions . . . . .	9-22
9-5	Soft Result Descriptor Error Locations . . . . .	9-26
9-6	Bit Positions for Soft Result Descriptor Exception Conditions . . . . .	9-26
10-1	Memory Map . . . . .	10-2
10-2	BCT Type Numbers and Operations . . . . .	10-10
10-3	Reporting Result Status Bits . . . . .	10-22
B-1	Branch Communicates (BCTs) for MCPVI . . . . .	B-1

## INTRODUCTION

This manual describes the various programmatic interface areas available with MCPVI. The intent of the information is not to detail the operations of the MCP, but to consolidate the information which a programmer might need, for example, in reading a dump, or in writing a user-coded utility program.

The user of this manual presumably has some programming knowledge and operating experience with B 4000/B 3000/B 2000 series MCP. Because of the constructs used in many explanations, a familiarity with Medium Systems Assembler will be useful.

The reader is also referred to the MCPVI Software Operations Guide, Volume 1 form number 1127529 for specific syntax and semantics of keyboard and control messages used in this manual.

The information contained in this manual replaces the information contained in the following Medium Systems Technical Newsletters: form numbers 1042272-010, -013, -024, -025, -027, and -028. The information contained in these technical newsletters still applies to MCPV.

## SECTION 1 DEFINITIONS

Throughout this manual, abbreviations and various notations will be used in the explanations and descriptions of the MCP tables and other information.

This section defines the notations that will be used in this manual.

### DATA DESCRIPTIONS

The data typing conventions are the same as those used in the Medium Systems Assembler. Alphanumeric data is indicated by UA (eight-bit form); unsigned numeric is indicated by UN (four-bit form); signed numeric by SN (four-bit form). Indirect addressing is indicated by the IA specification. SA means signed data in eight-bit form.

Data fields are located by name, position, and length. The notation used is:

name (position, length)

For example, FIBLRA (80,4) denotes that the field called FIBLRA is located at zero-relative position (digit) 80, with a length of four digits. The naming convention used indicates that the first two or three characters of the field name show what record the field belongs to. For example, FIBLRA is a field in the File Information Block (FIB); the prefix DF or DFH refers to information in the Disk File Header (DFH) and so on.

All positions are zero-relative and all lengths are in digits, unless otherwise noted.

To indicate a bit position within a digit, a colon (:) is used. For example, 4:1 means relative digit four, the one-bit. If the digit location is implied from a previous description, only :n will be shown.

The notation :n/ means that the n-bit of the digit is zero (off).

Bits are logically ANDed and ORed with \* and + respectively.

A value enclosed in @, such as @FOC1@, refers to a value equal to the hexadecimal digits indicated. Only the hexadecimal values 0-F (0-15) can be used.

### CONTROL SYNTAX

When the discussion pertains to control message syntax, the following notation is employed:

#### Capitals

Words shown in capital letters must be used as shown (example: SAVE).

#### Lower case

Words in lower case letters and enclosed in < > specify that a substitution of the indicated type should be made (example: <integer> indicates that an integer is to be used in that portion of a message).

#### [ ]

Words shown between brackets are optional within a message (example: [SAVE] or [<integer>]).

#### { }

Braces around two or more words indicate that a choice is to be made among words.

## HARDWARE CODES

The terms hardware code or hardware type refer to the encoding of the hardware device name to a numeric value. These values are used in such records as the File Information Block and the Device Assignment Table.

Table 1-1 contains the hardware codes and their mnemonic names.

**Table 1-1. Hardware Codes**

<b>Code</b>	<b>Mnemonic</b>	<b>Meaning</b>
01	CRD	Card Reader
02	PRN	Printer
03	PCH	Card Punch
04	MTP	Magnetic Tape
05	SPO	Teletype SPO
06	DSK	HPT or 100-Byte Mode Disk
08	SOR	Sorter/Reader
09	PTR	Paper Tape Reader
10	PTP	Paper Tape Punch
11	DPK	Disk Pack
13	TYP	Teletypewriter
14	OLB	On-Line Banking Unit
15	A3B	Model 28 Teletype
16	TWX	TWX
17	T50	IBM 1050
18	D20	UNIVAC DCT2000
19	T70	IBM 1070
20	B47, B35, B25	B 4700/B 3500/B 2500
21	AA1	AA1
22	VDD	Visual Display Unit
23	RJE	Remote Job Entry Terminal
24	T30	IBM 1030
25	BTT	Burroughs Touch Calling Unit
26	BDD	Burroughs Digital Display Unit
27	F13	Friden 7311
28	OCS	B 9348-X OCS
29	TC7	TC 700
30	TC5	TC 500
31	B05	B 500
39	DCP	B 774/B 874
40	PBD	Printer Backup Disk
41	PCD	Punch Backup Disk
42	PBT	Printer Backup Tape
43	PBTB	Blocked Printer Backup Tape
44	PBP	Printer Backup Disk Pack
45	PCP	Punch Backup Disk Pack
50	PCR	Pseudo Card Reader, Disk
51	PCRP	Pseudo Card Reader, Disk Pack

## SECTION 2

### PROGRAM INTERFACE

A user program communicates with the MCP using a Branch Communicate instruction (BCT, OP = 30). With a BCT, a program requests the MCP to perform one of a variety of functions such as file OPEN, sort calls, or physical I/O. This section describes the user program Branch Communicates.

Also included are the interface requirements for various system utilities that can be user-written.

#### PROGRAM BRANCH COMMUNICATE

The BCT instruction is used to transfer control from a user program to the MCP. The address specified in the BCT is an absolute memory address within the MCP. A BCT address is valid if the first digit at that address is an undigit @F@. If the first digit is not @F@ a processor interrupt occurs. The five digits following the @F@ is an address in the MCP which will handle that particular BCT. The remaining five digits of the address must also be valid.

Each Communicate is headed by the equivalent COBOLV/ COBOLL construct (where such exists). Following in parentheses, is the BPL term for the communicate.

The Communicate structure is described as follows:

```

BCT  Address
BUN
P1
  •
  •
  •
Pn
```

The BUN does not imply that the instruction must be an unconditional branch, but rather, a branch format (8-digit) instruction. However, an unconditional branch is generated by the compilers for all communicates where BUN is indicated.

All parameters are labelled P1 through Pn, including fillers. ACONs in the parameters must not have any address controller, extended address, or indexing specifications, and cannot exceed the program limit register. Addresses of File Information Blocks (FIBs) cannot exceed 399998. ( In some cases, ACONs in programs compiled prior to MCPV, contain UA address controller specifications. Earlier MCPs ignored the high-order digit in certain communicates. As this digit is significant for MCPVI, if the 6-digit address is found to exceed the limit of the program, the MCP ignores the 2-bit of the high-order digit.) In some cases ACON parameters are optional, signified by a zero value in the field. End-of-file (EOF) labels are an example. (However, in the case of EOF addresses, one must be present where needed, namely when EOF is reached.)



## ACCEPT

### ACCEPT (ACCEPT)

The ACCEPT BCT requests data from the Operator Control Station (OCS).

The format of this BCT is:

	BCT	0254
	BUN	
P1.	ACON	(input area, must be mod 2)
P2.	CNST 1	UN = 0 (flag indicating ACCEPT)
P3.	CNST 2	UN = XX (size of input area in bytes)
P4.	CNST 1	UN = 0 Filler

P1 must be MOD 2. P1 cannot exceed 60 bytes. If P3 is equal to zero or greater than 60, 60 is used.

The AX OCS command may be entered and saved prior to a program executing the ACCEPT BCT. If input text from a previously entered AX command is available, the text is inserted into the input area and the program is reinstated immediately. If the size of the text exceeds P3, the text is truncated and no error message is displayed.

If no previous text is available, the program is suspended until an AX command is entered. If the number of characters entered exceeds P3, the operator is notified with an error message on the OCS and ACCEPT is repeated.

In either case, if fewer than P3 characters are input, the data is left-justified in the buffer. An ETX (03) follows the last character input. The buffer contents following the ETX are undefined. If exactly P3 characters are input, no characters are inserted.

## ARM (NO COBOL SYNTAX)

The ARM BCT is used to enable a program to handle processor or program exceptions which may be experienced during the course of execution. The format of this BCT is:

	BCT	0214	
	BUN		
P1.	CNST 6	UN	(filler)
P2.	CNST 1	UN	= 5
P3.	CNST 1	UN	= 0 (complement current setting)
			= 1 (disable)
			= 2 (enable)

When a program is ARMed (ARM enabled), a program error causes the MCP to store the base relative value of the program address register in  $\text{BASE} + 64$ . The program is then reinstated at the address indicated by  $\text{BASE} + 94$ . If this field does not contain a valid address, the program will be terminated.

If a program is not ARMed, a program or processor error causes the MCP to terminate the job. Therefore, an ARMed program can recover from hardware exceptions and program errors.

The three categories of errors from which a program can recover are processor errors, breakout/restart, and MCP-detected program errors.

Each of these cases causes a result descriptor to be stored in  $\text{BASE} + 80$ . For processor errors, one of the following is stored:

- C800 indicates an invalid instruction.
- C400 indicates a memory parity error.
- C200 indicates an address error.
- C100 indicates an instruction timeout.
- E800 indicates an undigit arithmetic operation.

The breakout/restart indications are as follows:

- C010 indicates breakout occurred.
- C020 indicates restart occurred.
- C030 indicates operator breakout attempted.

MCP generated pre-terminate results (program errors) are returned to the program as 9XX0; the XX is an error number from table 2-1.

A program is no longer ARMed if the ARM branch has been taken. The program must reARM to handle a subsequent error.

In the following list, <F-N> means file name, <P-N> means program name, and <ADR> means program base relative address.

ARM

**Table 2-1. Pre-terminate Error Codes**

<b>Code</b>	<b>Meaning</b>
01	INV CLOSE <ADR> INV FIB ADCR
02	INV CLOSE <F-N> <ADR> FILE NOT OPEN
03	INV CLOSE <F-N> <ADR> INV USER DISK HEADER
04	INV CLOSE <F-N> <ADR> SMASHED FIB
05	INV CLOSE <F-N> <ADR> INV FILE ID
06	INV CLOSE <F-N> <ADR> INV LABEL USE ADDR
07	INV CLOSE <F-N> <ADR> FILE SIZE EXCEEDED
08	INV CRCR <ADR> INV SIZE
09	INV DCOM I/O <F-N> <ADR> FILE NOT OPEN
10	INV DCOM I/O <F-N> <ADR> INV PINGPONG ADDR
11	INV DCOM I/O <F-N> <ADR> INV PHONE NBR
14	EOF NO LABEL <F-N> <ADR>
15	I/O ERR <F-N> <ADR> INV RSLT DESC
17	INV I/O DESC <F-N> <ADR>
18	INV I/O LIMIT <F-N> <ADR>
19	MEM PAR ADDR <ADR>
20	PAR NO LABEL <F-N> <ADR>
21	PROG OVLY READ ERR
22	<F-N> INV OPR MTP REWIND
23	INV READ <F-N> <ADR> FILE RSTRCTD OR NOT OPEN
24	INV OPEN <ADR> INV FIB ADDR
25	INV OPEN <F-N> INV DFAC CODE
26	INV SEQ OPEN <F-N> <ADR> RANDOM REQD
27	INV OPEN <F-N> <ADR> FILE NOT CLOSED
28	INV OPEN <F-N> <ADR> INV DIAG CC/U
29	INV OPEN EXTEND <F-N> <ADR> NO WRITE RING
30	INV OPEN <F-N> <ADR> INV FILE ID
31	INV OPEN <F-N> <ADR> INV HDW / I/O
32	INV OPEN <F-N> <ADR> INV LOCK USE
33	INV OPEN <F-N> <ADR> INV RECD SIZE
34	INV OPEN <F-N> <ADR> INV RQSTR
35	INV OPEN <F-N> <ADR> INV REVRS
36	INV OPEN <F-N> <ADR> INV SEQ O/I
37	INV OPEN <F-N> <ADR> INV VAR RECD SIZE
38	INV OPEN <F-N> <ADR> NO I LABEL
39	INV OPEN <F-N> <ADR> OVRSIZ DISK AREA
40	INV OPEN <F-N> <ADR> OVRSIZ EOF
41	INV OPEN <F-N> <ADR> SMASHED FIB
42	INV OPEN <F-N> <ADR> INV BCT PARAM
43	INV OPEN <F-N> <ADR> FILE NOT RELEASED BY CLOSE
44	INV OPEN <F-N> <ADR> INV LABEL USE ADDR
45	INV OPEN <F-N> <ADR> INV SHRD RECD SIZE / BLOCK
47	ADDR ERR <ADR>
48	INSTR TIME OUT <ADR>
49	INV BCT ADDR
50	INV INSTR <ADR>
51	MEM PAR ADDR <ADR>
52	PROGRAM SNAPPED
53	INV READ <ADR> INV FIB ADDR

**Table 2-1. Pre-terminate Error Codes (Cont)**

<b>Code</b>	<b>Meaning</b>
54	INV READ <F-N> <ADR> FILE RSTRCTD OR NOT OPEN
55	INV READ <F-N> <ADR> INV BUFFR RSLT DESC
56	INV READ <F-N> <ADR> INV DISK ADDR
57	INV READ <F-N> <ADR> INV I/O OR FILE TYPE
58	INV READ <F-N> <ADR> INV RECD SIZE
59	INV READ <F-N> <ADR> SMASHED FIB
60	INV READ <F-N> <ADR> SMASHED KEY
61	INV POSITION <F-N> <ADR> POSITION OUT OF FILE
62	INV READ <F-N> <ADR> FILE NOT SHRD
63	INV WRITE OR UNLOCK <F-N> <ADR>
64	STATEMATE NO LABEL <F-N> <ADRS>
66	INV ARM ADDR <ADR>
67	INV BCT PARAM
68	OVER TIME <ADR>
70	INV STOQ <ADR> INV BLOCK ADDR
71	INV STOQ <ADR> INV ENT SIZE
72	INV STOQ <ADR> INV NAME
73	INV STOQ <ADR> INV NAME SIZE
75	INV MTP LABEL <F-N> <ADR>
76	MTP LABEL WRITE ERR <F-N> <ADR>
77	INV OPEN EXTEND <F-N> <ADR> INV LABEL
78	INV USE RET
79	INV CLOSE <F-N> <ADR> PACK <P.SER.NBR> NOT AVAIL
80	INV CLOSE <F-N> <ADR> INV USER PACK HEADER
81	INV OPEN <F-N> <ADR> OVRSIZE PACK AREA
82	INV READ <F-N> <ADR> INV PACK ADDR
83	LOST RJE HANDLER (JOB ABORTED)
84	INV BCT USE
85	SYS ACC ERR INV SYS ACC CODE
86	PROG STACK OFLOW
87	IN USE PACK POWERED OFF
88	INV OPEN <F-N> <ADR> INV MCS BFFR SIZE
89	INV BCT RQST - NO EXTRA MIX ALLOWED
90	DMS ERR <ADR> INV BCT PARAM
91	DMS ERR <ADR> NO ERR LABEL
92	DMS ERR <ADR> DATA BASE NOT OPEN
93	DMS ERR <ADR> USE PROC CALL FROM USE PROC
94	DMS ERR <ADR> USER TBL OFLOW
95	<P-N> DFS VIOLATION
A0 to B9	DIRECT I/O ERRORS

## BREAKOUT CONTROL

### BREAKOUT CONTROL (NO COBOL OR BPL SYNTAX)

This BCT can be used to programmatically inhibit operator BR and BD commands and, optionally, to notify the program that such an attempt was made. The format of this BCT is:

```
      BCT      0254
      BUN
P1.  CNST 6   UN  =  000000
P2.  CNST 1   UN  =   9
P3.  CNST 1   UN      :8 <available>
      :4 <available>
      :2 Pass breakout/restart R/Ds to program if ARMED
      :1 Disallow BR and BD commands
```

If the program has not performed Breakout Control, or if the last Breakout Control had P3 equal to zero, the program can be broken out and restarted at any time, by any method.

If the program has performed Breakout Control with P3 equal to one or three and the program is not ARMED, a BR or BD request is rejected with a NOT ALLOWED TO PROGRAM response. The program is unaware that a BR/BD request has been made. Programmatic breakout requests are honored.

If Breakout Control has been performed with P3 equal to two, and the program is ARMED, any breakout or restart is valid. However, at the completion of the breakout or restart, the program is reinstated at the ARM address with a Breakout Occurred or a Restart Occurred R/D.

If Breakout Control has been performed with P3 equal to three, a programmatic breakout or a restart results in the same actions as P3 = 2. If operator BR/BD is attempted, no MCP error message is given. The program is reinstated at its ARM address with an Operator Breakout Attempted R/D. The program can then handle this situation.

In any case where the program is reinstated at its ARM address, the program must reARM, if desired, before returning to mainline processing.

Refer to ARM BCT in this section.

## DATA BASE (NO COBOL OR BPL SYNTAX)

This BCT is the interface between user programs and the MCP when a DMS II data base is accessed. Only the COBOL-74 compiler can generate DMS II BCTs. The format of this BCT is:

BCT 0894  
BUN (around parameters)  
DATA (parameter list)

The length of the parameter list varies depending on the type of operation specified. The DMS II operations (along with the equivalent COBOL-74 verb) and the parameter information follow.

### CLOSE (CLOSE)

Exception routine address (6 UN)  
OP code = 02 (2 UN)

### CREATE (CREATE)

Exception routine address (6 UN)  
OP code = 03 (2 UN)  
Variant, 0 = CREATE, 1 = RECREATE (1 UN)  
Structure ID of data set (6 UN)  
Reserved (4 UN)  
Address of data set record area (6 UN)

### EQUATE PATH (SET)

Exception routine address (6 UN)  
OP code = 08 (2 UN)  
Structure ID of data set (6 UN)  
Structure ID of set (6 UN)

### FREE (FREE)

Exception routine address (6 UN)  
OP code = 13 (2 UN)  
Structure ID of data set (6 UN)

### INSERT (INSERT)

Exception routine address (6 UN)  
OP code = 09 (2 UN)  
Structure ID of data set (6 UN)  
Structure ID of set (6 UN)

### OPEN (OPEN)

Exception routine address (6 UN)  
OP code = 01 (2 UN)  
Variant: 0 = INQUIRY, 1 = UPDATE (1 UN)  
Length of SIB in words (3 UN)  
Address of DMERROR USE procedure (6 UN)  
Address of DMSTATUS register (6 UN)  
Address of SIB (6 UN)  
Data base name (6 UA)

## DATA BASE

### POSITION (SET)

Exception routine address (6 UN)  
OP code = 07 (2 UN)  
Variant: 0 = Set, 1 = Data set (1 UN)  
Option: 0 = BEGINNING, 1 = ENDING (1 UN)  
Structure ID of structure to be altered (6 UN)

### READ DIRECT (MODIFY)

Exception routine address (6 UN)  
OP code = 05 (2 UN)  
Variant: 0 = FIND, 1 = LOCK, 2 = DELETE (1 UN)  
Condition: 0 = NULL, 1 = SEQUENTIAL (1 UN)  
Motion: 0 = NEXT, 1 = PRIOR, 2 = FIRST, 3 = LAST (1 UN)  
Structure ID of data set (6 UN)  
Address of data set record area (6 UN)  
Reserved (1 UN)

### READ INDIRECT (MODIFY)

Exception routine address (6 UN)  
OP code = 06 (2 UN)  
Variant: 0 = FIND, 1 = LOCK, 2 = DELETE (1 UN)  
Condition: 0 = NULL, 1 = SEQUENTIAL, 2 = KEYED (1 UN)  
Motion: 0 = NEXT, 1 = PRIOR, 2 = FIRST, 3 = LAST (1 UN)  
Structure ID of data set (6 UN)  
Structure ID of set (6 UN)  
Set selection expression string size in words (4 UN)  
Address of set selection expression string (6 UN)  
Address of data set record area (6 UN)  
Reserved (1 UN)

### REMOVE (REMOVE)

Exception routine address (6 UN)  
OP code = 10 (2 UN)  
Variant: 0 = CURRENT, 1 = DATA SET (1 UN)  
Structure ID of data set (6 UN)  
Structure ID of set (6 UN)  
Reserved (1 UN)

### STORE (STORE)

Exception routine address (6 UN)  
OP code = 04 (2 UN)  
Structure ID of data set (6 UN)  
Address of data set record area (6 UN)

### RETURN FROM DMERROR USE PROCEDURE

Exception routine address (6 UN)  
OP code = 99 (2 UN)

## CLOSE (CLOSE)

The CLOSE BCT closes files and designates their disposition. The format of this BCT is:

```
      BCT      0154
      BUN
P1.  ACON      (FIB)
P2.  CNST 1    UN = X (type of CLOSE-Parameter 1)
      0 = FILE (REWIND)
      1 = REEL (REQIND)
      2 = FILE NO REWIND
      3 = REEL NO REWIND
      4 = FILE RELEASE
      5 = REEL RELEASE
      6 = FILE LOCK
      7 = REEL LOCK
      8 = FILE PURGE
      9 = REEL PURGE
      C = FILE CLOBBER
      E = FILE CLOBBER
P3.  CNST 1    UN = Y (type of CLOSE-Parameter 2)
      1 = CRUNCH P2 must be 4, 6, C, or E)
      2 = NO REQIND RELEASE (P2 must be 4)
      8 = NO DISCONNECT
```

CLOSE REEL and NO REWIND operations are defined for magnetic tape and paper tape only.

CLOSE CLOBBER and CRUNCH are defined for output disk files or disk pack only. CRUNCH can be ignored if more than one area of the file is allocated.

NO DISCONNECT applies only to dialed data communications devices.

CLOSE PURGE applies only to disk and disk pack files, and output magnetic tape files.



## COMPLEX WAIT

### COMPLEX WAIT (NO COBOL OR BPL SYNTAX)

The COMPLEX WAIT BCT allows a user to suspend execution until one of the specified events happens. If any event has happened at the time the BCT is executed, the program is reinstated immediately.

The COMPLEX WAIT BCT is not allowed to a program which is executing in the shared area.

The format of the BCT is:

```
      BCT          0994
      BUN
P1.  CNST 2UN    = XX (number of parms in list)
P2.  ACON        (address of parameter list)
P3.  ACON        (address of 2 UN using field or 0 if none)
P4.  ACON        (address of 2 UN giving field or 0 if none)
```

P2 points to the list of parameters which is defined as follows:

```
      CNST 4UN    = XXXX (event type)
                        0 = time (doze)
                        1 = ODT input present
      ACON        (depends on event type)
      ACON        (depends on event type)
```

If event type = 0, the first ACON points to a 5 UN time field in seconds and the second ACON is zero. If event type = 1, both ACONS are zero.

These three fields are repeated P1 number of times and must be contiguous.

When the BCT is executed, each event is tested to see whether it has already occurred. The order in which the events are tested is determined by P3. If P3 = 0, the events are tested in the order they are specified in the list. Otherwise, the first event tested is the one that occupies the P3-rd (1 relative) position in the list.

When the program is reinstated after executing a COMPLEX WAIT BCT, it has the option of requesting an indication of which event has happened. If P4 = 0 no indication is given; otherwise the (1-relative) position in the original list of the event that occurred is placed in the giving field.

When a program executes a COMPLEX WAIT BCT the MCP builds an entry in the Complex Wait Table for that program (refer to section 5). If there are not enough entries available in the Complex Wait table, the program is suspended while waiting for Wait Table space. However, if Wait Table space does not become available, the BCT is invalid and the program is terminated. The MCP's 100-second status routine allows any program waiting for Wait Table space to re-execute the COMPLEX WAIT BCT if space becomes available. The operator may cause an earlier re-execution of the BCT by issuing a <mix>OK command from the OCS.

## CORE-SIZE (MEMORY)

The CORE-SIZE BCT interrogates the amount of memory allocated to the user program. The format of this BCT is:

	BCT	0214
	BUN	
P1.	ACON	(response area)
P2.	CNST 1 UN	= 2 (flag specifying memory)
P3.	CNST 1 UN	= 0 (filler)

If the requestor is a compiler, the response area is a 7 UN field of the form CCCPEUS where:

CCC = Core in use (thousands) from base to limit. Does not include disk file header space.  
P = 1 - if Pack work file option is set (WRKP).  
2 - if code file is to be compiled to diskpack.  
EU = EU requested in COMPILE card for code file.  
S = 1 if COMPILE ...SYNTAX; otherwise 0.

If the requestor is not a compiler, the response area is a 6 UN field of the form CCC000.

B 2000/B 3000/B 4000 MCPVI Programmer's Guide  
Program Interface

DATE

DATE (JDATE)

The DATE BCT requests the Julian system date. The format of this BCT is:

	BCT	0214
	BUN	
P1.	ACON	(response area)
P2.	CNST 1 UN	= 4 (flag specifying DATE)

The response area is a 5 UN field where the Julian date is placed. This date is in the form YYDDD where YY is the year and DDD is the Julian day.

The reserved word for Julian date in COBOL-74 is DAY.

## DEBUG TRACE (NO COBOL OR BPL SYNTAX)

The DEBUG TRACE BCT is used to initiate tracing of the MCP. It cannot be used for tracing user programs. To trace user programs, see the TRACE BCT. The format of this BCT is:

```
      BCT 0514
P1.  ACON      (next instruction to be executed)
P2.  CNST 1    = X (state flag)
      UN
      0 = Zero base, control state
      1 = Non-zero base, control state
      2 = Zero base, normal state
      3 = Non-zero base, normal state
P3.  CNST 1    = 0 (filler)
      UN
```

P2 is ignored when the trace is called from other than the MCP; a value of 3 is always assumed.

The DEBUG module must be set (in memory).

## DIRECT I/O

### DIRECT I/O (NO COBOL OR BPL SYNTAX)

The DIRECT I/O BCT requests diagnostic file OPEN, CLOSE or I/O operation. The format of this BCT is:

	BCT	0434
	BUN	
P1.	ACON	(direct I/O FIB)
P2.	CNST 1 UN	= 8
P3.	CNST 1 UN	= X

0 = I/O Request  
1 = File OPEN Request  
2 = File CLOSE Request  
3 = Perform I/O and inhibit channel at IOC  
4 = Perform I/O and continue  
5 = Wait for I/O to complete  
6 = Cancel I/O  
7 = <<Reserved>>

This communicate is used to perform direct I/O operations in on-line diagnostic programs and certain utilities.

#### NOTE

Direct I/O is intended for use only in Burroughs supplied utilities and test programs. Burroughs Corporation will not be responsible for problems incurred by the use of Direct I/O in user programs.

## DISPLAY (DISPLAY)

The DISPLAY BCT displays low-volume data from a program. The format of this BCT is:

BCT	0254
BUN	
P1. ACON	(message area, must be mod 2)
P2. CNST 1 UN	= 1 (flag specifying DISPLAY)
P3. CNST 2 UN	= XX (size of message area in bytes)
P4. CNST 1 UN	Y
	:1 = Display library maintenance messages even if LIB = 0 (used only for LOADMP)
	:2 = Eliminate unnecessary blanks

P3 must not exceed 60 bytes (56 for programs having a multi-program identifier). If this limit is exceeded, only the leftmost 60 (or 56) characters are displayed.

Refer to DISPLAY UPON, DCOM, in this section.

## DISPLAY-LINES

### DISPLAY-LINES (NO COBOL SYNTAX)

The DISPLAY-LINES BCT allows a program to display multiple message lines on the local SPO or OCS. The format of this BCT is:

	BCT	0254
	BUN	
P1.	ACON	(message area, must be mod 2)
P2.	CNST 1 UN	= 8 (flag specifying DISPLAY-LINES
P3.	CNST 3 UN	= XXX (number of lines to display

This BCT ensures contiguity of all lines displayed on the SPO. Each line must be delimited by a NULL (00) character and must be 72 or fewer bytes in length.

#### NOTE

The display text is not prefaced by the program identification <P ID> = <mix-no.>.

## EXITROUTINE (NO SPECIFIC COBOL SYNTAX)

The EXITROUTINE BCT is used to exit from a USE routine. The format of this BCT is:

	BCT	0294
P1.	CNST 8 UN	= 0 (filler)
P2.	ACON	(FIB)

The compilers generate a BUN for P1; however, it is never executed.

P2 references the file (FIB) to which the USE routine applies.

Execution of this instruction causes the program to be reinstated at the address stored in FIBRCW of the first Buffer Status Block (refer to FIB in section 5 of this manual). This address is stored by the MCP when the routine is entered. Consequently USE routines must not perform operations which can result in the entry of other USE routines or the return linkage can be destroyed.



## FILL FROM – FILL INTO

### FILL FROM (FILL IN) – FILL INTO (FILL OUT)

The FILL BCT is used to receive/send data from/to a specified program by means of the MCP core-to-core (CRCR) mechanism. The format of this BCT is:

	BCT	0414
	BUN	
P1.	ACON	(send/receive buffer area)
P2.	ACON	(Program-name field)
P3.	ACON	(PROCEED to label)
P4.	CNST 4 UN	= XXXX (# of units to move)
P5.	CNST 1 UN	= Y (type of unit)
		0 = Bytes
		1 = Digits
		2 = Words
P6.	CNST 1 UN	= Z (action type)
		0 = Send
		1 = Receive

The program name field is assumed to be 6 UA and can contain either blanks or the name of the program with which the hook up is to be made. In the former case (GLOBAL FILL), any program in the appropriate state can be used for the hook up except one which is also doing a GLOBAL FILL.

If multiple programs are prepared for a hook up, the highest priority job is chosen.

If the action label (P3) is not present (is zeros), the program is suspended until the transfer can take place. If the field is non-zero, the program is reinstated at that address if a hook up cannot be made at that time.

If the buffer size (P4) of the sender and receiver are not equal, the shorter size is taken.

If a FILL INTO request is executed and no program is ready for the transaction; but a program of the appropriate name is in either a WAIT or SLEEP status, that program is marked ready to run, however, no data is transferred. Use of this mechanism can improve hook up response time while reducing wasted hook up attempts (buzzing). If multiple programs are in such a state, only the highest priority program is marked ready to run.

Data types of sender and receiver need not be the same, as the MCP chooses the transfer unit based on modularity of data addresses and length.

CRCR mod must be set (in memory).

The respective constructs, in COBOL-74, for FILL FROM and FILL TO are RECEIVE and SEND.

Refer to Interprogram Communication, section 3.

FILL FROM BOTTOM – FILL FROM TOP

FILL FROM BOTTOM (PULLQ) – FILL FROM TOP (POPQ),  
FILL INTO BOTTOM (PUTQ) – FILL INTO TOP (PUSHQ)

This FILL BCT is used to receive/send data from/to a storage queue by means of the MCP STOQUE mechanism. The format of this BCT is:

	BCT	0494
	BUN	
P1.	CNST 1 UN	= 0 (reserved)
P2.	CNST 1 UN	= X (request type)
		2 = PUSHQ (FILL INTO TOP)
		3 = POPQ (FILL FROM TOP)
		8 = PUTQ (FILL INTO BOTTOM)
		9 = PULLQ (FILL FROM BOTTOM)
P3.	ACON	(STOQUE parameter block)
P4.	ACON	(operation failed label)

P2 specifies the type of action requested. Values of 2 and 8 specify storage requests at the TOP or BOTTOM of the queue, respectively. Values 3 and 9 indicate a retrieval request from the queue head or tail respectively.

P3 references the Stoque Parameter Block which contains the name of the queue to be used for the operation, an individual queue entry name if desired, and a message area.

If P4 is zero, the program is suspended if a storage request cannot be accomplished due to lack of space (PUSHQ, PUTQ) or if a retrieval request cannot be satisfied because the desired element is not available (POPQ, PULLQ). If P4 is not zero, the program is reinstated at that address if the request cannot be satisfied at that time.

The COBOL-74 construct for FILL FROM is RECEIVE FROM; while SEND TO is used for FILL INTO.

STOQ mod must be set (in memory).

Refer to Interprogram Communication, STOQUE, section 3.

## FILL FROM POLL

### FILL FROM POLL (POLLQ)

The FILL FROM POLL BCT requests a count of the entries in a specific storage queue or part of a queue. The format of this BCT is:

	BCT	0494	
	BUN		
P1.	CNST 1 UN	= 0	(reserved)
P2.	CNST 1 UN	= 4	
P3.	ACON		(STOQUE parameter block)

P3 references the Stoque Parameter Block which contains the name of the queue to be polled, optionally an individual entry name, and space for the response.

STOQ mod must be set (in memory).

Refer to Interprogram Communication, section 3.

## INTERROGATE FILE (FIND)

The INTERROGATE FILE BCT requests the MCP to check for the presence of a disk or disk pack and optionally pass the disk or disk pack file DFH to the requestor. For disk files, the format of this BCT is:

	BCT	0214
	BUN	
P1.	ACON	(field containing name of file to find)
P2.	CNST 1 UN	= 3 (flag specifying INTERROGATE FILE)
P3.	CNST 1 UN	= X (answer digit and special parameter)
P4.	ACON	(response area – optional)

P1 addresses a 6-byte field (mod 2 address) which contains the name of the diskfile requested.

P3 has two functions:

Certain values cause DFH information to be returned into BASE + 100 if the file is found (Refer to section 5):

:2 = File attributes returned (40 digits)

:4 = File attributes plus all area addresses (40-840 digits)

:2\*:4 = File attributes plus the first 20 area addresses (200 digits).

If :8 is set, the answer will be placed in the 1-digit field addressed by P4; otherwise the answer is placed in P3 (P4 is not needed):

P3:1 set = File found in directory

:1 reset = file not in disk directory

All other bits are not disturbed.

For disk pack, the format of this BCT is:

	BCT	0214
	BUN	
P1.	ACON	(points to MFID/FID)
P2.	CNST 1 UN	= 9
P3.	CNST 1 UN	= X (answer digit and special parameter)
P4.	ACON	(response area – optional)

P1 must address a 16 UA field with the following format:

CNST	1 UA	= Space
CNST	6 UA	= <pack-ID> otherwise spaces
CNST	2 UA	= Space
CNST	6 UA	= <file-ID>
CNST	1 UA	= Space

## INTERROGATE FILE

This field is the same format as a standard LABEL.

P3 and P4 have the same meaning as with disk except that the pack file header is returned in the format of a disk file header (refer to section 5). The first 40 digits of the pack file header are reformatted as follows.

PF-RSZ	→	DF-RSZ	5 UN (Record size in digits)
PF-RPB	→	DF-RPB	3 UN (Records per block)
PF-#AR	→	DF-#AR	2 UN (Number of areas)
PF-EOF	→	DF-EOF	8 UN (End of file pointer)
PF-NU1	→	DF-USE	2 UN (Number of users on processor #0)
PF-NU2	→	DF-USE+02	2 UN (Number of users on processor #1)
PF-NU3	→	DF-USE+04	2 UN (Number of users on processor #2)
PF-NU4	→	DF-USE+06	2 UN (Number of users on processor #3)
			5 UN (Reserved)
PF-TP3	→	DF-ST1	1 UN (File type)
PF-TP2	→	DF-DKS	1 UN (File type)
PF-SPA	→	DF-DSA	7 UN (Sectors per area)

## READ WITH LOCK-ONLY (LOCK)

The READ WITH LOCK-ONLY BCT requests that a block of a shared disk or disk pack file be LOCKed without any transfer of data. The format of this BCT is:

```
          BCT    0114
          BUN
P1.  ACON  (FIB)
P2.  ACON  (EOF Address)
```

The four bit of the high-order digit of P1 must be set. This communicate is actually a variation of READ.

This operation causes a block to be LOCKed but no data is transferred to memory. If the block is already LOCKed by another program, the program waits until it has been UNLOCKed.

Refer to READ BCT in this section and to Appendix C, Programming Considerations for Shared Files.

## LOCK WITH SEEK

### LOCK WITH SEEK (NO COBOL SYNTAX)

The LOCK WITH SEEK BCT requests that a block of shared disk or disk pack file be LOCKed without any data transfer. The operation is identical to LOCK except that the program does not wait if the block is currently LOCKed. The format of this BCT is:

```
BCT 0314  
BUN  
P1. ACON (FIB)
```

The four bit of the high-order digit of P1 must be set. This communicate is actually a variation of SEEK.

Refer to SEEK BCT and Appendix C, Programming Considerations for Shared Files.

## OPEN (OPEN)

The OPEN BCT requests the association of a physical device or file with the user program file based on requested hardware type, specified OPEN mode, and file name. The format of this BCT is:

```
          BCT          0134
          BUN
P1.  ACON          (FIB)
P2.  CNST 1 UN    = X (type of OPEN)
          0 = INPUT
          1 = OUTPUT
          2 = INPUT/OUTPUT (I-O)
          3 = OUTPUT/INPUT (O-I)
          4 = EXTEND
P3.  CNST 1 UN    = Y
          0 = REWIND
          1 = NO REWIND
          2 = REVERSE
          4 = FLOW (MICR files)
          LOCK ACCESS (Other than MICR files)
          6 = REVERSE/LOCK ACCESS
          8 = LOCK
          A = REVERSE/LOCK
```

OPEN I/O is valid only for disk, disk pack, and DCOM files.

OPEN O/I is valid only for random disk and disk pack files.

OPEN NO REWIND applies only to magnetic tape.

OPEN REVERSE applies only to sequential disk and magnetic tape files.

LOCK/LOCK ACCESS applies only to permanent disk files.

EXTEND is valid for permanent disk, disk pack, and labelled magnetic tape files. This is essentially an output function where the file is positioned at its EOF such that subsequent WRITES will create new records beyond the prior EOF. As of the ASR 6.2 release, only the COBOL-74 compiler generates the EXTEND option.



OVLY

## OVLY (NO SPECIFIC COBOL OR BPL SYNTAX)

The OVLY BCT requests the MCP to call in a specified overlayable segment if the segment is not present in memory. When present, the program branches to a designated location. The format of this BCT is:

```
MVN  0303 NNN Segdict(0): +18  
BUN  Segdict(NNN):IA
```

NNN = segment# requested. Segdict(NNN) is the segment dictionary entry for the requested segment. Segdict(0) refers to the Master Segment Dictionary Entry for the program.

If the overlay is not in memory, the first six digits of the NNN segment dictionary entry contains the address of the Master Segment Dictionary Entry. This, in turn, contains the overlay communicate instruction (BCT 0174) and data used by the MCP overlay call routine. MCP then reads the requested segment into the appropriate memory area and marks the segment present by moving the address of the first executable instruction of the overlay to the first six digits of Segdict(NNN). Similarly, any segments overlaid by the current segment are marked absent by moving the address of Segdict(0) to the first six digits of Segdict(NNN). Thus, any future branch to the segment results in an indirect branch to the segment without MCP intervention, and any call on an absent overlay calls the MCP as described above.

The address of the first executable instruction mentioned above is accessed from Segdict(NNN):+12 which is actually the address of the instruction to be executed after the overlay becomes (or is found to be) present. For example, in COBOL programs, this field addresses a non-overlayable branch instruction which has been modified to point to the desired label in the segment.

Refer to Object Programs, section 6.

## MIXTBL (NO COBOL SYNTAX)

The MIXTBL BCT requests information about current jobs in the mix. Programs running in the time-share area that execute this BCT will only receive information about jobs in the time-sharing mix. The format of this BCT is:

	BCT	0214
	BUN	
P1.	ACON	(program ID field, table, or unused depending on P3)
P2.	CNST 1 UN	= 6 (specifies PROGRAM request)
P3.	CNST 1 UN	= X (defines type of request)
P4.	CNST 2 UN	= YY (holds response)

The values of P1 and P3 can be:

P3	P1	Request
0	Program ID Address	Place number of programs in mix with specified ID in P4.
1	Filler	Place number of programs in mix in P4.
2	Program ID Address	Place <mix-no.> of specified program in P4.
4	Table Address	Place mix information in specified table; also put number of programs in mix in P4.
8	Filler	Place <mix-no.> of caller in P4.
9	Response Area	Place caller information in P4.

Following is the table format (P3 = 4).

Header	
Reserved	1 UN
Jobs in Mix	2 UN (value also in P4)
Core Available	3 UN (mod 1000; total available memory)

Body (one entry for each program)	
MIX ID	6 UA (program name)
MIX-MF	6 UA (multi-program name)
MIX-NO	2 UN (mix number)
	1 UN (reserved)
	3 UN (memory used by job, base to limit only)

B 2000/B 3000/B 4000 MCPVI Programmer's Guide  
Program Interface

MIX IBL

The program ID field (where relevant) must be 6 UA, mod 2, and must contain the name of the program in question.

It is the responsibility of the programmer to allocate a table of sufficient size when  $P3 = 4$ .

For  $P3 = 9$  the response area contains the following information:

6 UA	Program ID
6 UA	Multi-program ID
2 UN	Mix number
4 UN	RLOG number
4 UN	RJE link
6 UN	Available

If the requestor has been initiated through RJE, the RJE link should be used when creating pseudo-decks.

## QUICKTIME (NO COBOL SYNTAX)

The QUICKTIME BCT requests the current system time from the MCP. The format of this BCT is:

	BCT	0454
	BUN	
P1.	ACON	(response area)
P2.	CNST 1 UN	= 1 (TIME request)
P3.	CNST 1 UN	= 4 (DATE and TIME request)

If P2=1, the response area is a 10 UN field and contains the current system time in milliseconds.

If P2=4, the response area is a 15 UN field and contains date and time in the following format:

YYDDmmmmmmmmmm

where YY is the last two digits of the year, DDD is the day in Julian format, and mmmmmmmmmmm is the time in milliseconds.

Because this BCT does not initiate an MCP overlay sequence such as that done by the TIME or TIME-60 BCTs, QUICKTIME requires less system overhead.

In COBOL-74, the reserved word TIMER is used for the QUICKTIME BCT.

RFAD

## READ (READ)

The READ BCT requests that a record from the designated file be delivered to the requesting program. The format of this BCT is:

```
          BCT    0114
          BUN
P1.  ACON  (FIB)
P2.  ACON  (EOF Address)
```

The eight and four bits of the high-order digit of P1 specify shared disk actions as follows:

Neither = Plain READ, ignore LOCK

:4 only = Refer to the READ WITH LOCK-ONLY BCT in this section

:8 only = LOCK, then plain READ

:8\*:4 = Lock, then plain READ, then UNLOCK

A zero value in P2 indicates that no EOF address is present.

Refer to Appendix C, Programming Considerations for Shared Files.

## SEEK (SEEK)

The SEEK BCT requests the MCP to make the requested random disk record available in the program buffer. If necessary, a physical I/O operation is initiated. The program processing is not suspended. The format for this BCT is:

BCT 0314  
BUN  
P1. ACON (FIB)

Invalid key conditions and irrecoverable I/O errors are ignored until the program executes a READ or WRITE on the record.

The eight and four bits of the high-order digit of P1 specify shared disk actions as follows:

Neither = Plain SEEK, ignore LOCK

:4 only = Refer to the LOCK WITH SEEK BCT in this section

:8 only = LOCK, then a plain SEEK.

:8\*:4 = LOCK, then a plain SEEK, then UNLOCK

Refer to Appendix C, Programming Considerations for Shared Files.

## SORT

### SORT (SORT)

The SORT BCT requests that a file be sorted. The format of this BCT is:

	BCT	0254
	BUN	
P1.	ACON	(sort parameter string)
P2.	CNST 1 UN	= 5 (specifies SORT)
P3.	CNST 1 UN	= 0 (filler)
P4.	ACON	(input FIB)
P5.	ACON	(output FIB)

P1 addresses the sort parameter specifications which consist of two to 50 successive 12-digit fields of the following format:

#### Type 1. General Sort Specifications (required)

8 UN = Total records or records/area for sort work files (see special actions digit). Ignored if input from disk.

1 UN = Parity action

0 = DROP

1 = USE

2 = END

1 UN = Input CLOSE type (see CLOSE for meanings). Valid values are 0, 2, 4, 6, 8, C, E. A zero value is converted to six by MCP before the sort is called.

1 UN = Output CLOSE type (as above)

1 UN = Special actions

:1 = Reserved

:2 = Userblock (both input and output)

:4 = Reserved

:8 = If set, record field contains the total number of records in the file; if not set, records/area.

Type 2. Work File Allocation Specification (optional)

1 UN = Flag digit

A = Identifies field as type 2 parameter

1 UN = Pack restrictions

:1 = Reserved

:2 = Reserved

:4 = Use single pack only for work files (ignored if :8 not set)

:8 = If set, work files allocated on disk packs, else HPT disk

1 UN = Reserved

1 UN = Disk assignment technique. Values correspond to FIBDTK (refer to FIBs, section 5)

2 UN = Disk assignment techniques. Values correspond to FIB-EU (refer to FIBs, section 5)

5 UN = Reserved

1 UN = Zero

Type 3. Key Specifications (one per key)

2 UN = Zeros

2 UN = Key Size (digits if class is 4-bit; bytes if 8-bit)

5 UN = Zero relative location of key in record (digits or bytes as appropriate)

1 UN = Key class

0 = UN

1 = SN

2 = UA

4 = SA

1 UN = Sort order



**SORT**

1 = Ascending

4 = Descending

1 UN = Flag

:1/ = More keys follow

:1 = No more keys follow

:2 = Name of translate table follows (valid only if :1 also set)

Type 4 = Translate Table ID (optional-valid only if :2 set in flag digit of last key specification)

6 UA = Name of 400-byte disk file to be used as translate table for sort keys (virtual collating sequence)

P4 points to the FIB of the file to be sorted. P5 points to the FIB of the file to be created as the output of the sort process.

## SORT RETURN (NO COBOL OR BPL SYNTAX)

The SORT RETURN requests a return from the sort intrinsic to the user program and is only used by SORT. The format of this BCT is:

	BCT	0254
	BUN	
P1.	CNST 6 UN	= 0 (filler)
P2.	CNST 1 UN	= 6 (specifies SORT RETURN)
P3.	CNST 1 UN	= 0 (filler)

SPACE

SPACE (NO SPECIFIC COBOL SYNTAX)

The SPACE BCT requests that the designated printer, magnetic tape, sequential disk, or sequential disk pack file be positioned to a specific point. The format of this BCT is:

	BCT	0394
	BUN	
P1.	ACON	(FIB address)
P2.	ACON	(EOF address; for printer, EOP address)
P3.	CNST 4 SN	= SUUVV (position parameters)
P4.	CNST 1 UN	= 0 (filler)

SPACE cannot be used on variable length records.

Output magnetic tape files can only be spaced in reverse.

The position parameters are of the format SUUVV where:

S = Sign digit (direction)

C = Forward

D = Reverse

The sign digit is not used for printer files.

UUVV = (should not be zero)

For magnetic tapes, disk, and disk pack files:

Number of records to space

For printer files:

UU = Number of lines to space (0-99)

VV = Channel number to which to skip; skipping overrides spacing.

When positioning a multi-pack disk pack file, all required packs must be on-line when they are needed by the positioning logic. A position may not be done to a pack which is off-line. Attempting this will terminate the job.

## SPOMESSAGE (NO COBOL SYNTAX)

The SPOMESSAGE BCT passes keyboard input messages to the MCP and requests that the response be returned to the requestor. The format of this BCT is:

BCT	0474
BUN	
P1. ACON	(buffer containing SPO input message)
P2. ACON	(buffer for reply)
P3. CNST 4 UN	= XXXX (length of reply buffer in digits)

The input message must be terminated by either a period (.) or an ETX character (03). The length of the input message must be less than 73 characters.

The length of the buffer (P2) must be at least 160 digits. If the input message is other than: AJ, BF, BP, CD, CK, CN, DB, DC, DQ, FI, FN, IR, MX, OL, OT, PD, RO, SO, SS, TI, TO, UR, WB, WC, WD, WJ, WM, WQ, WS, WT, WXD, WXM, WXP, WY, or XC, the reply from the MCP is:

BELBEL \*\* KBD IGNORED: REQUEST NOT ALLOWED.

Each response line is placed in the buffer as the line appears on the SPO if the job executing the SPO message is an RJE handler; if it is not, the leading space on the message is suppressed (to be compatible with MCPV). Each line is terminated by carriage return and line feed (@0D0A@); the last (or only) line is additionally terminated by ETX (@03@).

If the buffer is too small for all lines of the response, a NULL character (@00@) follows the last full line which fits into the area.

The length of the reply buffer must be at least 160 digits and must not overlap the input area as it is cleared to spaces before the input text is processed. The length of the output buffer must be an even number of digits.

STOP

## STOP (STOP)

The STOP BCT requests termination of the calling program. The format of this BCT is:

BCT 0194

Any disk or disk pack files still OPEN at EOJ are CLOSEd with the following dispositions:

Permanent files OPENed INPUT or I-O are CLOSEd RELEASE.

Files OPENed OUTPUT or O-I are CLOSEd PURGE.

This can be overridden by the setting of FIBST1:2 for a disk or disk pack file (refer to FIBs, section 5).

In timesharing, if a type 1 job (normal user program) executes a STOP BCT, it is handled as a Process Return BCT. All files belonging to the type 1 job are CLOSEd as described above and control is returned to the last type 2 job which executed a call-and-return Process Call BCT. (or to the MCP if no type 2 job was executed). If a type 2 job executes a STOP BCT, all OPEN files are CLOSEd (as above), and the entire job is terminated.

## TIME (TIME)

The TIME BCT requests the current system time in milliseconds from the MCP. The format of this BCT is:

	BCT	0214
	BUN	
P1.	ACON	(response area)
P2.	CNST 1 UN	= 1 (flag specifies time)
P3.	CNST 1 UN	= 0 (filler)

The response area must be a 10 UN field.

Refer to QUICKTIME and TIME-60 in this section.

TIME-60

## TIME-60 (TIME60)

The TIME-60 BCT requests the current system time from the MCP in hours, minutes, and seconds. The format for this BCT is:

	BCT	0214
	BUN	
P1.	ACON	(response area)
P2.	CNST 1 UN	= 8 (flag specifying TIME-60)
P3.	CNST 1 UN	= 0 (response as 00HHMMSS66) 1 (response as 00HHMMSSXX)

The response area must be a 10 UN field. The response is in the format 00HHMMSS66 or 00HHMMSSXX where:

HH = Hours  
MM = Minutes  
SS = Seconds  
66 = 60ths of a second (rounded)  
XX = Hundredths of a second

The 00HHMMSSXX format is returned when the COBOL-74 construct ACCEPT <data-name> FROM TIME is used.

Refer to QUICKTIME and TIME in this section.

TODAYS-DATE

## TODAYS-DATE (DATE)

The TODAYS-DATE BCT requests the current system date in the Gregorian format. The format of this BCT is:

	BCT	0214
	BUN	
P1.	ACON	(response area)
P2.	CNST 1 UN	= 0 (flag specifies TODAYS-DATE)
P3.	CNST 1 UN	= 0 (response in MMDDYY form) = 1 (response in YYMMDD form)

The response area is a 6 UN field where:

MM = Month  
DD = Day  
YY = Year

Response in the YYMMDD format is obtained only by using the COBOL-74 construct ACCEPT <data-name> FROM DATE.

Refer to the DATE (Julian date) BCT in this section.



TODAYS-NAME

## TODAYS-NAME (NO COBOL OR BPL SYNTAX)

This BCT can only be obtained thru the COBOL-74 construct TODAYS-NAME. The function of this BCT is to obtain the symbolic representation of the current day of the week, the date, and the time. The format of this BCT is:

	BCT	0214
	BUN	
P1.	ACON	(response area)
P2.	CNST 1 UN	= 7
P3.	CNST 1 UN	(function flag)

The COBOL-74 construct which generates this BCT is ACCEPT <data-name> FROM TODAYS-NAME.

The possible values and interpretations for P3 are:

- 0 = Day of the week in lower-case (9 UA)
- 1 = Date in the form "Dec 25, 1979" (no quotes) (12 UA)
- 2 = Time in the form "9:36 a.m." (10 UA)
- 3 = All of the above (excess spaces removed) (36 UA)

If P3:4 is on, then all responses will be in upper-case.

## TRACE (TRACE OR DUMP)

The TRACE BCT requests the MCP to initiate or terminate a program trace or to produce a memory dump of the program. The format of this BCT is:

	BCT	0334
	BUN	
P1.	CNST	= 00 (turns off trace) 01 (turns on default trace) 03 (turns on trace backup disk) 05 (turns on trace within limits) 07 (turn on backup trace within limits) 1X ((trace interrogate) (X = response) 20 (memory dump) 21 (partial memory dump) 22 (memory dump to disk) 23 (partial memory dump to disk)
P2.	ACON	(beginning address for trace/dump)
P3.	ACON	(ending address for trace/dump)
P4.	CNST 3 UN	(beginning segment number for trace)
P5.	CNST 3 UN	(ending segment number for trace)

The Trace Interrogate function returns zero if the program is not being traced; otherwise, one is returned.

For a partial memory dump, P2 and P3 are rounded down and up, respectively, to the next modulo 1000 address.

For trace, P2 and P3 specify the address of instructions where tracing is to start and end, respectively. If either address is zero, the corresponding limit is not enabled. The segment numbers correspond to the beginning and ending addresses and must be supplied. A segment number of zero means any segment, not just segment zero. The trace bounds can be altered while tracing.

P4 and P5 are not required for a Dump. P2 through P5 are not required if P1 = 00, 01, 03, 1X, 20, or 22.

UNLOCK

## UNLOCK (UNLOCK)

The UNLOCK BCT requests that a block of the designated shared disk file be UNLOCKed. The format of this BCT is:

```
BCT 0234  
BUN  
P1. ACON (FIB)
```

The 8-bit of the high-order digit of P1 must be set.

This operation causes a previously LOCKed block of a shared disk file to be UNLOCKed. If the block has not been previously LOCKed by the program, then the program will be terminated. This request is a variation of a WRITE request.

Refer to Appendix C, Programming Considerations for Shared Files.

## USERCHANGE (NO COBOL OR BPL SYNTAX)

The Userchange BCT allows a program to change the usercode and charge number under which it is running. It is used by certain system intrinsics, such as RJE, which are executed under the system default charge number but wish to charge time and jobs to the appropriate user. The format of this BCT is:

```
          BCT    0534
          BUN
P1.  ACON  (input parameters)
P2.  ACON  (response area)
```

P1 points to an alphanumeric string containing the following information:

- 6 UN = 000100 specifies Userchange function
- 10 UA = new usercode or spaces
- 10 UA = new password or spaces
- 6 UN = new charge number or zeros
- 6 UN = FIB address of remote device

P2 points to a two character alphanumeric reply area. Possible replies are:

- 00 = combination valid and change done
- 01 = invalid syntax
- 02 = combination syntax free but invalid
- 03 = system error
- 04 = smashed FIB

If the Security option is set, the contents of the input fields will be verified for syntax, and, if error free, will be validated against the usercode file. If Security is not set, only the syntax will be checked. If the input passes all applicable checks, the usercode, password, and charge number under which the program is running will be changed to the input values. A type 4/2 RLOG record will be written to the RLOG file.

## USERCODE

### USERCODE (NO COBOL OR BPL SYNTAX)

The Usercode BCT gives a program the ability to determine the usercode under which it is running. The format of this BCT is:

```
          BCT    0534
          BUN
P1.  ACON  (input parameters)
P2.  ACON  (response area)
```

P1 points to a 6-character alphanumeric string containing "000200".

P2 points to a 20-character response area in the following format:

2 UA (error flag)

"00" – no errors

"03" – system error

18 UA – FILLER

10 UA – usercode or spaces

## WAIT (DOZE)

The WAIT BCT requests that program execution be suspended (sleep) for at least a specified number of seconds. The format of this BCT is:

	BCT	0254
	BUN	
P1.	ACON	(field where time is stored)
P2.	CNST 1 UN	= 2 (specifies WAIT)
P3.	CNST 1 UN	= 0 (filler)

The time field is 5 UN and must contain a value from 1 to 86399. A value of 0 is the same as 1; undigit values result in no suspension. Values greater than 86399 are equivalent to 86400 (one day).

If another program attempts a CRCR SEND operation to a sleeping program, the WAIT is immediately terminated; also, if a program is attempting CRCR SEND to a program attempting a WAIT, the WAIT becomes, effectively, a no-operation.

The WAIT BCT is similar in function to the DATACOMM WAIT <time> request which is preferable to the DOZE request (on those systems which have either the DCOM or DCP module set) as it requires considerably less system overhead.

## WRITE

### WRITE (WRITE)

The WRITE BCT requests that a record of the designated file be written to the associated physical device. The format of this BCT is:

	BCT	0234
	BUN	
P1.	ACON	(FIB)
P2.	ACON	(EOF address; for printer files EOP address)
P3.	CNST 1 UN	= 0 (filler)
P4.	CNST 3 UN	= XYZ (printer/punch formatting)

The eight and four bits of the high-order digit of P1 specify shared disk actions as follows:

Neither = LOCK, then a plain WRITE, then UNLOCK

:4 only = Plain WRITE

:8 only = Refer to the UNLOCK BCT in this section.

Refer to Appendix C, Programming Considerations for Shared Files.

P2 applies to printer, disk, and disk pack files only. For printer files P2 specifies the address of an EOP routine to which control is given when a 12-punch is sensed on the printer. For disk and disk pack files, P2 specifies the address of the invalid key routine for random files or the EOF routine for sequential files (declared file bounds exceeded).

P4 is used only for printer and punch files. The X variant contains 0, 1, or 2 for print files and specifies spacing of 0, 1, or 2 lines; it is ignored for punch files. The YY variant contains a channel number (00-11) for printer skipping or a stacker number (00, 01, or 02) for punch files.

## WRITE BREAKOUT (BREAKOUT)

The WRITE BREAKOUT BCT requests the MCP to perform a breakout (checkpoint) operation on the requesting program. The format of this BCT is:

	BCT	0334	
	BUN		
P1.	CNST 1 UN	= 4	(specifies BREAKOUT)
P2.	CNST 1 UN	= X	(BREAKOUT file routing)
		8	= Use disk for BREAKOUT
		4	= Use tape for BREAKOUT
		2	= Use default device
		1	= Save previous disk BREAKOUT file
		8*4	= Use pack for BREAKOUT

If P2 is zero, the breakout file is written to disk if the BRDK system option is set; otherwise, magnetic tape is used.



## ZIP

### ZIP (ZIP)

The ZIP BCT passes control information to the MCP for processing. The format of this BCT is:

```
BCT 0274  
BUN  
P1. ACON (control information buffer)
```

Unless the first character of the buffer is a period (.), the buffer must be 72 or fewer characters in length. Further, the buffer must be terminated by a period. (If one cannot be found in the first 72 bytes of the buffer, the MCP inserts one into buffer location 72.) If the first character of the buffer is a period, the string can exceed 72 bytes, and the buffer must be terminated by a period. The following cannot be ZIPped: ALLOCATE, BINARY, CHANNEL, DATA(B), DISK, DISPLAY, DLP, LABELn, PATCH, QWKMEM, and UNIT control instructions; and the BO, CQ, DQ, HL, HM, KX, LH, LI, LO, and SM keyboard messages.

The MCP displays up to the first 30 characters of the ZIP control text buffer on the SPO if the ZIPM option is set, and the request is not START or STOP.

## ZIPSP0 (NO COBOL OR BPL SYNTAX)

The ZIPSP0 function gives the user the ability to pass control information to the MCP and to determine if any errors occurred during processing. The format of this BCT is:

BCT 0534  
BUN  
P1. ACON (input parameters)  
P2. ACON (response area)

P1 points to an alphanumeric area containing the word "ZIPSP0" followed by the text to be zipped. This text must be in the same format as for the ZIP BCT. Refer to ZIP in this section.

P2 points to an 82-character response area. The first two characters are a numeric error code.

If no errors are detected, the error code will contain "00" followed by a 4-character value containing the run log number of the last job initiated by the ZIPSP0 BCT or "0000" if no jobs were initiated. The remaining 76 characters are undefined.

If an error is detected, the first two characters are the non-zero error code and the following 80 characters will be a "\*\*\* ZIP IGNORED: ..." keyboard response terminated by ETX.

## DATA COMMUNICATIONS BRANCH COMMUNICATES

Most data communications communicates specify the same address (BCT 0354). The parameters following the communicate specify the particular operation requested.

Several of the requests can specify variants in the form of an 8-bit string which is inserted into the physical I/O descriptor for the operation. Table 2-2 summarizes the meaning of the variant bits. Bits are numbered from one to eight (right to left). Certain bits are not permitted for some operations and must be zero. In most cases the bits indicate special variants to the standard operation and correspond to variants specified in source language statements. In some cases (for example, WCRC) distinct source language syntax exists for an operation which differs from another (WC) only by a variant bit. Table 2-2 specifies the source language operations to which the variants are applicable rather than the actual I/O descriptor OP codes. In some cases the bits have more than one meaning depending on the OP code or the setting of other variant bits.

The OP codes and variants are for MLC/STC terminal adapters. For terminals attached to DLPs, the proper translation of OP code and variants must be performed.

**Table 2-2. Data Communications Descriptor Variants**

Bit	Meaning	Abbreviation	Application
8	TONE	R	WC, WCRC (Touch-Tone®)
7	NO TIME OUT BREAK	T B	RC, WCRC, WTRC, WCRT UNCL
6	END TEXT (delete ETX)	E	WC, WCRC, (8A1, 1050)
5	START TEXT (preset STX)	X	RC, WC, WCRC (8A1, 1050)
4	POLL (if bit 3 on) Read Transparent	P	WCRC (variant of WC) WCRT (variant of WC)
3	Flip Flag	F	WCRS (indicates flip to read after write – variant of WC)
2	STREAM	S	RC, WC, WCRC (polling operation only – indicates recirculating poll)
1	DIAL DISCONNECT (hang up)	D H	RC, WC, WCRC, WTRC, WCRT UNCL

DIAL must not occur with any variants except those specifying WCRC, WTRC, and WCRT.

POLL (flip flag must be on) must not occur with any other variants except STREAM.

START TEXT must not occur with END TEXT.

VOICE is defined as a variant in the programming languages and it is effectively noise as it merely sets the TONE flag off; consequently, VOICE and TONE are mutually exclusive.

\* Registered trademark of A.T. & T. Co.

BREAK applies only to full duplex lines.

NO TIME OUT for WCRC, WTRC, and WCRT operations applies only to the read portion.

Table 2-3 gives a quick reference to the Data Communications (DCOM) BCTs. Since all DCOM BCTs other than OPEN, CLOSE, ACCEPT FROM, and DISPLAY UPON are to address 354, the first parameter denotes the operation.

**Table 2-3. Data Communications BCT Quick Reference**

<b>Value</b>	<b>Meaning</b>
00	Read
01	Fill
02	Write
03	Write-Read
04	Write-Trans-Read
05	Write-Read-Trans
06	Enable
07	Interrogate
08	Disable on No-Data (condcancel)
09	Ready Buffer
10	Wait
11	Disable (cancel)
12	Interrogate End-Text
13	Translate Table Fetch

In addition, values from 20-26 are permitted and function as extended forms of values 00-06. These 20-series operations request automatic INTERROGATE and INTERROGATE-END-TEXT following the completion of the requested operation.

## ACCEPT FROM

### ACCEPT FROM (ACCEPT)

The ACCEPT FROM BCT requests a message to be input from the designated remote SPO. The format of this BCT is:

	BCT	0254
	BUN	
P1.	ACON	(input area, must be mod 2)
P2.	CNST 1	= 3 (specifies ACCEPT)
	UN	
P3.	CNST 2	= XX (size of input area in bytes)
	UN	
P4.	CNST 1	= 0 (filler)
	UN	
P5.	ACON	(remote identifier)

P3 must not specify more than 60 bytes.

P5 points to a 6 UA area containing either the cc/u designation or the adapter ID of the remote SPO from which data is to be requested.

If the value in the field to which P5 points is the special identifier REMSPO, the ACCEPT occurs from the remote SPO which initiated the job regardless of the actual adapter ID.

If the indicated device is not a remote SPO (including the initiator for REMSPO requests) or is not logged-in, the request is processed as if it were ACCEPT (local SPO).

#### NOTE

If a job has been executed from a remote SPO all input and output messages which would normally go to the local SPO will be directed to the remote SPO. Thus, it is not necessary to do an ACCEPT FROM REMSPO; a normal ACCEPT will have the same results.

DISABLE [ON BREAK] (DATACOMM CANCEL)

DISABLE [DISCONNECT] (DATACOMM CANCEL)

The DISABLE BCT requests that an unconditional cancel operation be performed on the designated file. The current I/O operation is cancelled if it is not yet completed; further, a break can be sent and/or a dialed line can be disconnected (if applicable). The format of this BCT is:

BCT	0354
BUN	
P1. CNST 2 UN	= 11 (specifies CANCEL)
P2. ACON	(FIB)
P3. CNST 2 UN	= 0B00000H (desc bits)
	OP code = 39

The cancel is always executed regardless of the line status.

## DISABLE ON NO-DATA

### DISABLE ON NO-DATA (DATACOMM CONDCANCEL)

The DISABLE ON NO-DATA BCT request that a conditional cancel of the current I/O operation be performed on the designated file. The format of this BCT is:

```
          BCT          0354
          BUN
P1.  CNST 2 UN  = 08 (specifies (CONDCANCEL)
P2.  ACON      (FIB)
          OP code = 37 (no variants)
```

If no I/O is in progress on the file, or if the I/O in progress is a WC, the MCP ignores the request and the program will be reinstated.

If an I/O is in progress, the line adapter will ignore a conditional cancel operation if it is executing the read portion of a WCRC and data transfer has started, or if input data is being received. If the adapter is executing the write portion of a WCRC, the cancel takes place at the completion of the write.

## DISPLAY UPON (DATACOMM DISPLAY)

The DISPLAY UPON BCT requests that the specified message be written to the designated remote SPO. If the request cannot be met (for instance, the remote SPO is not logged-in), the message is sent to the local SPO. The format of this BCT is:

	BCT	0254
	BUN	
P1.	ACON	(message area, must be mod 2)
P2.	CNST 1 UN	= 4 (specifies DISPLAY)
P3.	CNST 2 UN	= XX (size of message area in bytes)
P4.	CNST 1 UN	= 0 (filler)
P5.	ACON	(remote identifier)

P3 must not specify more than 60 bytes.

P5 points to a 6 UA area containing either the cc/u designation or the adapter ID of the remote SPO to which the data is to be sent.

If the value in the field to which P5 points is the special identifier REMSPO, the DISPLAY is done to the remote SPO which initiated the job regardless of the actual name.

If the indicated device is not a remote SPO (including the initiator for REMSPO requests) or is not logged-in, the request is processed as a DISPLAY (local SPO).

### NOTE

If a job has been executed from a remote SPO all input and output messages which would normally go to the local SPO will be directed to the remote SPO. Thus, it is not necessary to do an DISPLAY UPON REMSPO; a normal DISPLAY will have the same results.



ENABLE

## ENABLE (DATACOMM ENABLE)

The ENABLE BCT requests the enabling of the line associated with the designated file; I/O complete occurs when either ENQ or dial-up occurs as applicable. The format of this BCT is:

	BCT	0354
	BUN	
P1.	CNST 2 UN	= 06 (specifies ENABLE)
P2.	ACON	(FIB)
P3.	ACON	(proceed to label)

If the associated line is a dialed line, the execution of this request will result in a disconnect (hang up). The adapter is then sensitive to a ring signal (dial-up). The program is not suspended while the ENABLE is in process; when both the I/O is completed and the program is in WAIT status, the job is reinstated at the address specified by P3 or at the next instruction if P3 is zero.

## ENABLE EXTENDED (DATACOMM ENABLE EXTENDED; NO COBOL SYNTAX)

The ENABLE EXTENDED BCT requests the enabling of the line associated with the designated file; at the completion of the operation, INTERROGATE and INTERROGATE-END-TEXT operations are performed. The format of this BCT is:

	BCT	0354	
	BUN		
P1.	CNST 2 UN	= 26	(specifies ENABLE EXTENDED)
P2.	ACON	(FIB)	
P3.	ACON	(proceed to label)	
P4.	ACON	(response area)	

P4 addresses a 26-digit area of the following format:

I/O Character Count	- 6 UN
Result Descriptor	- 4 UN
Extended R/D	- 16 UN

The character count must always be one for this operation.

Refer to INTERROGATE and INTERROGATE-END-TEXT in this section.

## FILL

### FILL (DATACOMM FILL)

The FILL BCT requests the initiation of the specified DATA COMM operation without suspension of the requesting program. The format of this BCT is:

	BCT	0354
	BUN	
P1.	CNST 2 UN	= 01 (specifies FILL)
P2.	ACON	(FIB)
P3.	CNST 2 UN	= RTEXPFSD (desc bits)
P4.	CNST 2 UN	= XX (type of operation)
		00 = READ-TO-CONTROL
		02 = WRITE-TO-CONTROL
		03 = WRITE-READ
		04 = WRITE-TRANS-READ
		04 = WRITE-READ-TRANS
P5.	ACON	(PROCEED to label)

The F variant in P3 is optional for WRITE-READ and WRITE-TRANS-READ as the MCP sets the flag; and similarly for the P variant for WRITE-READ-TRANS.

Parameter P4 describes the type of I/O operation requested and the values correspond to those in P1 for the specific I/O requests.

Parameter P5 contains the action label for the request. When both the I/O operation is completed and the program is in a WAIT status, the job is reinstated at the address specified by P5.

## FILL EXTENDED (DATACOMM FILL EXTENDED; NO COBOL SYNTAX)

The FILL EXTENDED BCT requests the initiation of the specified DATA COMM operation without suspension of the requesting program. At the completion of the operation, INTERROGATE and INTERROGATE-END-TEXT operations are performed. The format of this BCT is:

	BCT	0354
	BUN	
P1.	CNST 2 UN	= 21 (specifies FILL EXTENDED)
P2.	ACON	(FIB)
P3.	CNST 2 UN	= RTEXPFSD (desc bits)
P4.	CNST 2 UN	= XX (type of operation)
		00 = READ-TO-CONTROL
		02 = WRITE-TO-CONTROL
		03 = WRITE-READ
		04 = WRITE-TRANS-READ
		04 = WRITE-READ-TRANS
P5.	ACON	(PROCEED to label)
P6.	ACON	(result area)

P6 addresses a 26-digit area of the following format:

I/O Character Count	- 6 UN
Result Descriptor	- 4 UN
Extended R/D	- 16 UN

Refer to INTERROGATE and INTERROGATE-END-TEXT in this section.

## INTERROGATE

### INTERROGATE (DATACOMM INTERROGATE)

The INTERROGATE BCT requests the MCP to translate the 16-bit result descriptor from the previous data communications request into a 16-digit string of ones and zeros corresponding to the bits ON and OFF in the R/D. The descriptor is accessed by the MCP from FIBBSW for the file. The format of this BCT is:

	BCT	0354
	BUN	
P1.	CNST 2 UN	= 07 (specifies INTERROGATE)
P2.	ACON	(FIB)
P3.	ACON	(response area)

The response area must be 16 digits long.

This operation is invoked automatically at the completion of extended I/O requests (types 20-26).

INTERROGATE-END-TEXT

## INTERROGATE-END-TEXT (DATACOMM INTERROGATE ADDRESS)

The INTERROGATE-END-TEXT BCT requests that the MCP return the count of characters transferred to and/or from memory on the previous I/O request (if complete). The format of this BCT is:

	BCT	0354
	BUN	
P1.	CNST 2 UN	= 12 (specifies INTERROGATE ADDRESS)
P2.	ACON	(FIB)
P3.	ACON	(response area)

The response area must be six digits in length.

This operation is invoked automatically at the completion of extended I/O requests (types 20-26).

Indeterminate results will occur if the previous I/O request is still in process when this request is invoked.

B 2000/B 3000/B 4000 MCPVI Programmer's Guide  
Program Interface

READ

READ (DATACOMM READ)

The READ BCT requests a READ-TO-CONTROL operation on the designated file. The program is suspended until the operation is completed. The format of this BCT is:

	BCT	0354
	BUN	
P1.	CNST 2 UN	= 00 (specifies READ)
P2.	ACON	(FIB)
P3.	CNST 2 UN	= 0T0X00SD (desc bits)
P4.	ACON	(zero)
		OP code = 32

## READ EXTENDED (DATACOMM READ EXTENDED; NO COBOL SYNTAX)

The READ EXTENDED BCT requests a READ-TO-CONTROL operation on the designated file, plus INTERROGATE and INTERROGATE-END-TEXT operations. The program is suspended until the operation is completed. The format of this BCT is:

	BCT	0354
	BUN	
P1.	CNST 2 UN	= 20 (specifies READ EXTENDED)
P2.	ACON	(FIB)
P3.	CNST 2 UN	= 0T0X00SD (desc bits)
P4.	ACON	(zero)
P5.	ACON	(result area)
		OP code = 32

P5 addresses a 26-digit area of the following format:

I/O Character Count	- 6 UN
Result Descriptor	- 4 UN
Extended R/D	- 16 UN

Refer to INTERROGATE and INTERROGATE-END-TEXT in this section.



## READY BUFFER

### READY BUFFER (DATACOMM READY BUFFER; NO COBOL SYNTAX)

The READY BUFFER BCT specifies that a stream mode buffer is empty and ready for more data. This operation is executed implicitly by the MCP as READ and WRITE requests are made to the file. Ready must be received by the I/O control within the required time period or data will be lost. The format of this BCT is:

	BCT	0354	
	BUN		
P1.	CNST 2 UN	= 09	(specifies READY BUFFER)
P2.	ACON	(FIB)	

## TRANSLATE-TABLE (DATACOMM TRANSTBL; NO COBOL SYNTAX)

The TRANSLATE-TABLE BCT requests the transfer of data communications translate tables from the MCP to the requestor. The format of this BCT is:

	BCT	0354
	BUN	
P1.	CNST 2 UN	= 13 (specifies TRANSLATE-TABLE)
P2.	ACON	(FIB)
P3.	ACON	(input table area)
P4.	ACON	(output table area)

P2 addresses the FIB for the data communications device. The specific translate table(s) retrieved is dependent on the values of FIBHDW and FIBTRN.

Both P3 and P4 must specify mod 4 addresses. They need not be mod 1000.

P3 or P4 can be zero if the corresponding table is not required.

### NOTE

The MCP does not maintain any non-standard output translate tables. If the FIB specifies non-standard translation the standard translate tables will be loaded.

Refer to FIB (FIBTRN), section 5.

## WAIT

### WAIT [UNTIL] (DATACOMM WAIT)

The WAIT BCT requests suspension of the calling program until the completion of any of the DATA COMM I/O requests currently pending for the program, the occurrence of a CRCLR SEND operation to the waiting program, or (optionally) a specified amount of time has passed, whichever occurs first. The format of this BCT is:

```
          BCT          0354
          BUN
P1.  CNST 2 UN  = 10 (specifies WAIT)
P2.  ACON      (field where time is stored)
```

If a DATA COMM I/O for the program is already complete at the time of the WAIT request, or when an I/O becomes complete, the program is reinstated at the PROCEED TO label of the FILL or ENABLE request associated with the I/O; if no branch were provided, the program resumes at the instruction following the WAIT.

P2 must be zero if WAIT UNTIL is not requested. Otherwise P2 points to a 5 UN field containing the maximum number of seconds to wait. If the time period elapses before any DATA COMM I/O becomes complete, the program is reinstated at the instruction following the WAIT. (See WAIT (DOZE) in this section for further details.)

If another program attempts a CRCLR operation to a WAITing program, the program is reinstated at the instruction following the WAIT.

This BCT is preferable to the DOZE BCT (in system configurations where it can be used) as it requires far less system overhead.

## WRITE (DATACOMM WRITE)

The WRITE BCT requests a WRITE-TO-CONTROL operation to the designated file. The format of this BCT is:

	BCT	0354
	BUN	
P1.	CNST 2 UN	= 02 (specifies WRITE)
P2.	ACON	(FIB)
P3.	CNST 2 UN	= R0EX00SD (desc bits)
P4.	ACON	(zero)

Refer to WRITE-READ and WRITE-READ-TRANS in this section, for other variations on WRITE.

WRITE EXTENDED

WRITE EXTENDED (DATACOMM WRITE EXTENDED; NO COBOL SYNTAX)

The WRITE EXTENDED BCT requests a WRITE-TO-CONTROL operation to the designated file. At the completion of the operation, INTERROGATE and INTERROGATE-END-TEXT operations are performed. The format for this BCT is:

	BCT	0354	
	BUN		
P1.	CNST 2 UN	= 22	(specifies WRITE EXTENDED)
P2.	ACON		(FIB)
P3.	CNST 2 UN	= R0EX00SD	(desc bits)
P4.	ACON		(zero)
P5.	ACON		(response area)

P4 addresses a 26-digit area of the following format:

I/O Character Count	- 6 UN
Result Descriptor	- 4 UN
Extended R/D	- 16 UN

Refer to INTERROGATE and INTERROGATE-END-TEXT in this section.



WRITEREAD EXTENDED

WRITEREAD EXTENDED (DATACOMM WRITEREAD EXTENDED; NO  
COBOL SYNTAX)

The WRITEREAD EXTENDED BCT requests a WRITE-TO-CONTROL/READ-TO-CONTROL operation on the designated file. At the completion of the operation, INTERROGATE AND INTERROGATE-END-TEXT operations are performed. The format of this BCT is:

	BCT	0354	
	BUN		
P1.	CNST 2 UN	= 23	(specifies WRITEREAD EXTENDED)
P2.	ACON		(FIB)
P3.	CNST 2 UN	= RTEXP1SD	(desc bits)
P4.	ACON		(zero)
P5.	ACON		(response area)

P5 addresses a 26-digit area of the following format:

I/O Character Count	- 6 UN
Result Descriptor	- 4 UN
Extended R/D	- 16 UN

Refer to WRITE-READ, INTERROGATE, and INTERROGATE-END-TEXT in this section.

## WRITEREADTRANS (DATACOMM WRITEREADTRANS)

The WRITEREADTRANS BCT requests a WRITE-TO-CONTROL/READ-TRANSPARENT operation on the designated file. The format of is BCT is:

	BCT	0354
	BUN	
P1.	CNST 2 UN	= 05 (specifies WRITEREADTRANS)
P2.	ACON	(FIB)
P3.	CNST 2 UN	= 0T00100D (desc bits)
P4.	ACON	(zero)
		OP code = 34 (note variant on WRITE)



WRITEREADTRANS EXTENDED

WRITEREADTRANS EXTENDED (DATACOMM WRITEREADTRANS  
EXTENDED; NO COBOL SYNTAX)

The WRITEREADTRANS EXTENDED BCT requests a WRITE-TO-CONTROL/READ TRANS-  
PARENT operation to the designated file. When the operation is completed, INTERROGATE and IN-  
TERROGATE-END-TEXT operations are performed. The format of this BCT is:

	BCT	0354	
	BUN		
P1.	CNST 2 UN	= 25	(specifies WRITEREADTRANS EXTENDED)
P2.	ACON	(FIB)	
P3.	CNST 2 UN	= 0T00100D	(desc bits)
P4.	ACON	(zero)	
P5.	ACON	(response area)	

The response area (P5) is a 26-digit field of the following format:

I/O Character Count	- 6 UN
Result Descriptor	- 4 UN
Extended R/D	- 16 UN

Refer to WRITEREADTRANS, INTERROGATE, and INTERROGATE-END-TEXT in this section.

## WRITETRANSREAD (DATACOMM WRITETRANSREAD)

The WRITETRANSREAD BCT requests a WRITE-TRANSPARENT/READ-TO-CONTROL operation on the designated file. The format of this BCT is:

	BCT	0354
	BUN	
P1.	CNST 2 UN	= 04 (specifies WRITETRANSREAD)
P2.	ACON	(FIB)
P3.	CNST 2 UN	= 0T00010D (desc bits)
P4.	ACON	(zero)

WRITETRANSREAD EXTENDED

WRITETRANSREAD EXTENDED (DATACOMM WRITETRANSREAD  
EXTENDED; NO COBOL SYNTAX)

The WRITETRANSREAD BCT requests a WRITE-TRANSPARENT/ READ-TO-CONTROL operation on the designated file. When the operation is completed, INTERROGATE and INTERROGATE-END-TEXT operations are performed. The format of this BCT is:

	BCT	0354	
	BUN		
P1.	CNST 2 UN	= 24	(specifies WRITETRANSREAD EXTENDED)
P2.	ACON		(FIB)
P3.	CNST 2 UN	= 0T00010D	
P4.	ACON		(zero)
P5.	ACON		(response area)

The response area (P5) is a 26-digit field of the following format:

I/O Character Count	- 6 UN
Result Descriptor	- 4 UN
Extended R/D	- 16 UN

Refer to WRITETRANSREAD, INTERROGATE, and INTERROGATE-END-TEXT in this section.

## **MICR BRANCH COMMUNICATES**

This section describes the communicates applicable to MICR files (Reader/Sorter files) not accessing a Reader/Sorter on a DLP or 4A control. Refer to sections 9 and 10 of this manual for a description of the BCTs applicable to a Reader/Sorter on a DLP or 4A control. All communicates are to the same address (0374) and carry one or more parameters. The first is always the address of the Reader/Sorter FIB; the second is a parameter specifying the requested action.

The headings used on these communicates do not always correspond to the equivalent source language syntax. Refer to the appropriate language manual for the specific syntax, (COBOL-68, form number 1108909; BPL, form number 1113735).

## CONTROL 4

### CONTROL 4 (ACTION 4)

The CONTROL 4 BCT requests that the light for the indicated pocket on the Reader/Sorter be lit. The format of this BCT is:

	BCT	0374
	BUN	
P1.	ACON	(FIB)
P2.	CNST 2 UN	= 64 (specifies POCKET LIGHT)
P3.	ACON	(pocket number)

The pocket number field should be 2 UN. Flow must be stopped prior to issuing this request.

## CONTROL 6 (ACTION 6)

The CONTROL 6 BCT requests that the batch counter be advanced on the Reader/Sorter. The format of this BCT is:

	BCT	0374
	BUN	
P1.	ACON	(FIB)
P2.	CNST 2 UN	= 66 (specifies advance counter)

Flow must be stopped prior to issuing this BCT.

B 2000/B 3000/B 4000 MCPVI Programmer's Guide  
Program Interface

READ

**READ (READ)**

The READ BCT requests the transfer of the current document to the program. The format of this BCT is:

	BCT	0374
	BUN	
P1.	ACON	(FIB)
P2.	CNST 2 UN	= 63 (specifies READ)
P3.	ACON	(FLOW STOPPED label)
P4.	ACON	(BATCH TICKET label)

## READ FLOW (READ FLOW)

The READ FLOW BCT requests the initiation of FLOW MODE operation on the indicated Reader/Sorter. The format of this BCT is:

	BCT	0374
	BUN	
P1.	ACON	(FIB)
P2.	CNST 2 UN	= 62 (specifies READ FLOW)
P3.	ACON	(FLOW STOPPED LABEL)
P4.	ACON	(BATCH TICKET ROUTINE)



## SELECT

### SELECT (ACTION 0)

The SELECT BCT requests routing of the current document to the indicated pocket of the Reader/Sorter specified. The format of this BCT is:

	BCT	0374
	BUN	
P1.	ACON	(FIB)
P2.	CNST 2 UN	= 60 (specifies SELECT)
P3.	ACON	(POCKET SELECT variants)
P4.	ACON	(TOO-LATE-TO-SELECT label)

The POCKET SELECT variants are a 4 UN field of the form NNRV where:

NN = Pocket number  
R = 0  
V = 0 = To continue flow  
1 = To stop flow

## OTHER USER PROGRAM INTERFACES

In addition to the specific information passed to the MCP in a BCT instruction, other interface areas are involved during the processing of a BCT. Two particular functions, USE routines and disk/disk pack file OPENS, modify or examine certain program locations. These additional parameters are presented here.

### USE ROUTINES

When certain USE routines are entered, the MCP passes information to the program which helps define the processing state at the time of entry. In all cases, the address to which return is made when the USE routine is exited, is placed in FIBRCW of the first Buffer Status Block following the FIB for which the USE routine was entered.

The following data is passed when a label USE routine is entered:

BASE: +34:1:UN = FIB-IO (value)  
BASE: +35:1:UN = 0 for begin label, 1 for end label  
BASE: +36:1:UN = 0 for file labels, 1 for reel labels

The following data is passed when an I/O error USE routine is entered:

BASE: +34:1:UN = FIBRWT bits :1 and :2  
BASE: +37:3:UN = Reel number (MTP)

### FILE HANDLING

When an existing disk or disk pack file is OPENed input or I/O, the program can request that the MCP present the disk file header (DFH) for the file in memory. This is done by setting certain bit combinations in the first digit of the FIB for that file (FIBST1). The specifications are:

<b>FIBST1 Value</b>	<b>Meaning</b>
:1	Request first 40 digits of header before modification (due to remapping according to record and block size specifications) at BASE: +100.
:4	Request first 40 digits of header after modification at BASE: +100.
:8	Request header addresses at BASE: +140; the number of addresses transferred is determined by the number of areas specification in the DFH. Bit 1 or 4 will also be on.

When a new output disk or disk pack file is CLOSEd WITH LOCK, RELEASE, or CLOBBER, the program can specify that the MCP is to access memory (at BASE: +100) for the first 40 digits of the DFH, rather than use the header maintained outside program bounds. This is specified by the 1-bit in FIBST1.

## MCP-UTILITY INTERFACES

The MCP provides several utility programs as intrinsics. Most of these programs can be user written and employ the standard MCP interface for that intrinsic. It must be understood that the interfaces for the intrinsics defined in the following are subject to change. This sub-section is intended primarily for information.

### Print/Punch Backup

A complete description of backup utilities interfaces can be found in Device Alternates, section 4.

### Log Programs

The names of the user-coded log analysis programs (initiated by the LNR, LNS, and LNM messages) must be RLGOUT, SLGOUT, and MLGOUT, respectively. The 2-digit log number (p1-p9) is inserted at BASE:+32 (p = processor number).

### Disk File LOAD/DUMP

The user-coded LOADMP is invoked in lieu of the MCP intrinsic except when the file name specified in the control syntax is SYSTEM (and /OWN is not specified), or when /MCP is specified.

The parameters passed to the program when it is initiated are shown in table 2-4.

**Table 2-4. Disk File LOAD/DUMP Parameters**

Location	Length	Meaning
BASE: +0	1UN	Execution digit 0 = LOAD 1 = DUMP 2 = ADD 3 = UNLOAD 4 = CHECK 5 = DUMP/CHECK 7 = UNLOAD/CHECK :8 = LOAD for ALLOCATE card (that is, OPEN files I/O)
BASE: +1	1 UN	:2 NEWLIST and LIB = 1, or LIST specified in control request :4 Abort on error :8 Tape FID = SYSTEM
BASE: +2	2 UN	Segments per block for 7-track (1-30)
BASE: +2	1 UN	High order bits of size field :4 DISKCHECK on LOAD/ADD :8 COMPARE on DUMP/UNLOAD
BASE: +4	2 UN	Specified EU for LOAD/ADD

In addition to the parameters passed to program memory, the MCP builds a work disk file for LOADMP. This file, which consists of 100-byte records (20 X 40), contains FIDs and other data needed for the requested operation. The file name is %nn0p0, where nn is the LOADMP mix number and p is the system processor number. The record format is shown in table 2-5.

**Table 2-5. LOADMP Pass File**

<b>Location (byte)</b>	<b>Length</b>	<b>Meaning</b>
0	1 UA	Record code 1 = Disk directory record (DUMP and UNLOAD only) 2 = Identifiers from LOADMP request
1	1 UA	Zero
2	2 UA	Name count (01-16)
4	6 UA	File identifier
10	90 UA	Up to 15 more identifiers

The first identifier of the work file (regardless of record code) is the name of the library tape specified in the control instruction.

Code 1 records contain the names of all permanent disk files except certain MCP files for which library maintenance is forbidden. All but the final record, will be full.

For the CHECK function, the first and only FID entry after the tape identifier, is three NULLS and three blanks; the same entry as for a /// group specifier.

Normal DISPLAY requests from LOADMP are ignored if the LIB option is not set. This can be overridden by setting the 1-bit in P4 of the DISPLAY communicate parameters; further, the 2-bit requests that the MCP eliminate extraneous blanks from the message.

The standard library tape as created by the LOADMP intrinsic is described as follows.

The library file, which can be multi-reel, consists of a single logical tape file. The file is written in odd parity and, for 7-track tape, no translation. Within the file are four types of records, each of different length. As many as 10000 disk files can be contained in a single library tape file.

**NOTE**

Reference is made in the following text to an expanded format of identifiers and data. The expansion is done by moving the data UN to UA, thus inserting an undigit F in every other digit. This is necessary to allow the full character set to be written to 7-track tape.

The FID is always FILE; byte eight of the MFID field contains a code, specifying the library format type (5 for tapes created by MCPVI/MCPV).

The first records on the tape are the directory blocks which contain the names, in order, of the files on the system tape. The record is 1204 bytes long and is shown in table 2-6.

**Table 2-6. LOADMP Type 1 Record**

Location	Length	Meaning
0	1 UA	1 – Record identifier
1	3 UA	Number of file names (001-100) in record
4	2 UA	First file identifier (expanded)
16	1188 UA	Up to 99 more identifiers (expanded)

Sufficient directory blocks are written to contain the names of all the disk files on the library tape.

A File Identifier Record precedes the data for each disk file on the library tape. This record is 28 bytes long and is shown in table 2-7.

**Table 2-7. LOADMP Type 21 Record**

Location	Length	Meaning
0	1 UA	2 – Record identifier
1	1 UA	1 – Subcode
2	2 UA	00
4	12 UA	File identifier (expanded)
16	1 UA	9
17	1 UA	Subsystem specification for file (0-3)
18	10 UA	Number of disk segments in file (expanded)

Following the File Identifier Record for each disk file on the library tape is a File Header Record. The security information is identical to the Security Attribute Storage Area (SASA) described in section 5. The record layout is shown in table 2-8.

**Table 2-8. LOADMP Type 22 Record**

Location	Length	Meaning
0	1 UA	2 – Record identifier
1	1 UA	2 – Subcode
2	2 UA	00
4	840 UA	Disk file header (expanded)
844	50 UA	Security information (expanded)
894	2 UA	Filler

The contents of the disk file is written in successive tape blocks following the File Header Record. Each record, which can vary from 104 to 6004 bytes, contains an integral number of disk segments (expanded format on 7-track tape). The maximum block size is constrained by the user Cold Start LOADMP declaration. Short records are written if the amount of data to be written is less than the maximum block size. (This can occur at the end of a disk area and/or at EOF.)

B 2000/B 3000/B 4000 MCPVI Programmer's Guide  
Program Interface

The third field contains the number of disk segments contained in the tape record. For 7-track system tapes, this will be a value from one to the value specified in the Cold Start LOADMP card (maximum 30). For 9-track system tapes, values from one to twice the value specified in the Cold Start LOADMP card will occur (maximum 60). The Data Block follows the description given in table 2-9.

**Table 2-9. LOADMP Data Block**

<b>Location</b>	<b>Length</b>	<b>Meaning</b>
0	1 UA	3 – Record identifier
1	1 UA	1 to 9 – Sequential block number. (used to detect block sequence errors; cycles from 1 to 9, 1 to 9, and so forth.
2	2 UA	Number of disk segments in this record (01-60).
4	100-6000 UA	File data (7-track expanded, minimum size 200 bytes).

## Disk Pack File LOAD/DUMP

The program PACKUP is used for transferring files from tape or disk to disk pack or from disk pack to tape or disk. Any LOAD or DUMP control syntax specifying /DPK will cause PACKUP to be executed.

The user-coded PACKUP is invoked instead of the MCP intrinsic except when /MCP is specified.

The parameters in table 2-10 are passed to the program when it is initiated.

**Table 2-10. PACKUP Parameters**

Location	Length	Information
BASE: +0	1 UN	0 = LOAD 1 = DUMP 2 = ADD 3 = UNLOAD 4 = CHECK 5 = DUMP/CHECK 7 = UNLOAD/CHECK
BASE: +1	1 UN	:8 Tape name is SYSTEM :4 ABORT option :2 NEWLIST option or /LIST :1 EU number specified on LOAD or ADD
BASE: +2	1 UN	:8 COMPARE option :4 DISKCHECK or LOAD/CHECK
BASE: +2	2 UN	7-track tape record size in sectors (10-30)
BASE: +4	1 UN	:8 Unused :4 Unused :2 Unused :1 Directory present in pass file
BASE: +5	1 UN	1 = LOAD/DPK 4 = DUMP/DPK 5 = LOAD or DUMP/DPKDPK 6 = LOAD or DUMP/DPKDSK 9 = LOAD or DUMP/DSKDPK D = LOAD or DUMP/MTPDPK E = LOAD or DUMP/MTPDSK

A pass file is also created by the MCP. This file contains additional information needed for the PACKUP operation, for example, the IDs of the files on disk or disk pack. The file ID is %nn0p0; nn is the PACKUP mix number and p is the processor number.

The pass file contains two record types, type 1 and type 2. Type 2 records occur first and contain information from the LOAD, DUMP, ADD, or UNLOAD statement. The first type 2 record has the format shown in table 2-11.

**Table 2-11. PACKUP Pass File**

Location (byte)	Length	Meaning
0	1 UA	Record code = 2
1	2 UN	Same as information passed to PACKUP at BASE: +4:2 in memory
2	2 UA	ID count (1-16)
4	6 UA	File ID 1; Tape ID, if operation involves tape; otherwise, value depends on operation
10	6 UA	File ID 2; Pack ID or blank depending on operation
16	84 UA	Up to 14 more file IDs; a masking character in the input is replaced by NULL (@00@

Successive type 2 records, if required, have the same format except that byte 1 contains a blank and all 16 file ID slots are used for file IDs listed in the input statement.

A type 1 record, if present, contains file IDs either from disk or disk pack. The format is the same as the format for type 2 records.

The library tape file consists of a single logical tape file. Within the file are four types of records, each of different length.

The MFID of the tape is specified in the DUMP or UNLOAD command. The FID is always FILE. PACKUP creates a type 6 tape (6 in byte eight of the MFID field in the tape label).

Reference is made, in the following paragraphs to an expanded format. This is done to accommodate 7-track library tapes and is effected by moving the data UN to UA, thus, inserting a @F@ in every other digit.

On 7-track tape, all information is expanded. On 9-track tape, all information except the file data is expanded. The following record descriptions are in unexpanded form.

The first record in the tape file must be a directory block. One or more directory blocks can exist. Each file DUMPed to tape has an entry in the directory block. The directory block format is shown in table 2-12.

**Table 2-12. PACKUP Type 1 Record**

Position	Length	Data
0	1 UA	1
1	3 UA	Number of file IDs in this block (100 maximum)
4	1200 UA	Up to 100 file IDs (expanded)



The sequence of the files on tape is the same as the sequence of the file IDs in the directory blocks.

A file dumped to tape is headed by two control block records. The first control block indicates the pack ID of the file. The other control block is a copy of the pack file header. Following the control blocks is zero or more blocks of file data. This sequence is repeated for every file DUMPed to tape. These three record types are described in tables 2-13, 2-14, and 2-15.

**Table 2-13. Control Block 1**

Position	Length	Data
0	4 UA	2100
4	12 UA	Pack ID (expanded)
16	12 UA	File ID (expanded)
28	10 UA	Total size of disk pack file in sectors
38	2 UA	Filler

**Table 2-14. Control Block 2**

Position	Length	Data
0	4 UA	2200
4	840 UA	Disk pack file header (expanded)
844	50 UA	Security information (expanded)
894	2 UA	Filler

The security information is the information which is stored in the SASA (refer to section 5).

**Table 2-15. Data Block**

Position	Length	Data
0	1 UA	3
2	1 UA	Sequential block number, 1-9, cyclic
4	2 UA	Number of disk pack sectors contained in this block (maximum 30)
8	5400 UA (max)	File data

## Automatic System Recovery Program

The SYSUP program, used with the automatic Halt/Load option (AUHL) and the System Up option (SYUP), allows the user to automatically recreate the system environment, including all user application programs, following any Halt/Load. This can be especially advantageous for a system working in an unattended on-line environment.

When the AUHL option is set, the operating system executes the SYSUP program at the completion of a Halt/Load resulting from system failure. If the SYUP option is set, SYSUP will be executed automatically at the end of any Halt/Load. However, the program may be executed at any time by program

or operator request. The user-coded SYSUP is invoked instead of the MCP intrinsic unless MCP is specified. Thus, the program may be used not only to restart the system in case of disaster, but also to bring up the system initially.

When executed automatically, the SYSUP program is passed four digits of information beginning at the base-relative address 32. The first digit will have the following values:

Digit	Meaning
0	Manual Halt/Load
1	SPO Halt/Load
4	Cold Start
5	Warm Start
9	Auto Halt/Load

The first digit may be used as a validity flag and also defines the type of Halt/Load. When the program is executed by the operator, other values may be input to indicate specific actions which must be taken. The next two digits are the mix number of the job which was actually executing at the moment the failure occurred. A user-written SYSUP program might use this information (presuming it can determine the name of the program associated with the mix number) to avoid restarting the specific program responsible for the system failure. The fourth digit, supplied at address 35, is the processor number on which SYSUP was executed. This is useful for shared systems where each system uses different software systems.

If a user-coded SYSUP program is executed as the result of an Auto Halt/Load, the program should execute DMPANL by ZIPing a PM1 request, then wait for DMPANL to complete before attempting to recreate the system environment. This ensures that even if the failure happens again, information about it will be preserved. Also, a full dump, or at least a dump specifying the relevant mix number, should be done because a TBL or MCP dump may not contain sufficient information to determine the cause of the failure. Of course, it is always possible to do two dumps: a short one for immediate analysis of the problem, and a full dump that is left in backup format and not printed unless it proves necessary.

The following program is an example of a SYSUP program written in BPL designed to restart an unattended time-sharing system.

```

SYSUP:
BEGIN
  CONTROL OP 94700;
  ALPHA  PARM1 (6)
        ,PARM2 (6);
  INTEGER MCP_FUNCTION (1)
        ,MCP_FALLT_MIXNO (2);
  DEFINE WAIT_FOR_EQJ(WHC) =
    BEGIN UNSEGMENTED
      INTEGER  STILL_RUNNING (2) := 0;
      DO BEGIN
        DOZE 2;
        STILL_RUNNING := MIXID WHC;
      END
      UNTIL STILL_RUNNING = 0;
    END;#
  ,LOAD_JOBDECK_FROM_PACK (FILE_NAME) =

```





**Table 2-16. Sort Parameter File**

<b>Record</b>	<b>Size</b>	<b>Contents</b>
1	200 UN	Input file FIB
2	400 UN	Input file label area
3	200 UN	Output file FIB
4	400 UN	Output file label area
5	12-600 UN	File and key specifications

## DMPALL

The MCP provides syntax to initiate the DMPALL media conversion program through the PERFORM (PFM) and GENERATE (GEN) control statements. The statement itself is placed into a disk work file without modification or syntax checking, except that any control text identifier (? or CC) is not included. The file can contain up to 20 areas of one segment each. The work file is named %nn0p0 (as for LOADMP) where nn is the program mix number and p is the processor number. In addition @FF@ is placed in BASE:+6:2:UN when DMPALL is initiated.

## MCP-COMPILER INTERFACES

Compilers or generators are any programs which are initiated by a COMPILE control statement. They are not generally user-written, but the special interface functions which the MCP provides for compilers can be utilized by most user programs. The following points are a few of the things which are unique to the treatment of compilers.

The CORE-SIZE (MEMORY) communicate will return the EU to which the code file is to be generated, the setting of the WRKP option, and flags specifying whether COMPILE ON <pack-id> or COMPILE for SYNTAX is requested.

When a compiler goes to EOJ, the MCP accesses IX1 in the compiler. If this field contains a non-zero value, syntax errors are assumed. This affects the EOJ message for the processing and card reader flushing operations. Further, at EOJ the MCP takes certain actions in respect of the code file. The compiler names the file "CODE" and defines it as 100-byte records, one record per block, and one disk area. The compiler should terminate with the CODE file still OPEN. If the compiler terminates normally (no DS or syntax errors) and the code file is requested (not COMPILE for SYNTAX), the MCP renames the file with the ID specified in the compile card.

Compiler work file OPEN and CLOSE records are not logged in the RUN log. For compile-and-go operations the requisite short schedule record is produced. Also, the SPO log will not contain compiler OPEN and CLOSE messages even if the OPEN and CLOSE options are set.

During execution, the program name is displayed in two parts: <program-id>/<compiler-id> where <program-id> is the name associated with the generated object code file.

For compile-and-go operations, necessary actions such as scheduling, file equating and return of code file space are handled.

## SECTION 3

# INTERPROGRAM COMMUNICATION

Among the many benefits of multiprogramming is the ability to have multiple programs in the mix performing different functions of the same task. A common example of such a system is an on-line inquiry and update operation where one program controls the data communications network, another program provides data management functions, and still other programs provide the actual inquiry and update functions. By structuring the system in this manner rather than employing one large program to accomplish all tasks, a number of benefits accrue. These include a number of design and implementation benefits such as automatic modularity, simultaneity of development, and maintenance ease. Most importantly, a greater transaction rate can be achieved by overlapping the processing of transactions.

Necessary to an efficient implementation of such decentralized systems, is an effective and flexible interprocess control and communication mechanism. The MCP provides a number of tools for such control, notably the Core-to-Core (CRCR) and Storage Queue (STOQUE) data transfer mechanisms. By using these mechanisms, the user can achieve many benefits of independently compiled modules without the need for a linkage function. Further, because each program in the system is an independent entity, no restrictions are placed on the locations of the programs within memory, as is the case with some implementations of sub-tasking.

Further still, the CRCR and STOQUE functions provide a wide range of facilities permitting application to a great variety of problems with minimal programming effort. Inter-process control, program synchronization, data transfer and transaction queuing are all provided within these MCP functions.

### **BASIC CONCEPTS**

Often, when multiple processes (programs) are involved in accomplishing a task, one process drives one or more of the other programs. The basic requirements of interprocess control such as initiation, synchronization, and data transfer are provided by MCP facilities. These functions reside in the MCP rather than in the programs to insure the integrity of the mix. Thus, while data can be passed from one program to another, it must be done by the MCP, according to a specific set of rules, and only when programs are willing to accept such data.

Three basic types of relationships are defined for processes: 1) independent, 2) synchronous, 3) asynchronous.

An independent process which is initiated by another program, can have parameters passed to it, and runs independent of the parent process. Independent processes allow the user to achieve the overlap of independent functions within a programming system. A typical example of an independent process mechanism, occurs when program A which reads file X, creates a master file and an exception file. Both files are used to create reports. Program A can initiate an independent process (program B) to create one report while A creates the other. Further, parameters might be passed to B when initiated, describing any special actions or exception conditions.

Independent processes are generally initiated by programmatic passage (ZIP) of an EXECUTE request to the MCP.

A synchronous process runs serially with a parent process and acts as a subroutine within the parent. First one process executes to a given point, then passes control (and parameters) to the other. The secondary process executes to a point, then returns control to the parent. The programs never execute in parallel.

Synchronous processing provides the capability to have programs act as subroutines to other programs without the need for any physical linkage between the processes. The sub-program can be used by several parent processes at the same time, thus obviating the need for a copy of the routine to be coded in each master process. Further, the individual processes can be coded in different languages, thus allowing (for example) a FORTRAN program to use the data manipulation capabilities of COBOL, or a COBOL program to employ the computational strength of FORTRAN. As a typical example of synchronous processing, consider a COBOL program which accesses various files, performs various update and data manipulation operations, and at some point requires the results of an extensive calculation. Thus, when the COBOL process requires those results, raw data can be passed to the FORTRAN program and that program can perform the necessary operations. Meanwhile, the COBOL program is suspended until the answer becomes available. The control of synchronous processes is generally accomplished through the CRCR function of the MCP. This function provides the necessary facilities for data transfer and process synchronization.

An asynchronous process runs simultaneously with the parent, each overlapping execution with the other. Parameters can be passed between the processes, and interlocking (synchronization) can occur when necessary.

Asynchronous processing provides the major benefit of an increased overlap of processing for a given task. Multi-programming is often understood as a means of achieving overlap in the processing of multiple independent programming systems. The mechanisms of asynchronous processing afford this throughput increase to individual systems as well.

Both the CRCR and Storage Queue functions provide the necessary facilities to accomplish asynchronous interprocess control. Each function, however, has individual strengths and the choice of mechanisms is dependent on the application and the degree of sophistication required.

## **INTERPROCESS CONTROL TECHNIQUES**

This sub-section describes the mechanisms provided by the MCP for interprocess control. A more complete technical discussion of these functions is provided in this section under Interprocess Control Mechanisms.

The MCP provides several capabilities which can be used to initiate independent processes. The primary mechanism is the ZIP operation which allows a program to pass control information to the MCP for processing. Typically, this would be an EXECUTE request; parameters can be passed through a VALUE or INSERT program parameter associated with the execution request.

The CRCR function is the primary means of accomplishing synchronous interprocess control. While similar process interlocking could be done with STOQUE, CRCR is more efficient when transactions are handled in a strictly serial mode with no overlap of processing.

The CRCR transfer functions are performed by the CRCR extension module of the MCP. If these facilities are to be used, the module must be loaded into memory by setting the CRCR system option.

The CRCR module has the basic function of transferring information from a data area in one program to a data area in another program. The data is transferred directly, without using any intermediate storage area, and only when both parties to the transfer acknowledge that they are prepared for the exchange.

Programs can execute two types of calls on the CRCR module. A program can request that data be sent to another program (send), or a program can acknowledge acceptance of data from another job (receive).

B 2000/B 3000/B 4000 MCPVI Programmer's Guide  
Interprogram Communication

The CRCR function requires the synchronization of CRCR transfers signifying that data is moved from the sender to the receiver only when both processes are simultaneously ready to enter into the transaction. Further, at least one of the parties to the transfer must be willing to wait until the other is prepared to complete the transfer. This synchronization of the programs is also known as interlocking. The process of synchronizing the programs and accomplishing the data transfer is known as a CRCR operation as a hook up.

The two basic requests which can be made of the CRCR function can be illustrated by the COBOL FILL...INTO and FILL...FROM constructs which request, respectively, that data be sent to or received from another program.

To send data, the COBOL language syntax is:

```
FILL  data-name  INTO  program-name  [PROCEED TO paragraph-name].
```

The data-name references the data to be sent to a program concurrently operating in a mix whose 6-character name is specified by the program-name. Another possibility is to specify a program-name of blanks which implies that any program is a candidate for the hook up (Global request). When the data transfer is completed, the program resumes at the next instruction. The optional PROCEED TO clause can be used if the sending program wishes to continue processing (at paragraph-name) when the receiving program is not ready for the data transfer. If this clause is omitted, the sender is suspended until the receiver is prepared to accept the data. The PROCEED TO clause is generally omitted for synchronous operations.

To receive data, the COBOL language syntax is:

```
FILL  data-name  FROM  program-name  [PROCEED TO paragraph-name].
```

This format indicates the readiness of the requestor to receive data. The data is to be transferred from the program-name data field and placed in data-name. The PROCEED TO clause functions in the same way as the corresponding option in the FILL...INTO format.

As an example of a strictly synchronous interprocess communication problem, consider a COBOL program employing a FORTRAN program for calculations as described above.

COBOL	FORTRAN
01 SEND-AREA.	DIMENSION ITERM (10)
03 TERM1 PIC S9(7) COMP.	INTEGER ANSR
03 TERM2 PIC S9(7) COMP.	. . .
. . .	
03 TERM10 PIC S9(7) COMP.	10 CALL FILL (ITERM, 40,
MOVE ... TO TERM1.	6HCOABPRG)
. . .	{process}
	. . .
FILL SEND-AREA INTO "FORCAL".	CALL SEND (ANSR, 4,
	6HCOBPRG)
FILL RECV-AREA FROM "FORCAL".	GO TO 10



B 2000/B 3000/B 4000 MCPVI Programmer's Guide  
Interprogram Communication

The COBOL program (COBPRG) builds the terms to be used in the calculation. The FILL...INTO statement causes SEND-AREA to be passed to the FORTRAN program (FORCAL) in array ITERM. (CALL FILL is the FORTRAN syntax for a receive request.)

(The COBOL data areas are declared S9(7) COMP. Assume that the integer precision in the FORTRAN program is 7.)

As another example of synchronous interprocess communications, consider a COBOL program employing a BPL program as a scanner for free format input. As each record is accessed by the COBOL program, the relevant portion is passed to the BPL program which scans and reformats the data and returns fixed length results.

The structures of the programs might be:

<pre> COBPRG. 01 SEND AREA.    03 FIXED-PART PIC X(10).    03 FREE-PART PIC X(70).  01 RECV-AREA.    03 WORD PIC X(10) OCCURS 10       TIMES INDEXED BY WORD-IND.       . . .  GETRECORD.   READ FILEA INTO SENDAREA.   FILL FREE-PART INTO "BPLSCN".   FILL RECV-AREA FROM "BPLSCN".  PROCESS.   MOVE WORD (WORD-IND) ...   . . .    GO TO GETRECORD.         </pre>	<pre> BPLSCN. ALPHA RECVAREA (70),   SENDAREA (100);  . . .  GETRECORD:   FILL IN RECVAREA "COBPRG"; PROCESS:   SCAN ...;  . . .    FILL OUT SENDAREA "COBPRG";  GC GETRECORD:         </pre>
---	---

(FILL IN... is the BPL syntax for data receiving; FILL OUT... is the data sending construct.)

Other designs are also possible including one in which the BPL program does the I/O as well as the reformatting. In this case, the program mechanism might be:

<pre> WAKE-UP.   FILL DUMM INTO "BPLSCN". GET-DATA.   FILL RECV-AREA FROM "BPLSCN". PROCESS.   MOVE WORD (WORD-IND) ...   .   .   .   GO TO WAKE-UP.         </pre>	<pre> WAITFORNEXT:   FILL IN DUMMY "COBPRG";   READ FILEA ...;   SCAN ...   .   .   .   FILL OUT SENDAREA "COBPRG";   GO TO WAITFORNEXT;         </pre>
---	---

The FILL...INTO at WAKE-UP is used strictly as a means to interlock with BPLSCN and to alert the program that a new record is to be obtained. It is possible to have multiple programs using BPLSCN simultaneously (for example, by making the BPLSCN FILL's global requests or by passing the program name with the COBOL FILL INTO).

Observe that asynchronous communications yields considerable benefits for the above example. The following sub-section examines this example further.

## ASYNCHRONOUS PROCESSES

Both CRCR and STOQUE provide tools for efficient asynchronous process control; each mechanism has applicational advantages. The two mechanisms can be distinguished by the fact that CRCR provides synchronized data transfer while STOQUE permits completely asynchronous exchanges. Programs using CRCR can process any given transaction asynchronously but must be synchronized at the time of data transfer. The latter is not necessary when using STOQUE. In addition, each mechanism has a number of other unique characteristics.

When CRCR is used for asynchronous processing, the PROCEED TO... clause of the FILL...INTO and FILL...FROM constructs are often employed. This facility allows a process to pursue other activities while the other program completes processing preparatory to the hook up. This capability is important in applications requiring a handler program. Such a program can be servicing many users (peripheral devices and/or programs) and cannot wait for a hook up with one program while other users require servicing. At a later time the CRCR request can be repeated; this process of executing the PROCEED TO... clause and later reinitiating the request is called buzzing. Since CRCR requires that the data transfer be synchronized, only one of the processes involved can specify the PROCEED TO... clause for a given transfer; otherwise a hook up would never occur. Thus, at times, one process must be waiting for the other to catch up.

The CRCR requests have another function which is sometimes valuable in asynchronous processing, usually in conjunction with the normal data transfer facility. If a program attempts to FILL...INTO or FILL...FROM another program and no process of the specified name is prepared to receive or send data, but a program of that name is either a WAIT or SLEEP status, the WAITing/SLEEPing program is marked ready-to-run. Thus, non-Global CRCR operations can be used to wake up another program. (A WAIT [UNTIL <time>] or WAIT <time> request, respectively.)

The STOQUE functions of the MCP are performed by the STOQUE extension module of the MCP. If these facilities are used, they must be loaded into memory by setting the STOQ system option.

The STOQUE module performs the basic functions of transferring data from a data area in a program to an external memory buffer and retrieving that data upon request. The mechanism can be used simultaneously by any number of programs as a means to transfer data between processes, or even as temporary storage for a single process. The data elements placed into the memory buffer are organized into one or more program-independent, symbolically-named lists called storage queues (STOQUE).

The STOQUE mechanism differs significantly from CRCR in that no synchronization of the sending and receiving programs is required. This is due to the fact that STOQUE does not transfer data directly from one program to another, but instead stores the message in an external memory area until requested. Therefore, multiple transactions can be in the storage queue simultaneously, thus, the use of STOQUE permits the complete overlap of processing between programs, with no necessity to interlock for each transaction.

Programs can execute three types of calls on the STOQUE module: store data, retrieve data, and queue inquiry. Both the storage and retrieval functions have important variations which permit the mechanism

B 2000/B 3000/B 4000 MCPVI Programmer's Guide  
Interprogram Communication

to be used in many ways. Rather than permitting only simple queuing of data, STOQUE allows elements to be added to or removed from a list at either the top or bottom yielding the benefits of both first-in first-out (FIFO) and last-in first-out (LIFO) mechanisms. In addition, queues can optionally be given a substructure to allow priority handling of data elements by associating a name with the individual elements of a list in addition to the individual name of the queue.

The queue inquiry function provides a rapid means of determining the size of a list without disturbing any elements.

All requests made of the STOQUE function, reference an area in the user program called the STOQUE Parameter Block. This program-maintained area contains the information needed by STOQUE to control the data elements, and is described as follows:

QUEUE NAME	PIC X(6).	Identifies individual queue
ENTRY NAME LENGTH	PIC 9(2) COMP.	Name LGH in bytes (00=NULL)
ENTRY NAME	PIC X(NN).	Entry name (optional)
ENTRY DATA LENGTH	PIC 9(4) COMP.	Data LGH in bytes (0000=NULL)
ENTRY DATA	PIC X(NNNN).	Data field (optional)

The queue name identifies the programmatically assigned symbolic name of the list to which the request pertains. The entry name length field specifies the size of the optional entry name field. If present, the entry name specifies the name associated with the individual queue entry. This name can be used to provide a substructure to a list and provides the means to access data elements which are at other than the top or bottom of the queue. The entry data length field specifies the size of the entry data area which contains the transaction to be accessed for a storage request or is the area to which data is placed in a retrieval operation. The entry data length field functions as a response area for a queue inquiry request. The entry data field, however, is not applicable to a queue inquiry request. The Stoque Parameter Block must begin on a mod-4 address.

The following paragraphs provide a brief description of the STOQUE requests as illustrated by the COBOL language syntax.

To store data in a STOQUE, the COBOL syntax is:

$$\text{FILL data-name INTO } \left\{ \begin{array}{c} \text{TOP} \\ \text{BOTTOM} \end{array} \right\} \text{ [PROCEED TO paragraph-name].}$$

The keyword INTO specifies that the request is for data storage. If the data element is to be queued at the head of the queue, TOP is specified; BOTTOM indicates that the entry is to be queued at the base of the list.

The data-name references the STOQUE Parameter Block described above. This block contains the data to be stored, and information needed by STOQUE to identify the queue and control the entry.

When the request is completed, the program is resumed at the next instruction.

If sufficient space is not available for the storage request, the sending program is suspended until space becomes available, unless the optional PROCEED TO clause is specified. In that case, the program does not WAIT, but resumes at the paragraph-name specified.

The COBOL syntax for data retrieval from STOQUE is:

FILL data-name FROM { TOP  
BOTTOM } [PROCEED TO paragraph-name].

The keyword FROM indicates that the request is for data retrieval. The TOP and BOTTOM specifications indicate whether data is to be retrieved from the head or base of the queue, respectively.

The data-name references the STOQUE Parameter Block which contains the storage space for the message and the information needed by STOQUE to identify the queue entry desired.

Retrieval of an element can be accomplished by specifying only the queue name (which accesses the first entry at the head or base as required), or by specifying both the queue name and an entry name (which accesses the first entry of the head or base meeting the name constraint). The latter permits entry name group specifications. For example, in a queue containing elements named A, AB, B, and ABC, entries A, AB, and ABC, are all candidates for a retrieval request for A, while both AB and ABC can be accessed only by a specific request for B or by omitting the entry name from the retrieval request. When the request is completed, the program is resumed at the next instruction.

If the designated queue is empty or no individual entry satisfies any specified name constraint, the program is suspended until the desired element is placed into the queue, unless the PROCEED TO clause is specified. In that case, the program does not WAIT, but resumes at the paragraph-name specified.

The inquiry (POLL) construct allows a program to determine the number of entries in a queue. The COBOL syntax is:

FILL data-name FROM POLL.

The data-name designates the STOQUE Parameter Block which contains the queue name, a length field, optionally the entry name, and the entry data length field which functions as a response area. If the entry name is omitted, STOQUE provides the count of all entries in the queue; if the name is specified, the count gives the number of entries of that name or name group only. A response of zero means the queue or the designated portion of the queue is empty.

## COMPARISON OF MECHANISMS

As both CRCR and STOQUE provide tools for efficient asynchronous process control, understanding of the relative merits of each method is necessary. There are several reasons for choosing one mechanism or the other.

Since a CRCR hook up requires less MCP processing than a STOQUE storage/retrieval pair, in most cases it provides faster response than STOQUE when data transfer is the primary function required. In a number of important instances, however, STOQUE can provide greater throughput, particularly in time-dependent applications or in those systems where programs can take advantage of the complete overlapping of processing available with STOQUE. Certain characteristics of each mechanism can affect the throughput rate. For example, buzzing can generate a significant amount of processing overhead if this operation is required frequently. While the processing requirements for any single unsuccessful request is quite similar for either CRCR or STOQUE, each has characteristics which can be

used to reduce such expenditures. For CRCR, the WAKE-UP and Global FILL capabilities can often be used to minimize buzzing. Further, the WAKE-UP facility, which is present only in CRCR, provides excellent responsiveness in certain applications, notably data communications. For STOQUE, the buffering facilities significantly reduce buzzing if the data rate is high enough to maintain multiple elements in a queue. Because indiscriminate buzzing is costly, applications must be examined carefully to determine which mechanism would yield the least number of unsuccessful data transfer requests.

While STOQUE yields a throughput increase in certain instances, this function has been implemented primarily to provide the user with a greater range of data manipulation capabilities including LIFO/FIFO queuing, transaction buffering, and priority handling of messages. The inclusion of these functions in the MCP provides the programmer with a number of powerful tools with little coding effort required.

Both CRCR and STOQUE may be performed by a program executing in the timesharing area.

## EXAMPLES OF CRCR AND STOQUE PROCESSING

The following examples show how CRCR and STOQUE can be applied.

### Example 1:

Many cases of simple asynchronous data transfer are handled by the CRCR function. These include most cases of unidirectional data flow. The second example in Interprocess Control Techniques on synchronous control can be reformulated as a typical example of this type of asynchronous processing. The program structures could be:

```
COBPRG.                                BPLSCN.
GET DATA.A.                            GET CARD:
  FILL RECV-AREA FROM "BPLSCN".        READ FILEA ...;
PROCESS.                                SCAN ...
  MOVE ...                               . . .
. . .                                    FILL OUT SENDAREA "COBPRG";
GO TO GET-DATA                          GO GETCARD;
```

Thus, the processing of one card by the COBOL program overlaps the reading and scanning of the next record by BPLSCN. Note that this example would not benefit from the use of STOQUE. If COBPRG processes faster than BPLSCN, a maximum of one item could be in the queue at any time. Conversely, if BPLSCN processed faster than COBPRG, the queue would be quickly filled and processing would be limited by the rate of COBPRG.

### Example 2:

Another example of the use of CRCR can be seen in a handler program. A common example of such a program would be a data communications network handler which receives messages from remote terminals, passes the transactions to programs, and returns data to the terminals. Such a program services multiple terminals and one or more programs. Consider a data communications handler controlling a network of remote teletypes (one per line) and an equivalent number of programs, one program associated with each line. The following skeleton program might be used to control the communications system. For simplification, each input from a terminal, one line of response is always generated and messages are sent from the programs only when input has been received from a terminal.

B 2000/B 3000/B 4000 MCPVI Programmer's Guide  
Interprogram Communication

```
01 RECV-AREA.
   03 WHO-FOR PIC 99.
   03 OUTPUT-DATA PIC X(70).

01 WAIT-PROG-FLAGS. .
   03 WAIT-OUTPUT-FLAGS PIC 9(20) COMP.
   03 WAIT-PROG REDEFINES WAIT-OUTPUT-FLAGS PIC 9 COMP OCCURS
     20 TIMES.

. . .

{housekeeping code}

ANY-MESSAGE.
  IF WAIT-OUTPUT-FLAGS = ALL 0 GO TO WAITER.
  FILL RECV-AREA FROM " " PROCEED TO WAITER.
  MOVE 0 TO WAIT-PROG (WHO-FOR).
  GO TO LINE1, LINE2-FILL... DEPENDING ON WHO-FOR.

WAITER.
  WAIT.
  GO TO ANY-MESSAGE.

LINE1-ENTERED-INPUT.
  READ LINE1.
  {process input}
  FILL LINE1-RECORD INTO "LINE1P" PROCEED TO CHECK-IF-PROGRAM-DOWN.
  IF WAIT-OUTPUT-FLAGS = 0 MOVE 1 TO WAIT-PROG(1)
  GO TO WAITER.
  MOVE 1 TO WAIT-PROG(1).
  GO TO ANY-MESSAGE.

LINE2-ENTERED-INPUT.
  . . .

LINE1-FILL.
  MOVE OUTPUT-DATA TO LINE1-RECORD.
  FILL LINE1 WITH WRITE-READ PROCEED TO LINE1-ENTERED-INPUT.
  GO TO ANY-MESSAGE.

LINE2-FILL.
  . . .
```

The logic of the programs would be:

```
GET-TRANSACTION.
  FILL RECV-AREA FROM "HANDLR".
  . . .

  {process transaction}
  FILL SEND-AREA INTO "HANDLR".
  GO TO GET-TRANSACTION.
```

The system illustrates several features of CRCR as used for asynchronous process control. First, the FILL...FROM in paragraph ANY-MESSAGE is a Global FILL. Use of this format makes it unnecessary for the handler to request data of each program individually. Rather the sender supplies the correct value in WHO-FOR so the handler can associate the message with the proper terminal. Second, the WAIT statement at WAITER causes the handler to be suspended temporarily either until one of the terminals completes a request or one of the programs attempts to send data by way of CRCR. In the former case, the handler resumes operation at any of several places depending on the line and operation involved. In the latter case, the program resumes immediately after the WAIT statement and a FILL...FROM request can be initiated. Both of these features are used to reduce buzzing (the WAIT-PROG-FLAGS are used for the same purpose). Third, the FILL...INTO operations must always be successful according to the definition of the system. If the operation does not succeed, the program has not executed a FILL...FROM because of an error in the process or lack of control since the FILL...INTO request. This latter case is unlikely, but possible, and must be allowed for in the handler.

Note again, that STOQUE would not improve throughput for this system. Only one transaction for a given line and program is present at any time. Consequently, CRCR yields the maximum overlap possible to the process, and because of the CRCR WAKE-UP capability, incurs very little buzzing overhead. While STOQUE can provide many important benefits to data communications systems, these are usually in the area of convenience rather than throughput. An important exception to this is considered in the following example.

**Example 3:**

As in the first example, consider that in certain applications, several input records must be collected for a given transaction. Frequently, the processing rates of the programs vary relative to each other during certain periods.

In example 1, COBPRG might require three input records from BPLSCN before the transaction can be completed. During the time the records are being collected, COBPRG processes faster than BPLSCN; the reverse might be true after the transaction has been collected. When processing occurs in this way, throughput is improved if BPLSCN inputs some or all of the records needed for the next transaction while COBPRG processes the current one. While STOQUE could be used for this case, a simpler and less costly solution would be to declare the number of card reader buffers in BPLSCN to be equal to the number of records needed for a transaction. In some applications, however, the number of input records needed for a transaction can vary considerably or can even exceed the maximum number of buffers (10) supported by the MCP. For such cases STOQUE can be used to achieve the affect of a variable number of buffers. Note that throughput improvements can occur only if the input and processing rates vary, and the data rate is high enough so that either program would need to wait for the other if CRCR were used. STOQUE causes the programs to be unconcerned with the problems of interlocking, buffering, and queuing of records.

While STOQUE can be used for applications such as this, more commonly, STOQUE would be employed for more complex handler systems. The next two examples present throughput considerations

which would indicate STOQUE as the mechanism of choice. Following the examples is a discussion of some of the other STOQUE mechanisms which are valuable for many applications.

STOQUE can be important in data communications applications although usually not for reasons of throughput. For STOQUE to produce better throughput responsiveness than CRCR, the system must possess certain characteristics.

First, the data rate must be high; otherwise, the STOQUE buffering capability is not needed. Second, and most important, the programs in the system must be able to completely overlap operations with each other. Consequently, all programs, especially the handler, must be able to lose control during the processing of a transaction. (For working programs, control is usually lost whenever a data communication I/O operation is completed and the higher priority handler can be given control, or when the program must wait for the own I/Os to be completed. For the handler, this loss of control might occur when transactions are audited and the program must wait for an I/O operation to be completed.) This consideration can be seen in the following example.

Example 4:

Consider a data communications handler passing input transactions to various working programs including program A. While A is processing transaction, another message is received. Since the handler has higher processing priority, it is given control and begins processing. If CRCR is used, the handler must hold (queue) the second transaction because program A is not yet ready. (This implies that STOQUE could be useful to relieve the handler of this responsibility.) Then, the handler might write the transaction to an audit file and have to wait for the completion of this I/O. During this interval, A could be given control and complete the processing of transaction, following which A would initiate a FILL...INTO request (the response for the current transaction). Neither A nor the handler would be working. If STOQUE is used, A could release the response, retrieve and begin the processing of the next transaction immediately, completely overlapping processing with the handler. (Note that processing can begin before auditing is completed. This may not be desirable in all cases.)

Example four illustrates both considerations stated above; the data rate is high enough to require transaction queuing and the working programs can gain control to take advantage of the buffering feature.

However, if the latter does not occur, the handler is relieved of many of the responsibilities of queuing and problem management, which offset the time and memory costs of STOQUE. The WAKE-UP facility of CRCR is not present here. Thus, when transaction volume is too low to keep either the handler or the working program busy, responsiveness is slightly less than that of a CRCR mechanism. For this reason, the response from the working program to the handler might be accomplished through CRCR rather than STOQUE.



B 2000/B 3000/B 4000 MCPVI Programmer's Guide  
Interprogram Communication

A typical organization of the working program/handler combination using only STOQUE might be:

```

01 STQIN.
  03 NAMEIN PIC X(6) VA "MESSAG".
  . . .

01 STQOUT.
  03 NAMEOUT PIC X(6) VA "RESPON".
  . . .

NEXT-TRAN.
  FILL STQIN FROM TOP.
  {process transaction}

  FILL STQOUT INTO BOTTOM.
  GO TO NEXT-TRAN.

01 STQBLKIN.
  03 NAMEIN PIC X(6) VA
    "RESPON".
  . . .

01 STQBLKOUT.
  03 NAMEOUT PIC X(6) VA
    "MESSAG".
  . . .

INIT-LINES.
  {initialize lines}
  . . .

WAITER.
  WAIT UNTIL SOME-TIME.

ANY-RESPONSE.
  FILL STQBLKIN FROM TOP
  PROCEED TO WAITER.
  {process response,
  initiate next request}
  GO TO ANY-RESPONSE.

LINEA-RESPONDED.
  READ LINEA.
  {process input}
  FILL STQBLKOUT INTO
  BOTTOM PROCEED TO
  NDRROOM.
  WRITE AUDIT-TAPE.
  GO TO ANY-RESPONSE.

LINEB-RESPONDED.
  . . .

```

Example 5:

A Reader/Sorter handler provides another example of a handler program. Such a program typically reads a number of documents from the time-dependent Reader/Sorter and passes them to another program for more complete processing. The STOQUE buffering facilities can provide significant improvements in throughput by providing a place for data when the processing program is not ready for a hook up with the handler. This reduces the number of times flow must be stopped on the Reader/Sorter. The throughput benefits possible with STOQUE become even greater if multiple handlers transfer data to one or more working programs. While this situation is most common with Reader/Sorter operations, some data communications systems also employ more than one handler.

Example 6:

A final example of the use of STOQUE for asynchronous processing, considers a data communication system consisting of a handler, a working program Y, and possibly other processes. As transactions are received from the remote devices, information is passed to program Y for processing. Because of the volume of traffic on certain lines, or for other reasons, it might be desirable to process some transactions before others when multiple messages are present simultaneously (for example, some transactions have a higher priority than others).

A queue can be given a substructure by associating a programmatically specified name with any individual entries of the list. This name, in addition to the name of the entire list, allows the retrieval of items which are within, rather than at an end of the list. This feature provides the means to accomplish the priority handling desired for this example. In particular, the handler might store all transactions in a queue called LIST1, all high priority items being given an entry name of H and the lower priority items a name of L. When program Y is ready to process a transaction, it requests an H item from LIST1; only when no high priority elements are in the queue, would Y request the L items. This mechanism allows all H items, whether placed into the queue after some L items or not, to be processed first.

A similar result could also be obtained without the use of entry names by storing high priority items at the head of the queue, low priority items at the base and always retrieving from the head of the queue. This mechanism gives highest priority to the most recent high priority items while the oldest low priority items have preference in that group. If this disadvantage is not serious, a slightly greater retrieval efficiency can be achieved. If more than two priority levels are used, this method cannot be employed.

Priority handling of transactions can be achieved by establishing a distinct queue for each priority level. As there is a limit to the number of queues which STOQUE can maintain, however, the user must be careful that an excessive number of queues not be required.

## **INTERPROCESS CONTROL MECHANISMS**

Additional technical details of the CRCR and STOQUE mechanisms include such areas as restrictions, operational details, and efficiency considerations are presented in this sub-section.

### **CORE-TO-CORE TRANSFER**

The following variables must be considered when using CRCR.

#### **Data Length and Types**

Among the parameters passed to the MCP in a CRCR request is the description of the data to be transferred: type (digit, bytes, or words), length (in units of the type), and location (address).

The characteristics of the data to be transferred by the CRCR mechanism is almost unrestricted. The data can be located in any portion of the program's memory and can be of any length up to 9999 digits, bytes, or words, depending on the address of the data (for example, if the data occurs at a modulo 4 address, it can be up to 9999 words in length). The description of the data areas need not be specified in the same way by both the sender and receiver. If the absolute data lengths (adjusted

according to type) are different, only the shorter size is moved. Similarly, the buffer locations in the two programs need not be the same, nor must they be address synchronized in the same manner. The transfer is most efficient when both programs specify a data area which is word aligned and an even number of bytes in length. The declared data type is unimportant and is only used to determine the meaning of the data length field.

If the data structure does not permit a word move, the transfer is made on a least common denominator method; if the data in one or both programs is at an odd address, a numeric move is necessary; if the data is at an even address but is not word aligned or is not an even number of bytes in length, an alphanumeric move is performed.

A conversion operation (UN to UA or UA to UN) is never performed on the data. The information is always moved as is, without regard to the data type specified in the parameters.

## Program Names

Since any number of program pairs can be using the CRCR functions concurrently, it is necessary that a job identify the program with which it is to communicate. One of the parameters of the CRCR request is the address of a 6-byte field containing the appropriate program name.

The sender or receiver can place six blanks in the program name field. This format of the parameter means that any program in the appropriate state (ready to send or receive) will be used in the transaction regardless of name. When the program name field contains blanks, the CRCR request is known as a Global FILL. This option can be useful in many instances, but care must be exercised when multiple independent programming systems are using the CRCR functions so that transactions from one system are not sent to another. A hook up will not occur between a sender and receiver if both are requesting a Global FILL.

## Program Synchronization

One of the major features of CRCR is that data transfer does not occur unless both programs involved are prepared to enter into the transaction. This mechanism insures the integrity of both programs and further allows the processes to use the CRCR functions as a means of interprocess synchronization (or interlocking). Because this interlocking is necessary with CRCR, it is often necessary (especially in asynchronous processing) to reinstate a CRCR request as the other program is not ready for the data transmission.

Care must be exercised in the design of a programming system in order to minimize buzzing. If, for example, the program using the PROCEED TO option processes for a much shorter time between transactions than the other program, a significant amount of overhead can be incurred by the execution of repeated unsuccessful FILL requests. Where possible, either the burden of processing must be placed on the program which does not WAIT, or an attempt must be made to balance the processing time between the communicating programs.

When such balancing is not feasible, it is often possible to employ the WAKE-UP facility of CRCR as an alternate interlocking mechanism. This mechanism is most often used by data communications handlers and allows the program to be suspended until any of three events occurs: 1) a device needs servicing, 2) a program attempts a non-Global FILL or, 3) optionally, a specified time period elapses. This mechanism can be used in certain other areas to reduce buzzing overhead. However, care must be exercised as the order of events is important. One program can WAKE-UP another by a FILL request, but a program will be allowed to SLEEP if it executes a WAIT or SLEEP request, and all other

programs WAITing CRCR on this program are in receive mode. For that reason, the WAKE-UP mechanism is generally used in conjunction with the other functions of WAIT or SLEEP, or when proper timing can be assured. Note that the WAKE-UP occurs only on non-Global SEND or RECEIVE requests; the PROCEED TO option can be used or not, as desired.

A program must never buzz without giving up control to another process. If a program simply loops on one or more FILL requests, it is always reinstatable; further, as the highest priority job on the system, no other program, including the one for which the buzzing job is waiting, can ever execute. Even if it is not the highest priority job, the program would expend less overhead if other tasks could process between the FILL requests.

## STORAGE QUEUE

The following variables must be considered when using STOQUE.

### STOQUE Parameter Block

When a program executes a STOQUE request, several parameters must be supplied: the request type, the name of the queue to which the request pertains, the entry name (if used), and the data area to be used in conjunction with the storage or retrieval request. The request type parameter is part of the STOQUE statement. All other elements are constructed by the program in a STOQUE Parameter Block which is referenced in the STOQUE request. The block must begin at a mod-4 address (for example, a COBOL 01 level). The first six bytes contain the name of the queue (Stoqname) to which the request pertains; this name must be left justified and blank filled if less than six bytes long. The name can contain any EBCDIC characters, including embedded blanks and non-graphic combinations, except that the entire name cannot consist of all NULLs or be MCPQUE (reserved for MCP). The next element is a 2-digit field containing the size of the entry name in bytes. The field can contain any value from 0 to 99; if no entry name is used, the length should be zero. The entry name field then follows. The name, which can consist of any EDCDIC characters, can be used to provide a substructure to a queue. Multiple entries in a queue can have the same entry name.

Retrieval requests need not specify the complete name associated with an entry. If a shorter name is used, the name acts as a group specifier as described earlier.

The length of the message area is specified in a 4-digit field containing the size of the buffer in bytes. The length can be any value from 1 to 2000. For a POLL operation, this field is used as the response area rather than a data length field as no message area exists for this operation.

The message area follows the length field. No restrictions are placed on the content of the field. A maximum length of 2000 bytes is permitted.

When a storage request is made, all fields from the entry name length field to the end of the entry data field are moved to the queue. When a retrieval request is made, these fields are moved from the queue entry to the program STOQUE Parameter Block Area. In certain cases, care must be exercised in retrieval request using group entry names. For example:

```
01  STOQ-PAR-BLOCK.
    03  QNAME PIC X(6) VALUE "QNAME1"
    03  QENT-NAME-LENGTH PIC 99 COMP VALUE 03.
    03  QENT-NAME PIC XXX VALUE "ABC".
    03  QENT-DATA-LENGTH PIC 9(4) COMP VALUE 0100.
    03  QENT-DATA PIC X(100).
```

If an entry in QNAME1 has an entry name of ABCD, a retrieval request for ABC can cause an unexpectedly large entry to be accessed. For this case, QENT-NAME-LENGTH would contain 04, QENT-NAME would be a 4-byte field containing ABCD (spilling over into QENT-DATA-LENGTH) and all other fields would be shifted over one byte.

Proper access of the data can be accomplished with a redefinition of STOQ-PAR-BLK. Retrieval requests using an entry name are slightly less efficient than simple requests. Storage requests are equally efficient, regardless of type.

## Buffer

The memory buffer maintained by STOQUE for the storage of data is allocated in user memory external to all programs. The space is allocated as needed and deallocated when possible. The user can control the amount of memory which STOQUE can use for storage space through Cold Start LIMIT specifications.

Memory is obtained in fixed size blocks (5 KD each) called Stoqblocks. The number of these Stoqblocks which can be allocated, up to a maximum of 60, is controlled by a LIMIT specification. If it is not possible to get enough memory space for a Stoqblock, the message "\*\*\* NO MEM FOR STOQ" is displayed on the OCS. To minimize the overhead of frequent memory allocation and deallocation, the MCP checks for empty Stoqblocks on a periodic basis rather than returning them when empty.

## Queue Names

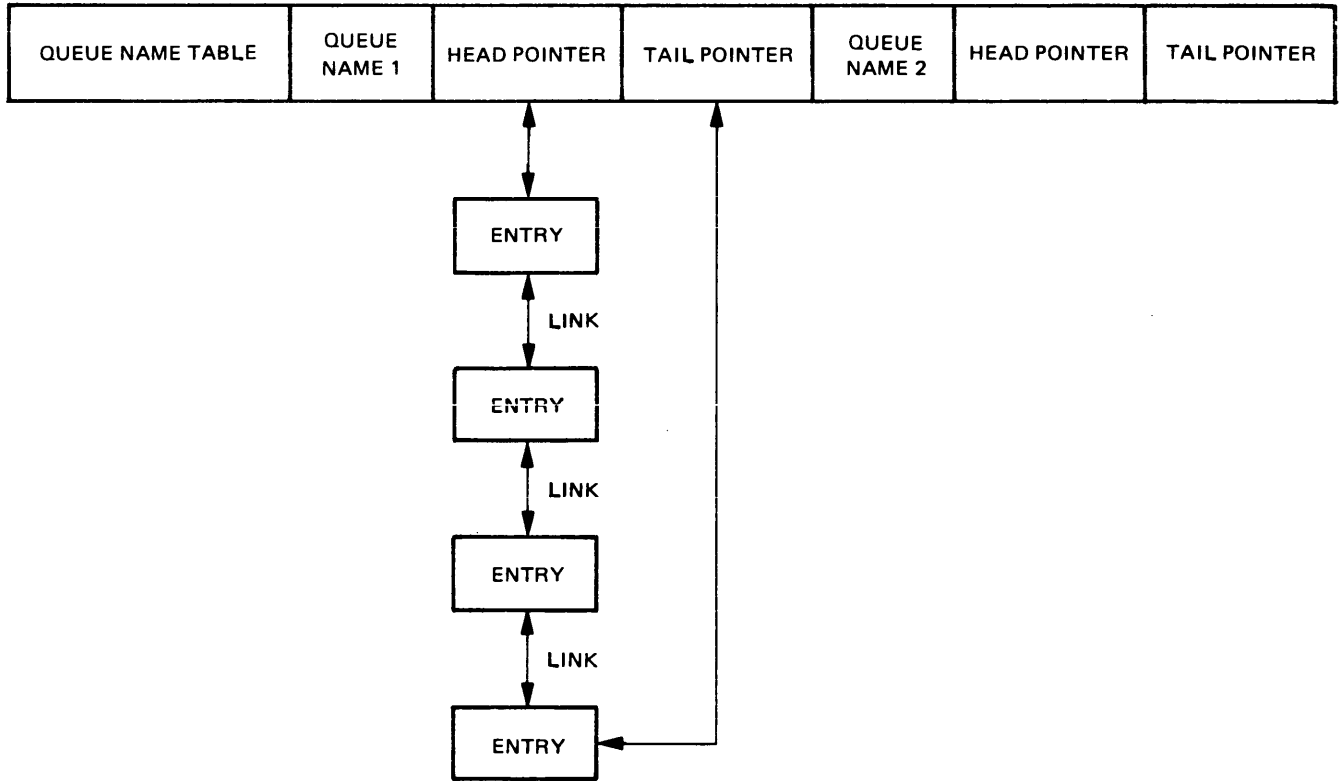
Data elements placed into the memory buffer by the STOQUE module carry no information about the program from which it was sent or destined. Instead, the elements are organized into a queue identified by the programatically specified Stoqname. The Stoqnames are maintained in a table contiguous to the STOQUE module. Because the Stoqname defines a queue, the capacity of the name table limits the number of separate queues which can be maintained simultaneously. The STOQUE module provides a table with a capacity of 10 names by default; a LIMIT STOQ NAMES parameter card can be used to increase the table size beyond the default value up to a maximum of 99 entries. Because memory for any additional table space is allocated in 1 KD increments, maximum efficiency of memory utilization occurs when the Stoqnames table is 10, 43, or 99 entries in size as each entry is 30 digits long.

A new entry is made in the Stoqnames table when the first storage or retrieval request is made for an entry of that name. An entry is not deallocated until the name table is full, a program executes a request for a new queue name and a queue is completely inactive (no entries exist and no programs are waiting for entries), or until an RQ keyboard message is entered.

## Queue Organization

The queue entries can occur anywhere in the STOQUE buffer space. Inter-element links (addresses) provide the structure and organization of a queue and thus allow a simple means to access elements or place new items into a queue. This type of structure is known as a linked list.

The organization of the storage queues is shown graphically in figure 3-1. The entries of the Stoqnames table provide rapid access to the list at either the head or base. In addition, each list element points to both the next and previous entry, allowing access to any individual item in the queue through either the head or base. The list, then, is bi-directional.



P1267

**Figure 3-1. Storage Queue Organization**

The flexible structure of the queues permits a wide range of applications. The existence of both forward and backward links allows items to be added to or retrieved from either the head or base and thus permits a queue to be used as a sequential list (FIFO), or as a push-down stack (LIFO), or both. Extremely complex queuing and retrieval mechanisms can be developed by combining these operations. Even more sophisticated designs are possible by using queue entry names as well.

### Storage Queue Entries

When a program executes a storage request, STOQUE obtains the necessary buffer space and moves the program data to the entry. In addition, a number of control fields are established in the entry and are used to manage the empty space. STOQUE then links the entry into the proper queue in the manner specified by the program.

Similarly, a retrieval request causes data to be moved from an entry to the requestor. The entry is then returned to the available space list.

The queue entry contains several important components which are used for the following purposes:

#### Space management

Links to the next and previous physical space in the Stoqblock and size specification are used when entries are retrieved and the space is to be returned to the available list.

#### Queue management

Links to the next and previous entries in the queue are maintained to permit access to elements in either direction.

#### Entry management

The size of the user portion of the entry and the entry is present.

#### Filler

Filler may be present to make the entry mod 4 digits in size and/or because the original available slot was not large enough for both the entry and new available links for the remainder.

The user portion of the entry is simply the program specified STOQUE Parameter Block (except the queue name portion). The MCP is unconcerned with the contents of this area except at the time of a retrieval request with an entry name specification. The requested name is checked against a corresponding number of characters of the entry name in the queue entry until a match is found or the queue is exhausted.

The user portion of an entry can range from 3 to 2102 bytes in length depending on the size of the entry name and data area. Retrieval operations are more efficient if the size is an even number of bytes.

### Available Space List

In addition to the active queues, available areas within the buffer space are maintained in a linked list. When a storage request is made, the STOQUE module attempts to find the smallest available area which can contain the entry. If no available space is large enough, a new Stoqblock is obtained (subject to the user-specified block limit and memory availability) and linked into the available space list.

As entries are removed from a queue, the MCP returns the space to the available list, consolidating the entry with surrounding space as applicable.

If an entire block remains in the available list for 40 seconds, the memory space is returned to the system, unless this would reduce the number of remaining STOQUE blocks below the minimum specified on a Cold Start LIMIT card. This mechanism minimizes frequent memory allocation and deallocation operations.

At any given moment, available space can be extensively checker-boarded throughout the Stoqblocks; because of the transient nature of the individual entries, the available list changes rapidly. Thus, no attempt is made to consolidate the available areas within a block.

## LANGUAGE CONSTRUCTS FOR CRCR AND STOQUE

The language constructs for CRCR and STOQUE are presented here.

### CORE-TO-CORE CONSTRUCTS

In COBOL ANSI-68, the Core-to-Core construct is

$$\text{FILL data-name-1} \left\{ \begin{array}{l} \text{FROM} \\ \text{INTO} \end{array} \right\} \left\{ \begin{array}{l} \text{literal-1} \\ \text{data-name-2} \end{array} \right\} [\text{PROCEED TO paragraph-name}].$$

FROM is specified for data receiving; INTO indicates data sending.

Data-name-1 addresses the alphanumeric data area to be used for the operation and can be subscripted or indexed. Literal-1, if used, is a non-numeric literal or figurative constant (spaces) specifying the name of the program with which communication is desired. Data-name-2, if used, must be described PICTURE X(6) and contain (left justified) a program name or blanks. If literal-1 or data-name-2 has a value of SPACES, a Global FILL is indicated.

The optional PROCEED TO clause is used if the requestor does not wish to wait until the other party to the transfer is prepared.

In COBOL ANSI-74, the Core-to-Core constructs are:

$$\begin{array}{c} \underline{\text{SEND}} \left\{ \begin{array}{l} \text{literal-1} \\ \text{data-name-1} \end{array} \right\} \underline{\text{FROM}} \text{ [data-name-2]} \\ \text{[ON } \underline{\text{EXCEPTION}} \text{ imperative-statement]} \\ \\ \underline{\text{RECEIVE}} \left\{ \begin{array}{l} \text{literal-1} \\ \text{data-name-1} \end{array} \right\} \underline{\text{INTO}} \text{ [data-name-2]} \\ \text{[ON } \underline{\text{EXCEPTION}} \text{ imperative-statement]} \end{array}$$

The SEND format is for sending data; RECEIVE is for receiving data.

Literal-1 and data-name-1 specify the name of the program with which communication is desired. Data-name-2 addresses the data area to be used for operation.

The optional ON EXCEPTION clause has the same meaning as the PROCEED TO clause has in COBOL ANSI-68.

In BPL, the Core-to-Core construct is:

$$\underline{\text{FILL}} \left\{ \begin{array}{l} \underline{\text{IN}} \\ \underline{\text{OUT}} \end{array} \right\} \text{data-name-1} \left\{ \begin{array}{l} \text{literal-1} \\ \text{data-name-2} \end{array} \right\} [ \text{ [label] } ]$$

The meanings of all elements are similar to the COBOL definition. The word IN specifies data receiving and OUT indicates data sending. The data-names cannot be subscripted. The action label is optional but, if present, must be enclosed in brackets.

In FORTRAN, there are three Core-to-Core constructs.

For receiving data, the formats are:

CALL ACCEPT (data field, length, logical variable, program name)



CALL FILL (data field, length, program name)

The FILL subroutine is similar to the ACCEPT subroutine except that a logical variable is not present in FILL. Because of this, FILL requires the program to wait until it receives a message from another program.

The data field parameter references the variable or array into which the data is to be placed. The length specification can be a constant or variable specifying the size of the message in bytes. When the ACCEPT subroutine is called, the logical variable is set .TRUE. if data was received and .FALSE. if no program was ready to send the message. The program name parameter can be a constant or variable containing the program name used for the hook-up.

For data sending the format is:

CALL SEND (data field, length, program name)

The SEND parameters have the same meanings as the respective ACCEPT parameters. No provision is made to buzz on data sending. If the receiver is not ready to receive, the sender must wait.

## STOQUE CONSTRUCTS

In COBOL ANSI-68, the STOQUE is used with:

$$\text{FILL data-name} \left\{ \begin{array}{l} \text{FROM} \\ \text{INTO} \end{array} \right\} \left\{ \begin{array}{l} \text{TOP} \\ \text{BOTTOM} \end{array} \right\} [\text{PROCEED TO paragraph-name}].$$

FROM is specified for data storage requests; INTO indicates a data retrieval request. The data-name references the STOQUE Parameter Block which describes the queue and the data entry. TOP specifies queuing or retrieval from the head of the queue (PUSHQ/POPQ); BOTTOM designates an operation at the base of the queue (PUTQ/PULLQ). The optional PROCEED TO clause is used with a storage request when the requestor does not wish to wait if storage space is unavailable for the data. The clause is also used with a retrieval request when the program does not wish to wait if the requested data is not in the queue.

The queue inquiry format is:

FILL data name FROM POLL .

In COBOL ANSI-74, the STOQUE constructs are:

$$\text{SEND TO} \left\{ \begin{array}{l} \text{TOP} \\ \text{BOTTOM} \end{array} \right\} \text{data-name}$$

[ON EXCEPTION imperative-statement]

B 2000/B 3000/B 4000 MCPVI Programmer's Guide  
Interprogram Communication

RECEIVE FROM { TOP  
BOTTOM } data-name

[ON EXCEPTION imperative-statement]

SEND TO is equivalent to FILL INTO in COBOL ANSI-68, and RECEIVE FROM is the same as FILL FROM. The meaning of all other elements is similar to those in COBOL ANSI-68.

BPL-STOQUE interaction is defined by:

{ PUSHQ  
PUTQ  
POPQ  
PULLQ  
POLLQ } data-name [ [ label ] ]

The PUSHQ and PUTQ requests specify storage requests at the TOP or BOTTOM of the queue, respectively. The POPQ and PULLQ requests designate retrieval from the TOP or BOTTOM, respectively, while POLLQ specifies queue inquiry. The data-name addresses the STOQUE Parameter Block. The action label (not valid for POLLQ, optional otherwise) has the same meaning as the COBOL PROCEED TO clause. If the action label is provided, it must be enclosed in brackets.

## SECTION 4

### DEVICE ALTERNATES

#### GENERAL CONCEPTS

One of the major measures of system throughput is the efficiency with which system resources are utilized. This implies that peripheral resources must operate as close to rated speed as possible; that memory space must be conserved by keeping jobs as small as possible and large jobs must be directed through the system as fast as possible; that processor usage by a given job must be minimized. In summary, programs must use resources only to the extent needed. Often, maximizing the use of one resource involves a trade-off with the most efficient use of another.

The implementation of device alternates (backup devices) provides a means of significantly increasing system resource utilization with a small penalty.

Several input/output devices on the system, particularly the card reader, line printer and card punch, are slow and are frequently the factors which most determine the speed of a program. This is undesirable since other valuable resources may be used inefficiently; the speed of other peripheral resources effectively restricted to that of the slowest device and memory is occupied by a large program which is usually idle, waiting for the slow device. In other cases, the slow device is only used sparingly (for example, an error listing) as inefficiency results by using the device at only a fraction of rated speed, and prevents other jobs from obtaining it.

When multiprogramming, several jobs which simultaneously need the same resource type can be found, especially with the slow devices. If multiprogramming is to be efficient, provision must be made to accommodate such simultaneous requests even if more requests than physical devices are available. These considerations indicate the need for alternate handling of line printer, card punch and card reader files. If these can be shunted to or from another device such as tape or disk, the following benefits will be obtained:

#### Speed

Other files used by the program can be processed closer to the rated speed of the devices concerned. Thus the program can release critical resources such as tape and memory more quickly. In all cases file handling is slower through the primary physical device than through an alternate and in all cases the difference in speed is significant.

#### Device Utilization

Programs will not tie up slow devices used sparingly. Instead the files will usually be placed on disk or disk pack devices which can be in use simultaneously by many jobs, and thus, will not interfere with the performance and requirements of any other programs.

#### Simultaneity

One or many programs can activate almost any number of these unit record files regardless of the number of physical devices of that type available. For example, several programs can be reading card input files even though only one hardware card reader is attached to the system.

Programs can simultaneously generate more print or punch files than there are printers or punches available. This often permits one program to exist where several would have been needed previously, thus reducing run times and programming effort.

Programs can read several card files simultaneously. This is especially valuable in a conversion environment.

## CARD READER ALTERNATES

Programs can be written without significant concern for physical limitations. Do not be too concerned with differences in the configuration of backup systems.

Systems need not be physically configured for the maximum requirements of a small number of jobs, thus, wasting money on infrequently needed peripherals.

Scheduling becomes easier as there is no need to be concerned with conflicts in resource requirements.

Printer or punch files can be generated if a printer or punch is not available or is not present on the system. This is common for punch files (particularly in conversion environments) whereby files can be generated without the presence of a punch. The generated files might be punched on another system, or simply used directly from disk.

### Convenience

Printer and card punch files can be physically output when convenient to the user rather than when the program is run. Files need not be output in any particular chronological order of programmatic creation. Similarly, card reader files can be placed on the alternate device at the convenience of the user, before the file is needed by a program but at a time when the file is available and a physical card reader is free.

The particular implementations of device alternates by the MCP provides a number of other benefits which are detailed later in this section.

## CARD READER ALTERNATES

Because the card reader is often a serious system bottleneck, the MCP permits card input to be read from an alternate medium. The card images to be processed, including all control cards, are transferred from the physical card reader to disk or disk pack by a utility program named LDCNTL. The disk files created by LDCNTL are called pseudo card decks or pseudo-readers. Until the pseudo-reader is activated, it is simply a dormant file analogous to a physical reader which is not yet turned on. When needed, these readers can be activated and the decks processed by programs as if being read from a physical card reader.

The user is unconcerned with the mechanism of the pseudo-reader. Operator control is exercised primarily over the choice of medium for the card file, and the sequence in which the decks are to be processed by the MCP.

Every card deck on the system is independent of all other decks, physical or pseudo. Several card decks can be in use simultaneously, by one or more programs, from any combination of card media even while other pseudo-decks are being created. Pseudo-decks may or may not be processed in the order in which created, as the user requires. On multiple processor systems, pseudo-decks can be created on one processor and used by another.

## BASIC MECHANISM OF CARD READER ALTERNATES

The first stage of pseudo-reader processing is similar to the operator placing physical card decks into physical card readers. The corresponding pseudo-reader step determines the transfer of physical card images into a properly formatted disk or disk pack file, thus creating an inactive pseudo-reader. As each pseudo-reader is created, it is given a unique numeric designation for operator interface. The deck (reader) number is the means to reference the pseudo-reader through which the deck is processed in much the same way as a channel/unit designation addresses a physical card reader.

CARD READER ALTERNATE

The second stage of deck processing activates the readers. For a physical reader the START button is depressed; for pseudo-readers, the RN input message is used. This message addresses the specific pseudo-reader which is to be activated; the MCP processes the control information in the deck in the same way as a deck passing through a physical reader. Readers which have been RNeD are called active readers. Readers can become associated with programs only after being activated.

These stages of the process are controlled by the system operator as for physical card decks. However, provision is made for automatic MCP control of the second stage through the Automatic Pseudo-reader Initiation (APCR) system option (that is, the MCP can be instructed to activate pseudo-readers as they become present).

The last stage of pseudo-deck processing consists of the usual programmatic file OPEN and file READ statements. The user program is unconcerned with the physical source of the file.

Often, the operator wishes to inquire into the status of pseudo-decks. The CD message is used to inspect the image of first card in inactive pseudo readers. This message corresponds to the physical action of inspecting the first card of a card deck in the physical card reader before the reader is started. Since such a physical action is impossible for pseudo-readers, the CD message is provided. The OL message can be used to inquire into the status of decks which are active. This corresponds again to the physical card reader operation.

Another function which may be necessary for the operator to perform is the removal of decks. Inactive decks are removed by the RD message. Decks which are in use are removed automatically when the reader becomes empty.

To convert a deck from active status to inactive status, the DA message is used. The deck must not be in use at the time of the DA request.

The following sub-sections give a more detailed explanation of each component involved in card reader alternate handling.

Table 4-1 summarizes the relationships between the physical-reader/physical-card-file and the pseudo-reader/ pseudo-deck concepts.

**Table 4-1. Analogies Between Physical and Pseudo Components**

Concept	Physical	Pseudo
Deck preparation	Place deck in card reader	LDCNTL action
Deck	Control cards and data file in card reader	Card images on disk or disk pack
Device	Card Reader	Pseudo-Reader (disk file and MCP)
Initiation of deck processing	Push START button on card reader	Key in RN message or have APCR option set
Initiation of data deck processing	Program file OPEN	Program file OPEN

## LDCNTL UTILITY

### LDCNTL (LOAD CONTROL) UTILITY

The Load Control program (LDCNTL) creates pseudo-decks (readers) from physical card decks. LDCNTL is a small straightforward card-to-disk program with the following special features:

1. The name of the physical card deck for LDCNTL is the special identifier CTLDCK. The first card in the reader must read:

```
? DATA CTLDCK
```

Following this file specifier are the normal decks of cards which one would usually pass through the reader, including all control cards. A typical CTLDCK might be:

```
? DATA CTLDCK
? EXECUTE PROGA CHARGE 9
? PRIORITY = 5
? DATA CARDS
  data file
? END . END DECK 1
? EXECUTE PROGB CHARGE 345
? END . END DECK 2
? EXECUTE PROGC CHARGE 99963
? DATA X
  data file
? END . END DECK 3
. . .
```

2. The MCP does not process control cards within the CTLDCK file with the sole exception of SPO control cards. This allows LDCNTL to read control statements (cards with a invalid punch in column 1) like data. However, the MCP does establish that the first word in the control statement is a valid keyword so that data card with spurious invalid characters in column 1 are not passed to LDCNTL. LDCNTL will include all valid control statements, except SPO cards, in the pseudo-deck. Since the MCP allows the control statements to be read as data (including the END statements), a special terminator is necessary to denote the end of the CTLDCK data file. This card is:

```
? ENDCTL
```

The concepts of deck and file are slightly different for the CTLDCK file. Deck includes all physical cards from the ? DATA CTLDCK statement through the ? ENDCTL card. Data file includes all cards between those statements (except SPO cards) even though some contain invalid characters in column 1.

3. One execution of LDCNTL can create many pseudo-decks. As each ? END card (end of deck) is sensed, LDCNTL closes the pseudo-deck (disk file). If more decks are present, following the one just read, a new pseudo-deck is initiated. One or more sets of control cards and/or data files can be contained in a single pseudo-deck. Typical decks may be:

```
? EXECUTE PROG1
? CHARGE 9
? DATA INFO1
  data file
? END
```

The preceding is the most common format of a deck and contains only one data file and one set of control statements. Another deck might be:

```
? EXECUTE PROGRAM CHARGE 91010  
? FILE A = B  
? END
```

This deck contains no data file. Presumably PROGRAM has no card input or is to be loaded into a separate pseudo-deck.

```
? EXECUTE PROG2 CHARGE 91026  
? DATA INFO2  
  data file  
? EXECUTE PROG3 CHARGE 91107  
? END
```

This deck contains one data file and two sets of program execution control cards. Since the pseudo-deck is processed serially, the second EXECUTE request will not be processed by the MCP until all cards in file INFO2 have been read. Thus, PROG3 will not be scheduled until this time. The second EXECUTE card will function as an end-of-file (EOF) delimiter for the file INFO2.

Other examples include more than one data file:

```
? EXECUTE PROG4 CHARGE 9  
? DATA INFO4  
  data file  
? DATA INFO5. THIS ENDS INFO4.  
  data file  
? END. THIS ENDS INFO5.
```

A pseudo-deck containing more than a single set of control statements and/or a single data file is known as a stacked pseudo-deck.

4. LDCNTL can be initiated at any time by either the LD message or an EXECUTE LDCNTL control instruction. Frequently, the command is entered by the operator when the system is first initialized and the program is not terminated until processing is complete for the day. At that (or any) time LDCNTL can be brought to end-of-job (EOJ) by passing the ENDCTL statement through the card reader. LDCNTL remains idle most of the time if in the mix all day. Since no processor time will be expended by LDCNTL during this period and since LDCNTL requires little memory, the benefits of operations ease usually outweigh the minor memory penalty. More than one LDCNTL program can be entered into the mix by entering multiple LD messages when the system has more than one physical card reader.
5. Data files processed by LDCNTL can be punched in EBCDIC, BCL or some of each. (LDCNTL is not able to properly process BINARY card files.) The only requirement is that of providing the proper ?DATA or ?DATAB control card to label the file. The MCP has the capability to change the mode of reading according to the requirements of the label card.

PSEUDO-READERS

PSEUDO-READERS

The pseudo-reader (pseudo-deck) is a file consisting of card images, both program data and MCP control statements, in a format acceptable to the MCP. The file characteristics are:

File Attribute	Disk	Disk Pack
Record Size	100 UA	120 UA
Blocking Factor	Multiple of 8	Multiple of 6
Records per Area	Multiple of 1000	Multiple of 1200

A file must contain an integral number of blocks per area.

LDCNTL creates decks 100 X 100 blocked 8 on disk and 100 X 1200 blocked 6 on disk pack. The record format is given in table 4-2.

**Table 4-2. Pseudo-Reader Card Format**

Field	Size	Notes
Record Type	1 UN	D for control cards; 8 for data records
Flag Digit	1 UN	F (control cards only) – specifies link present
Link	6 UN	Zero relative key to next control card
Data	96 UA	Card Image
Filler	20 UA	Present for disk pack file, only

The record type field contains the first digit of the result descriptor which occurred when the physical card was read. D (undigit) means invalid character detected. The flag digit is present in control records if the control card link is present. This optional link is used during deck flushing operations to find the next control card. The use of the link is described below.

Invalid characters in control records are replaced by an EBCDIC question mark character (6F).

When the reader is created, the output disk file-identifier must be @00000. This special identifier tells the MCP that when the file is OPENed the next sequential pseudo-deck name is to be assigned to the file. All decks have a name of the form:

@pnnnn

where p is the number of the processor on which the file was created and nnnn is a unique 4-digit number assigned by the MCP. Numbers are assigned starting with 0001 and incremented as each new file is created. The counter is reset at 9999. This assignment mechanism insures file uniqueness within and across processors and allows the user to create pseudo-decks.

Provision is made to allow decks created on one processor of a multiple processor system to be used on another processor. For most purposes, the MCP assumes that only those files created on a given processor are to be used on that processor unless specifically informed otherwise.

Care must be taken to distinguish the name of the disk file which is the pseudo-reader from the name of any card data file within the reader. Within the pseudo-deck can be one or more data files, each



with a DATA control card identifying the name of the card file which follows. This is the only name of concern to the program; the name by which to access the data in the card file. This mechanism is analogous to the handling of physical card readers whereby the unit is addressed by channel/unit designation which is independent of the name of any data file associated with the unit.

## User Creation of Pseudo-Decks

A user program can create pseudo-decks independently of LDCNTL. This may be desirable when the input is from a source other than the card reader, or when the user wishes more stringent control over some aspects of the input. The output disk file (pseudo-deck) must be defined as described in preceding text, and must be given the special name #00000.

If input is to be through the card reader, the input file identifier must be CTLDCK so that control cards can be accessed. The user program need not be concerned with mixed BCL and EBCDIC decks as the MCP automatically changes the mode of reading cards according to the requirements of the label cards detected (DATA or DATAB). For other card files, any buffering technique or number of buffers is allowed.

Control cards can be recognized by examining the result descriptor for the record. The first three digits of the result descriptor for control cards is D40; all other cards yield an 800 descriptor. In addition, column 1 of all control cards in memory must contain an EBCDIC question mark character (6F). If the user wishes the program to be initiated by the LD console message instead of the MCP intrinsic, the user program must be given the name LDCNTL.

If the input from which a pseudo-deck is made comes from RJE, the program should execute a PROGRAM BCT to obtain its RJE link. This RJE link value should be placed in columns 75-76 of the control cards in the pseudo-deck (a move of 2 UA to 2 UA), and column 74 should be @FF@. This allows messages generated by the pseudo-deck to be routed back to the RJE station.

Figure 4-1 shows a sample program to create pseudo decks.

## Pseudo-Reader Processing

The following describes the basic functional areas of pseudo-reader processing: reader activation, control card processing, programmatic access of card decks, and reader deactivation. The mechanism can best be seen through an example. The following deck might be loaded into a pseudo reader on disk by LDCNTL.

```
? EXECUTE A CHARGE 3
? FILE ABC = DEF MT9
? DATA X
.
.
.
? END
```

When this reader is created, the MCP checks the APCR option. If on, and table space and other limitations permit, an automatic RN is invoked; otherwise, the reader remains inactive until turned on by an operator RN request or until a reader becomes available.

When the activation request is made, the MCP enters information about the deck into a Pseudo-Reader Directory on disk and begins to read the control cards. Program A is scheduled for execution in the

# B 2000/B 3000/B 4000 MCPVI Programmer's Guide Device Alternates

## PSEUDO-READERS

```

000100 IDENTIFICATION DIVISION.
000110 PROGRAM-ID. LDCNTX.
000120 REMARKS. SORTS TAPE INPUT (80 CHARACTER RECORDS), FORMATS
000130 INTO PSEUDO DECKS WITH APPROPRIATE CONTROL INFO.
000140 FIRST RECORD OF SORTFL IS PARAMETER CARD.
000150 DECK ORDER IS
000160 ? EX PROG DATA PARAMS
000170 ? PARAMETER CARD
000180 ? DATA COIN
000190 ? DATA DECK
000200 ? END
000210 FIRST CONTROL CARD LINKS TO SECOND. SECOND CONTROL
000220 STATEMENT DOES NOT NEED A LINK AS THE NEXT CONTROL
000230 CARD IS THE END STATEMENT.
000240 ENVIRONMENT DIVISION.
000250 INPUT-OUTPUT SECTION.
000260 FILE-CONTROL.
000270 SELECT TPIN ASSIGN TO TAPE RESERVE 1.
000280 SELECT PSEUDO ASSIGN DISK RESERVE 1.
000290 SELECT SORTFL ASSIGN SORT DISK RESERVE 1.
000300 DATA DIVISION.
000310 FILE SECTION.
000320 FD TPIN.
000330 J1 TP PIC X(80).
000340 FD PSEUDO FILE CONTAINS 20 BY 1000 RECORDS
000350 VALUE OF ID "000000".
000360 J1 LU-REC.
000370 J2 LU-FLK.
000380 * JDFR = CONTROL CARD AND LINK, JDDR = CONTROL CARD
000390 * NO LINK, 80 = DATA RECORD
000400 J3 CNTRL-FLAG PIC 99 CUMP.
000410 * USED AS LINK FOR CONTROL RECORDS
000420 J3 SEQ PIC 9(6) CUMP.
000430 J3 CDATA PIC X(80).
000440 J3 FILLER PIC X(16).
000450 SD SORTFL FILE CONTAINS 20 BY 1000 RECORDS
000460 BLOCK CONTAINS 10 RECORDS.
000470 G1 SORT-REC.
000480 J2 SORT-CODE PIC 9999.
000490 J2 FILLER PIC X(75).
000500 WORKING-STORAGE SECTION.
000510 77 EX-CARD PIC X(22)
000520 VALUE "? EX PROG DATA PARAMS.".
000530 77 JECK-LABEL PIC X(12) VALUE "? DATA COIN".
000540 77 NO PIC X(6) VALUE "? END.".
000550 PROCEDURE DIVISION.
000560 SORTIT SECTION.
000570 SORTING.
000580 * SORT TAPE - FIRST RECORD WILL BE PARAMETER FOR PROG
000590 * SORT SORTFL ON ASCENDING KEY SORT-CODE
000600 USING TPIN OUTPUT PROCEDURE CREATE-DECK.
000610 END-IT.
000620 STOP RUN.
000630 CREATE-DECK SECTION.
000640 OPEN-DISK.
000650 OPEN OUTPUT PSEUDO.
000660 * SET UP EXECUTE CARD, FLAGS AND ZERO RELATIVE LINK
000670 * MOVE EX-CARD TO CDATA.
000680 * MOVE JDFR TO CNTRL-FLAG.
000690 * MOVE 2 TO SEQ.
000700 * WRITE LU-REC.
000710 * GET PARAMETER CARD AND INSERT INTO DECK
000720 * RETURN SORTFL.
000730 * MOVE SORT-REC TO CDATA.
000740 * MOVE 80 TO CNTRL-FLAG.
000750 * WRITE LU-REC.
000760 * MOVE JECK-LABEL TO CDATA.
000770 * MOVE JDDR TO CNTRL-FLAG.
000780 * WRITE LU-REC.
000790 RETURNIT.
000800 * RETURN SORTFL AT END TO PUT-END-CO.
000810 * MOVE SORT-REC TO CDATA.
000820 * MOVE 80 TO CNTRL-FLAG.
000830 * WRITE LU-REC.
000840 GO TO RETURNIT.
000850 PUT-END-CO.
000860 * MOVE NO TO CDATA.
000870 * MOVE JDFR TO CNTRL-FLAG.
000880 * WRITE LU-REC.
000890 * CLOSE PSEUDO LOCK.
000900 END-OF-JOB.
PROGRAM ID LDCNTX.
COMPILE DATE 14:04 12/23/78 USING 368778 COMPILER.
NO WARNINGS.
NO SEQUENCE ERRORS.
81 SYMBOLIC RECORDS COMPILED AT 607 RECORDS PER MINUTE.
8 SECONDS TOTAL ELAPSED CLOCK TIME.
17 DISK SEGMENTS REQUIRED FOR THIS PROGRAM.
15000 BYTES TOTAL CORE REQUIRED.
45000 BYTE COBOLY COMPILER. RELEASE NUMBER: ASK 6-2 .
LBA ADDRESS HIGH ADDRESS LENGTH IN DIGITS
RESERVED MEMORY 000000 00198 000198
DATA DIVISION 00198 00252 00054
FIXED SEGMENT CONSTANTS 00252 00252 00000
FIXED SEGMENT INSTRUCTIONS 00252 00264 00012
INPUT OUTPUT BUFFERS 00264 00316 00052
STACK 00616 00000 00000
MAXIMUM DISK FILE HEADER SPACE 00000

```

P1268

**Figure 4-1. Example Pseudo-Deck Creation Program**

usual way. When the DATA statement is reached, the Pseudo-Reader Directory entry is updated to reflect the file name X. The initial processing of the deck is then complete.

Program A is eventually brought into the mix and opens card file X. The MCP checks the main memory tables for a physical card file and the Pseudo-Reader Directory for a pseudo card file with this name. When X is found in the directory, a memory area is obtained and a portion of the Disk File Header (DFH) for the deck and an Input-Output Assignment Table entry is established in this area. The remainder of the space is used as a buffer for a block of card images from the reader. Programmatic READ requests then result in the transfer of images from the block to the program buffer. When the END card is read through the reader, the program EOF branch is taken and the reader including all memory space, is released.

### Pseudo-Reader Activation

Each pseudo-deck is created in an inactive state; it must first be activated before the MCP can begin to process the control statements in the deck or assign any data file in the deck to a program. The user is given a number of options regarding pseudo-reader activation, the basic choice being whether the MCP or the operator is to control this area. If the user elects to have the operator control reader activation, the APCR system option must be reset. The operator can then turn on the desired readers as needed by using the RN keyboard message. This message can address the specific inactive pseudo-reader to be made ready by the format:

RN deck-number

where the deck-number specifies the 4-digit number assigned to that reader. A deck created on a different processor of a multiple processor configuration can be activated by specifying both the processor number and deck number:

RN processor-number deck-number

All inactive decks currently on disk can be made ready by the format:

RN /

Only decks created on the processor through which the RN request is made, are activated unless a processor number precedes the slash.

If the user wishes pseudo-deck handling to be completely automatic, the APCR system option must be set. This flag informs the MCP that as each new pseudo-deck is CLOSED by the creator, an automatic RN for that deck is to be invoked.

Activation of pseudo-readers causes no main memory space to be allocated. Instead the MCP creates a record in the disk resident Pseudo-Reader Cross-Reference Directory (PCRXRf) which has a capacity of 80 readers for each processor. While as many as 9999 decks can be on disk for each processor, a maximum of 80 can be active at one time.

The user can further restrict the number of readers the MCP can automatically activate at one time by the (set deck limits) SD input message. This message, which is of the format SD <integer>, specifies the maximum number of readers which can be activated simultaneously by the APCR facility. The integer can have any value from 1 to 80. If the Directory becomes full or the deck limit is reached, the MCP temporarily stops activating readers. Even if the deck limit is reached, the operator can still

## PSEUDO-READERS

employ the RN message to activate additional readers (up to an absolute limit of 80 total readers). Such RNs do not in any way affect APCR processing as the MCP keeps track of automatically RNeD decks, separately.

### Control Card Processing

The activation of a pseudo-reader is similar to the corresponding action for a physical reader.

When the reader is activated the MCP reads the first card image (which must be a control record). This statement, and all subsequent control syntax, is processed in the usual way. If an END statement is reached, the reader is considered empty and is removed from the Cross Reference and Disk Directories.

If a DATA(B) statement is accessed, the name of the data file and location in the reader is entered into the Cross-Reference Directory. The data file can then be accessed by a program.

### Programmatic File Access

Until a program executes a file OPEN request for a file in an active pseudo-reader, that file is in a state analogous to a file in a physical reader: positioned past the DATA statement and is unassigned. When a card file is OPENed, the MCP checks both the IOAT and the Pseudo-Reader Directory for the appropriate file identifier. If only one file of that name is found, assignment proceeds as described below. If more than one reader contains a file of the requested name but the EXECUTE (or COMPILE) request for the program was entered through one of them, that reader is assigned. Otherwise, file assignment requires operator intervention. File assignment for a physical reader involves connection of the program File Information Block (FIB) of the card file to the appropriate MCP IOAT entry and performing the initial loading of the program buffers. Pseudo-reader assignment requires more activities.

First the MCP must obtain a 2 KD main memory buffer for reader use. In this area the MCP builds an IOAT entry and DFH skeleton for the file. (Internally the MCP treats the device alternate file as a type of disk file.) The reader is marked in use in the Directory and is associated with the program FIB. Because the external memory area functions as buffers, the program FIB is modified to reduce the file to a single internal buffer. A block of card images is read into the external buffer; the program internal buffer is filled (if necessary), completing the OPEN request.

READ operations cause the transfer of records from the external buffer to the program buffer, and if necessary, to the program work area for as the external buffer is emptied, a disk read is initiated to refill. All I/O operations are overlapped with the program execution.

### Reader Deallocation

Deallocation of a card reader (physical or pseudo) means that the association between the device and the program using the data file is terminated. This term does not imply that the reader is deactivated or removed in any way. Deallocation of a card reader device occurs when an EOF condition is detected, when the program executes a CLOSE request prior to this time, or when the program terminates before either of these events has occurred.

EOF detection means that a control card (other than a SPO card) is sensed while the program is associated with the device. Unlike other devices, card readers are detached from the program at this point rather than at the file CLOSE request. For pseudo-readers, the MCP releases all memory space allocated to the reader and performs certain maintenance functions on the Pseudo-Reader Directory entry.

If the program executes a CLOSE request or terminates normally before EOF is sensed, the MCP de-allocates the reader and flushes the deck to the next control card. Flushing a physical reader requires passage of cards through the reader until a control card is sensed or the reader is made not ready.

Pseudo reader flushing gives similar results but is accomplished in a different manner. When the reader is to be flushed, the MCP examines the next card image in the reader; if a control card, flushing is complete. If that card is not a control card flushing is accomplished through the control card link found in the last DATA statement accessed. This link provides a rapid means of finding the next control record without the overhead of reading each record from disk. If the link is not present, the MCP assumes that the next control record is an END card and removes the reader as described in the next sub-section.

If the program terminates abnormally (for example, is DSed), the MCP deactivates the reader, removes the deck from the Pseudo-Reader Directory but not the Disk Directory. The reader is made not ready and can be reactivated by an RN request.

## Removing Pseudo-Readers

When a pseudo-reader becomes empty (such as when an END card is detected), the MCP removes the reader from the system (unless it was RNed with SAVE). The reader is removed from the Pseudo-Reader and Disk Directories, thus becoming inaccessible for further use. This can occur during control card processing, normal reading of a deck by a program, or during reader flushing.

In addition, if a source deck for a COMPILE and GO operation is contained in a pseudo-reader and the compilation is unsuccessful (for example, syntax errors were detected), the reader is removed if the control card found during a flushing operation is either an END or DATA statement. In the latter case, the MCP assumes that the DATA deck was included for the program which would have executed had the compilation been successful.

Decks can also be removed by explicit operator directive as described below.

## Recovery of Pseudo-Decks

An important feature of pseudo-deck handling occurs when the system is aborted while pseudo-readers are active. When a processor is reinitialized after an abort (Halt/Load), the Pseudo-Reader Directory is searched and all active readers for that processor are also reinitialized and placed into a not ready state. Such decks are not reactivated until the operator executes an RN request, regardless of the setting of the APCR option.

## Operator Interface To Pseudo-Readers

The operator has control over the following pseudo-decks (specific syntax and options can be found in the MCPVI Software Operations Guide, form number 1108987):

### Display Inactive Readers

In response to the CD input request, the MCP displays the number of each requested pseudo-reader and up to the first 50 characters from the first card in the deck. Only decks in inactive pseudo-readers are displayed. The password field of USER and BEGINUSER cards is replaced with asterisks.

### Monitor Active Pseudo-Readers

Pseudo-readers are monitored in the same manner as physical devices, through the OL message. The format OL PCR lists the status of all readers; OL PCR <deck-number> reports the status



### PCRM System Option

This option causes the MCP to inform the operator of the location and contents of the first control card and each DATA(B) card in a control card group as processed from pseudo-decks. This information can be used with the CONTROL option of the RN input message.

### Stacked Pseudo-Decks

While the MCP does not handle stacked pseudo decks in a special manner, mechanisms described previously have additional significance for such decks.

Stacked pseudo-decks are primarily intended for COMPILE and GO runs because of the MCP action if the compilation is not successful. Misuse of the facility can result in operational problems. For illustrative purposes consider a stacked deck containing the following:

```
? EXECUTE P1 CHARGE 60606
? DATA P1FILE
  data file
? EXECUTE P2 CHARGE 60606
? DATA P2 FILE
  data file
? END. END OF PSEUDO DECK
```

If program P1 is terminated abnormally (DSed) while file P1FILE is OPEN, the pseudo-deck is not flushed. Instead, the reader is deactivated as described above and an RN request results in the re-execution of P1. Conversely, if P1 completes the processing of P1FILE before it aborts, P2 will begin execution, and the termination of P1 will have no effect on the reader. Frequently, this would be undesirable but could be corrected in many cases by specifying that P2 be executed AFTER P1.

Note that the entire deck is recovered if the system aborts (Halt/Load) or if the program aborts (DS/DP). In either case, when the deck is reactivated (the operator keys in an RN message), processing begins again with the first card: EXECUTE P1... .

If program P2 is executing when the abort occurs, program P1 was completed. However, if the PCRM option was on, it is possible to restart the deck from ?EXECUTE P2. PCRM causes the deck location of the ?EXECUTE P2 card to be displayed. An RN with the CONTROL option can be used to bypass the execution of P1.

Due to these considerations, stacked decks must not be used indiscriminately.

### CREATION OF PSEUDO-DECKS FROM BACKUP PUNCH FILES

The MCP provides the ability to convert a file in punch backup format to a file in pseudo-deck format. This is useful when existing programs generate card files which are to be read by subsequent programs, but the user does not need the physical cards for any purpose. This is a common occurrence in a conversion environment.

The LDCNTL utility is initiated by the Convert (CV) console message. The full syntax of the message is:

```
CV [ /OWN ] [ * ] <file-number> [ /MCP ] [SAVE] [ <integer> ] [ ; parameter instructions ]
```

## PSEUDO READERS

The operator can include /OWN or /MCP parameter to require the initiation of either the user-coded or MCP version of the utility. If this parameter is omitted, the user-coded version is initiated if present on disk; otherwise, the MCP intrinsic is initiated. The <file-number> designates a punch backup file number and SAVE instructs LDCNTL not to PURGE the input (punch backup file) when the pseudo-deck has been created. The MCP intrinsic does not use the \*, /, or <integer> parameters. If the Cold Start CHRGE ALL specification has been made, the CHARGE parameter instruction must be included.

The MCP intrinsic LDCNTL OPENS the selected backup file, OPENS an output disk or disk pack file with the name #00000 and reformats the label and data records into acceptable pseudo-deck format. See User Creation of Pseudo-Decks in this section. LDCNTL does not include any control cards in the pseudo-deck beyond the initial DATA and terminating END statements. Special handling of stacker selected card punch records is not provided. A user-coded version of LDCNTL can be written if other features are desired.

The interface between LDCNTL and the MCP is identical to the PCHOUT interface described in Punch Alternate in this section.

## SUGGESTIONS AND SPECIAL METHODS

It is often worthwhile to keep LDCNTL permanently resident when card reader usage is fairly constant. This mechanism avoids any operator concern for this area of operation and insures that the reader is kept as busy as possible. A simple mechanism for initiating the process is to create the following deck:

```
? SPO SO APCR  
? EXECUTE LDCNTL ...  
? DATA CTLDCK
```

Following a Halt/Load, the deck can be entered through the card reader, thus, initiating both LDCNTL and automatic MCP reader activation without console operation. Whenever decks become available, the operator simply places them in the physical reader and need not be concerned with further procedures.

If reader usage is sporadic and/or the small amount of memory used by LDCNTL is of concern, the process is initiated in the same manner when required.

## Deferral of Decks

The mechanism of pseudo-reader handling permits decks to be processed in any order, regardless of the sequence of creation by LDCNTL or other means. Frequently, physical card decks are available to the operator before being needed by a program. If a card reader is available, it is beneficial to load the deck to disk or disk pack at that time rather than wait until the program requires the data.

Two basic methods are available to defer the processing of the deck until needed. If the deck is loaded with the APCR option reset, the MCP does not initiate the reader until the operator executes an RN request for that deck. After the deck is loaded, the option can be set again to allow automatic activation of subsequent decks. This method might be selected if desired to include the program control statements with the data deck.

Alternately, the deck can be loaded with the option set if the control statements (other than the DATA control card) are removed from the front of the deck. The MCP then activates the reader when the deck is loaded and the file is ready to be accessed. This method does not require operator intervention to activate the reader, but does require that the control cards be entered separately from the data deck.



## Cautions and Capacities

The capacity of a pseudo-deck is essentially unlimited; a deck created by the MCP LDCNTL utility contains as many as 100,000 card images. The maximum number of pseudo-readers which can be active simultaneously is limited by the MCP Pseudo-Reader Cross-Reference Directory (PCRXRF) to 80 per processor. Note that this number equals the capacity of the largest possible schedule table (JRT) and consequently, it is possible to load many decks to disk and activate a great many readers to completely fill the schedule. If this occurs, the MCP notifies the operator and does not activate any decks which would overflow the schedule. Any such decks will be activated automatically when the schedule is no longer full. This does not preclude the activation of readers containing only DATA files or other control statements which do not cause the initiation of a job.

If this condition occurs due to APCR processing, the user can employ the SD keyboard message to set a deck limit below the size of the schedule table.

The maximum number of pseudo-decks which can be on disk simultaneously is 9999 for each processor. This number exceeds the number of readers which can be active simultaneously. If the PCRXRF file becomes full or the user specified deck limit is reached, the MCP stops reader activation until space becomes available in the PCRXRF. Activation then resumes at the point of interruption if the APCR option is set. (The MCP cannot detect decks loaded with APCR reset under these conditions.)

## Control Card Linkage

Readers containing more than one set of control instructions (stacked decks) must have proper linkages between control cards. A typical stacked deck is:

```
? COMPILE A WITH COBOLL. COMPILE AND GO
? DATA CARD
. . .
? DATA X . DATA FOR GO-PHASE
. . .
? END
```

Proper handling by the MCP requires that the first DATA statement be linked to the second.

## When Not to Use Pseudo-Readers

Physical readers can be used instead of pseudo-readers if the card reader is not frequently used and no conflict exists with its use (for example, if little multiprogramming is done or if few programs use card files).

The additional disk or disk pack channel loading due to the use of pseudo-readers might also be a consideration. If disk is the device used most frequently by a mix of programs and only one job needs a card deck (no card reader conflicts) disk pseudo-readers will provide little benefit and can decrease throughput due to increased disk channel competition. The same is true for disk pack channel loading. Another reason for not using pseudo-readers is time considerations; the additional time necessary to load a deck to disk might extend the time to completion of a run beyond the desired time. This occurs rarely, however.

On occasion, the amount of disk or disk pack necessary to hold a large card deck is prohibitive and pseudo-readers might not be used for this reason; similarly, the small amount of memory needed for pseudo-reader processing can be a consideration. Some of this memory can be offset by declaring no alternate areas for the card file. however, this will probably reduce the card reader throughput.

## PRINTER ALTERNATES

### PRINTER ALTERNATES

Printer alternates are often called printer backup. The MCP provides three alternate media for handling print files: tape, disk, and disk pack. They are known respectively as Printer Backup Tape, Printer Backup Disk, and Printer Backup Disk pack, and are referred to as PBT, PBD, and PBP. (Care must be exercised to avoid confusion with the PBT, PBD, and PBP system options and the PBD and PBP input messages described below. Context will usually make the meaning clear.) Many considerations dictate user choice of one alternate medium or another. Significant user control can be exercised in respect to printer backup.

The user is unconcerned with the mechanism of printer backup and exercises control primarily over the disposition of the file to the primary device or alternate medium; how automatically this is to be done; and how the backup file is to be output to the primary device.

Every print file on the system is independent of all other print files. Several can be created simultaneously, from one or more programs, using any combination of media. A single program can simultaneously create printer files directed to all four possible media: the printer, backup disk, backup tape, and backup disk pack.

The ultimate printing of any backup file is also independent of that for any other backup file. The files may or may not be printed in the sequence created, as the user requires.

Other facilities include: multiple backup files can be created; multiple copies of backup files can be created; a backup file created on one processor of a multiple processor system can be printed by another processor.

Unlike card file processing, the user often wishes to control the disposition of a printer file to one of the four possible media: printer, backup tape, backup disk, or backup disk pack. A printer file can be routed to any of four media, rather than one which is primarily a line printer file with tape, disk, and disk pack as alternates.

Consequently, significant user control is provided for printer files, even to the extent of forbidding the use of the primary device. The user can control printer file disposition at three levels: by the source program file declaration; with execution time control cards; and/or when the file is OPENed by the program. This control can be summarized as:

#### Program (file declaration)

Files are assignable to a specific medium: printer, backup tape, backup disk, and backup disk pack; any available medium; or backup medium only.

#### Execution Time (file equate control card)

Files are selectable as for source program declarations.

#### File OPEN Time

A file which is not selected to a specific printer medium (that is, alternates are allowed) by the source declaration or FILE equation can be controlled at file OPEN time by the system level options and/or operator console syntax.

## ESTABLISHING A PRINTER MEDIUM

When a program executes an OPEN request for a printer file, the MCP attempts to find the selected device. The result of the search is dependent on the source program declaration, the setting of the system level options, and the availability of resources. The user has extensive control over the disposition of the printer file from the object program.

The source program specifies a file type for all declared files. While there is no reason for a source program to be concerned with the medium for card input, it is often desirable for the program to specify or restrict the use of media for printer files. Consequently, software provides the capability to make such an assignment in the file declaration. For a printer file the user has several choices:

The file must be assigned to a physical printer.

(SELECT <file-name> ASSIGN TO PRINTER NO BACKUP.)

The file must be routed to a printer backup tape (PBT).

(SELECT <file-name> ASSIGN TO PRINTER BACKUP TAPE.)

The file must be routed to printer backup disk (PBD).

(SELECT <file-name> ASSIGN TO PRINTER BACKUP DISK.)

The file must be routed to printer backup disk pack (PBP)

(SELECT <file-name> ASSIGN TO PRINTER BACKUP DISK PACK.)

The file can be directed to either the physical printer, printer backup tape or printer backup disk/disk pack. This can be done automatically by the MCP according to the availability of resources or by the system operator.

(SELECT <file-name> ASSIGN TO PRINTER.)

The file must not be assigned to a physical printer. Either printer backup tape or printer backup disk/disk pack is acceptable. The selection of the medium can be done automatically by the MCP or by the operator.

(SELECT <file-name> ASSIGN TO PRINTER BACKUP.)

On occasion the user wishes to override the source program file type declaration for a particular execution of the job. (To change the file type permanently, a source program change and recompilation is necessary.) This override is accomplished by the FILE (label) equate control statement which accompanies the EXECUTE or COMPILE control card. For example, a program might have declared:

```
SELECT PRINFL ASSIGN TO PRINTER.
```

## PRINTER ALTERNATES

This implies that a physical line printer or any backup medium is acceptable. The user might wish to change this for a particular execution. The possibilities are:

- ? FILE PRINFL = <file-ID> PRO (printer only)
- ? FILE PRINFL = <file-ID> PBT (printer backup tape)
- ? FILE PRINFL = <file-ID> PBD (printer backup disk)
- ? FILE PRINFL = <file-ID> PBP (printer backup disk pack)
- ? FILE PRINFL = <file-ID> PRN (printer)
- ? FILE PRINFL = <file-ID> PBK (backup printer)

The <file-identifier> is the VALUE OF ID desired for the file. The operation of the execution time override shown above makes the file appear to have been SELECTed to PRINTER NO BACKUP, BACKUP TAPE, BACKUP DISK, BACKUP DISK PACK, or BACKUP, respectively.

When a program executes an OPEN request for a printer file, the MCP must assign the file to a particular medium. A source language declaration or a FILE equate override of a specific medium fully satisfies the responsibility of selecting the desired medium, and the MCP need only find a device of the type specified. For PRINTER NO BACKUP files, the MCP searches the Input/Output Assignment Table (IOAT) for an available physical printer. For BACKUP TAPE files, the IOAT is first searched for an existing printer backup tape on which the current file may be stacked (unassigned and not re-wound), or failing this, an available scratch tape. In addition, if the printer backup tape file is blocked, a variable size memory area must be obtained. For BACKUP DISK files, the MCP searches for 4 KD of user memory for pseudo printer use. If the necessary resource can be found, the file is assigned. If file assignment cannot be completed, the operator is notified of the need for the selected medium and the program is suspended until the needed resource becomes available or the operator forces assignment to a different medium.

If alternate handling of the file is allowed (ASSIGN TO PRINTER or ASSIGN TO BACKUP), the responsibility for selecting the final medium is placed on the MCP or the system operator. The third level of user control specifies the limits of automatic MCP assignment of such files. Four system level options known as PRN, PBT, PBP, and PBD can be set or reset at any time to permit or forbid automatic assignment of the corresponding medium by the MCP. The possible combinations are summarized in table 4-3.

When a PRINTER file is OPENed, the MCP evaluates the system options in the following order: PRN, PBT, PBP, then PBD and attempts to establish the appropriate device. Failing this (because of insufficient resources and/or because the options are reset), the operator is notified and the program is suspended. When resources become available and/or the system options permit, automatic assignment is completed. During this interim the operator can use a console message to direct the file to the desired medium.

The same procedure occurs for BACKUP files except that any check for a physical line printer is bypassed.

A zero (0) indicates that the option is reset, a one (1) indicates the option is set and automatic assignment is allowed.

These options do not apply to files assigned to PRINTER NO BACKUP, PRINTER BACKUP TAPE, PRINTER BACKUP DISK, and PRINTER BACKUP DISK PACK, as no choice of medium is allowed to the MCP for these cases. However, if the requested medium is not available when the file is OPENed, the operator can force assignment to a different medium.

Table 4-3. Printer Backup Combinations

System Level Options in priority order				Automatic MCP Assignment Limit PRINTER/BACKUP Files
PRN	PBT	PBP	PBD	
0	0	0	0	No automatic assignment, complete operator control.
1	0	0	0	Only a printer can be used for PRINTER files. BACKUP files cannot be automatically assigned.
0	1	0	0	Only backup tape can be used for PRINTER and BACKUP files.
0	0	1	0	Only backup disk pack can be used for PRINTER and BACKUP files.
0	0	0	1	Only backup disk can be used for PRINTER and BACKUP files.
1	1	0	0	A printer or backup tape can be used for PRINTER files. Only backup tape can be used for BACKUP files.
1	0	1	0	A printer or backup disk pack can be used for PRINTER files. Only backup disk pack can be used for BACKUP files.
1	0	0	1	A printer or backup disk can be used for PRINTER files. Only backup disk can be used for BACKUP files.
0	1	1	1	Any backup medium can be used for PRINTER and BACKUP files.
1	1	1	1	Any medium can be used for PRINTER. Any medium is allowed for BACKUP files.

Print files can be declared as requiring special forms either in the source program file declaration or in a FILE equate override. This specification is in addition to any selection of printer medium. If the file is assigned to PRINTER NO BACKUP, or if no backup system options are set before searching for an available device, the MCP must inform the operator that special forms are required for the file. The operator can then mount forms on a line printer and direct the file to that printer, or divert the file to an alternate medium. If a backup file is produced, it contains a flag specifying that special forms are required when the file is ultimately printed on a line printer.

## LABEL PROCESSING

When a suitable device has been assigned to the program, the MCP executes the appropriate label procedures.

For a physical printer, the label is written unless the file is declared unlabeled, or the file-identifier is the name ABSENT. This special identifier informs the MCP that, even though the file is declared labeled, labels are not to be written to the physical printer. For backup files, labels are always written but the beginning label is flagged if the file is declared and unlabeled. This informs the printer utilities to bypass label processing when the file is printed.

## PRINTER ALTERNATES

If the file has been declared as FORM (special forms required), the backup file label is marked to reflect this. When the file is printed the print utility causes the operator to be informed that forms are required and must be mounted.

Program beginning and ending label handling USE routines are permitted regardless of medium.

## CLOSING PRINT FILES

When a program executes a CLOSE request for a print file, or when a program terminates (normally or abnormally) with a file still OPEN, file CLOSE routines of the MCP are invoked. The action taken is dependent on the CLOSE type and the file medium.

For physical printer files, two close types are meaningful: CLOSE or CLOSE WITH RELEASE. For files directed to the physical printer, CLOSE directs the MCP to perform the normal close mechanism (including the ending label if applicable) but not to release the printer to the system. The device remains attached to the program print file FIB and can be reOPENed at any time. This is done when the printer file is to be reOPENed in a short time. CLOSE WITH RELEASE accomplishes the same action as CLOSE except that the device is released to the system for possible reassignment.

Files routed to backup tape act in much the same way except that labels are always written. When a printer backup file on tape is CLOSED, the tape is not rewound (whether RELEASEd or not), to permit other print files to stack on the same tape. The simple CLOSE syntax is valuable when the user wishes to stack several print files for the same program on a single tape, perhaps for printing off-line. The same file description (FIB) must be used for each file since the device is attached to a particular FIB.

Files routed to backup tape act in much the same way except that labels are always written. When a printer backup file on tape is CLOSED, the tape is not rewound (whether RELEASEd or not), to permit other print files to stack on the same tape. The simple CLOSE syntax is valuable when the user wishes to stack several print files for the same program on a single tape, perhaps for printing off-line. The same file description (FIB) must be used for each file since the device is attached to a particular FIB.

Files routed to backup disk are automatically CLOSED WITH RELEASE by the MCP regardless of the user declaration of CLOSE type, since a simple CLOSE is meaningless for such a file. Whenever a PBD or PBP is CLOSED, the MCP inserts the name assigned to the file (@pnnnn) into the LABMFD label area field of the program.

In all cases, if a program terminates without having CLOSED a print file, the file is automatically CLOSED WITH RELEASE in the manner described above. Allowing the MCP to CLOSE print files bypasses any program USE routine.

Whenever a printer backup file (tape or disk) is CLOSED, the operator is informed through the SPO. Whenever a FORM file attached to a physical printer is CLOSED WITH RELEASE, the operator is informed and the device is SAVEd so that the forms can be removed and no other files can be assigned to that printer until it is made ready by the operator or until a file is specifically directed to it.

## REOPENING PRINT FILES

Print files which are CLOSED (plain or RELEASE) do not revert to original programmatic (or FILE equated) status. Thus a PRINTER BACKUP file which is OPENed, routed to PBT and then CLOSED WITH RELEASE, will be routed to PBT if OPENed again.

## BACKUP PRINT FILES

There are two types of backup print files: printer backup tape and printer backup disk or disk pack.

### Printer Backup Tape

The printer backup tape can produce either blocked or unblocked backup files. Blocked printer backup file creation requires a memory area outside the program bounds used to hold a block of twelve print images plus pointers needed for printer backup control. The backup tape routines consist of MCP coding which divert printer images and page control information to a magnetic tape. Provision is made for multiple reel printer backup tape files. Further, multiple print files (blocked and/or unblocked) can reside on the same magnetic tape, one file stacked behind the previous one (stacked printer backup tape).

### Backup Printer Disk or Disk Pack

Printer backup disk file creation requires a memory area (outside of program bounds) which is used as a buffer and holds a block of 10 print images as well as storage for the DFH for the backup print file. The backup disk or disk pack routines consist of the MCP coding which diverts print images to the memory buffer area and writes the block to disk or disk pack as the buffer becomes full.

The user (operator) is not involved with any aspect of the mechanism of printer backup except when the installation has elected to reset the system options and allow the operator to control assignment by the OU/FM console syntax. This mechanism permits the operator to divert print files to the desired medium at file OPEN time.

## BACKUP TAPE FILES

Printer backup files can reside on any type of magnetic tape. Seven-track PBT files are written non-standard (even parity) to accommodate off-line printing on a satellite system (such as a B 300).

A printer backup tape file has the standard format for tape files on the system, and contains records consisting of control information and print images. The format of the file is:

- Beginning label
- Tape mark
- Data Records
- Tape mark
- Ending label

Other files of the same format can follow the first. (A tape mark follows the last, or only, ending label.) This multiple file printer backup tape is known as a stacked printer backup tape.

All print files routed to backup tape are labeled. If the program declared an unlabeled printer file, the beginning label of the backup file is flagged so that the labels are not output by the print utility.

The beginning label has the format given in table 4-4.

The Creator and Charge Number fields are not placed in the label if the necessary area contains any non-zero or non-blank characters. This avoids overlaying any user inserted data.

Printer backup tapes always have a multifile identifier of BACKUP to define the tape as a PBT and to allow the tape to be multifile if stacking is to take place. To avoid operator confusion, installations

PRINTER ALTERNATES

**Table 4-4. Backup Print File Label Format**

Name	Location	Length	Contents/Function
	LABEL: +0	1 UA	Blank
LABEL	LABEL: +1	7 UA	LABEL to verify presence of label record.
MFID	LABEL: +9	6 UA	BACKUP – multiple file identifier.
FID	LABEL: +17	6 UA	User declared file identifier.
Creator	LABEL: +58	13 UA	Program name/compiler name if compile listing; program name if user program output.
Charge Number	LABEL: +72	6 UA	Charge number of creator.
Form Flag	LABEL: +78	1 UA	1 = forms required.
Label Flag	LABEL: +79	1 UA	:1 = print file declared labels omitted :4 = ignore channel 12 (RPG)

must avoid using the file name BACKUP for any other tape files. The data records are blocked depending on the setting of the LIMIT PBTBLK specification (refer to the MCPVI Software Operations Guide, Volume 1 form number 1127529) when the printer file was OPENed.

The format of the data record is

Field	Length	Contents/Function
Control Information	4 UA	Paper Motion Control.
Print Image	132 UA	Meaningless for skip records; otherwise, user created print line.

There are two types of data records: print records and skip records. Both are of the same format and are distinguished by a bit in the control information field. The printer records specify that a line is to be printed, and paper motion might occur. The skip records indicate only paper motion; consequently the contents of the print image field is immaterial. The detailed meaning of the control information field is explained in this section under Printer Control Data.

## BACKUP DISK FILE

Printer backup disk and disk pack files consist of label records and data records. The data records consist of control information and printer images. The important characteristics of the file are

File Attribute	Disk	Disk Pack
Record Size	140 UA	144 UA
Blocking Factor	10	10



File Attribute	Disk	Disk Pack
Records per Area	2250	2500
Segments per Area	3150	2000
Number of Areas	100	100
Maximum File Capacity	225000 Records	250000 Records

Since disk and disk pack files have no labels and a PBD file is essentially a normal disk file, the first record of the PBD file is the beginning label of the print file and the last record is the ending label. The labels are of standard format and are identical to PBT labels except that the MFID need not be BACKUP.

Unlabeled files can be routed to backup disk; however, the first and last records of the backup file are MCP generated label records, flagged to indicate that the labels are not to be output by the print utility.

Note that the first and last records of the backup disk files are simply disk records like any other, but are created from the label records which apply to the printer file. The other records in the file are data records containing control information and print images. The format is identical to PBT records except that PBD records contain a 4-byte filler at the end and PBP records contain eight bytes of filler.

Printer backup disk files are identified in the disk or disk pack directories by names of the form:

@pnnnn

where p is the number of the processor on which the file was created and nnnn is a unique 4-digit number assigned by the MCP when the file is entered into the Disk Directory. Numbers are assigned starting with 0001 and incremented as each new file is created. The counter is reset when it reaches 9999.

As with other pseudo files on disk, the name of the pseudo file must be carefully distinguished from any other data file within it. Since the printer backup disk file is simply a disk file as far as the Disk Directory is concerned, it must have a unique name. This name is of no concern to the program creating the file, but necessary for operator reference and for program output from the backup file to the primary device. Within the backup file is the print file, which carries a user assigned name.

### Printer Control Data

The first four bytes of printer backup records are printer control data. This printer control information consists of four characters extracted from the parameters which accompany the branch communicate (BCT) instruction generated by the compilers from the user program WRITE syntax. The code generated is:

BCT 0234 (call MCP WRITE routine)  
 BUN \*: +24 (to next instruction)  
 ACON FIB (pointer of file)  
 ACON EOP (ASSEMBLER only: end of page (EOP) branch if 12-punch sensed)  
 CNST 4 UN (space and skip variants)

## PRINTER ALTERNATES

The space and skip variant field from the communicate is placed in the backup file records. The CNST contains the necessary printer control information for the WRITE operation. If the line is to be output to the printer, the 4-digit field is placed into the variant field of the printer I/O descriptor. The format of the field is:

0XYY

where X is the number of lines the printer spaces after the line is printed (X can be 0, 1, or 2) and YY is the carriage tape channel to which the printer skips after the line is printed. (YY can be from 00 to 11; 00 specifies that channel skipping does not take place.) If both spacing and skipping variants are present, skipping takes precedence. For example, the COBOL statement:

```
WRITE RECA OF FILEA BEFORE ADVANCING 2 LINES. would generate:  
BCT 0234  
BUN *: +24  
address of FILEA FIB  
000000 (no EOP branch in COBOL)  
0200
```

The last field specifies to print and space two lines. Another example, the statement:

```
WRITE ... BEFORE ADVANCING CHANNEL 1.
```

would generate a variant field of 0001. A variant of 0011 could have been generated by:

```
WRITE ... BEFORE ADVANCING CHANNEL 11.
```

and a variant of 0100 could have occurred due to the statement:

```
WRITE RECA BEFORE ADVANCING 1.
```

Since the printer hardware can space only zero, one, or two lines per operation (print and space, or space only), and because the device first prints and then spaces, certain operations cannot be handled simply by a WRITE request. For example, to write a line and then skip three lines, a single physical WRITE cannot accomplish the action; both a write and a space operation are required. The WRITE has been previously discussed. The MCP also provides the capability to specify that printer positioning is desired with no printing. The format of the communicate for a position (POSN) request is:

```
BCT 0394 (call MCP POSN routine)  
BUN *: +26 (to next instruction)  
ACON FIB (pointer to file)  
ACON EOP (ASSEMBLER only: EOP branch)  
CNST 6 UN SXXYY0 (space and skip variants)
```

This communicate is used for a request to position either printer, magnetic tape, or disk files. For printer files, the S (sign) variant is not used and must be zero. The SS and YY fields are similar to the corresponding variants for the WRITE communicate except that XX can take on values from 00 to 99 but the entire variant cannot be zeros.

The variant field in the position communicate is not placed directly into the I/O descriptor as for a WRITE, since the printer can space only two or fewer lines. Instead, the MCP evaluates the number

## PRINTER ALTERNATES

and (for the physical printer) builds sufficient I/O requests to accomplish the desired action (for instance, a POSN four lines requires two physical space operations, each spacing two lines).

The position request might occur if by spacing some number of lines and then printing, or skipping to a channel and then printing. These actions are opposite to the mechanism of the printer. (the COBOL programmer does not directly request a position operation. Instead, certain variants of the syntax will cause a POSN communicate to be generated in addition to a WRITE request, for example, WRITE...AFTER.)

To print a line and space five lines, the code might be:

```
BCT 0234
BUN *: +24
ACON FIB
ACON EOF
CNST 4 UN = 0200 (write and space two lines)
BCT 0394
BUN *: +26
ACON FIB
ACON EOF
CNST 6 UN = 0030000 (space three lines)
```

If the file was attached to a physical printer, the first communicate would cause the MCP WRITE routine to place the variants into the printer I/O descriptor and issue a physical I/O. This would cause the line to be printed and two lines spaced. The second communicate tells the MCP POSN routine to initiate sufficient spacing operations to space the desired number of lines. For example, the routine would space the printer two lines, and then space one line to give the total of three.

If the printer file is being passed to an alternate medium, several special considerations must be noted.

The printer control information from the WRITE or POSN request must be placed into the backup file record.

The records must distinguish between WRITE records and POSN records.

The MCP need not generate the individual physical operations for a position request which is diverted to backup; this can be done when the backup utility prints the file. Instead, the variant field can be passed to the backup file rather than causing one or more separate skip requests.

These operations are accomplished by the pseudo printer logic of the MCP. When the user requests a printer operation and the file is being diverted to a backup medium, the MCP extracts the 4-digit control field from the communicate parameters and moves them to a 4-byte field just before the print line. (Before each buffer, the compilers allocate four bytes of space used during backup operation to hold the control variants.) If the record is destined for a backup medium and the request is for a SPACE or SKIP operation, this fact must be flagged by the MCP position routine which turns on the 2-bit in the third variant digit (the high order digit of the skip portion of the field). The 4-digit variant moves to the 4-byte field ahead of the print buffer.

For example, variant 0120 means space one line, 0520 means space five lines. Variant 0021 specifies skip to channel one; 0030 specifies skip to channel 10. In all cases, the contents of the print line is immaterial as the 2-bit specifies that paper motion is to occur without printing.

## PRINTER ALTERNATES

When the WRITE and POSN routines finish any needed procedures with the control variants, other printer backup logic decrements the beginning address of the buffer in the I/O descriptor by eight (digits) to include the variant field in the buffer and in the backup file.

## ACCESS AND CONTROL OF PRINTER BACKUP FILES

The following describe various details in handling printer backup files.

### Accessing Printer Backup Files Programmatically

Programmatic accessing of backup files is described in detail in User Supplied Print Utilities.

Backup tape files can be accessed by executing an OPEN request for a file with the name BACKUP/ bbbbbb (blanks).

Backup disk files are accessed either by specific name (@pnnnn) or by the generalized printer backup name (@00000) which requests the next file in sequence.

### Operator Interface to Printer Backup Files

The operator has control over printer backup files through keyboard input messages as follows:

#### Printer Backup Tape Files

##### Display Backup Tape Label

OL <cc/u> directs the MCP to display the identifier of the indicated unit (which may contain a backup tape). If the tape is stacked, only the identifier of the first file is displayed.

##### Direct a File to Backup Tape: <mix no> OU <cc/u>

If a program is suspended due to the lack of a suitable printer medium, the operator can direct the file to backup tape. This format can also be used to route the file to a physical printer. The FM message can be used to route a file to a printer or backup tape if special forms are required.

##### Remove a Backup Tape File

PG <cc/u> directs the MCP to purge the tape on the designated unit. All files on the tape are then unavailable.

##### Initiate Printer Backup Tape Utility

$$\text{PB} \left[ \begin{array}{l} /OWN \\ /MCP \end{array} \right] \langle \text{cc/u} \rangle \quad [ * ] \quad [ \langle \text{file-identifier} \rangle ] \quad [ \langle \text{integer} \rangle ]$$

directs the MCP to initiate the backup tape print utility to output the designated backup tape file on the designated unit, or all files on the backup tape.

##### Bypass Part of a Backup Tape File During Processing

<mix no> SK <integer> directs the specified print utility to skip the designated number of backup file records, then resume printing.

Terminate Processing of a Backup Tape File:

$$\langle \text{mix no} \rangle \quad \text{QT} \quad \left[ \left[ \begin{array}{c} \text{END} \\ \text{ALL} \end{array} \right] \right]$$

directs the specified utility to terminate processing immediately (QT ALL); terminate at the end of the current file even if more files remain to be processed (QT END); or terminate processing of the current file and go on to the next (QT).

Printer Backup Disk Files

Display Backup Files on Disk or Disk Pack

BF PRN/ or BFP PRN/ causes the MCP to list all printer backup disk or disk pack file numbers and the names of the printer files within them. BF [P] PRN <file-identifier> or BF [P] <file-number> can be used to list specific backup files.

Direct a Print File to Backup Disk or Disk Pack: <mix no> OUDK

If a program is suspended because the MCP cannot complete assignment on the printer files, the operator can direct the file to backup disk. <mix no> OUPK is used for assignment to disk pack.

Change a File Name to a Backup File Name

CHANGE <file-identifier> TO @00000 changes file-identifier to a backup disk file name (@nnnn). The file must be in the correct backup disk file format as only the name is changed.

$$\text{CH} \quad \langle \text{file-identifier} \rangle \quad \text{TO} \quad @ \quad \text{ON} \quad \left\{ \begin{array}{c} \text{cc/u} \\ \text{pack-ID} \end{array} \right\}$$

is used for disk pack.

Remove a Printer Backup Disk File

RF [P] PRN <file-number>, RF [P] PRN <file-identifier>, or RF [P] PRN/ directs the MCP to remove the specified printer backup disk or disk pack files from the directory if not in use.

Initiate Printer Backup Disk Utility

$$\underline{\text{PB}} \quad \left[ \begin{array}{c} \text{P} \\ \text{D} \end{array} \right] \quad \left[ \begin{array}{c} \text{/OWN} \\ \text{/MCP} \end{array} \right] \quad [ * ] \quad \left\{ \begin{array}{c} / \\ \langle \text{file-number} \rangle \end{array} \right\} \quad [\text{SAVE}] \quad [ \langle \text{integer} \rangle ]$$

initiates a print utility to output the designated print backup disk file or print backup disk files.

Bypass Part of a Backup Disk File During Processing:

$$\langle \text{mix no} \rangle \quad \text{SK} \quad [ - ] \quad \langle \text{integer} \rangle$$

directs the specified print utility to skip forward or backward the designated number or backup file records, then resume printing.

## PRINTER ALTERNATES

Terminate Processing of a Backup Disk File:

$$\langle \text{mix no} \rangle \quad \text{QT} \quad \left[ \left\{ \begin{array}{l} \text{END} \\ \text{ALL} \end{array} \right\} \right]$$

See above for backup tape.

## PRINTER BACKUP UTILITIES

The MCP provides intrinsic utility programs to output backup files to the physical printer. The programs used to output printer backup tape files and printer backup disk and disk pack files are named, respectively, PBTOUT and PBDOUT. Both programs have several characteristics in common:

Each can output a single selected backup file or several files in sequence.

Either can be interrupted during processing with considerable flexibility. See SK and QT messages above.

Both print programs can be restarted, beginning printing at almost any point in the backup file. If restart is requested, the print program requests the necessary data from the operator and positions the file to the restart point. Two bypass mechanisms are available:

1. The operator can request a restart by specifying that positioning is to occur to a specific record based on the contents.
2. The operator can request that a specific number of backup file records (both data and paper-motion records) be bypassed.
3. The PBTOUT intrinsic can, optionally, be directed to restart the print file from the point at which the file was QTed.

Each program can accommodate backup files which are user declared to require the use of special forms.

### Backup Tape (PBTOUT)

The following tells how the PBTOUT program can be used.

#### *INITIATION*

The user has two options for the initiation of the PBTOUT utility: printing of the file(s) can be automatically initiated by the MCP or by the operator through a keyboard message. If the user wishes automatic initiation, the system option APBT must be set by the SO console message. The MCP will initiate PBTOUT whenever a printer is available and a non-rewound and accessible printer backup tape is found. If such a condition exists, the print program is scheduled for execution, the tape is rewound and both the tape and the printer are reserved to insure availability when required by the print program.

If direct operator control is desired, the APBT option must be reset (RO message) and the PB <cc/u> keyboard message is used to initiate the utility. When the PB message is entered, the print utility is scheduled for execution, the designated drive is reserved for the program and, if necessary, the tape is rewound. Two basic options are provided when the operator initiates printing: 1) all files on a

PRINTER ALTERNATES

(stacked) tape can be printed (PB <cc/u>) or 2) a specific file can be printed (PB <cc/u> <file-identifier>).

The operator can include a /OWN or /MCP parameter to require the initiation of either the user-coded or MCP version of the utility. In the absence of this parameter, the user-coded version is initiated if present on disk; otherwise, the MCP intrinsic is scheduled.

When the operator initiates printing, restart or skipping can be selected. This is done by a variant of the PB message: PB <cc/u> \* <file-identifier> where the asterisk indicates that the operator wishes to specify a restart point in the file, or a number of tape records are to be bypassed before printing begins. (PB <cc/u> \* without a file-identifier indicates a restart for all files on the tape.) The next specification in the PB request can be a 1 or 2-digit integer (0-99) which is passed to the print program. This number is not currently used by the MCP supplied utility but can be employed in any manner desired in a user-coded PBTOUT (see below). Full syntax for the PB message is:

$$\text{PB} \left[ \begin{array}{c} \text{/OWN} \\ \text{/MCP} \end{array} \right] \text{ <cc/u> } [ * ] [ \text{<file-identifier>} ] [ \text{<integer>} ] [ ; \text{ parameter instructions} ]$$

The parameter instructions can include any valid program parameter statement such as CHARGE, LOCK, AFTER, and so on. If the Cold Start CHR G ALL specification has been made, the CHARGE parameter instruction must be included.

*OPERATION AND CONTROL*

When the print program begins execution, the tape on the appropriate channel and unit is OPENed. If the operator has requested the printing of a single file on a stacked printer backup tape, the MCP multifile search routine finds the appropriate file; otherwise, the first file on the tape is selected. The label record is accessed to determine if FORMs and/or labels are required for the printer file. If a restart (\*) is specified, the print program requests the necessary data from the operator and positions the file to the desired point.

After necessary positioning of the file, a printer is OPENed and printing begins. During printing, the operator can use the SK and QT messages to initiate special actions in the print program as described above. (When printing a PBT, for the last (or only) file on the tape, QT and QT ALL mean the same thing; however, if QT is used, PBTOUT may have to space to the end of the current file to determine if any more files are present.) When printing of the files on the tape is completed, the tape is rewound, but not purged, to allow multiple printings.

**Backup Disk/Disk Pack (PBDOUT)**

The following tells how PBDOUT can be used.

*INITIATION*

PBDOUT is initiated by a keyboard message (PB, PBD, or PBP...) The operator can indicate that all files are to be printed in sequence according to numeric designations (PB/ , PBD/ , or PBP/), or that a specific file is to be printed (PB , PBD , or PBP <integer> ) where the integer is the numeric portion-nnnn or pnnnn-of the PBD file name. The pnnnn format can be used to print a file on a processor other than its creator.

## PRINTER ALTERNATES

The operator can include a /OWN or /MCP parameter to require the initiation of either the user-coded or MCP version of the utility. In the absence of this parameter, the user-coded version is initiated if present on disk; otherwise, the MCP intrinsic is initiated.

Restart or skipping can be specified by typing PB\*, PBD\*, or PBP\*... ; the asterisk indicates that the operator wishes to specify a restart point or a number of records to be bypassed before printing begins. PB\*/, PBD\*/, or PBP\*/ indicates restart for all backup disk/pack files.

The operator can also specify that the PBD files are to remain on or off disk after printing by specifying the word SAVE if the files are not to be purged. If no specification is made, the files are not SAVEd.

The next specification in the PBD request can be a 1 or 2-digit integer (0-99) which is passed to the print program and indicates the total number of copies of the backup file to be printed. If either a zero or a one is entered, or if the parameter is omitted, PBDOUT generates one copy of the file. Full system syntax for the PBD message is

$$\text{PB} \begin{bmatrix} \text{P} \\ \text{---} \\ \text{D} \end{bmatrix} \begin{bmatrix} \text{/OWN} \\ \text{---} \\ \text{/MCP} \end{bmatrix} \left[ \text{*} \right] \left\{ \begin{array}{c} \text{/} \\ \text{---} \\ \text{<file-number>} \end{array} \right\} \left[ \text{SAVE} \right] \left[ \text{<integer>} \right] \left[ \text{; parameter instructions} \right]$$

If the Cold Start CHRG ALL specification has been made, the CHARGE parameter instruction must be included.

When the printer program is initiated, the MCP passes it the file number (zeros if the message was PB/, PBP/, or PBD/), flags if \*, and/or SAVE were specified and the copy parameter. The MCP does not establish that the desired file is in the Disk Directory before initiating the print program.

### OPERATION AND CONTROL

When the print program begins execution the appropriate disk file is OPENed. If a particular file is specified, it is accessed; otherwise, the PBD file with the lowest number for that processor is requested of the MCP.

The first record is read to determine if FORMs and/or labels are required for the printer file. If restart (\*) is requested, the print program requests the necessary data from the operator and positions the file to the desired point.

During the printing the operator can use the SK and QT message to initiate special actions in the print program as described above.

When a file has been completely printed, it is CLOSED WITH PURGE unless the operator has specified SAVE in the PBD or PBP message. (The file is not PURGEd if QT or QT ALL is specified since the file is not completely printed for these cases.) If the original request was for the printing of a single file, PBDOUT terminates.

If the operator requests that all files be printed, PBDOUT requests the MCP to OPEN the backup print file with the next higher number for that processor. The process continues as described above until all PBD or PBP files on the processor have been printed or the operator terminates the program. (any PBD files CLOSED by programs during the processing of a PBD/ request are also accessed and printed in turn without operator intervention.)



## USER SUPPLIED PRINT UTILITIES

The user may wish to write his own version of a print utility. The following information may be helpful when writing a user-coded version of the utility.

### General Considerations

The MCP provides the user with the means to write individualized versions of the print intrinsics. These program can use the same interface with the MCP and the operator as the MCP intrinsics. Keyboard input to initiate and control user-coded utilities is the same for the MCP utilities, as are the parameters passed to the utilities by the MCP.

When the operator request a print utility and does not specify the /OWN or /MCP parameters, the MCP scans the Disk Directory for a program with the appropriate name for the utility. If such a file can be found, the user utility is invoked in the same manner as would be the MCP intrinsic; if no user-coded version is present on disk, the MCP utility is initiated.

If the /OWN parameter is specified, the user-coded version must be present or no utility is initiated. If /MCP is specified, the MCP intrinsic is scheduled whether a user-coded version is present or not.

As there are some differences in the methods of accessing PBT and PBD files, separate discussions follow. The record access and handling mechanisms are similar. The individual PBT and PBD discussions indicate where differences exist, or where special methods can be applied to improve the performance of the print program.

### General File Handling

After the file has been accessed (as described below) the utility executes READs on the data records. The print program must determine whether the record is a data record or a papermotion record specifying spacing or skipping, by testing the 2-bit in the third control digit. If the bit is not on, the control information fields must be moved to the variant digits of a printer WRITE communicate. If the bit is on, the bit must be reset and the field moved to the variant digits of a POSN communicate.

### User-Coded PBTOUT

The user-coded version of the printer backup tape utility, which accesses printer backup tapes, must be named PBTOUT to use the same operator interface as the MCP utility. However, any program can access a printer backup tape file as the file access mechanism is not dependent on the program name (except where operator communication is involved: PB, QT, and so on). The MCP passes the following data to the print program when initiated by the PB message:

Name	Location	Length	Contents/Meaning
Index Field	BASE: + 32	5 UN	Always zeros for MCPVI
Variants Flag	BASE: + 37	1 UN	
Star Flag		:1	Reserved
File Flag		:2	* specified in PB message
Code Flag		:4	Single file printout
		:8	Code entered
Code	BASE: + 38	2 UN	00-99; value keyed by operator in PB message. Contains 00 if nothing entered by operator (Code Flag off)

PRINT UTILITIES

The Index Field is not used by MCPVI and is set to zeros.

The Star Flag specifies that PB <cc/u> \*...was entered by the operator and used to indicate a restart, though the user can employ the flag in any manner desired.

If the operator entered the Code PB parameter, the number is placed in the Code field (right-justified) and the Code Flag is set. The code can be used any way the user wishes; typical uses might include a specification of the number of copies to be printed or a coding for a group of files to be printed together.

If a file identifier was included in the PB request, the File Flag is set. The setting of this flag affects the PBDOUT file handling mechanism.

Normally, the print program will have one tape and one printer file declared.

The input tape file must be declared as follows:

```

Hardware type:      TAPE
Multifile identifier:  BACKUP
File identifier:    Must be blanks
Record size:       136 UA
Blocking factor:   30
Label records:     Standard
Ignore short/long records  True
    
```

The printer file must be declared:

```

Hardware type:      PRINTER NO BACKUP
File identifier:    Any
Record size:       132 UA
Blocking factor:   1
Label records:     Standard
    
```

When the tape file is OPENed, the MCP finds and attaches the required file. If a single file printout was requested in the PB message, the MCP searches the tape for the appropriate file. If no file name was requested, the first file on the tape is selected.

After OPENing the tape file, the label must be accessed to find the file-identifier, the FORM flag and the label-type flag. (The label area is addressed by the field FIBLAB--FIB: +188. In particular, this address points to the ninth character position of the label which is the byte before the multifile identifier field (LABMFD). The file-identifier (LABFID) begins eight bytes beyond the MFID field.)

The following fields in the printer FIB must be modified if necessary:

Field	Location	Length	Contents/Meaning
FIBLBL*	FIB; +13	1 UN	0 = Standard label 1 = Omitted label
FIBSPF	FIB: +33	1 UN	:1 = If FORM is required
FIBFLM	FIB: +128	1 UN	:4 = Ignore channel 12

NOTE

\* Rather than changing FIBLBL, omitted labels can be indicated by specifying a print file identifier of ABSENT which causes the MCP to bypass physical printer label processing even if the FIB specifies a labeled file.

After modifying the printer FIB, the desired portions of the tape label must be moved to the printer label area, as well as any other data the user wishes to insert. The printer file can then be OPENed.

Other fields with which the user might be concerned include:

Field	Location	Length	Contents/Meaning
Quit Flag	BASE: +0	1 UN	Set if QT message entered by operator :1 = QT :2 = QT END :4 = QT ALL
Skip Count	BASE: +2	6 un	Integer entered in SK message (First digit = @D@ IF SK-nnnnn)
FIBST2	FIB: +1	1 UN	:1 = File OPENed optionally; set when all files on tape have been accessed
FIBCHN	FIB: +148	2 UN	After OPEN, channel on which tape located
FIBUNT	FIB: +150	1 UN	Tape drive unit designation

The tape records can be accessed and printed in the manner in this section under General Considerations. The print program need not be concerned with the blocking factor of the file. However, greater efficiency can be achieved if the print program performs the record unblocking. The size of the block can be determined by subtracting the contents of the FIBACE field in the Buffer Block for the current block, from the beginning address in the corresponding I/O descriptor.

When EOF is reached, the tape file must be CLOSED WITH RELEASE if a single file printout is requested (File Flag set); otherwise, the file must be CLOSED NO REWIND and PBTOU then moves spaces to the file identifier field of the tape label area. The program executes an OPEN NO REWIND to access the next file. This blank identifier requests that the MCP assign the next sequential file on the multifile backup tape. If all files on the tape have been accessed and the program requests the next file, the MCP executes an optional OPEN for the file, causing PBTOU to take an EOF branch on the first tape READ. The program can distinguish between this condition and an empty print file. FIBST2:1 will be set for the end-of-tape (EOT) case.

When multiple files are being printed, the printer must be CLOSED after each file unless the current file or the next file is declared FORM in which case the printer must be CLOSED WITH RELEASE. In the former case, CLOSE prevents any other program from obtaining the printer. In the latter case, the printer must be RELEASEd so that the operator can be notified to dismount the form (if the current file is FORM) or can select the correct printer (if the next file is FORM).

### User-Coded PBDOUT

A user-coded version of the printer backup disk utility must be named PBDOUT to avail the same operator interface as the MCP utility. The file access mechanism is not dependent on the program

PRINT UTILITIES

name, and any program can access a PBD or PBP file. When PBDOUT is initiated by the PB, PBD, or PBP console message, the MCP passes it the following data:

Name	Location	Length	Contents/Meaning
File Number	BASE: + 32	5 UN	File number requested in PBD message or zeros if PBD/ requested
Variants Flag Save Flag  Star Flag  Code Flag	BASE: + 37	1 UN	:1 SAVE specified in PBD message :2 * Specified in PBD message :4 Disk Pack backup (PBP entered) :8 Code entered
Code	BASE: + 38	2 UN	00-99 as entered by operator, 00 if no specification is made (Code Flag off)

The File Number is the numeric portion of the name of the PBD file (@pnnnn); if PB/, PBD/, or PBP/ is requested, the field contains zeros. The use of this field is described below.

The Variant Flag digit specifies that other data was entered by the operator. While normally used to control file disposition and/or restart, the user can place any meaning on the flags.

The Code parameter can be used any way the user wishes. Typical uses might include a specification of the number of copies desired, a coding for a group of files to be printed together, or a code to specify headings and formatting required for the file.

Normally, the print program will have one disk and one printer file declared.

The input file specifications are:

Hardware Type	DISK (random or sequential)	DISK PACK
File identifier:	@00000	@00000
Record size:	140 UA	144 UA
Blocking factor:	10	10

The printer file must be declared as described above for PBTOUT.

Before OPENING the disk file, the program must specify the name of the desired file. The name can be constructed by moving the file number to the low order characters of the file identifier portion of the disk file label area (LABFID) (refer to the description for PBTOUT) and by inserting a @ into the high order character position. For a PB/, PBD/, or PBP/ request, this constructs a @00000 identifier; for a specific file request, this mechanism specifies the name of that file only.

The special identifier @00000 instructs the MCP to find the next sequential printer backup file on disk (if PBD was entered, or disk pack if PBP was entered) for that processor. The MCP uses the LABMFD field as a starting point for the file search. If the first character of LABMFD is not @,

the MCP assumes that the first PBD file is to be accessed. Thereafter, the field contains the current PBD file name and is maintained by the MCP. When the @00000 file is OPENed, the file identifier is placed in both the file identifier and multifile identifier fields of the label area.

The first record of the file is the printer file label record. This record must be accessed to find the file identifier of the print file, the FORM flag, and the label flag.

The following fields in the printer FIB must be modified if necessary:

Field	Location	Length	Contents/Meaning
FIBLBL	FIB: + 13	1 UN	Label Convention 0 = Standard label 1 = Label omitted
FIBSPF	FIB: + 33		:1 = If form is required
FIBFLM	FIB: + 128	1 UN	:4 = Ignore channel 12

The desired portion of the printer label record from the PBD file, as well as any other data must be moved to the printer label area.

Other fields that might concern the user are:

Field	Location	Length	Contents/Meaning
Quit Flag	BASE: + 0	1 UN	Set if QT message entered :1 = QT :2 = QT END :4 = QT ALL
Skip Count	BASE: + 2	6 UN	Integer entered in SK message
FIBST2	FIB: + 1	1 UN	:1 Set when all PBD files have been accessed on @00000 request
LABMFD	LABEL: + 9	6 UA	Name of current PBD used as starting point to find next file on @00000 OPEN request
LABFID	LABEL: + 17	6 UA	Must contain @00000 before OPEN of PBD/ operation to access next file; after OPEN contains current file being accessed

After the file has been OPENed and the label processing completed, records are accessed and printed in the manner described under General Considerations in this section. The print program must recognize the ending printer label record so as not to output the label as a data record. (This can be done

## PRINT UTILITIES

by programmatically testing the first characters of the record for the field bLABELbb.) While the utility can allow the MCP to perform the record unblocking, greater efficiency is achieved if done in the print program.

When EOF is reached, the disk file must be CLOSED WITH RELEASE or PURGE according to the PBD message input.

If PB/, PBD/ or PBP/ is being processed, @00000 must be moved to the file identifier field of the disk file label area and the next PBD file OPENed. When all files have been exhausted and the user requests the next file, the MCP executes an optional OPEN for the file, causing the user program to take an EOF branch on the first disk file READ. (The user can confirm the condition as FIBST2:1 will be set.)

The printer file CLOSE mechanism to accommodate FORMs must be handled as described for PBTOUT.

## SUGGESTIONS AND SPECIAL METHODS

The major consideration in choosing between tape and disk or disk pack are:

If the file must be printed on another (for example, non-shared) system PBT (or in some cases PBP) is generally indicated.

If the file is large, PBT may be necessary.

If there is considerable competition on the tape or disk channels, or the tape units one of the media are excluded.

If speed is a consideration, blocked PBT is slightly faster than PBD. PBD is considerably faster than unblocked PBT.

If memory space is a serious consideration, unblocked PBT may be indicated. (This is the ONLY advantage.)

Installation standards and the characteristics of the file in question often dictate source program file declarations. Some installations require all files to be diverted to backup for simplicity or recovery purposes; some never use PBT because of configuration limitations. Considerations such as these restrict programmatic choice of printer medium.

Certain file types dictate one medium or another. Small intermittent listings (errors, batch totals, and so forth) are most suitable for PBD. Security files (checks, and so forth) are suitable for printer only. PRINTER NO BACKUP must be used if the source program employs an end-of-page (EOP) USE routine for the file. (For this reason, the EOP declarative is rarely used. Line counters must be used if backup is anticipated.) Files to be printed on another system are assigned to PBT.

The PRINTER and BACKUP declarations must be used if unsure which medium is best. Experience with the particular mix in which the program runs often indicates which declaration is optimal. Once determined, the source program must be changed accordingly.

Never declare alternate areas for printer or punch backup files (except unblocked backup tape) as there is no benefit. The buffer in the user program is used as an intermediate work area from which records

are transferred to an external buffer area. However, physical printer, physical punch and unblocked PBT files benefit from one alternate area unless the file processing is intermittent or small. Rarely is more than one alternate area beneficial.

The FILE equate control statements are rarely needed. They are most useful to override the source declaration for a particular run. For example, if a program declares a PRINTER file, but for a particular run a printer backup tape is desirable (perhaps to create a tape to be shipped elsewhere), label equation is used.

To place a PBT file on a particular physical tape, the file can be label equated to PRINT FORM and, when the MCP informs the operator that a printer is required for the printer file, the OU message can be used to force the file to the desired tape.

### When Not To Use Printer/Punch Alternates

The considerations presented in Card Reader Alternates also pertains to printer and punch alternates. The major reasons not to use alternate media are time, channel, and resource usage.

### Special Source Program Methods

When a program creates a large PBD or PCD file the primary device is available during a part of the creation of the backup file. Thus, print images may be present on disk at the same time a printer is idle, but the records cannot be printed until the PBD file is CLOSED. Conversely, when a large PBD file is being printed, other smaller files are present and printing is held until the printer becomes free.

Both problems can be solved if the source program which creates the PBD/PCD file periodically CLOSEs and reOPENs the printer (or punch) file and results in the creation of several smaller PBD/PCD files. As each file is CLOSED it becomes available for printing, coincident with the creation of the remainder of the file. Similarly, because the individual parts of the file are smaller, the printer is not tied up for lengthy periods and other files can be printed between sections of the large file.

This technique is applicable primarily to PBD and PCD files but can be used to solve the second problem for PBT files as well. Programs which allow alternate handling of the file must use the simple CLOSE construct; otherwise, the pieces of the file could be assigned to a mixture of media.

A user-coded PBDOUT is desirable to gather the parts of the file for ultimate printing.

### User-Coded Utilities

There are many reasons to employ a user-coded utility. Some of the special purposes and possible mechanisms are described here.

As the data entered through the various keyboard messages is transmitted to the utility program, the meaning attached to the data is solely a function of the program.

Often, it is desirable that a group of print files be output in sequence. This might include all files created by a specific program, all files pertaining to a system, or the compile listing and output of a program being tested.

Several mechanisms for specification exist, but most rely on the capability to programmatically determine the group of files desired. Significant information exists in the first record of PBD and PCD

## PRINT UTILITIES

files and in the label of PBT files to permit this. The user program can OPEN and check the data in a printer or punch file, bypassing those which do not meet the criteria. This determination can be done by file identifier (if there is a structure to the file IDs), the creator, or the charge number fields.

For example, the keyboard input PBD/ 5 could tell a user-coded PBDOUT to output all files in group five. (For this case the code specification would not mean five copies of all files as this would be undesirable.) Output would be from all files which have a five in some position of the file name, all files with charge number five or a specific group which is keyed by a five.

The print utility would access all PBD files (using the @00000 file identifier), check the label records for the necessary criterion and bypass those which fail the test. Similar mechanisms could use the file number field in the PBD message to indicate a group if two digits are insufficient for specification. (For example, the Star Flag could be used to differentiate between a normal PBD 345 and a PBD group 345, or PBDOUT could be file equated to the correct file.)

Similar syntax could be used by PBTOUT to specify groups on stacked tapes.

Some installations want PBDOUT to be permanently resident accessing all files as they become present (or some variation, perhaps bypassing certain groups), as well as having normal facilities. This might be keyed by a PBD/ or some other means. The user PBDOUT would output all files currently on disk, when finished, would not terminate, but (DOZE) for a period of time and then try to OPEN @00000 again.

Other keyboard messages can be given different meanings. For example, one variant of the QT message could indicate quit printing and restart rather than simply quit which would permit on-the-fly restart without the necessity for a file search to the restart point.

Similarly, the integer in the SK message could be interpreted as a record number to skip TO, rather than a number of records to skip. Random access of the PBD file would allow forward or backward skipping, permitting simple partial file printouts, multiple copies of partial file printouts, manual file searching and so on.

## Other Uses

A user-coded PBDOUT can be useful for purposes other than those previously indicated. Such a program might have the capability of selecting certain records from a print file (totals), rather than printing the entire file. This can be useful in several areas, particularly during a conversion or testing session.

A user-coded utility allows program mechanisms to significantly improve the performance over the MCP version of the program. Most print programs require coding to control page headings and/or footings, line counts and so on. Codes can be eliminated if the first records output to a PBD or PBT file contain the headings/footings to be used, line count information, and any other data necessary to define page control. The program can then WRITE data records ignoring headings, line counts and other data. A user utility can access these records and do the page control. Such a mechanism reduces programming effort, program size, program execution time, and the size of the backup file.

## Special Uses of Printer and Punch Alternates

The MCP directly charges time for the printer and punch utilities. Time may be charged to the specific charge number entered with the PBD/PC message, or to the default system charge number if no



## CARD PUNCH ALTERNATE

specification is made and the user permits default charging (Cold Start USE CHRG card). If the user wishes a different mechanism, a utility program can accommodate the scheme. For example an installation may wish to charge the PBDOUT/PCHOUT time to the same number as the creator of the file. A utility can be written to tell which PBD or PCD file to output (for example VALUE or ACCEPT), OPEN the file, access the charge number from the first record and ZIP a PBD/PC request with the appropriate charge number.

This mechanism has the further benefit that operator intervention is minimal and by using methods described above, can be completely eliminated. For example, a particular mechanism indicates the use of a user-coded PBDOUT as a resident printer. This is also done by a separate utility which accesses the PBD/PC file and ZIPs the necessary PBD/PC messages. Similar techniques are used to eliminate the need for separate printer and punch utilities and allow a single program to control the output of either or both types of backup disk files.

### CARD PUNCH ALTERNATE

The card punch alternate is often called punch backup (abbreviated PCD), for which the alternate medium is either disk or disk pack. As for printer backup, the user is given significant control over the handling of punch files. In many respects, backup punch files are handled in the same way as printer backup disk files, and references are made to that area where relevant. The major difference between printer and punch alternates is the absence of tape as an alternate medium for punch files.

The user is unconcerned with the MCP mechanism which transfers records from user programs to the alternate medium. The user can exercise control over the disposition of the file to the primary or alternate medium, how this is done, and how the backup file is output to the primary device.

Every punch file is independent of all other punch files. Several can be created simultaneously from one or more programs.

The ultimate punching of a backup file is also independent of that for any other backup file. The files can be output in the sequence in which created or not as the user desires.

The user can control file disposition at three levels: in the source program file declaration, with execution time control cards, and/or when the file is OPENed by the program. This control can be summarized as follows:

#### Source Program (File Declaration)

The file is assignable to specifically punch or punch backup; or to either medium as available.

#### Execution Time (FILE Equate Control Card)

File selectable as for source program declarations.

#### File OPEN Time

A file which is not selected to a specific punch medium (for example, no alternate allowed) by the source declaration or FILE equation can be controlled at file OPEN time by system level options and/or operator console syntax.

While the punch backup process may be different depending on the extent of user control exercised, the following example is illustrative of the mechanism.

When a program executes an OPEN request for a card punch file the MCP searches for an available card punch. When found, it is assigned to the requesting program; otherwise, the MCP attempts to

## CARD PUNCH ALTERNATE

establish a pseudo punch so that punch images can be diverted to disk or disk pack, allowing the program to continue.

When the program completes and CLOSEs the punch file, it is available for physical punching. This action is initiated by a keyboard message which causes the execution of a punch utility.

## ESTABLISHING A PUNCH MEDIUM

When a program executes an OPEN request for a punch file, the MCP attempts to find the selected device. The result of the search is dependent on the source program declaration, the setting of the system level options, and the availability of resources. The user has extensive control over the disposition of the punch file from the program.

The system software permits the user to select the desired punch medium in the source language declaration of the file. The user can declare the file in any one of the following ways:

The file must be assigned to a physical card punch.

(SELECT file-name ASSIGN TO PUNCH NO BACKUP.)

The file must not be assigned to a physical card punch, but must be routed to punch backup.

(SELECT file-name ASSIGN TO PUNCH BACKUP.)

The file can be directed to either the physical card punch or to backup. This can be done automatically by the MCP according to the availability of resources, or by the system operator.

(SELECT file-name ASSIGN TO PUNCH.)

The user may wish to override the source program file type declaration for a particular execution of the job. This override is accomplished by the FILE (label) equate control statement which specifies that the files must be assigned specifically to a physical punch, punch backup disk only, punch backup disk pack only, or to any punch medium by the control cards:

```
FILE < file-name > = file-identifier PCO  
FILE < file-name > = file-identifier PCD  
FILE < file-name > = file-identifier PCH  
FILE < file-name > = file-identifier PCP
```

## File Open Time Control

When a program executes an OPEN request for a punch file, the MCP will assign the file to a particular medium. A source language declaration or a FILE equate override of a specific medium fully satisfies the responsibility of selecting the desired medium and the MCP need only find a device of the type specified.

For PUNCH NO BACKUP files, the MCP searches the IOAT for an available physical punch. For PUNCH BACKUP disk files, the MCP searches for 2 KD of user memory (for punch backup use). If the necessary resource can be found, the file is assigned. If file assignment cannot be completed the operator is notified of the need for the selected medium and the program is suspended until the needed resource becomes available or the operator forces assignment to a different medium.

If alternate handling of the file is allowed (ASSIGN TO PUNCH), the responsibility for selecting the final medium is placed on the MCP or the operator. The third level of user control specifies the limits of automatic MCP assignment of such files. The setting of the system level options PCH, PCD, and PCP allow or forbid the automatic assignment of a physical punch, backup disk, or backup disk pack, respectively, as a medium for PUNCH files. The meanings of the option settings are summarized in table 4-5.

When a PUNCH file is OPENed, the MCP checks the PCH option. If set, the MCP searches the IOAT for an available physical punch. If found, it is assigned; otherwise, the MCP checks the PCP option, then the PCD option to determine if the file can be assigned to backup. If this cannot be done (because the options are reset or insufficient space exists) the operator is notified and the program is suspended. When resources become available and/or the appropriate system option is set, automatic assignment is completed. During the interim the operator can employ console syntax to direct the file to the desired medium.

**Table 4-5. Punch Backup Options**

PCH	PCP	PCD	MCP Action
0	0	0	No automatic assignment.
1	0	0	Only a card punch can be used for PUNCH files.
0	1	0	Only punch backup disk pack can be used for PUNCH files.
0	0	1	Only punch backup disk can be used for PUNCH files.
1	1	0	If a punch is available, it is automatically assigned; otherwise, backup disk pack is used.
1	0	1	Use backup disk if a card punch is not available.
1	1	1	If a card punch is not available, use disk pack. If a disk pack is not available, use disk.

A zero (0) indicates that the option is reset; a one (1) indicates that the option is set and automatic assignment is allowed.

These options do not apply to files assigned to PUNCH NO BACKUP, PUNCH BACKUP DISK, or PUNCH BACKUP DISK PACK, as no choice of medium is allowed to the MCP for these cases. However, if the requisite medium is not available when the file is OPENed, the operator can employ a keyboard message to route the file to the other medium.

In addition to any selection of punch medium, card punch files can be declared as requiring special forms either in the source program file declaration or in a FILE equate override. Whenever the file is assigned to PUNCH or to PUNCH NO BACKUP, the MCP must inform the operator that special forms are required. The operator then places the necessary cards in a card punch and directs the file to that device, or diverts the file to backup. If a backup file is produced, it contains a flag specifying that special forms are required when the file is output to a card punch.

## LABEL PROCESSING

When a suitable device has been assigned to a program, the MCP executes the appropriate label procedures.

## CARD PUNCH ALTERNATE

For a physical punch, the label is written unless the file is declared unlabeled. For backup files, a Burroughs standard label is always produced, but it is flagged if the file is declared unlabeled. This informs the punch utility to bypass label processing when the file is output.

If the file has been declared as FORM, the backup file label contains a flag which reflects this fact.

Program beginning and ending label procedures are permitted regardless of medium. Label considerations for printer backup files also pertain to punch backup files.

## CLOSING PUNCH FILES

When a program executes a CLOSE request for a punch file, or when a program terminates with a file still OPEN, the MCP file close routines are invoked. The action taken is dependent on the CLOSE type and the file medium.

Two CLOSE types are meaningful for punch files, CLOSE and CLOSE WITH RELEASE. If the file is attached to a physical punch, CLOSE directs the MCP to perform the normal CLOSE procedures (including an ending label if applicable), but not to release the unit to the system. This is done when the punch file is to be reOPENed in a short time. A CLOSE WITH RELEASE accomplishes the same actions except that the unit is released to the system for possible reassignment.

Files diverted to backup disk are automatically CLOSED WITH RELEASE regardless of the program specified CLOSE type.

In all cases, if a program terminates without having CLOSED a punch file, the file is automatically CLOSED WITH RELEASE. Whenever a punch backup file is CLOSED, the operator is notified.

## PUNCH BACKUP FILES

The pseudo punch has two components. A memory area (outside of program bounds) is used as a buffer which holds a block of card images as well as storage for the disk file header for the backup punch file. The second component is the MCP logic which diverts card images to the memory buffer area and writes the block to disk as the buffer becomes full.

The user is not involved with any aspect of the mechanism of punch backup except when the installation has elected to reset the system options and force the operator to control file assignment by the OU console syntax.

Punch backup files are composed of records which contain both card punch images and stacker select control information. The first and last records of the backup file are label records which contain the user declared file identifiers as well as flags which specify whether the file requires special forms, whether the labels are to be physically punched, and the punch mode (BCL or EDCDIC).

The characteristics of a punch backup file are:

File Attribute	Disk	Disk Pack
Record Size	100 UA	120 UA
Blocking Factor	16	14
Records per Area	2000	1500
Segments per Area	2000	1000
Number of Areas	100	100
Maximum File Capacity	200000 Records	150000 Records

Since disk and disk pack files have no labels and a PCD file is essentially a normal disk or disk pack file, the punch label records occur as the first and last records of the backup file. The labels are in standard format except that the fields in the beginning labels given in table 4-6 have special significance:

**Table 4-6. Backup Print File Label Format**

Name	Location	Length	Contents/Function
Mode	LABEL: +0	1 UA	0 = Punch BCL 2 = Punch EBCDIC
Label	LABEL: +1	7 UA	LABELbb – Verify presence of label
MFID	LABEL: +9	6 UA	Multifile ID if present, otherwise, zeros
FID	LABEL: +17	6 UA	File identifier
Creator	LABEL: +58	13 UA	Program name/compiler name if compile listing; program name if user program output.
Charge Number	LABEL: +72	6 UA	Charge number of creator.
Form Flag	LABEL: +78	1 UA	1 = Forms required.
Label Flag	LABEL: +79	1 UA	1 = File declared unlabeled

The Creator and Charge Number fields are not placed in the label if the necessary area contains any non-zero or non-blank characters.

All other backup file records have the following format:

Field	Length	Contents/Function
Filler	3 UA	
Control Information	1 UA	Stacker Select Control
Card Image	96 UA	User Created Card Image
Filler	20 UA	(only if backup disk pack)

The control field specifies stacker selection for those card punches with alternate stackers, and can contain only a 0, 1, or 2. The number specifies primary stacker, alternate stacker one, or alternate stacker two, respectively.

Punch backup files are identified in the directory by names of the form:

\*pnnnn

where p is the number of the processor on which the file by the MCP when the file is entered in the Disk Directory.

Numbers are assigned beginning with 0000 and incremented as each new file is created. The counter is reset at 9999.

The name of the punch backup file is completely independent of the identifier of the punch file within, and is used only for operator reference and for the utility which outputs the backup file to the primary device.

## CARD PUNCH ALTERNATE

### ACCESS AND CONTROL OF PUNCH BACKUP FILES

The following describe various means used in handling punch backup files.

#### Accessing Punch Backup File Programmatically

Programmatic accessing of backup files is described in detail later in this sub-section on punch utilities. They can be accessed by specific name (\*pnnnn) or by the generalized punch backup name (\*00000) which requests the next sequential file.

#### Removal of Punch Backup Files

Punch backup files can be removed automatically when the files are output by the punch utility (by closing the file with PURGE) or by the keyboard message RF described below.

#### Operator Interface to Punch Backup Files

The operator has the following control over punch files through keyboard messages:

##### Display Backup Files on Disk: BF PCH/

The MCP lists all punch backup file numbers and the name of the punch file within them. BF PCH <file-number> or BF PCH <file-identifier> can be used to list specific backup files. BFP PCH... is used for disk pack.

##### Direct a Punch File to Punch Medium:

$$\langle \text{mix no} \rangle \quad \text{OU} \quad \left\{ \begin{array}{l} \langle \text{cc/u} \rangle \\ \text{DK} \\ \text{PK} \end{array} \right\}$$

If a program is suspended because the MCP cannot complete assignment of a punch file, the operator can direct the file to a physical punch (cc/u), backup disk (DK), or backup disk pack (PK). <mix no> FM<cc/u> can also be used to direct output to a punch.

##### Change a File Name to a Backup File Name

CHANGE <file-identifier> TO \*00000 changes <file-identifier> to a punch backup disk file name (\*pnnnn). The file must be in the correct disk file format as only the name is changed.

$$\text{CHANGE } \langle \text{file-identifier} \rangle \quad \text{TO } * \text{ ON } \left\{ \begin{array}{l} \langle \text{cc/u} \rangle \\ \text{PACK} \\ \langle \text{pack-ID} \rangle \end{array} \right\}$$

is used for disk pack.

##### Remove a Punch Backup Disk File

RF PCH <file-number>, RF PCH <file-identifier>, or RF PCH/ directs the MCP to remove the specified backup files from the Disk Directory if not in use. RFP PCH... is used for disk pack.

### Initiate Punch Backup Disk Utility

$$PC \quad [P] \quad \left[ \begin{array}{l} /OWN \\ /MCP \end{array} \right] \quad [*] \quad \left\{ \begin{array}{l} / \\ <file-number> \end{array} \right\} \quad [SAVE] \quad [ <integer> ]$$

initiates a punch utility to output the designated punch backup file or all punch backup files.

### Bypass Part of a Backup File During Processing

<mix no> SK [-] <integer> directs the specified utility to skip (forward or backward) the designated number of backup file records and then resume processing.

### Terminate Processing of a Backup File

$$<mix no> \quad \left[ \begin{array}{l} END \\ ALL \end{array} \right]$$

directs the specified utility to terminate processing immediately (QT ALL), terminate at the end of the current file (QT END), or terminate processing of the current file and go on to the next, if any (QT).

### Convert Backup File to Pseudo-Deck

$$CV \quad [P] \quad \left[ \begin{array}{l} /OWN \\ /MCP \end{array} \right] \quad <file-number> \quad [SAVE]$$

initiates a utility which copies the designated punch backup file, reformatting into a pseudo-deck format.

## PUNCH BACKUP UTILITY (PCHOUT)

The MCP utility known as PCHOUT, provides the capability to output punch backup files to the primary device. Many of the characteristics of PBDOUT are also common to PCHOUT.

PCHOUT can output a single selected file or several files in sequence.

The program can be interrupted during processing and directed to skip a number of records or to terminate processing in any of three ways (Refer to Bypass Part of a Backup File During Processing, and Terminate Processing of a Backup File, in this sub-section.)

PCHOUT can be directed to execute a skip or restart operation when initiated, allowing recovery from system malfunctions.

Either labeled or unlabeled files can be handled and the use of special forms is allowed.

As for many other MCP utilities, the user can develop an individualized punch utility to be used in lieu of the MCP intrinsic.

## CARD PUNCH ALTERNATE

Operator communication with the punch backup program normally involves no more than the initiation of the utility. As applicable, a restart point can be specified, or processing can be interrupted.

PCHOUT is initiated by a keyboard message (PC [P] ...). The operator can indicate that all files are to be punched in sequence according to numeric designations (PC/), or that a specific file is to be output (PC <integer> where <integer> is the numeric--nnnn or pnnnn--portion of the backup file name. The PC pnnnn format can be used to punch a file on a processor other than the creator).

The operator can include a /OWN or /MCP parameter to require the initiation of either the user-coded or MCP version of the utility. In the absence of this parameter, the user-coded version is initiated if present on disk; otherwise, the MCP intrinsic is initiated.

When the keyboard message is entered, restart or skipping can be specified by the form PC [P] \*... . The asterisk indicates that the operator wishes to specify a restart point or number of disk records to be bypassed before punching begins.

The program can also specify that the backup files are not to be purged after being punched by including the word SAVE in the PC request.

The next specification in the request can be a 1 or 2-digit integer (0-99) which is passed to the punch program. This parameter is used by PCHOUT to determine the number of copies of the punch file to be output.

The complete syntax for the PC message is:

$$\left\{ \begin{array}{l} \text{PCP} \\ \text{PC} \end{array} \right\} \left[ \begin{array}{l} \text{/OWN} \\ \text{/MCP} \end{array} \right] \left[ \begin{array}{l} * \\ \text{<file-number>} \end{array} \right] \left[ \text{SAVE} \right] \left[ \text{<integer>} \right] \left[ \text{; parameter instructions} \right]$$

If the Cold Start CHR<sub>G</sub> ALL parameter has been specified, the CHARGE parameter instruction must be included.

When the punch program is initiated, the MCP passes it the file number (zeros if the message was PC/), flags if \* and/or SAVE were specified, and the copy parameter.

## USER SUPPLIED PUNCH UTILITY

The following information pertains to a user supplied version of PCHOUT.

### General Considerations

The MCP provides the user with the means to write individualized versions of the punch intrinsic. This program can use the same interface with the MCP and the operator as the MCP intrinsics. Keyboard inputs to initiate the control of user-coded utility are the same for the MCP utilities as are the parameters passed to the PCHOUT program.

When the operator requests a punch utility, the MCP scans the Disk Directory for a file with the appropriate name for the user-coded version. When a file is found, the user utility is invoked in much the same manner as the MCP intrinsic; if no user-coded version is present on disk, the MCP utility is initiated.



## User-Coded PCHOUT

The user-coded version of the punch utility must be named PCHOUT to use the same operator interface as the MCP intrinsic. The file access mechanism, however, is not dependent on the program name, and any program can access a punch backup file. When PCHOUT is initiated, the MCP passes the following data:

Name	Location	Length	Contents/Meaning
File Number	BASE: + 32	5 UN	File number requested in PC [P] message, or zeros for PC [P]/
Variants Flag Save Flag Star Flag  Code Flag	BASE: + 37	1 UN	:1 SAVE specified :2 * specified :4 File is on disk pack (PCP...) :8 CODE supplied
Code	BASE: + 38	2 UN	00-99 As entered by operator, 00 if no parameter entered (Code Flag off)

The file number is the numeric portion of the name of the punch backup file (\*pnnnn); if PC/ is requested, the field contains zeros.

The Variant Flag digit specifies that other data was entered by the operator. While normally used to control file disposition and/or restart, the user can place any desired meaning on these flags. The Code parameter can also be used in any way.

Normally, the punch program will have one disk and one punch file declared.

The input file specifications must be:

Hardware type	Disk (random or sequential)	Disk Pack
File identifier:	*00000	*00000
Record size:	100 UA	120 UA
Blocking factor:	16	15

The punch file is declared:

Hardware type:	PUNCH NO BACKUP
File identifier:	Any
Record size:	80 UA
Blocking factor:	1
Label records:	Standard

Before OPENing the input disk or disk pack file the program must specify the file name.

The name is constructed by moving the file number to the low order characters in the file identifier portion of the file label area, and inserting a \* character into the high order character position. For a PC/ request, this constructs the \*00000 identifier; for a specific file request, this mechanism specifies the name of that file only.

The special identifier \*00000 specifies that, when the file is OPENed, the MCP is to find the next punch backup file for that processor.

CARD PUNCH ALTERNATE

The first record of the file is the punch file label record. This record (previously described) must be accessed to determine the file identifier of the punch file, the FORM indicator, the label convention, and the recording mode.

The following fields in the punch file FIB must be modified if necessary:

Field	Location	Length	Contents/Meaning
FIBLBL	FIB: + 13	1 UN	Label Convention 0 = Standard label 1 = Omitted label
FIBSPF	FIB: + 33		:1 = If Form is required
FIBMOD	FIB: + 43	1 UN	0 = BCL file 2 = EBCDIC file

The desired portion of the label record from the backup file, as well as any other data inserted, must be moved to the punch label area.

Other fields with which the user might be concerned include:

Field	Location	Length	Contents/Meaning
Quit Flag	BASE: + 0	1 UN	Set if QT entered :1 = QT :2 = QT END :4 = QT ALL
Skip Count	BASE: + 2	6 UN	Integer entered in SK request
FIBST2	FIB: + 1	1 UN	:1 Set when all backup files have been accessed (PC/)
LABMFD	LABEL: + 9	6 UA	Name of current PCD file being accessed; used as starting point to locate next file on *00000 request
LABFID	LABEL: + 17	6 UA	Must contain *00000 before OPEN for PC/ operation to access next file; after OPEN contains current file being accessed

After the files have been OPENed and label processing completed, data records must be accessed, the punch information moved to a buffer area, and the stacker select control digit moved to the variant field of a WRITE communicate.

The last record in the backup file is a label record and must be recognized prior to executing a WRITE, to avoid punching the label as a data record. This is done by programmatically checking for the field bLABELbb in the first eight characters of the record.

The input file can be CLOSED WITH RELEASE or PURGE according to the PC message input.

If PC/ is being processed, the program simply reconstructs the \*00000 identifier and OPENs the next backup file. When all files have been exhausted and the utility requests the next file, the MCP executes an optional OPEN for the file, causing the user program to take an EOF branch on the first disk file READ. (The program can confirm the condition as FIBST2:1 will be set.)

## **SECTION 5**

### **MCPVI TABLES**

The MCP tables presented in this section represent the data areas maintained by the MCP for a user program. Much of this information can be found in a program memory dump and is presented for this reason.

The information described does not represent all of the tables maintained by the MCP. Rather, the tables included are ones which relate directly to a program. These tables are:

- The Disk/Disk Pack File Header
- The File Information Block
- The File Buffer Descriptor
- The Input/Output Assignment Table
- The Job Reference Table
- File Labels
- The Mix Table
- The Security Attributes Storage Area
- The Complex Wait Table

## DISK FILE HEADER

### DISK FILE HEADERS

A file that resides on disk or disk pack has three components. Two of these are maintained by the MCP: Disk Directory entry and file header. The third, file data, is user program created and altered.

The Disk Directory entry simply indicates that a particular file resides on disk or disk pack. Presence of a Disk Directory entry implies the existence of a File Header and possibly, file data.

The File Header contains information about the file and acts as a road map for the file data. Pointers to the allocated areas of the file (pages) are in the File Header. Also contained in the header are record length, blocking factor, number of areas declared, number of users of the file, and various other information about the file.

Different information is maintained in the File Headers for disk files than for disk pack files. However, after a file is OPENed, file processing requires similar information whether the file is on disk or disk pack. Consequently, when a file is OPENed, the two file header types are reformatted into a common description when the header is loaded into memory.

A user program can request disk file header information in two ways. 1) the INTERROGATE FILE BCT; 2) load file header information at file OPEN time (FIBST1 set). Refer to Program Interface, section 2, and and FIBs in this section, for details.

A disk/disk pack file OPEN requires 200 digits of memory. 100 digits is allocated to an IOAT, the other 100 is the reformatted file header (refer to IOAT in this section).

The area pointers to the file (address blocks) are allocated in 200 digit increments. Since an area pointer is eight digits long, a file of 100 areas, requires 800 digits for the area pointers. Thus a maximum of 1 KD is required for a disk/disk pack file OPEN; the actual memory required depends on the maximum number of areas declared for the file.

The following subsections contain descriptions of the headers on disk and disk pack, as well as the reformatted header after the file has been OPENed.

DISK FILE HEADER (ON DISK)

The following descriptions apply to the disk version of the Disk File Header.

Label	Relative Location /Size	Content/Meaning
DF-RSZ	0,5	Record size in digits
DF-RPB	5,3	Maximum number of records per block
DF-#AR	8,2	Maximum number of disk areas assigned to the file (0 Relative)
DF-EOF	10,8	End of file pointer
DF-USE	18,2	Number of users on processor 0
	20,2	Number of users on processor 1
	22,2	Number of users on processor 2
	24,2	Number of users on processor 3
DF-SYS	26,1	System number of locking program
DF-MIX	27,2	Mix number of locking program
DF-ORG	29,1	:8 Relative I/O Data File
		:4 Indexed I/O Data File
		:2 Indexed I/O key file
		:1 << Available >>
	30,1	Not used
DF-ST1	31,1	:8 Get high disk areas
		:8/ Get low disk areas
		:4 The command PM/ is inhibited. If APCR = 1, then RN/ is inhibited
		If APBD = 1, then PBD/ is inhibited If APCM = 1, then PC/ is inhibited
		:2 Remove on HL even if marked permanent
:1 Do not squash file		
DF-DKS	32,1	Disk subsystem assignment
		0 = Default disk subsystems
		1 = Primary disk subsystem
		2 = Disk subsystem #2
		3 = Disk subsystem #3
		4 = Disk subsystem #4
		5 = Disk subsystem #5
		6 = Disk subsystem #6
7 = Disk subsystem #7		

DISK FILE HEADER

<b>Label</b>	<b>Relative Location /Size</b>	<b>Content/Meaning</b>
		8 = Disk subsystem #8 E = Common (shared) disk subsystem
DF-DSA	33,7	Number of disk segments per disk area
DF-AR1	40,8	Disk address of area # 1 of file
	48,792	Disk address of areas # 2 through # 100

DISK PACK FILE HEADER (ON DISK PACK)

The following descriptions apply to the disk pack version of the Disk File Header.

Label	Relative Location /Size	Content/Meaning
PF-CAD	0,6	Zeros
PF-SPT	6,8	Address of this sector
PF-SIZ	14,4	Header size in bytes
PF-TP1	18,1	File type
		:8 << Available >>
		:4 File name change in progress
		:2 Incomplete file (partially removed)
PF-TP2	19,1	File type
		:8 Assign by space available file
		:4 Assign by area file
		:2 Single pack
		:2/ Multipack file
PF-TP3	20,1	File type
		:8 << Available >>
		:4 If APCR is set, RN/ is inhibited If APBD is set, PBP/ is inhibited If APCM is set, PCP/ is inhibited
		:2 << Available >>
		:1 No squash file
	21,3	Not used
PF-BEN	24,8	Block EOF pointer
PF-RSZ	32,6	Record size in digits
PF-RPB	38,3	Records per block
PF-BSZ	41,9	Block size in digits
PF-BPA	50,6	Blocks per area
PF-SPA	56,6	Sectors per area
PF-#AR	62,4	Areas requested
PF-UAR	66,4	Area counter
PF-EOF	70,10	EOF pointer

B 2000/B 3000/B 4000 MCPVI Programmer's Guide  
MCPVI Tables

DISK PACK FILE HEADER

Label	Relative Location /Size	Content/Meaning
PF-FRM	80,2	Record format
PF-USH	82,8	User header link
PF-PPA	90,3	Partitions per area (split cylinder files)
PF-ORG	93,1	:8 Relative I/O Data file :4 Indexed I/O Key file :2 Indexed I/O Data file :1 <<Available>>
PF-CDT	94,5	Creation date
PF-LAD	99,5	Last access date
PF-SAV	104,5	Save factor
PF-ADB	109,6	Base pack header address
	115,11	Not used
PF-SNS	126,1	Sensitivedata flag
PF-STY	127,1	Security type :8 Reserved :4 Guarded :2 Public :1 Private 0 None
PF-SUS	128,1	Security use 6 IO (default) 4 IN 2 OUT 1 SECURED
	129,1	Reserved
PF-SUC	130,20	Usercode
PF-GRD	150,12	Guard file ID
PF-OTY	162,1	Open type
PF-PRM	163,1	Permanent flag
PF-NU1	164,2	Number of users processor 0
PF-OO1	166,2	Number of open out processor 0
PF-NU2	168,2	Number of users processor 1
PF-OO2	170,2	Number of open out processor 1
PF-NU3	172,2	Number of users processor 2
PF-OO3	174,2	Number of open out processor 2



B 2000/B 3000/B 4000 MCPVI Programmer's Guide  
MCPVI Tables

DISK PACK FILE HEADER

<b>Label</b>	<b>Relative Location /Size</b>	<b>Content/Meaning</b>
PF-NU4	176,2	Number of users processor 3
PF-OO4	178,2	Number of open out processor 3
PF-BPP	180,4	Blocks per partition
PF-FAM	184,12	Pack ID of the guard file
PF-MIX	196,2	Mix number of generator or locking program
PF-SYS	198,1	Number of generator or locking system
PF-NRM	199,1	
		:8 Remove this file on null activity
PF-AR1	200,8	First area link
	208,nn	19-99 more area links

DISK PACK FILE HEADER

FILE HEADER IN MEMORY

The following information represents the reformatted disk and disk pack file headers when the headers are in memory.

Label	Relative Location /Size	Content/Meaning
DF-RSZ	0,5	Record size in digits
DF-RPB	5,3	Number of records per block
DF-#AR	8,2	Maximum number of areas
DF-EOF	10,8	End of file pointer
DF-ORS	18,5	Original record size in digits
DF-ORB	23,3	Original number of records per block
DF-DSB	26,4	Number of disk segments per block or pack sectors per block
DF-PAK	30,1	File assignment type :8 Assign by space available pack file :4 Assign by area pack file :2 Single pack :2/ Multipack file :1 Cylinder bound pack file
DF-ST1	31,1	:8 Get high disk address :8/ Get low disk address :4 The command PM/ is inhibited. If APCR = 1, then RN/ is inhibited If APBD = 1, then PBD/ is inhibited If APCH = 1, then PC/ is inhibited :2 Remove on HL even if marked permanent :1 Do not squash file
DF-DKS	32,1	Disk subsystem assignment 0 = Default disk subsystems 1 = Primary disk subsystem 2 = Disk subsystem #2 3 = Disk subsystem #3 4 = Disk subsystem #4 5 = Disk subsystem #5 6 = Disk subsystem #6 7 = Disk subsystem #7

DISK PACK FILE HEADER

Label	Relative Location /Size	Content/Meaning
		8 = Disk subsystem #8 E = Common (shared) disk subsystem
DF-DSA	33,7	Number of disk segments per disk area
DF-ADR	40,6	Memory address of address block #1
DF-ADR	46,6	Memory address of address block #2
DF-ADR	52,6	Memory address of address block #3
DF-ADR	58,6	Memory address of address block #4
DF-DIR	64,8	Disk address of Disk Directory header block or pack address of Pack Directory sector
DF-DFH	72,8	Disk address of disk file header block or pack address of pack file header
DF-DRX	80,4	Index to file in Disk Directory header block or pack directory header sector
DF-BPA	84,7	Number of blocks per disk area
DF-SIZ	91,3	Disk file header size in digits, 840 digits max; 0 Pack file header size in bytes, 500 bytes max
DF-USR	94,2	Total number of users sharing in-core DFHDR
DF-RND	96,2	Total number of users with random access
DF-ST2	98,1	:8 In-core area address blocks present :4 Temporary disk or pack file :2 <<Available>> :1 <<Available>>
DF-OR1	99,1	:8 Relative I/O Data File :4 Indexed I/O Data File :2 Indexed I/O Key File :1 <<Available>>

For a split cylinder disk pack file, the following redefinitions to the file header apply.

Label	Relative Location /Size	Content/Meaning
DF-PPA	18,4	Partitions per area
DF-BPP	22,4	Blocks per partition

FIB

**FILE INFORMATION BLOCK (FIB)**

When a program is compiled, each source language file declaration causes the compiler to generate a block of data which describes the file for the MCP. This 200-digit area is known as the File Information Block (FIB) and contains flags, counters, addresses, and sizes which are used by the MCP during all phases of file processing.

Many of the fields are directly constructed by the compiler according to the source program declaration. These include record size, blocking factor, file hardware type, recording mode, label convention, and so on. Other fields affected when the file is OPENed are input/output mode, file status flag, and link to a physical device table entry. Several fields are dynamically changed during file processing such as the block and record counts; record and buffer pointers.

Several fields are used only for certain types of files; other fields have multiple uses depending on the file type or the particular stage of file processing.

Any modification of the FIB is emphatically discouraged. Such actions can be dangerous to the program attempting the modification because the information can cause MCP or program failure. Further, as MCP mechanisms are subject to change, there is no guarantee that programs which modify FIBs will continue to function properly under future releases. The information in the following sub-sections is provided primarily to give greater insight into the FIB-MCP interface. To a lesser degree, the information is provided for those who are willing to risk the dangers inherent in FIB modifications. The release of such information does not constitute endorsement of its application, does not imply support of its use, and does not guarantee that programs which employ these or similar mechanisms will continue to function.

**FILE INFORMATION BLOCK LAYOUT**

The FIB is the file handling interface between program and MCP. The 200-digit area is used for READ/WRITE, OPEN, and CLOSE. The following descriptions indicate the functions of the FIB fields. Note that some fields have different meanings depending on hardware type.

Label	Relative Location /Size	Content/Meaning
FIBST1	0,1	<p>Controls tape assignment, DFH access, and file recovery</p> <p>:1 DSK (input, I/O): At file OPEN, MCP puts first 40 digits of DFH as exist in directory into requestors BASE: +100. See FIBST1:8.</p> <p>DSK (output): At file CLOSE, requestor wants MCP to use 18 digits at BASE: +100 rather than MCP maintained DFH.</p> <p>:2 DSK, MTP (output): File is CLOSEd with RELEASE if not done by program prior to EOJ (normal or abnormal).</p>

Label	Relative Location /Size	Content/Meaning
		<p>:4 DSK: At file OPEN, MCP puts first 40 digits of DFH as exist in memory into requestors BASE: +100 (includes any modification made to accommodate programmatically declared redefinition in block size, and so on). MTP: See below.</p> <p>:8 DSK (input, I/O): At file OPEN, MCP puts disk addresses for file starting at BASE: +140; number of addresses passed depends on number allowed for file as specified in DFH; bit is reset during file OPEN procedure.</p> <p>:4, :8 MTP (output): The following bits control output MTP track requirements. In addition to the meaning described in the following, certain special combinations pertain at file OPEN. If neither bit is set, and the MPE bit is not set (FIBOPT:4) all three bits are set. If FIBST1:8 is set, but the MPE bit is not set, FIBOPT:4 is automatically set. At the end of file OPEN, type of tape actually assigned reflected in bits left on in FIBST1, FIBOPT.</p> <p>:4 MTP: Specifies MT7 acceptable for file.</p> <p>:8 MTP: Specifies MT9 (NRZ) acceptable.</p> <p>:8 SOR: Specifies 4A type Sorter Control required at OPEN.</p>
FIBST2	1,1	<p>Controls Printer/Punch assignments; miscellaneous.</p> <p>:1 Input: Indicates no physical file to process or CLOSE; set if OF or FR keyboard message entered, all backup files accessed (for example, @00000 request) or last file on multifile tape accessed (FID = spaces request).</p>

B 2000/B 3000/B 4000 MCPVI Programmer's Guide  
MCPVI Tables

FIB

Label	Relative Location /Size	Content/Meaning
		<p>:2 PRN, PCH: File must be assigned to primary device.</p> <p>:4 PRN, PCH: File must be assigned to backup device. (If bit set, and FIBHDW = 04, means PBT only; bit reset and 42 moved to FIBHDW.)</p> <p>:8 Specifies that program is controlling block size either by changing descriptor addresses or by the @CF@ character (MT7); MCP does not affect FIB-BB or use full MTP I/O error recovery facilities.</p>
FIBRRN	2,5	<p>Rerun Number</p> <p>CRD, PCH, PRN, MTP, PTR, SEQ DSK: Number of records left to process before next breakout (checkpoint); decremented to zero, then reset from FIBRRC.</p>
FIBRRC	7,5	<p>Rerun Control</p> <p>CRD, PCH, PRN, MTP, PTR, SEQ DSK: Number of records to process between breakouts; moved to FIBRRN at each breakout.</p>
FIB-BA	12,1	<p>Buffering Technique</p> <p>All files: :1 Buffer(s) only, program works directly in buffer. :1/ Buffer(s) and work area, program accesses record work area. Value affects meaning of FIBARB, FIB-NB, FIB-WA. :2 Restart running. :4 DPK: File must reside on a single pack; otherwise, areas are spread over available packs (as constrained by FIBDTK). :8 MTP: During automatic reel swap, buffers currently queued for I/O are to be placed on next reel to insure that file is not written off end of current reel (occurs only if program has no label USE routine).</p>

Label	Relative Location /Size	Content/Meaning
FIBLBL	13,1	<p>File Label Convention</p> <p>Non-DSK output: Specifies type of label to create.</p> <p>0 = Standard Burroughs label. 1 = Label omitted. 2 = USASII standard label. 4 = Label as per Installation Label Card specification. 8 = MTP output only – use first scratch tape available and maintain same label type. Type inserted into FIBLBL.</p>
FIBALT	14,1	<p>Number of Alternate Buffers</p> <p>All files: Total # of buffers – 1 (only one buffer used for PCR, PCD, PBD, DCM, PBTB regardless of number declared.</p>
FIBSTA	15,1	<p>File Status</p> <p>Used to establish validity of I/O requests or special handling at OPEN, CLOSE, and EOJ.</p> <p>0 = All files: File is OPEN.</p> <p>1 = All files: File never OPENed. This value is generally found only before the first OPEN of the file.</p> <p>2 = All input files except RND, DCM, SOR: File access restricted. I/O requests are restricted to file CLOSE because EOF has been sensed.</p> <p>3 = All files: File CLOSEd. Program has OPENed, then CLOSEd file; file can still be attached to device if request was simple CLOSE. See FIBIOA.</p> <p>4 = CRD: File prematurely CLOSEd. The MCP has CLOSEd and RELEASEd the unit before the program CLOSE request because a control card has been sensed.</p>

B 2000/B 3000/B 4000 MCPVI Programmer's Guide  
MCPVI Tables

FIB

Label	Relative Location /Size	Content/Meaning
		<p>5 = MTP, PTR: Automatic reel swap in process. When end of reel is sensed, the current reel is CLOSED and the next reel OPENed.</p> <p>6 = Premature CLOSE and EOF label taken, or file CLOSED with LOCK by a COBOL ANSI-74 program</p> <p>7 = MTP, PTR: Reel swap in process. When program requests CLOSE REEL, current reel is CLOSED and next reel OPENed.</p> <p>9 = MTP: Multifile search in process. MCP in process of looking for requested file on multifile tape.</p>
FIBSVF	16,3	Save Factor
		<p>MTP: Number of days beyond creation date that file can be purged automatically by MCP.</p>
FIBMRL	19,5	Maximum Record Length
		<p>All files: Record size in digits. For variable length records FIBMRL gives maximum record size allowed. The MCP forbids a value of zero and, for work area access files, a value of 40,000 or greater.</p>
FIBRPB	24,3	Records per Block
		<p>DSK, MTP: Number of fixed length records per block; used to create or recalculate DFH for disk files during MTP/DSK positioning and RND READ/WRITE.</p>
FIBARB	27,6	Addresses of Record in Buffer
		<p>All files: Meaning depends on FIB-BA. If FIB-BA:1 set, after READ, FIBARB points to record requested; prior to WRITE points to available space in buffer (buffer is available at this point). If FIB-BA:1 not set, prior to READ points to record to be</p>



Label	Relative Location /Size	Content/Meaning
		requested; prior to WRITE points to space for next output record (buffer may not be available at this point). If FIBIX2:1, value placed in program IX2 after each I/O request.
FIBSPF	33,1	<p>Flags for Special Forms, Miscellaneous</p> <p>:1 PRN, PCH: Special forms required. Automatic assignment of output device (except forced backup) cannot be done because of need for special forms; PRN, PCH device saved (SV) automatically when device released. If backup file created, flag carried to file.</p> <p>:2 DSK: Directs MCP to insert the pass file number into the second and third characters of the file ID. This number is maintained within the MCP and is incremented at each use. Used to ensure file name uniqueness when &lt;mix no&gt; is not satisfactory.</p> <p>:4 Waiting MCS buffer (DCP)</p> <p>:8 DCM: Internal flag specifying that current request is for transparent I/O (WTRC or WCRT).</p>
FIB-WA	34,6	<p>Work Area Address</p> <p>All files: If FIB-BA:1/, address of work area. If FIB-BA:1, unused.</p>
FIBHDW	40,2	<p>Hardware Type</p> <p>All files: Prior to file OPEN, specifies hardware type requested (See FIBST1:4 and :8; FIBST2 :2 and :4 for supplementary specifications). Can be modified in file OPEN due to file equate or IL. Refer to Table 1-1 for the specific values.</p>
FIB-IO	42,1	<p>Input/Output Mode Flag</p> <p>Establishes validity of current I/O request.</p>

FIB

Label	Relative Location /Size	Content/Meaning
		<p>0 = File OPENed input.            1 = File OPENed output.            2 = File OPENed I/O (DSK, DPK, DCM only).            3 = File OPENed O/I (RND only); changed to two during file OPEN.</p>
FIBMOD	43,1	<p>Recording of File</p> <p>Applies to output files only. Input file recording mode determined from file (MT7 or label (CRD)); For input MTP, parity reflected in FIBMOD.</p> <p>0 = Non-standard recording.            PCH: File to be punched in BCL.            MT7: File to be written in even parity.</p> <p>1 = BINARY recording.            PCH: File to be punched BINARY (low order six bits of even bytes to be punched in top rows of card, low order six bits of odd bytes in bottom rows); file cannot go to backup.            MTP7: File to be written odd parity.</p> <p>2 = Standard recording.            PCH: File to be written in EDCDIC.</p>
FIBBLK	44,1	<p>Blocking Technique</p> <p>0 = All files:            Records unblocked.</p> <p>1 = All files:            Fixed number of fixed length records</p> <p>2 = All files except DSK, DCM, SOR:            Variable number of variable length records per block. For work area access files, output blocks are packed to maximum possible (written only when current record will not fit in space remaining in buffer). For buffer access files, output blocks are written when the maximum record size (FIBMRL) exceeds the remaining space and are less efficient.</p>
FIBFNM	45,2	<p>File Number</p> <p>File number assigned to file by compiler; value assigned by order in which files</p>

Label	Relative Location /Size	Content/Meaning
		declared; used in file equating as index to proper file equate block on disk.
FIBLBA	47,2	Last Buffer to Access  Non-DCM: FIB relative index (trailing zero assumed) to last buffer status block; used in buffer rotation. See FIB-NB.
FIBCBS	49,6	Current Buffer Size  All files except RND, SOR, DCM: Size in digits of remaining space in buffer when record length (FIBMRL for fixed length records or current record size for variable length output) exceeds FIBCBS, physical I/O is triggered (for input files, FIBARB is also compared to FIBACE).
FIBRCT	55,8	Record Count (redefines FIBACT)  Non-RND: Number of unique data records accessed (not access to record) during file processing; placed in output MTP ending labels; EOF pointer for output sequential DSK files.
FIBMBS	63,6	Maximum Block Size  All files except SOR, DCM: Buffer size (block size) in digits; gives actual size of blocks of fixed length records (except final block of MTP reel which may be short). Gives maximum block size for variable length records. Moved to FIBCBS at physical I/O.
FIB-NB	69,3	Next Buffer Pointer  All files: FIB relative index to current BSB; meaning depends on FIB-BA. If FIB-BA:1 set: after READ and prior to WRITE points to BSB for current buffer (buffer is available). If FIB-BA:1 not set: prior to READ and WRITE points to BSB for next request (buffer may not be available). Buffer rotation involves modifying FIB-NB

B 2000/B 3000/B 4000 MCPVI Programmer's Guide  
MCPVI Tables

FIB

Label	Relative Location /Size	Content/Meaning
		(incrementing by 40 or resetting to 200 when value exceeds FIBLBA x 10.).
FIBIOA	72,6	Input/Output Assignment Table Index
		All files: Address of IOAT (physical file table) entry for device attached to FIB; file can be OPEN or CLOSEd and not released; contains zeros if no file attached.
FIBTRN	78,1	Hardware or MCP Controlled Translation Type
		MTP, PTP, PTR, DCM: Type of code translation to be done on data before releasing buffer to program; for MTP, PTR, PTP, moved to FIB-D3 except as noted.  0 = PTR, PTP: Process 7-bit odd parity (append or strip high order bit). All files except PTP, PTR: No translation.  1 = PTR, PTP: Translate BCL/EBCDIC (6-bit odd parity).  2 = PTR, PTP: Process 8-bit, no parity (no translation).  DCM: (input): Non-standard translation, codes which have both upper and lower case characters, such as PTTC/6 and ASCII, have the lower case set translated to upper case EBCDIC.  4 = DCM: Standard MCP translation. MT7: Enable hardware translation for BCL/EBCDIC conversion.  5 = MT9: Enable hardware ASCII/EBCDIC translation.
FIBOPT	79,1	Optional File Flag and Miscellaneous
		Input files:  :1 File declared OPTIONAL: need not be present (end of file is forced on first access if OF message entered).  :2 MTP: MCP should display file label information on SPO at file OPEN.

Label	Relative Location /Size	Content/Meaning
		<p>:4 MTP (output): 1600 BPI tape (9-track) permitted for file. Bit is set at file OPEN if FIBST1:8 already set. At end of file OPEN, bit left on if MPE assigned (input or output).</p> <p>:8 MTP (output): 250 IPS tape (GCR) permitted for file.</p>
FIBLRA	80,4	<p>Logical Records per Area</p> <p>DSK (output, O/I only): Maximum number of records per disk area; MCP uses field only if FIBRPA is zero.</p>
FIBNAR	84,2	<p>Number of Areas</p> <p>DSK (output, O/I only): Maximum number of areas for disk file; ignored on input and I/O files (value in DFH is used).</p>
FIB-DA	86,1	<p>Disk Access Technique</p> <p>DSK:</p> <p>:1 Random.</p> <p>:1/ Sequential.</p> <p>:2 Sequential I/O.</p> <p>:4 Usercode flag for LOADMP and PACKUP.</p> <p>:8 File declared SHARED (HPT only) indicates that FIBMRL and FIBRPB must match sized in DFH at file OPEN; allows shared disk requests when SHRD option set.</p>
FIBDFN	87,2	<p>Disk File Number</p> <p>DSK: File Number of disk file (independent of FIBFNM) assigned sequentially according to order of declaration of disk files in source program; used in HPT space assignment if FIBDTK:1.</p>
FIBRSW	89,5	<p>Record Size in Words</p> <p>Size of record in words for fixed length, work area access records (FIBMRL/4); (size for variable length calculated from size</p>

FIB

Label	Relative Location /Size	Content/Meaning
		field in each record); cannot be zero or greater than 9999.
FIBEXT	94,6	Pointer to Security Attribute Storage Area
FIBRAD	100,1	<p>Read Address Flag; Block Access Flags</p> <p>Input files: Bits :1 and :2 control the method of determining the ending block address and the direction in which the file is accessed. If neither bit is set, the file is implied to be read forward; fixed length records and the ending address are taken from FIB-BB since the hardware RAD is not needed (that is, unblocked file or DSK). If the bits are set, RAD is used.</p> <p>:1 MTP: Read file forward. :2 MTP, SEQ DSK: Read file reverse.</p> <p>The following bits apply to DSK files:</p> <p>:4 DSK: After WRITE, perform READ CHECK operation for parity check.</p> <p>:8 RND: Do not examine buffers for block; physical read must be performed (READ and blocked WRITE); explicit SEEK request are ignored. Used to force physical I/O, for example, when programs modify data in buffers and require a fresh copy of the record.</p>
FIBOPN	101,1	<p>Internal File OPEN Flag</p> <p>All files: Miscellaneous flags used internally during file OPEN.</p> <p>:1 Initial file OPEN. :2 HPT output: file is output pseudo deck (ID = #00000). :4 File is output breakout file. :8 Reserved.</p>
FIBADR	102,8	<p>Disk Address of Current Block (redefines FIBSQ1, FIBSQ2, FIBSQL, FIBQAD)</p> <p>RND:</p>

Label	Relative Location /Size	Content/Meaning
		Used as work area to build the absolute disk address for the current operation. SEQ DSK: Used as a work area to build the absolute disk address during disk positioning.
FIBORG	110,1	File Type DSK: :2 Indexed I/O key file. :4 Indexed I/O data file. :8 Relative I/O data file.
	111,1	Reserved
FIBDKB	112,8	Relative Block Number (redefines FIBDCF) RND: Temporary internal storage for relative block number of file during READ/WRITE requests.
FIBKEY	120,6	Actual Key Location (redefines FIBABS, FIBJAM, FIBMTL) RND: Address of actual key (8 UN). MTP: Used internally to store first six digits of I/O descriptor during tape position.
FIBSBL	126,1	Buffer Flag :1 New area OPENed by GETDSK. :2 DPK: File is DPK file. :4 HPT: File is HPT or backup disk. :8 All files except RND, DCM, SOR: Buffer required flag; buffer exhausted on previous I/O request, new buffer needed for further processing (used for all work area access files except variable length).
FIBUNF	127,1	Random Disk Wait Flag; Tape Position Flag RND: Specifies program waiting for RND processing.

B 2000/B 3000/B 4000 MCPVI Programmer's Guide  
MCPVI Tables

FIB

Label	Relative Location /Size	Content/Meaning
		<p>1 = Buffer required for current request (I/O not yet initiated).</p> <p>2 = Program waiting I/O on current request (READ or implicit SEEK for blocked WRITE).</p> <p>3 = Program waiting buffer availability for next I/O (WRITE in process on buffer access file).</p> <p>MTP: Specifies program waiting for MTP position.</p> <p>2 = Waiting for all in process I/Os to complete before positioning begins.</p> <p>4 = Waiting for I/O on SPACE operation when moving tape.</p> <p>6 = Input file buffer fill in process after positioning complete.</p>
FIBFLM	128,1	<p>:1 File limits specified (DISK).</p> <p>:2 Search for reserved scratch tape.</p> <p>:4 Ignore channel 12 on printer (RPG).</p> <p>:8 DMS-II file (MCP only flag).</p>
FIBIX2	129,1	<p>IX2 Flag: Breakout Flags</p> <p>All files:</p> <p>:1 Directs MCP to move FIBARB to program IX2 after each READ or WRITE request (see FIBARB); generally used for buffer access files.</p> <p>:2 Save previous breakout disk file.</p> <p>:4 Use MTP for breakout (rerun every n records).</p> <p>:8 Use DSK for breakout (ignored if :4 set).</p>
FIBBCT	130,8	<p>Block Count</p> <p>All files: Number of unique blocks of data read or written during file processing (value for RLOG taken from IOAT).</p>



Label	Relative Location /Size	Content/Meaning
		For MTP, value inserted into output ending labels and checked when tape used as input.
FIBPOS	138,5	<p>File Position data (redefines FIBNAU, FIBBFF, FIBFFL, FIBDCO, FIBWTF, FIBBCF)</p> <p>PRN, SEQ DSK, MTP: Field used as 4 SN to hold data from POSN communicate.</p> <p>PRN: Channel or line skipping data (sign digit ignored).</p> <p>MTP,DSK: Number of records to skip (sign digit gives direction).</p> <p>DSK: @F@ in last digit causes EOF action to be taken on next disk request.</p> <p>Second digit used as internal flag; @F@ means all buffers refilled.</p>
FIBUSE	143,2	<p>USE Routine Exit Handling</p> <p>Used as branch table key when USE routine exited; identifies type of routing in process.</p> <p>00 = No further MCP action needed (input begin label); stalemate.</p> <p>06 = Output begin label (label still to be written).</p> <p>12 = Output end label (label still to be written).</p> <p>18 = Input end label (CLOSE to be completed).</p> <p>24 = End-of-page routine (no further action needed).</p> <p>30 = WRITE parity routine (logical I/O to be completed if work area access).</p> <p>36 = MCS DCP – write error.</p> <p>42 = READ parity routine.</p>
FIBRWT	145,1	<p>READ/WRITE Type</p> <p>All files except SOR, DCM: Type of I/O request to begin process. Bits 1 and 2 specify the general type of request as follows.</p> <p>0 READ request.</p> <p>:1 WRITE request.</p>

FTR

Label	Relative Location /Size	Content/Meaning
		<p>:2 RND: SEEK request. :1*:2 PRN, MTP, SEQ DSK: Position request.</p> <p>RND: In addition to the above, bits 4 and 8 specify shared disk operations yielding the following possible digit values:</p> <p>0 = Plain READ (ignore lock status). 1 = Plain WRITE (unlock implied if relevant). 2 = Plain SEEK (ignore lock status). 3 = Not valid for RND. 4 = LOCK (wait if currently locked). 5 = WRITE NO UNLOCK (block must have been locked previously by requestor). 6 = LOCK SEEK (do not wait if currently locked). 7 = Not valid. 8 = READ WITH LOCK (wait if currently locked). 9 = UNLOCK (block must have been locked previously by requestor). A = SEEK WITH LOCK (do not wait if locked currently). B = Not valid. C = READ UNTIL UNLOCKED (plain READ, wait if currently locked). D = Not valid. E = SEEK UNTIL UNLOCKED (plain SEEK, retry if currently locked). F = Not valid.</p>
FIBWKF	146,1	<p>Work File Flag; Miscellaneous</p> <p>:1 DSK: Insert requestor's mix number in second and third characters of file ID. Used to ensure file ID uniqueness, primarily for work files.</p> <p>:2 DSK: Insert processor number on which requestor running into fifth character of file ID. Used to ensure file ID uniqueness in shared disk systems.</p> <p>:4 Call TERM return at end of CLOSE. :8 Call TERM at end of CLOSE.</p>

Label	Relative Location /Size	Content/Meaning
FIBDTK	147,1	<p>Disk Assignment Technique</p> <p>DSK (output, O/I): Directs MCP to assign any disk needed during execution of program in particular ways. Used to improve I/O overlap on multi-channel or multi-subsystem configurations and/or to assign files to particular subsystems.</p> <p>0 = Use random disk assignment method. Assign each area (page) to a randomly selected EU by taking last two digits from interval timer, dividing by total number of EUs on subsystem(s) and using remainder to select an EU of those on subsystem(s). (Often good method for random files.)</p> <p>1 = Use FIBDFN as seed number for method above rather than random number. This method attempts to place entire file on same EU. (Often good method for programs with multiple output sequential files.)</p> <p>2 = Use area number for seed number. This method attempts to place each area of a file on a different EU. (Often good method for random files.)</p> <p>4 = Select EU from value in FIB-EU. If EU not on system use value as seed number. Method attempts to place all new areas on same EU. (Often good method for either random or sequential files when mix is well defined.)</p> <p>:8 Unused.</p> <p>DPK (output), O/I, I/O: Controls the area assignment of DPK files.</p> <p>:1 Assign areas beginning at a cylinder boundary; otherwise, space is assigned from the beginning of an available area.</p> <p>:2 Assign each area to successive packs (multipack files only); otherwise, assign space as available (for multipack files, preference given to the pack with the largest amount of available space).</p>

B 2000/B 3000/B 4000 MCPVI Programmer's Guide  
MCPVI Tables

FIB

Label	Relative Location /Size	Content/Meaning
		:4 Unused. :8 Unused.
FIBCHN	148,2	Channel Number (redefines FIB-EU)
		Non-DSK: Primary channel for I/O on this file.
FIBUNT	150,1	Unit Number
		Non-DSK: Unit Number of device assigned.
FIBAUD	151,1	Reserved
FIBULB	152,6	Label USE Routine Address (redefines FIBUST, FIBURE)
		All labeled files: Address of USE routine for user label handling. Values put in BASE: +34 through BASE: +39 by MCP, can be used to evaluate file status: BASE: +34:1 = FIB-IO (distinguish input/output). BASE: +35:1 = 0 for begin label, 1 for end label. BASE: +36:1 = 0 for file labels, 1 for reel labels (MTP). BASE: +37:3 = Reel number (MTP).
FIBUER	158,6	I/O Error USE Routine Address (redefines FIBUER)
		MTP, DSK, PRN, PTP, PTR: Address of USE routine for I/O error handling. Values put in BASE: +34 through BASE: +39 by MCP, can be used to evaluate reason for entry: BASE: +34:1 = FIBRWT bits :1 and :2 (distinguish READ/WRITE errors). BASE: +37:3 = Reel number (MTP).
FIBUEP	164,6	End-of-page USE routine Address (redefines FIBUTE, FIBPIN)
		PRN: Address of USE routine entered when 12-punch sensed in printer carriage tape; meaningless if file diverted to backup.

Label	Relative Location /Size	Content/Meaning
		<p>MTP: Temporary storage for number of blocks to space during positioning.</p> <p>DSK: Address of file limit table (COBOL files having file limits).</p> <p>SEQ DSK (input): EOF pointer value (8SN) from DFH (obtained at OPEN) used by COBOL DEBLOCK routine (also redefines FIBUPS).</p>
FIBUPS	170,6	<p>Pocket Select USE Routine Address (redefines FIBPON)</p> <p>SOR: Address of pocket select USE routine (SORTER 4).</p> <p>HPT: Current file limit table entry address (COBOL programs having file limits).</p>
FIBRPA	176,8	<p>Number of Records per Disk Area</p> <p>DSK (output, O/I): Number of records declared per disk area.</p> <p>DSK (input, I/O): Value calculated from values in DFH and FIBRPB; used only if FIB specifies FIBMRL and FIBRPB different from values in DFH.</p> <p>MTP: Temporary storage for number of blocks to move tape during positioning.</p>
FIBCOD	184,2	<p>Descriptor OP Code Storage</p> <p>All files except RND, DCM, SOR, SEQ I/O: Contains OP code for processing of file; placed into FIB-OP during READ/WRITE; OP code for other file types developed at READ/WRITE time.</p> <p>DCM: Temporary storage for request type from communicate.</p>
FIBBC1	186,1	<p>OPEN/CLOSE Communicate Variant</p> <p>All files: Holds first variant digit from OPEN/CLOSE</p>

FIB

Label	Relative Location /Size	Content/Meaning
		communicate during OPEN and CLOSE. See FIBBC2.
FIBBC2	187,1	OPEN/CLOSE Communicate Variant All files: Holds second variant digit from OPEN/CLOSE communicate during OPEN and CLOSE. See FIBBC1.
FIBLAB	188,6	All files: Addresses the byte before the MFID field of the label area. The size and format of the area is dependent on the file and label type definitions. All files have at least a minimum label area containing the file name fields. Addresses must be modulo 4.
FIBLAE	194,6	All files: This field addresses the 1st digit past the label area. Depending on the file and label type definitions, it may function as the end address in I/O descriptors for label operations. Addresses must be modulo 2 or 4 depending on hardware and label type declared.

FIB REDEFINITIONS

The following are redefinitions to some of the FIB fields previously described.

Label	Relative Location /Size	Content/Meaning
FIBACT	55,8	Actual Key Storage for RND (redefines FIBRCT) RND: Used as work area to hold the current record number (actual key).
FIBQAD	102,6	Address of SOR Queue Element (redefines FIBADR, FIBSQ1, FIBSQ2, FIBSQ3, FIBSQL) SOR: Absolute address of IOQ element reserved at file OPEN for SOR.

Label	Relative Location /Size	Content/Meaning
FIBSQ1	102,1	Previous I/O Request Type (redefines FIBADR, FIBQAD)
		<p>SEQ I/O DSK: Storage of value of FIBRWT for previous I/O request; set from FIBSQ2 at each request.</p> <p>0 = Request was READ; if current request also a READ, must change pointers to access proper record.</p> <p>1 = Request was WRITE; pointers are correct for current request.</p>
FIBSQ2	103,1	Current I/O Request Type (redefines FIBADR, FIBQAD)
		<p>Storage of value of FIBRWT for current I/O request.</p> <p>0 = Request is READ; if previous request was also a READ, pointers (FIBARB, FIBCBS) still point at previous request and must be changed; otherwise, pointers are correct.</p> <p>1 = Request is WRITE; must set FIBSQ3:2 to force physical I/O when block is completed; advance pointers to next record.</p>
FIBSQ3	104,1	Current I/O Request Type (redefines FIBADR, FIBQAD)
		<p>SEQ I/O DSK:</p> <p>:1 Internal flag specifying both FIBSQ1 and FIBSQ1 indicates READ and must cycle back through READ/WRITE to access correct record; bit reset during return cycle.</p> <p>:2 Specifies write activity on current block and when the block is exhausted, must be written.</p> <p>MTP:</p> <p>:4 Retry short/long records.</p> <p>:8 Ignore short/long records.</p>
FIBSQL	105,5	Sequential I/O Record Length (redefines FIBQAD, FIBADR)
		SEQ I/O DSK:

FIB

Label	Relative Location /Size	Content/Meaning
		Storage area for record length of last record accessed.
FIBDCF	112,1	DCM FILL Flag (redefines FIBDKB) DCM: 0 = No FILL given. 1 = FILL initiated.
FIBABS	120,6	DCM actual Key location (redefines FIBKEY, FIBJAM) DCM (transparent): Address of field containing record size for WCRT.
FIBJAM	120,6	Address of Jam/Missort USE routine (redefines FIBABS, FIBKEY) SOR: Address of USE routine for handling jam/missort/EOF (SORTER 5).
FIBPSN	138,5	Redefine FIBPOS TAPE: FIBPOS as 4SN.
FIBNAU	138,3	Number of Disk Areas (redefines FIBPOS, FIBBFF, FIBFFL). DSK (output only): Number of disk areas assigned during run. Used internally during CLOSE.
FIBBFF	138,1	Buffer Status Flag (redefines FIBPOS, FIBNAU) DCM (stream mode): Program has initiated I/O request (buffer is ready for data).
FIBFFL	139,1	Stream Mode Flip Flag (redefines FIBPOS, FIBNAU). DCM (stream mode): Specifies operation is WCRC with stream.
FIBDCO	140,1	Stream Mode OP Storage (redefines FIBPOS, FIBNAU) Holds low order digit of DCM OP code for current request.
FIBWTF	141,1	Wait Flag (redefines FIBPOS) DCM (stream mode):



Label	Relative Location /Size	Content/Meaning
		1 = Waiting IOC.
FIBBCF	142,1	Buffer IOC Flag (redefines FIBPOS)
		DCM (stream mode): IOC has occurred before program request for record.
FIB-EU	148,2	Selected EU for HPT Assignment (redefines FIBCHN)
		DSK: Used for subsystem and disk space assignment if FIBDTK:4. At file OPEN (output, O/I) If first digit is nine, second digit gives relative number of subsystem desired for file (value must be eight or less; zero means default). Indicates all areas of the file are to be allocated on the designated subsystem(s). (Subsystem designation remains with file in DFH.) If first digit is not nine, entire field designates specific EU (by number) for file. If EU on system, associated subsystem is selected; else default subsystem(s) are used.
FIBUST	152,6	RND DSK: Address of USE routine for shared disk stalemate handling.
FIBURE	152,6	Read Error USE Routine (redefines FIBUST, FIBULB)
		SOR: Address of general read error USE routine. (unencoded, cannot read, and so on, SORTER 1).
FIBUAE	158,6	Amount Error USE Routine (redefines FIBUER)
		SOR: Address of USE routine for amount error handling (SORTER 2).
FIBUTE	164,6	Address of Transit Error USE Routine (redefines FIBUEP, FIBPIN)
		SOR: Address of USE routine to handle transit field errors (SORTER 3).

FIB

Label	Relative Location /Size	Content/Meaning
FIBPIN	164,6	Ping Address for DCM Stream Mode (redefines FIBUEP, FIBUTE)
		DCM (stream mode): Address of beginning of buffer area.
FIBEOF	164,8	Disk File EOF Pointer (redefines FIBUEP)
		DSK: Disk File EOF pointer.
FIBPON	170,6	Pong Address for DCM Stream Mode (redefines FIBUPS)
		DCM (stream mode only): Address of second half of buffer area (put in FIB-BB).

## FILE BUFFER DESCRIPTORS

File Buffer Descriptors (also called Buffer Status Blocks) are located immediately following the FIB for a file. There is one Buffer Status Block (BSB) for each buffer declared for the file. (Note that three alternate areas means four buffers.)

Each BSB is 40 digits long. Each is set up by the compiler and used to handle the I/O operation that is to be performed on that buffer.

A BSB can be addressed relative to the FIB. For example, if a file has three buffers, there is a BSB at FIB: +200, FIB: +240, and FIB: +280.

## BUFFER STATUS BLOCK LAYOUT

The following descriptions are BSB-relative instead of FIB-relative.

Label	Relative Location /Size	Content/Meaning
FIBBSW	0,4	Buffer Status Word
		All files: The first three digits of this field contain the high order digits of the hardware result descriptor (R/D) which occurred at the completion of the physical I/O operation for this buffer. If the R/D indicates an error condition, the last digit contains a code describing the particular error. See table 5-1.
FIB-OP	4,2	I/O Descriptor Code
		All files: OP code for descriptor. Set from FIBCOD for files other than RND, SEQ I/O DSK, SOR and DCM. For these files, value developed during READ/WRITE. Contains @FF@ if file OPENed by optional file mechanism.
FIB-D1	6,1	Variant Digit for Descriptor
		Value depends on file type; not used for some types. Some uses are: Part of HPT EU specification. MTP density specification. PRN/PCH spacing/stacker select.
FIB-D2	7,1	Variant Digit for Descriptor
		DSK, MTP, PRN: Generally contains unit number designation

B 2000/B 3000/B 4000 MCPVI Programmer's Guide  
MCPVI Tables

BSB

Label	Relative Location /Size	Content/Meaning
		of device attached. Meaningless for some hardware types.
FIB-D3	8,1	Variant Digit for Descriptor
		Meaning varies with hardware type.
FIB-D4	9,1	Variant Digit for Descriptor
		Meaning varies with hardware type.
FIB-AA	10,6	Beginning Address of Buffer
		Address of beginning of buffer. Value must be modulo 4. Acts as terminating address for read backward operations (MTP).
FIB-BB	16,6	Terminating Address of Buffer
		Address of digit following buffer. For stream mode DCM, address of second half of buffer. Value must be even, and must be modulo 4 for MTP, DSK. Acts as terminating address for all input operations and all output operations except PRN, PCH.
FIB-AD	22,6	Disk File Address
		DSK: Low order six digits of disk address requested on READ or WRITE operation (high order digit consists of low order two bits of FIB-D2). Temporary storage for enable R/D (low order four digits of field).
FIBRCW	28,6	Return Control Word (redefines FIBBL#, FIBFSA).
		Files having USE routines except SOR: Program address to which return must be made after USE routine exited. (FIBRCW used in first BSB only.)
FIBBL#	28,8	Random Block Number (redefines FIBRCW, FIBACE, FIBFSA, FIBBBA).
		RND: Zero relative block number of block currently in buffer; used to determine if physical I/O needed to access desired record (if FIBRAD:8 not set).

Label	Relative Location /Size	Content/Meaning
FIBFSA	28,6	Flow Stopped Address (redefines FIBBL#, FIBRCW)
		SOR (first BSB only): Address of flow stopped label taken from communicate parameters.
FIBACE	34,6	Actual Ending Address of Buffer (redefines FIBBBA, FIBBL#, FIBHDK)
		All files except SOR, RND: Ending Address of block in buffer. May differ from FIB-BB for short blocks (for example, final blocks on MTP reel or variable length records). Set from hardware RAD operation or FIB-BB depending on FIBRAD value. Used to detect end of buffer and need for physical I/O on input files. See FIBCBS.  SOR: Storage for time interval passed if program is to be aborted due to pocket select routine taking too much time.
FIBBBA	34,6	Black Band Address (redefines FIBACE, FIBBL#, FIBAADK)
		SOR (first BSB only): Address of black band label taken from communicate parameters.
FIBHDK	36,3	Hashed Disk Address (redefines FIBACE, FIBBBA)
		RND: Randomized value of disk address for current buffer (first three digits and last three digits of FIB-AD NOred together); used as check beyond FIBBL# to determine presence of requested block.
FIBSEK	39,1	SEEK Buffer Flag (redefines FIBACE, FIBBBA)
		:8 Buffer used for explicit SEEK. :4 <<Available>> :2 <<Available>> :1 <<Available>>

BSB

**Table 5-1. Result Descriptor Codes**

<b>Digit 4 Value</b>	<b>Meaning</b>
0	No exception
1	Invalid R/D
2	Invalid I/O Descriptor (see table 5-2)
3	Parity error
4	Memory parity error during I/O
5	Not ready
6	End of file (for example, tape mark)
7	End of medium (for example, EOT)
8	Reserved
9	Reserved
A	Short record (MTP)
B	Long record (MTP)
C	End-of-page (PRN)
D	Reserved
E	Special error ignored
F	All errors ignored

If digit four is a 2 (invalid I/O descriptor), digit one is a 9 and digits two and three further define the error as shown in table 5-2.

**Table 5-2. I/O Descriptor Errors**

<b>Digits 2 and 3 Value</b>	<b>Meaning</b>
01	Write or unlock to record not locked by requestor (shared disk)
02	I/O timeout (no R/D returned)
10	End address limit error
20	Attempt to write MCP disk
40	Invalid I/O (hardware detected)

## INPUT/OUTPUT ASSIGNMENT TABLE

The Input/Output Assignment Table (IOAT) is also known as the Device Assignment Table (DAT). For each hardware unit (except disk and pack) declared to the MCP, there is a corresponding entry in the IOAT. For disk and pack, entries are maintained in the EU Table (EUTAB).

The physical attributes of a hardware device are encoded into an IOAT entry. Some of the information contained in an IOAT entry includes hardware type, channel, unit, device status, and translation type.

When a program OPENS a file, an IOAT entry, corresponding to the hardware type requested or allowed to the file, is assigned to the program. This same IOAT entry remains attached to the program until the associated file is CLOSED RELEASE or the job goes to EOJ.

Most hardware devices on a system can only be accessed by one program at a time; for example, only one program can be using a printer. In such cases, the IOAT entry is linked to the program. Disk type devices and the DCP can be accessed by multiple users. But the MCP only maintains one IOAT or EUTAB entry for each of these units. When a program OPENS a file assigned to one of these hardware types, the MCP makes a copy of the actual IOAT entry and attaches the copied version to the program. All the file actions that the program performs will refer to this copied (soft) IOAT.

For disk type files, the soft IOAT is allocated along with the file header in 200 digits of contiguous memory. For a DCP file, the IOAT is allocated in the program external DCP buffer.

Programmatic access to the IOAT of a file is not possible since the IOAT entry is outside the bounds of the program BASE and LIMIT.

The MCP also maintains an IOAT on disk. This IOAT is a skeleton table which indicates what channels, units, and device type are declared to the system. From the disk version, the in-memory IOAT is built when the MCP is Halt/Loaded.

## IOAT LAYOUT

The following describes the information found in an IOAT entry.

Label	Relative Location /Size	Content/Meaning
IO-HDW	0,2	Hardware Type (Refer to Table 1-1 for Specific Values)
IO-HDS	2,1	Supplementary Hardware Type TAPE: :8 GCR :4 PE :2 9-Track :1 7-Track CARD READER/PUNCH: 0 = Standard 80 COL card reader/punch

B 2000/B 3000/B 4000 MCPVI Programmer's Guide  
MCPVI Tables

IOAT

Label	Relative Location /Size	Content/Meaning
		PRINTER: 0 = Standard Printer :8 <<Available>> :4 Translate Table Printer :2 <<Available>> :1 Train Printer SORTER: 0 = All others 1 = 4A SYSTEM SPO 1 = TC 4000 0 = All others DISK PACK: 0 = Type unknown 1 = 215 Pack 2 = 225 Pack 3 = 235 Pack 4 = 206 Pack (interlaced) 5 = 206 Pack (sequential) 6 = 207 Pack (interlaced) 7 = 207 Pack (sequential)
IO-UNT	3,1	Unit Number
IO-CHN	4,2	Primary I/O Channel
IO-LNK	6,2	I/O Queue Access Link
IO-STA	8,2	Device Status Digit Link
IO-QUE	10,2	I/O Queue Element Count
IO-MIX	12,2	Mix Number of User
IO-FIB	14,6	User FIB Address (base relative)
IO-MSK	20,4	I/O Error Ignore Mask
IO-NSC	24,1	Unit Status Digit 0 = Unit ready and unassigned 1 = Unit not ready 2 = Control load required (TPR, DPK, DCP)



Label	Relative Location /Size	Content/Meaning
		3 = Waiting, not ready 4 = <<Available>> 5 = DPK waiting, ready for initial status 6 = MTP waiting, ready for initial status 7 = PTP waiting, ready for rewind 8 = <<Available>> 9 = Device initialization requested A = Unit bypass (not tested or assigned) B = Unit ready (assigned: not to be tested) C = Unit not available: slow/no IOC during status D = Unit not available: Invalid I/O during status E = Unit requires STAT-2 F = Unit not available: XU-ed
IO-ERC	25,3	Total Error Count on File
IO-ERT	28,2	Total Retry Count
IO-BCT	30,8	Block Count
IO-ID	38,12	File Identifier
IO-MFD	50,12	Multifile Identifier
IO-USE	62,1	Device Usage 0 = Standard usage 1 = Direct I/O 2 = Trace Printer 3 = Pseudo-reader deck (pack output) 4 = Pseudo-card reader (pack) 5 = Punch backup pack 6 = Printer backup pack 7 = Pseudo-reader deck (disk output) 8 = Pseudo-card reader (disk) 9 = Punch backup disk A = Printer backup disk

B 2000/B 3000/B 4000 MCPVI Programmer's Guide  
MCPVI Tables

IOAT

Label	Relative Location /Size	Content/Meaning
		B = Printer backup tape blocked C = Printer backup tape unblocked D = <<Available>>
IO-WIO	63,1	:8 Waiting close queue flush :4 Queued I/O required for N-SEC testing :2 Waiting position or file flush :1 Waiting I/O complete
IO-ST1	64,1	:8 Inhibit I/O on this file :4 Unit in use :4/ Unit available :2 Input :2/ Output :1 Open :1/ Closed
IO-ST2	65,1	:8 Call terminate at end of CLOSE :4 CLOSE called by terminate :2 CLOSE called by exception processing :1 EOF sensed
IO-ST3	66,1	:8 File in use by SORT intrinsic :4 Unit is DLP type device :2 User table entry has been made (unit logged) :1 User program not in memory
IO-ST4	67,1	:8 Time-sharing IOAT :4 <<Available>> :2 TSM IOAT pushed but FIBIOA not updated :1 TSM process = type I :1/ TSM process, type II

The following descriptions are for disk.

Label	Relative Location /Size	Content/Meaning
IO-ADR	68,8	Disk Address Next I/O
IO-AR#	76,2	Current Area Number
IO-FS1	78,2	FPM Slots Assigned
IO-FS2	80,2	FPM Slots In Use
IO-RBA	82,7	Remaining blocks in area
IO-DSK	89,1	OPEN Type
		:8 Random :8/ Sequential :4 OPENed reverse :2 COBOL code file :1 Standard code file
IO-DK2	90,1	
		:8 Waiting address block core for file OPEN :4 File declared as SHARED (SHRD DISK) :2 File had breakout :1 <<Available>>
IO-CLA	91,1	File Classification
		:8 Private :4 Information :2 Public :1 Free :1/ Control
IO-PK1	92,1	
		:8 Base pack not resident on system :4 Base pack type restricted :2 Base pack type master :1 Reserved
IO-PK2	93,1	
		:8 Pack overflow specified :4 Waiting delayed OPEN (Pack) :2 Waiting powered off in use pack :1 <<Available>>
IO-HPT	94,6	Disk File Header Address

IOAT

The following descriptions are for split cylinder disk pack files.

Label	Relative Location /Size	Content/Meaning
IO-CYL	78,1	<<Available>>
IO-RPA	79,3	Remaining Partitions in Area
IO-RBP	82,7	Remaining Blocks in Partition

The following descriptions are for standard devices.

Label	Relative Location /Size	Content/Meaning
IO-UST	68,3	Result Descriptor Last Status Update
IO-LKS	71,1	Unit Status :8 Unit Saved :4 Unit to be Saved :2 Saved by MCP (secure or backup) :1 Unit Locked
IO-LBL	72,1	:8 Label sensed :4,2,1 0 = Omitted label 1 = Burroughs standard label 2 = ANSI standard label 3 = B 6700 ANSI label 4 = ANSI label, current MCP 5 = B 3500 modified ANSI label, MCP and CP 6 = B 3500 modified ANSI label, MCPV 7 = LABEL1 installation label
IO-MOD	73,1	:8 BINARY card input file :4 READ with translation :2 Status change :1 ENABLE allowed for STATUS and I/O error
	74,1	<<Available>>

Label	Relative Location /Size	Content/Meaning
IO-AUT	75,1	Miscellaneous
		:8 Train printer auto train load flag :4 SHARED tape flag (Save on CLOSE and H/L) :2 << Available >> :1 << Available >>

The following descriptions are for data communications devices.

Label	Relative Location /Size	Content/Meaning
IO-DCN	76,2	Buffer Number
IO-DCR	78,2	Number of I/O Requests on Disk
IO-DCK	80,2	Next Disk Request to Read
IO-ACT	82,6	Action Label Storage
IO-RDA	88,6	Result Descriptor/Address Label Storage
IO-TRN	94,1	Translate Table Index
IO-SPO	95,1	Remote SPO Capability Level
IO-DC1	96,1	:8 Dialed line :8/ Leased line :4 Print system messages on remote SPO :2 ENABLE/FILL completed :1 ENABLE/FILL initiated
		:8 Need overlap message :4 Remote SPO waiting LOG-OFF ENABLE :2 Device is on multi-line control :1 RJE adapter type: 0 = SYNC 1 = ASYNC
IO-DC2	97,1	

IOAT

Label	Relative Location /Size	Content/Meaning
IO-DC3	98,1	:8 Remote SPO
		:4 Data stream operations in process
		:2 <<Available>>
		:1 <<Available>>
	99,1	<<Available>>

The following descriptions are for an OCS display device.

Label	Relative Location /Size	Content/Meaning
	60,22	<<Available>>
IO-DQ	82,6	Last Automatic Message Display Time in seconds
IO-OCS	88,4	Address of OCS Buffer in KD
	92,2	<<Available>>
	94,6	Identical to Data Communications Device

The following descriptions are for magnetic tape devices.

Label	Relative Location /Size	Content/Meaning
IO-RL#	76,3	Reel Number
IO-CAN	79,5	Physical Tape Number
IO-VAR	84,1	Density/Parity Variant for Descriptor
IO-MTS	85,1	:8 Purge after rewind
		:4 Hard ASCII translate (Mod IV Mag tape)
		:2 Previous OPEN type: OUT
		:2/ Previous OPEN type: IN
		:1 Scratch tape
IO-MTT	86,1	:8 Unload tape after rewind
		:4,2,1 Translate type (labels):

Label	Relative Location /Size	Content/Meaning
		0 = No translate required 1 = Internal BCL 2 = External BCL 3 = EBCDIC 4 = 7-bit ASCII 5 = 8-bit ASCII 6 = <<Available>> 7 = <<Available>>
IO-PSR	88,1	Branch Table Address for Position Return
IO-PSF	89,1	:8 <<Available>> :4 Ignore tapemarks during position :2 Hold overlay during position :1 Reverse position
IO-SKP	90,4	Skip Block Count on OPEN or Position
IO-HPT	94,6	Pseudo-Device Attribute Pointer

CHART DELETED

The following descriptions are for Reader/Sorter devices.

Label	Relative Location /Size	Content/Meaning
	68,18	<<Available>>
IO-SX1	86,4	Invalid I/O Count (PCKT-SLCT)
IO-SX2	90,4	Invalid I/O Count (ENABLEs)
IO-QAD	94,6	Queue Element Address (absolute)

The following descriptions are for train printers.

Label	Relative Location /Size	Content/Meaning
IO-DEF	76,12	Default Translator File-ID
IO-LDT	88,12	File-ID Presently in Translator

IOAT

The following descriptions are for DCPs

<b>Label</b>	<b>Relative Location /Size</b>	<b>Content/Meaning</b>
IO-FFD	50,12	Firmware File-ID for this DCP
IO-LHR	74,1	Halt/Load Requirements :8 <<Available>> :4 <<Available>> :2 LH required for this DCP :1 Warm Load LH permitted
IO-ND#	75,1	Number of this DCP
IO-BUF	76,8	DCP Entry Address (absolute)
IO-DCF	84,1	:8 Host output suspended (S-memory full) :4 Cancel of default read initiated :2 Default read in progress :1 <<Available>>
	85,1	<<Available>>
IO-ERF	86,2	Diagnostic Error Code
IO-MEM	88,4	Highest S-memory address on DCP in Hexadecimal Words
IO-OPN	92,2	Number of MCS files in use on DCP
	94,2	<<Available>>
IO-ERI	96,4	Diagnostic Error Information



## JOB REFERENCE TABLE

The Job Reference Table (JRT) is considered the schedule table for the MCP. When a COMPILE or EXECUTE request is entered, the MCP creates a JRT entry for the program requested. A JRT entry must be made in order for a program to enter the mix.

A program JRT entry contains information on priority, core requirements, precedence links, location of the code file on disk, the manner in which the program is initiated, and various other information about the program. Some of the information is retained for the Run Log; some for program execution.

The MCP handles one JRT entry in memory at a time. The entire JRT is on disk; an entry is read as needed.

## JRT LAYOUT

The following describes the fields in the JRT.

Label	Relative Location /Size	Content/Meaning
JRT-SC	0,1	Entry Status Code
		0 = Not active 1 = Schedule in process 2 = Program ready for mix entry 4 = Program executing 8 = Program terminating
JRT-ET	1,1	Program Initiate Code
		:8 EXECUTE phase to be scheduled (terminate) :4,2,1 0 = EXECUTE 1 = COMPILE and GO (COMPILE phase) 2 = COMPILE SYNTAX 3 = COMPILE LIBRARY 4 = COMPILE and GO (EXECUTE phase) 5 = RUN 6 = Share 7 = COMPILE SAVE (COMPILE phase)
JRT-PG	2,1	Special Program Code
		1 = Program is a generator 2 = Program is DMPALL 3 = Program is LOADMP 4 = Program is PACKUP 5 = Program is DSKOUT 6 = Program is an MCS (DCP) 7 = Time-sharing process 8 = Shared area (TSM)

JRT

Label	Relative Location /Size	Content/Meaning
		9 = <<Available>> A = Compiler in shared area B = DMS control program
JRT-DO	3,1	:8 Initiated through ZIP communicate :4 Execute with LOCK :2 Initiated through pseudo-card reader :1 Charge number supplied
JRT-LQ	4,1	:8 Memory dump request on abnormal termination :4 Initiated from remote SPO :2 Time limit supplied :1 Label equate supplied
JRT-FG	5,1	:8 Bound flag (MCP intrinsic) :4 Test flag :2 Debug flag :1 Rerun flag
JRT-TM	6,1	:8 Set trace at BOJ :4 MCP work file in use by program (%MX0P0) :2 No code file at terminate :1 Syntax error
JYT-FF	7,1	:8 User program must be used :4 MCP intrinsic must be used :2 Secondary execution label equate disk obtained :1 Primary execution label equate disk obtained
JRT-SA	8,1	:8 Extension table assigned :4 Privileged program

B 2000/B 3000/B 4000 MCPVI Programmer's Guide  
MCPVI Tables

JRT

Label	Relative Location /Size	Content/Meaning
		:2 << Available >> :1 << Available >>
JRT-CL	9,1	Classification :8 Private file :4 Information file :2 Public file :1 Free file :0 Control
JRT-US	10,2	Maximum Number of Users in Shared Area
JRT-ID	12,12	Program Identification
JRT-MF	24,12	Multiprogram Identification
JRT-RQ	36,2	DAT/MIX/PCR Requestor Code
JRT-EU	38,2	Disk EU for Generator Code File
JRT-RL	40,4	Run Log ID Number
JRT-CN	44,6	Charge Number
JRT-SP	50,1	Schedule Priority
JRT-MP	51,1	Memory Priority
JRT-RP	52,1	Process Priority
JRT-TL	53,5	Time Limit (in seconds)
JRT-DF	58,2	Disk File Count
JRT-IC	60,2	Number of IOATs Assigned (during schedule)
JRT-DS	60,2	User DS Code (from DS/DP), redefines JRT-IC
JRT-RJ	62,2	RJE Originator Key
JRT-PX	64,1	:4 Code file is on disk pack
JRT-XX	65,6	Standard core requirement (JRT in process); Terminate stack address (during term close)
JRT-YY	66,5	Time Job Entered Schedule (schedule)
JRT-VA	71,2	Value Address
JRT-VL	73,1	Value Length
JRT-VD	74,8	Value Data
JRT-FC	74,2	Number of files declared (when JRT-VL = 0)
JRT-SG	76,6	Disk Segments in program (when JRT-VL = 0)
JRT-IN	82,8	Disk Address of First Insert (or value) Segment

B 2000/B 3000/B 4000 MCPVI Programmer's Guide  
MCPVI Tables

JRT

<b>Label</b>	<b>Relative Location /Size</b>	<b>Content/Meaning</b>
JRT-FH	90,8	Disk File Header Address: Program File
JRT-FD	98,3	Decrement to Directory Header: Program File
JRT-FX	101,4	Index into Directory Header: Program File
JRT-EH	105,8	Disk File Header Address: Label Equate File
JRT-ED	113,3	Decrement to Directory Header: Label Equate
JRT-EX	116,4	Index into Directory Header: Label Equate
JRT-SH	120,8	Disk File Header Address: Stopped File
JRT-SD	128,3	Decrement to Directory Header: Stopped
JRT-SX	131,4	Index into Directory Header: Stopped File
JRT-RH	135,8	Disk File Header Address: Rollout File
JRT-RD	143,3	Decrement to Directory Header: Rollout File
JRT-RX	146,4	Index into Directory Header: Rollout File
JRT-BJ	150,5	Beginning of Job Time (in seconds)
JRT-BR	155,5	Previous Breakout Disk File Number
JRT-DT	160,8	Accumulated Direct Time
JRT-PT	168,8	Accumulated Prorated Time
JRT-WT	176,8	Accumulated Waiting IOC Time
JRT-NM	184,12	Breakout Tape/Pack Name
JRT-TI	184,8	Total Stopped Time (in milliseconds)
JRT-BK	196,2	Breakout Parameters
	198,2	<< Available >> (during execution)
JRT-GT	116,20	Trace Parameters (during schedule)
JRT-SU	156,1	Generated Program's Security use
JRT-SN	157,1	Generated Program's Sensitive data flag
JRT-GD	158,12	Generated Program's Security guard
JRT-FA	170,12	Generated Program's Security family

## LABELS

The file labeling mechanism of the MCP creates the sophisticated hardware/software/systems operator/user program interface necessary to provide all Burroughs customers with unexcelled programming and systems operational ease.

An MCP primary concern is to create an efficient multiprogramming operation where external intervention is held to an absolute minimum and maximum throughput is attained by incorporating automatic file recognition and orderly file creation methods without resorting to a job control language with an inherent confusion factor.

The file labeling techniques used by the MCP provide the system operator with a magnetic tape pre-mount capability (on any tape unit available) so that jobs residing in the schedule will not be delayed upon entering the mix, because of the need to mount a required input file or scratch tape on a specific unit before a given job can commence.

Label handling is a function of the MCP. However, provisions have been made to allow user access to file labels upon input/output through USE routines as specified in the various programming languages.

## EXTERNAL LABEL FORMATS

Most file types can accommodate external labels. Exceptions on the system include disk and data communications files. The MCP recognizes several label types; unlabeled files with unrecognizable labels are acceptable to the system, though operator intervention is required to achieve file assignment to a program.

### Burroughs Standard Label

This label is the standard label for Burroughs B 4000/B 3000/B 2000 Series systems and serves as both the beginning and ending label for all reels of a file (where applicable). Beginning file and beginning reel labels are distinguished by the value of the reel number field. Ending file and ending reel labels are distinguished by a flag in the label field. The label format is given in table 5-3.

**Table 5-3. Burroughs Standard Label Format**

<b>Position (Byte)</b>	<b>Length (UA)</b>	<b>Contents</b>
0	1	Blank
1	7	LABELbb
8	1	Zero
9	6	Multifile-ID field; zeros if no MFID
15	1	Blank
16	1	Zero
17	6	File identifier (FID); blanks for scratch files
23	1	Blank
24	3	Reel number for tape files
27	5	Creation date (Julian format)

LABELS

**Table 5-3. Burroughs Standard Label Format (Cont)**

<b>Position (Byte)</b>	<b>Length (UA)</b>	<b>Contents</b>
32	2	Reserved for cycle; accessible to user to distinguish multiple runs of a program on a single day. Contains 01 by default
34	5	Purge date (Julian); date on which MCP may use tape as scratch (if write ring present)
39	1	Sentinel (0 = end-of-file; 1 = end-of-reel, ending label only)
40	5	Block count; ending label only
45	7	Record count; ending label only
52	1	Reserved (flag for checkpoint tapes on B 5500)
53	5	Physical tape number (inserted by operator command)
58	21	Reserved

When a label is created, the size may be declared to be greater than the 80 characters described above. Any other space allocated may be employed by the user in any manner desired. The MCP does not interrogate or use any area beyond the normal 80-byte label area.

Certain differences exist between labels created on the various Burroughs systems including recording mode, code, and identifier sizes. The MCP recognizes standard labels created on any of the systems, though identifiers longer than six characters are truncated.

A tape mark is written after the beginning label and before the ending label, and after the last ending label of a physical reel. Figure 5-1 gives some examples of tapes created with a Burroughs standard label.

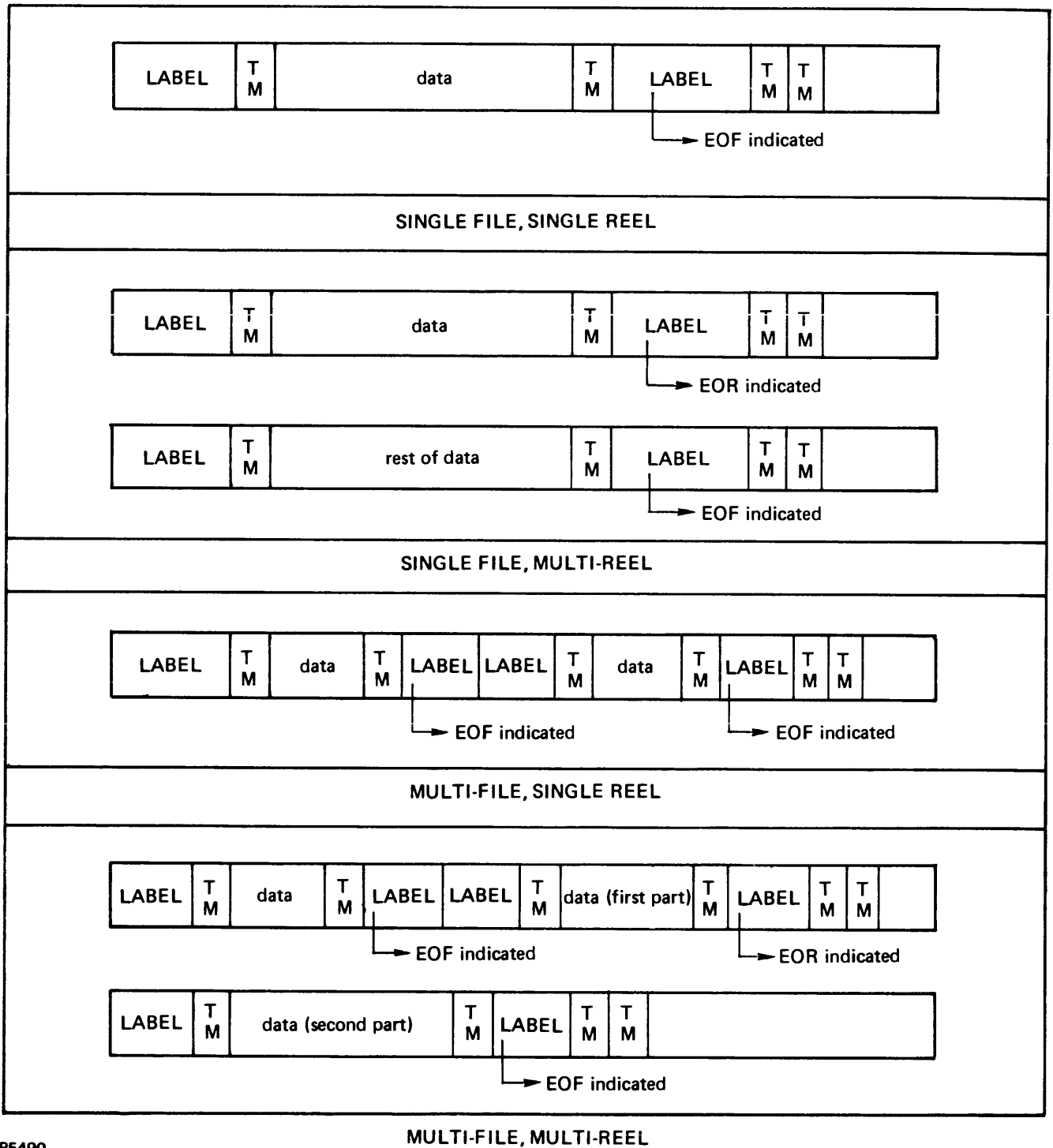
### USASI Standard Label

The USASI standard label is recognized by the MCP. Further, this label type can be generated by the MCP if such is declared by a program.

The USASI label consists of two or more physical records. The first is a volume record which defines the physical tape reel. (This record is not present in user programs but is recognized or created by the MCP directly.) The format of this record on the system is shown in table 5-4.

**Table 5-4. USASI Label Record 1**

<b>Position (Byte)</b>	<b>Length (UA)</b>	<b>Contents</b>
0	4	VOL1
4	1	Blank
5	5	Volume identifier
10	69	Unused
79	1	1



P5490

MULTI-FILE, MULTI-REEL

Figure 5-1. Examples of Tapes Created with Burroughs Standard Label Format

LABELS

The second record, following the VOL label is the first file header label. Refer to table 5-5.

**Table 5-5. USASI File Header Label Record**

<b>Position (Byte)</b>	<b>Length (UA)</b>	<b>Contents</b>
0	4	HDR1 for beginning label EOR1 for end-of-reel label EOF1 for end-of-file label
4	6	File ID
10	11	Blank
21	6	Multifile ID
27	1	0
28	3	Reel number
31	4	0001
35	7	Blank
42	5	Creation date (YYDDD)
47	1	Blank
48	5	Purge date (YYDDD)
53	1	Blank
54	6	Ending label block count
60	6	MCPV-1
66	14	Blank

On tapes from other systems, additional HDR labels may follow the first. For example, the B 6700 system creates an HDR2 label that defines the file physical characteristics.

Following the HDR labels are from zero to nine 80-character user header labels (UHL). These labels are user created and contain any data the user wishes to insert.

A tape mark follows the beginning label group, followed by file data which is terminated by a single tape mark.

End-of-file (EOF) labels are placed at the end of a logical file. These records are of the same format as the corresponding HDR records except that EOF replaces HDR. Further, block and record counts are present in the EOF1 record at positions 54 and 60, respectively.

An end-of-reel (EOR) label is placed at the end of a physical reel and is used during reel switching of multi-reel files. This record is in the same format as the EOF record except that EOR replaces EOF.

User trailer records (UTL) may follow either the EOF record(s) or EOR record(s) in a manner analogous to UHL records.

A tape mark follows the ending label group. An additional tape mark follows the last ending label group on a physical reel.



## Installation Labels

The MCP provides the capability of recognizing and creating labels defined by the user. This is particularly valuable when tapes from non-Burroughs systems are to be used or created on the system. The label definition is specified in an installation label card which is acceptable input to the CSTRT or WSTRT loaders. The card format is given in table 5-6.

**Table 5-6. Installation Label Card Format**

<b>Position (Byte)</b>	<b>Length (UA)</b>	<b>Contents</b>
1	8	LABEL1 – identifier card as installation label definition (LABEL1 may appear anywhere on the field)
9	1	BLANK
10	10	Text of beginning-of-file identifier (for example, HDR1)
20	2	Relative location of beginning-of-file identifier
22	2	Length of beginning-of-file identifier
24	2	Location of multifile-ID field
26	1	Length of MFID field (0 = no MFID field)
27	2	Location of file-ID field
29	1	Length of file-ID field
30	2	Location of reel number field
32	1	Length of reel number field
33	2	Location of creation date field (Julian format)
35	2	Location of purge data field
37	1	Length of purge date field (value of 5 or greater means purge date in Julian form; a value less than 5 implies retention period in days)
38	2	Location of physical reel ID field (physical tape number field)
40	1	Length of physical reel ID field
41	2	Location of EOR or EOF identifier
43	1	Length of EOR or EOF identifier
44	5	Text of EOR ID (left-justified)
49	5	Text of EOF ID (left-justified)
54	3	Length of label record
57	2	Number of label records (0 = variable, in which case MCP assumes maximum of 99)

## LABELS

**Table 5-6. Installation Label Card Format (Cont)**

<b>Position (Byte)</b>	<b>Length (UA)</b>	<b>Contents</b>
59	1	1 = Tape mark occurs after ending labels 0 = No tape mark
60	1	1 = Tape mark occurs after beginning labels; 0 = no tape mark
61	2	Location of block count field (ending label)
63	1	Length of block count field
64	2	Location of record count field (ending label)
66	2	Length of record count field

All location fields must contain values less than 80. A variable number of labels must not be specified unless tape marks are present after both beginning and ending labels. Creation dates must be present in Julian format. One tape mark is expected before the ending label.

All label information must be in the first physical label record. If multiple records are defined, the user is responsible for proper format on output files, and for extraction and verification of any data on input files.

### Unlabeled Files

A tape which does not contain a standard, USASI, or installation label is considered to be unlabeled. While such a tape might contain labels recognizable to another system, the MCP considers any tape with unrecognizable initial records to be unlabeled.

### Unreadable Labels

If the first records of a tape file cannot be physically read by a tape drive, the tape is said to have an unreadable label. ( This is not the same as unrecognizable.) This condition can be due to parity errors in the label records, density or track incompatibilities, or a long blank space on the tape.

### Scratch Tape Files

A physical tape reel is considered scratch (usable as an output file) if:

1. The label identifier contains blanks and contains a write ring.
2. The current date is equal to or beyond the purge date on the tape and the tape contains a write ring.
3. The tape is unlabeled and contains a write ring.

Case 1, above occurs when the system operator explicitly directs the MCP to purge a tape. Tapes which meet the other criteria are not explicitly purged when detected, but are merely noted as being usable for output if a program requests an output tape.

When a program requests an output tape file, the MCP assigns a tape in the following way. If the relevant FIB is already attached to a tape, that file is OPENed and the IOAT is searched for a scratch tape of the same label type declared in the source program. If one can be found it is assigned; otherwise, the first available scratch file is selected.

## PROGRAM LABEL DEFINITIONS

Four different label specifications are permitted on the system: standard Burroughs system labels, USASI standard labels, installation defined labels, and omitted labels.

The meaning of these specifications depends on the label type and the I/O mode (input or output). For example, a file of any label type is acceptable as input (regardless of the user program specification), but output files always follow the program label specifications.

Within a program, all defined files have at least a minimal label area to hold the file identifier. This area can be extended for certain file types or label declarations.

The relative location of label areas within a program differs with different compilers. For example, the COBOL compiler locates the label immediately after the FIB for the file, while the Assembler locates the label beyond the file buffers. However, the label can always be located from the FIB pointers FIBLAB and FIBLAE. The former addresses the byte prior to the multifile-ID field of the basic label area, and the latter references the byte immediately following the label.

## Unlabeled Files

Any file type can be declared unlabeled. The area allocated by the compilers for unlabeled files is a 19-byte extract of the standard Burroughs label and is used to hold the file identifier and reel number (where relevant). While unlabeled tapes do not have a reel number written on the tape, the MCP keeps track of the number of tapes accessed for various purposes. Further, the field is needed if a labeled tape is manually assigned to the requesting program.

An additional byte is allocated beyond this area to make this entire field modulo four in size. This 20-byte label area can be called the Basic Label Area (BLA) and is of the format shown in table 5-7.

**Table 5-7. Basic Label Area**

Position (Byte)	Length (UA)	Contents
0	1	Zero: position to which FIBLAB points
1	6	Multifile identifier: zeros if no MFID declared
7	1	Blank
8	1	Zero
9	6	File identifier
15	1	Blank
16	3	Reel number
19	1	Filler

The Basic Label Area is the nucleus for all other label types and is accessed by the MCP for file name identification during file OPEN and CLOSE operations and other miscellaneous purposes.

## LABELS

The description is slightly modified for disk files, in that the last 4 bytes are not allocated (total size: 16 bytes). Further, the multifile-ID field is not used for disk files since only a single identifier is permitted.

The label area for a disk pack file is in the standard label format, as described in the following text. However, the fields following the Basic Label Area are not used as they are not relevant for disk pack files.

### Standard Labels

The default label specification in source programs is the Standard Label. The memory area allocated is basically constructed by extending the Basic Label Area in both directions to give an area of the size and format of the Standard Label. Table 5-8 shows the format of a Standard Label.

**Table 5-8. Standard Label Format**

Position (Byte)	Length (UA)	Contents
0	8	bLABELbb
8	20	Basic Label Area
28	52+	Remainder of Standard Label

The address of the label is calculated by subtracting 16 digits from FIBLAB, the address of the Basic Label Area. The delimiting address is in FIBLAE giving a total area of 80 or more bytes (depending on any user specifications of a user label area). The label addresses in the FIB are used to construct an I/O descriptor for label processing.

### USASI Labels

Regarding the other label types, the USASI Label Area is built around the Basic Label Area. The entire field is shown in table 5-9.

**Table 5-9. USASI Label in Program Area**

Position (Byte)	Length (UA)	Contents
0	8	bLABELbb
8	20	Basic Label Area
28	80+	USASI Label formatting area

The first field is not allocated by all compilers (such as, ASMBLR).

The third field is the area into which the USASI label is read on input, or in which output labels are formatted.

If the field is 80 characters in size, only the HDR label can fit the space. More space can be allocated to permit User Header Labels (UHL) or User Trailer Labels (UTL). On input files, after bypassing the VOL label, the MCP reads successive 80-character label records into ascending locations of the program label area until the space is exhausted, a tape mark is sensed, or an unrecognizable record is read (does not contain HDR, UHL, or UTL in the first positions). In the latter case, successive records are read in the same location until a record of the expected type is found or a tape mark is sensed. (Label records can never be read into locations beyond the label area.)

On output, after creating the VOL label, the MCP formats an HDR label in the first 80 characters of the label area and then writes successive 80-character records until the space is exhausted, when a tape mark is written.

**CAUTION**

Sufficient space must be allocated for at least the HDR record or contiguous memory areas may be destroyed when the MCP formats this record.

It is the responsibility of the user program to create UHL and/or UTL records including the correct record identification characters (UHL or UTL) in the first positions.

An example of UHLs in a COBOL program follows:

```
FD FILENAME
    LABEL RECRDS USASI.
01 LABEL.
03 HDR-LABEL PICTURE X(80).
03 UHL-LABEL OCCURS n TIMES.
    05 UHL PICTURE X(3).
    05 UHL-CNT PICTURE 9.
    05 USER-DATA PICTURE X(76).
```

**NOTE**

Each record must be exactly 80 characters long; n is the number of UHL labels to be written.

DECLARATIVES.

UHL-LABELER SECTION. USE AFTER STANDARD BEGINNING LABEL PROCEDURE ON FILENAME.

UHLER. ADD 1 TO UHL-INDEX.

```
MOVE UHL TO UHL (UHL-INDEX).
MOVE UHL-INDEX TO UHL-CNT (UHL INDEX).
MOVE . . . TO USER-DATA (UHL-INDEX).
IF UHL-INDEX IS LESS THAN n GO TO UHLER.
```

## Installation Labels

The label area allocated for an Installation (non-standard) Label is quite similar to that allocated for USASI labels. The Installation Label Area follows the Basic Label Area.

The actual size of installation defined labels is specified to the MCP (through the Installation Label Card); consequently, the compilers are unaware of the actual size and allocate 80 bytes unless a different size is declared. For files with multiple label records, sufficient space must be allocated to hold all records.

## LABELS

For input files, successive label records are read into the Installation Label formatting area until the space or the declared label count is exhausted or a tape mark is sensed. (FIBLAE is used as the delimiting address for all reads; in no case can any contiguous memory area be destroyed.) If memory space is exhausted first, additional records are read into an MCP buffer until sufficient records are bypassed or a tape mark is sensed.

When all label records have been read and/or bypassed, the MCP gives control to the program label USE routine, if present.

For output files, the MCP formats the label record in the first portion of the Installation Label formatting area.

### CAUTION

Sufficient space must be allocated for at least one record, or contiguous memory locations may be destroyed when the MCP formats the label record.

If a label USE routine is provided, it is entered at this point. The user then creates any records beyond the initial label record and/or modifies the initial label record. As previously stated, all data specified in the Installation Label Parameter Card is formatted in the first label record by the MCP. If some of the data is to be in other label records, the appropriate information must be move in the USE routine. However, the data must not be cleared in the first record if the MCP is to recognize the tape label.

After the USE routine is exited, or if none exists, the MCP physically writes the first record. (The size is controlled by the value declared in the Installation Label Parameter Card.)

If multiple labels are defined and space exists in the label area, successive records are written until the label area or the defined count of label records is exhausted. (If a variable number of labels is declared, the MCP sets the label counter to 99, relying solely on the end label address, FIBLAE, and the declared label size to control the number of records to be written.)

## Special Cases

Certain tape files created on non-Burroughs systems are not handled automatically by the MCP and special programmatic attention is required. This includes tapes with multiple tape marks before the beginning label, and multifile tapes which are not recognized by the MCP (unrecognizable labels or no labels). This sub-section describes a general method for handling such files; an unlabeled multifile tape will be used as an illustration.

When an unlabeled file is assigned to a program (ULed) and processed by the job, the MCP does not know when EOF occurs. Whenever a tape mark is sensed, the MCP assumes an EOR condition, rewinds the current reel, and requests the next. The operator is responsible for notifying the MCP when no further reels are present (FR message).

The assumption of EOR at each tape mark signifies that straightforward handling of multifile unlabeled tapes is difficult. Such tapes, which have only tape marks between the files, can be successfully read to the first tape mark, but at this point, the MCP rewinds the tape. The key to handling such a tape lies in recognizing that the rewind occurs not when the tape mark is physically sensed, but rather when a logical READ is done on the buffer which yielded a tape mark result descriptor.

The programmatic method is to examine the result descriptor (FIBBSW) for the record about to be read. If the descriptor shows a tape mark (C40), the file is CLOSEd NO REWIND to access the next records. (For easiest handling, the file must have two or more buffers assigned.) Thus, a READ is never done on the tape mark buffer and the reel is not rewound.

## MIX TABLE

The MIX table (MIX) is used by the MCP to maintain information about active programs. The presence of a MIX entry indicates that a program has begun execution.

Among the information stored in the MIX table are program location in memory, program address, wait indicators (why a program cannot continue), various times, and program status. The first eight digits of the MIX entry determine the reinstatability of a program. If no bits are on in the first eight digits, the program can be reinstated. If any bit is on, the program is awaiting MCP or operator action.

The MIX table only exists in memory. No copy is stored on disk. The MCP is assigned a MIX number of zero.

## MIX TABLE LAYOUT

The following describes the fields in the MIX table. The information is used by the MCP in maintaining a job while executing in the system.

Label	Relative Location /Size	Content/Meaning
MIX-OV	0,1	Wait count for MCP overlay area
MIX-IO	1,1	Waiting I/O processing 1 = READ/WRITE IOC (also DCOM file IOC) 2 = File CLOSE queue flush 3 = Stoppage 5 = Positioning 6 = Data Comm FILL/ENABLE (WAIT status) 7 = Program overlay 8 = <<Available>> 9 = Terminate queue flush
MIX-WM	2,1	Waiting module processing 1 = Data Comm for BCT processing 2 = Core-to-Core send/receive 3 = <<Available>> 4 = STOQUE data entry 5 = STOQUE core 6 = STOQUE name slot 7 = STOQUE for processing 8 = Waiting trace 9 = DMS DBP waiting module function

MIX

Label	Relative Location /Size	Content/Meaning
		A = TSM for handler transfer B = Waiting DCPC
MIX-OK	3,1	Waiting Operator/MCP Action 1 = Duplicate file (non-disk) 2 = Duplicate library on disk (or pack) CLOSE 3 = No user disk or pack 4 = No file on disk or pack 5 = Locked file on disk or pack 6 = No file (non-disk) 7 = Output device required 8 = Extension module not in core (MIX-HW indicates module waiting for) MIX-HW = 1 DCOM = 2 MICR = 3 CRCR = 4 STOQ = 5 DCP = 6 Reserved = 7 Pack = 8 <<Available>> = 9 <<Available>> 9 = Waiting DMS function A = Local SPO ACCEPT B = Remote SPO ACCEPT C = Waiting OPEN hardware D = Waiting missing pack E = Waiting block count error action F = ZIP/SPOM processing
MIX-CR	4,1	:8 Waiting trace backup memory (may be set with other values) 0 = Not waiting (or trace) 1 = Disk or backup OPEN 2 = Direct I/O OPEN 3 = DCP OPEN



Label	Relative Location /Size	Content/Meaning
		4 = DCOM OPEN 5 = Share MIX PIB process OPEN
MIX-WA	5,1	:8 Push in process :4 Program stopped :2 Waiting IOC on MICR/OCR :1 Shared area waiting RUN :1 Time-sharing process waiting I/O queue flush
MIX-WB	6,1	:8 Program sleeping (DOZE or DCOM WAIT) :4 Trace print in process :2 Trace printer required :1 Trace disk required
MIX-WC	7,1	:8 Time-sharing process waiting terminal IOC :4 Program suspended :2 Time-sharing process waiting swapping :1 Waiting DIRECT I/O IOC
MIX-RP	8,1	:8 Process priority or time-sharing scheduling queue number
MIX-MP	9,1	Memory priority or PRIORITY within time-sharing scheduling queue
MIX-QP	10,2	Position in priority queue (time-sharing)
MIX-RN	12,1	Program Execution Flags :8 User Program flag (run light) :4 Snap Gate Enable flag :2 BCT Override flag (always off) :1 <<Reserved>>
MIX-BH	13,1	High order digit of BASE register
MIX-LH	14,1	High order digit of LIMIT register
MIX-PA	15,7	Program address register
MIX-BL	22,3	Low order digits of BASE register
MIX-LL	25,3	Low order digits of LIMIT register
MIX-TG	28,1	MODE/OVERFLOW/CONDITION toggle
MIX-EX	29,1	HALT execution ASCII toggle
MIX-NO	30,2	Job number

B 2000/B 3000/B 4000 MCPVI Programmer's Guide  
MCPVI Tables

MIX

Label	Relative Location /Size	Content/Meaning
MIX-BA	32,4	Base address
MIX-LM	36,4	Limit address
MIX-AC	40,20	Fixed arithmetics accumulator storage Code file parameter storage (terminate)
MIX-XX	60,6	IOAT address of working file Waiting core: core required Waiting DCP: DCP number
MIX-YY	61,5	Time to wake up from DOZE Time waiting IOC initiated
MIX-TP	66,1	:8 Tracing program (overlayable trace) :4 Trace heading required :2 TSM main MIX (MIX-PG = 8) :1 CLOSE in process
MIX-TP	66,1	Temporary terminate status for COMPILER and GO 1 = Code file PURGE 2 = Code file SAVE 4 = Code file BYPASS
MIX-PR	67,1	:8 Parameters Passed :4 <<Available>> :2 <<Available>> :1 <<Available>>
MIX-IH	68,2	Data Comm wait pointer (IO-STA of next unit)
MIX-HW	70,2	Hardware type required; terminate code
MIX-HI	72,1	MIX-HO value when HIHO is waiting disk/memory
MIX-HO	73,1	HIHO Status Digit 1 = Stop in process: KBD/ZIP step request 2 = Stop in process: priority crashout 3 = Stop in process: push initiated 4 = Stop in process: SORT: rollout 5 = Stop in process: breakout request 6 = Stop in process: memory dump to disk request

Label	Relative Location /Size	Content/Meaning
		7 = Sort specification file build in process 8 = Program is KBD/ZIP stopped 9 = Program is priority stopped A = Push roll-in phase in process B = Sort roll-in phase in process C = Reserved D = Reserved E = Waiting disk F = Waiting core
MIX-IC	74,3	Number of IOATs assigned to program
MIX-TL	77,5	Time Limit (remaining in seconds)
MIX-ID	82,12	Program identification
MIX-MF	94,12	Multi-program identification
MIX-PB	106,8	Disk address of program parameter block
MIX-RQ	114,2	DAT/MIX/PCR requestor code
MIX-RJ	116,2	RJE originator key
MIX-DD	118,1	:8 Initiated through ZIP communicate :4 EXECUTE with LOCK :2 Initiated from pseudo-card reader :1 Charge number supplied
MIX-LQ	119,1	:8 Memory dump request on abnormal termination :4 Initiated from a remote SPO :2 Time limit supplied :1 Label equate supplied
MIX-SR	120,1	:8 No pushdown/stop :4 Breaking out :2,1 SORT/Breakout program index: 0 = User program 1 = SORT.V (disk sort) 2 = SORT: (tape sort) 3 = <<Available>>
MIX-DC	121,1	:8 Simulate processor interrupt in program memory

B 2000/B 3000/B 4000 MCPVI Programmer's Guide  
MCPVI Tables

MIX

Label	Relative Location /Size	Content/Meaning
		:4 Midnight overlap for DOZE :2 Disallow operator Breakout requests :1 USE procedure in process
MIX-TR	122,1	:8 Terminate running :4 Call terminate at completion of DUMP/Breakout :2 Pass Breakout R/Ds to program if ARMED :1 Breakout not allowed (pack or direct I/O file)
MIX-ET	123,1	:8 USRTBL entry has been made :4 USERFL maintenance possible :2 User card entry made for this program :1 Reserved for SYAC
MIX-FL	124,1	:8 Previous ACCEPT and DISPLAY messages on disk :4 Code file is on disk pack :2 <<Available>> :1 Missing base pack (PSN unknown)
MIX-PG	125,1	Special Program Code 1 = Program is a generator 2 = Program is a DMPALL 3 = Program is LOADMP 4 = Program is PACKUP 5 = Program is DSKOUT 6 = Program has DCP MCS status 7 = Time-sharing process 8 = Time-sharing main MIX entry A = Generator in shared area B = DMS control program

B 2000/B 3000/B 4000 MCPVI Programmer's Guide  
MCPVI Tables

MIX

<b>Label</b>	<b>Relative Location /Size</b>	<b>Content/Meaning</b>
MIX-RT	126,4	Accumulated Run time this N-second
MIX-WT	130,4	Accumulated I/O wait time this N-second
MIX-DT	134,6	Accumulated Direct time for period
MIX-PT	140,6	Accumulated Prorated time for period
MIX-BT	146,6	Accumulated I/O wait time for period
MIX-AV	152,4	Average Run/Wait time; Time-slice for time-sharing process
MIX-RL	156,4	Run Log ID number

SASA

**SECURITY ATTRIBUTES STORAGE AREA**

The Security Attributes Storage Area (SASA) is a 25-byte field which is generated by the compiler whenever security attributes are declared for a file. A file's security attributes are stored in the SASA. These attributes may come from label equate information, program specified information, or the disk directory. The source of the information depends upon the type of open performed on the file and on file equate parameters supplied with the execution.

**SECURITY ATTRIBUTES STORAGE AREA LAYOUT**

The SASA is addressed using the pointer stored in FIBEXT. If no SASA is present, FIBEXT must be zero.

The SASA has the following layout.

<b>Label</b>	<b>Relative Location /Size</b>	<b>Content/Meaning</b>
SA-REV	0,2	Revision level of SASA (UA)
SA-GRD	2,12	Guard file ID
SA-FAM	14,12	Pack name for guard file (default = DISK)
SA-STY	26,1	Security type 4 = Guarded 2 = Public 1 = Private (default) F = None
SA-SUS	27,1	Security use 6 = IO (default) 4 = IN 2 = OUT 1 = SECURED
SA-SNS	28,1	Sensitive data flag 0 = Not sensitive (default) 1 = Overwrite data with random pattern
SA-MAI	29,1	Reserved
SA-UCO	30,20	Usercode of the creator

## COMPLEX WAIT TABLE

The MCP builds entries in the Complex Wait Table (CWT) when a program executes a Complex Wait BCT.

There are two types of entries in the CWT. For each program there is a program entry. Following each program entry is a series of event entries. The end of the table is indicated by a program entry with a run log number of zero.

The program entry has the following layout.

<b>Label</b>	<b>Relative Location /Size</b>	<b>Content/Meaning</b>
CWT-FG	0,1	Program header flag = F
CWT-RL	1,4	Run log number
CWT-GV	5,5	Giving address
	10,5	<< Available >>

The event entry has the following layout.

<b>Label</b>	<b>Relative Location/Size</b>	<b>Content/Meaning</b>
CWT-TP	0,4	Entry type 0000 = time 0001 = ODT input
CWT-FA	5,6	File address
	10,4	<< Reserved >>
CWT-NX	14,2	Index of item in list

## SECTION 6

### OBJECT PROGRAMS ON DISK

When a program is compiled, the generator creates an executable object program on disk. The format of that code file is fixed so the MCP can easily determine the necessary program requirements and load the program into memory. The code file has the following components:

- Program Parameter Block
- File Parameter Block(s)
- Object Program

The code file occupies one disk area, no greater than 9999 segments and subject to the restrictions noted in the following sub-sections. Compilers generate only one-area code files.

#### PROGRAM PARAMETER BLOCK

The Program Parameter Block (PPB) of the code file contains information needed to schedule and load the object program and always occurs at the beginning (zero-th segment) of the code file. The PPB contains two basic groups of data; general program information and the program Segment Dictionary. The former is used primarily to schedule and load the program into memory; the latter is primarily used during the load procedure to build the program Segment Dictionary in memory.

Aside from the Segment Dictionary, the fields of the PPB do not reside in memory.

#### PROGRAM PARAMETER BLOCK LAYOUT

The layout of the Program Parameter Block follows:

Label	Relative Location /Size	Content/Meaning
PB-PRN	0,12	Program name or time of compile
PB-SGS	12,3	Number of overlayable segments
PB-INS	15,5	Address of first executable instruction
PB-COR	20,6	Memory requirements
PB-SDA	26,6	Relative memory address of Segment Dictionary

The following information is the Program Segment Dictionary master entry.

Label	Relative Location /Size	Content/Meaning
PB-BCT	32,6	BCT instruction 300174
PB-DFD	38,2	Number of disk files declared
PB-FPF	40,1	File Parameter Block Flag
		0 = NO FPBs



B 2000/B 3000/B 4000 MCPVI Programmer's Guide  
Object Programs on Disk

Label	Relative Location /Size	Content/Meaning
		1 = 100 digits (usual size) 2,3 = 200 digits
PB-OPS	41,3	Number of logical segments
PB-WFL	44,6	Workflow Language flag (= "WFL")
PB-OVN	50,3	Segment number requested for overlay
PB-BSG	53,3	001
PB-CDT	56,6	Date program compiled
PB-CPL	62,2	Compiler of this Program A = ASSMBLR B = BPL C = COBOL D = DASDL F = FORTRAN R = RPG W = WFL X = XFORTN

The following are Program Segment Dictionary working entries.

Label	Relative Location /Size	Content/Meaning
PB-PRB	64,6	Address of PB-BCT or first instruction of segment
PB-RDA	70,6	Disk address of program segment relative to beginning of code file
PB-SLO	76,6	Address of first instruction to be executed after overlay call
PB-BEG	82,6	Beginning memory address of overlay
PB-END	88,6	Ending memory address of overlay
PB-LVL	94,2	After program load, EU and high order digit of absolute disk address of program

The working entries are repeated for each overlay in the program using the first 192 digits of additional disk segments, if necessary.

B 2000/B 3000/B 4000 MCPVI Programmer's Guide  
Object Programs on Disk

The following two fields are found only in the first disk segment of Program Parameter Block.

<b>Label</b>	<b>Relative Location /Size</b>	<b>Content/Meaning</b>
PB-FIL	192,2	Number of files declared in program
PB-MSZ	194,6	Memory size of global segment (digits)

The following data is used during program scheduling:

**PB-BCT**

Must contain 300174 (branch communicate instruction). Identifies file as a program.

**PB-COR**

Memory required for program.

## PROGRAM LOADING

The following fields are used during the loading of the first segment of the program into memory:

**PB-SGS**

To calculate Segment Dictionary size.

**PB-SDA**

Address where Segment Dictionary to be built.

The first 23 digits of the master Segment Dictionary entry and all working Segment Dictionary entries are placed into memory during program loading.

The following fields are used to skip past the FPBs and find the beginning of the non-overlayable segment of the program code.

**PB-FPF**

Flag defining size of individual file parameter blocks (FPB).

**PB-FIL**

Number of FPBs.

The following fields are used to load the program and begin its execution:

**PB-INS**

Relative memory address of first program instruction.

**PB-MSZ**

Size of non-overlayable segment is used to construct disk read descriptor to load program.

## MISCELLANEOUS

The following fields are used by the DC console message:

### PB-BCT

As above.

### PB-PRN

Program name as defined in source program, or the time of the compile.

### PB-COR, PB-DFD

Memory required.

### PB-CDT

Date compiled.

### PB-CPL

First character of compiler name.

The remainder of the PPB fields constitutes the Segment Dictionary which is used for program overlay requests.

Immediately following the PPB is a full disk segment which is not currently used.

## FILE PARAMETER BLOCK

If a program declares any files, the compiler builds File Parameter Blocks (FPB) following the empty segment after the PPB. These blocks are used only for FILE (label) equating. When a FILE equate control card is entered with a program request, the MCP checks the internal file name in the FPB against the name in the FILE equate card. If a match is found, all FPBs are copied into a separate disk area (if an area has not been obtained for a previous label equation request for the program), and the file equate information (name change, device change) is written into the appropriate record in the copy.

At file OPEN time, the MCP accesses the appropriate record in this label equate block (using FIBFNM as an index) and, if FILE equation was done for the file being OPENed, the necessary modifications are made to the FIB and/or label areas.

FPBs are 100 or 200 digits in size depending on the compiler. PB-FPF in the PPB defines the size. If a program code file contains an odd number of 100-digit FPBs, the last 100 digits at the end of the FPB area are unused.

### FPB on Disk

The format of the FPB on disk is as follows.

Label	Relative Location /Size	Content/Meaning
FP-FNM	0,12	Internal file name
FP-MFD	12,12	Multifile-ID

B 2000/B 3000/B 4000 MCPVI Programmer's Guide  
Object Programs on Disk

Label	Relative Location /Size	Content/Meaning
FP-FID	24,12	File-ID
FP-HWR	36,2	Hardware type
FP-BUP	38,1	Backup flag
FP-LEQ	39,1	Label Equate flag
FP-SPF	40,1	Special Forms flag
FP-TRK	41,1	Magnetic tape track type
FP-GRD	42,12	Guard file ID
FP-STY	54,1	Security type (refer to SASA)
FP-FIB	55,6	Base-relative FIB address
FP-SEG	61,3	Number of logical segments containing this FIB
FP-SUS	64,1	Security use (refer to SASA)
	65,12	Reserved
FP-SNS	77,1	Sensitive data flag
FP-FAM	78,12	Pack ID of guard file (default = DISK)
	90,10	Reserved

## PROGRAM CODE

Following the FPBs is the object program's non-overlayable segment and then the program overlays. All program segments begin on disk segment boundaries; thus a small amount of unused space can exist between program segments.

The program code file relative addresses of the overlayable segments can be found in the individual segment dictionary entries (PB-RDA). (All addresses are zero relative.)

## PROGRAM SEGMENT DICTIONARY

The Program Segment Dictionary is created from information in the Program Parameter Block. This dictionary is used for all program overlay requests.

A Segment Dictionary is always at least 64 digits long. In other words it always has at least two 32-digit entries. The first 32 digits is the base, or header entry. The second and subsequent 32-digit groups correspond in sequence, to each program segment which exists.

### Segment Dictionary Header Entry

The format of the Segment Dictionary header entry is as follows.

<b>Label</b>	<b>Relative Location /Size</b>	<b>Content/Meaning</b>
PSD-BC	0,6	300174, overlay BCT
PSD-DF	6,2	Number of disk files declared
PSD-FP	8,1	File Parameter Block flag
PSD-#S	9,3	Number of logical segments in program
PSD-OC	12,6	Overlay call counter
PSD-SG	18,3	Requested logical segment number
PSD-BS	21,3	Base logical segment number
	24,2	Reserved
PSD-SZ	26,6	Segment Dictionary size in digits

The subsequent Segment Dictionary entries have the following format.

<b>Label</b>	<b>Relative Location /Size</b>	<b>Content/Meaning</b>
PSD-OV	0,6	Address of overlay BCT or first instruction
PSD-DA	6,6	Low order disk address of logical segment
PSD-FI	12,6	Memory address of first instruction to execute
PSD-BE	18,6	Beginning memory address of logical segment
PSD-EN	24,6	Ending memory address of segment
PSD-EU	30,2	Disk EU number of logical segment

## **PROGRAM OVERLAY MECHANISM**

When a program needs a logical segment (in COBOL, GO TO or PERFORM to an overlayable section, in BPL, a procedure reference), the compiler generated code performs the following actions.

The appropriate logical segment number is moved to PSD-SG and an indirect branch (BUN OP = 27) is executed to PSD-OV for the segment requested. Two results are possible.

If the segment is already in memory from a prior call, then PSD-OV for the segment contains the address of the first instruction in the segment. The indirect branch, in effect, will be a branch to the first instruction.

If the segment is not in memory, then PSD-OV contains the address of the overlay BCT, PSD-BC. Then, the indirect branch will be to the overlay BCT instead of the first instruction. THE BCT 0174 is executed and the MCP reads the requested program segment from disk. After loading PSD-OV with PSD-FI, the program is reinstated at the PSD-FI address.

## SECTION 7

### HOW TO READ A DUMP

A program dump is usually taken to find a program problem. This problem can be either a program fault which causes the program to abort or a logic error which causes the program to loop or to perform unexpected logic sequences.

A dump is a snapshot of program memory and shows the contents of memory at a particular point in time. Retracing a program's execution can be done with a dump if the correct program listing is used with it. It is also possible to predict a program's sequence if the necessary information is in memory when a dump is taken.

Program debugging is a subjective topic. Like programming, it is never done alike. However, information presented in this section is useful for reading any program dump. All the information will not be applicable for a particular program failure, but having more facts will enable a more complete picture of the program status at the time of the dump.

#### OBTAINING A PROGRAM DUMP

A program memory dump can be obtained in several ways. If a program experiences a hardware fault or attempts an illegal action upon a data file, the MCP will terminate the program. If the program had been executed with the MEMDUMP control statement, a program fault will cause the program to abort with a memory dump. If neither MEMDUMP was specified nor the system option TERM was set, the program will be subject to a DS or DP option. A memory dump can be obtained with a DP. The TERM option causes an automatic DS on program faults.

If a program does not abort, a dump can be obtained by a DM or DP keyboard input message. Dumps can also be produced programmatically. In COBOL, the construct is TRACE 20 or TRACE 22.

Memory dumps are automatically directed to backup disk, and can be identified by IDs of the form \$pnxxx. The PM keyboard command is used to initiate the DMPOUT intrinsic to print the dump.

#### READING THE DUMP

The following text refers to information in the sample program dump in appendix A.

The printout produced by DMPOUT can be divided into two parts. The first part of the dump shows information which a program does not have access to: IOATs, DFHs, and MIX table entry. The contents of these entries are described in section 5. The second part is the contents of program memory from BASE to LIMIT.

The first pages of the dump give overall information about the program (MIX number, compile data, run date and time, MCP release level, program status, Segment Dictionary, and so on are shown). Following this information, each FIB still open at the time of the dump is printed. IOAT, FIB, and DFH (if applicable) are included.

The Dump Control Segments give the uninterpreted values of program JRT and MIX entries.

The memory contents can be used to determine values at particular memory locations and also to verify the instruction address.

B 2000/B 3000/B 4000 MCPVI Programmer's Guide  
How to Read a Dump

If the dump was produced from a program abort, the MCP gives the reason for and the address of the fault in a message displayed upon the OCS. The message has the following format:

— < program-id > < error condition > < address > < segment # >

The value in <address> is the base relative address of the instruction. This value appears in the dump under Run Control Word and is labeled PAR.

If the dump is taken programmatically or through the OCS while the program is running, no error message is displayed, and the PAR value is the address of the next instruction to be executed.

The processor result descriptor at address 80-83 on the program's memory should be examined to determine what type of error occurred. If the result descriptor is zero, the error was MCP-generated (for instance, invalid OPENS and I/O errors). A non-zero value indicates a processor interrupt such as an address error, instruction timeout, or undigit arithmetic.

Also, in the memory portion of the dump, the instruction at the address given by PAR should be checked for validity. If the error was MCP generated, the PAR value points at the instruction following the failing instruction. This is also true if the error was a processor interrupt. For the processor error invalid instruction, PAR points at the failing instruction.

If the failure is an address error, the instruction in question must be decoded. Address errors include attempting to access outside of BASE and LIMIT, odd address for a UA field, non-mod 4 address for a word field, and branching to an odd address. The address controllers in the instruction indicate whether or not any indexing or indirect addressing is needed. The index registers (IX1, IX2, and IX3) under the heading Reserved Memory, are also found in the memory portion at base relative locations 8, 16, and 24, respectively.

If the program failure is an MCP detected error (for example, file processing errors), the PAR will point at a branch instruction following a Branch Communicate Instruction BCT (OP = 30). This type of failure indicates that a parameter passed to the MCP is incorrect or that one of the other program MCP interface areas is in error. (Refer to the Program Interface, section 2 for BCT formats.)

For a file processing error such as invalid OPEN/CLOSE or invalid READ/WRITE, a FIB is involved. Associated with the FIB are a file header and an IOAT.

Information in the IOAT is not useful in most cases, in dump reading. Fields of interest include IO-ST1, IO-ID, IO-PK1, IO-PK2, and IO-RDA. In the disk file header, the information pertaining to the file descriptions for record length and blocking factor are useful (DF-RSZ and DF-RPB).

Several of the FIB fields are useful during program debugging. Some of them are:

Field	Meaning
FIBSTA	Describes whether file is OPEN; and if CLOSED, whether previously OPENed.
FIBARB	Depending on buffer technique, current or next record to process.



B 2000/B 3000/B 4000 MCPVI Programmer's Guide  
How to Read a Dump

Field	Meaning
FIB-WA	Work area location for current record.
FIB-IO	If the file is processed in multiple mode, mode that was most recently used.
FIBRCT	Number of records if any processed before the failure.
FIBBCT	Number of blocks processed.

Aside from the value in determining the general program state, the FIB can be useful in debugging invalid I/O operations (invalid I/O descriptor). This occurs when a program inadvertently destroys the Buffer Status Blocks.

The invalid I/O descriptor can be found by examining the result descriptor (FIBBSW) to which FIB-NB points. For invalid I/O operations, FIBBSW contains a special R/D of the form 9XX2 where the second and third digits specify the specific error in the I/O descriptor, refer to File Buffer Descriptors in section 5.

Invalid programmatic actions detected by the MCP include invalid file OPENs, CLOSEs, READs, WRITEs, and so on. Many failures of this type occur due to inadvertent programmatic destructions of a FIB (for instance, a runaway subscript). Consequently, check the FIB first for such a condition. If the FIB has not been destroyed, several individual fields can be checked. The most valuable FIB field for this type of debugging is FIBSTA, which describes the file status. Most invalid file CLOSEs, READs, and WRITEs occur because the file is not OPEN, while invalid OPENs occur because the file is not CLOSED.

On occasion other fields can be useful. For invalid READ and WRITE these include:

Field	Meaning
FIBMRL	Maximum record size; see FIBARB below.
FIBARB	For READ of variable length records, pointer to current record which may be an invalid size.
FIB-WA	For variable length WRITE analogous to FIBARB above.
FIB-IO	Mode declared at OPEN may be incompatible with I/O request. This often occurs with a COBOL sort when a coding error inadvertently causes the program to fall out of an INPUT PROCEDURE directly into an OUTPUT PROCEDURE.
FIBBLK	2 = variable length; occasionally files are declared variable length inadvertently.
FIBRSW	Work area access records cannot exceed 9999 words.

An invalid OPEN may be observed due to invalid record size, oversize or incompatible disk file declarations, incorrect data communications file declarations, an attempt to use output installations labels without a system installation label definition, or a number of other errors in the FIB. Often the MCP output message on the OCS describes the error.

## STACK OPERATION

A stack is a programmatic method of data storage and retrieval. On B 4000/B 3000/B 2000 series systems, this facility is effected by use of the NTR instruction (OP - 31). The stack is simply a memory area set aside to be used as a stack. The memory area is fixed (once established, cannot change). Depending on the stack location, the upper limit can be either a physical limit (program LIMIT register value) or a logical limit (some predetermined address). In the latter case, there is no absolute protection against stack overflowing (exceeding maximum size).

Figures 7-1, 7-2, and 7-3 illustrate the physical concept of stacks. Figure 7-1 shows an empty stack. Two pointer values (TOP-OF-STACK and CURRENT-ENTRY) are used in stack manipulation. Since the stack is empty, the value of CURRENT-ENTRY is unknown or irrelevant. Figure 7-2 shows the stack after an entry has been placed into it. TOP-OF-STACK now points to the location following the stack entry.

In figure 7-3, another entry has been placed into the stack. CURRENT-ENTRY now points to this entry and TOP-OF-STACK is updated to point at the location where a new stack entry would appear.

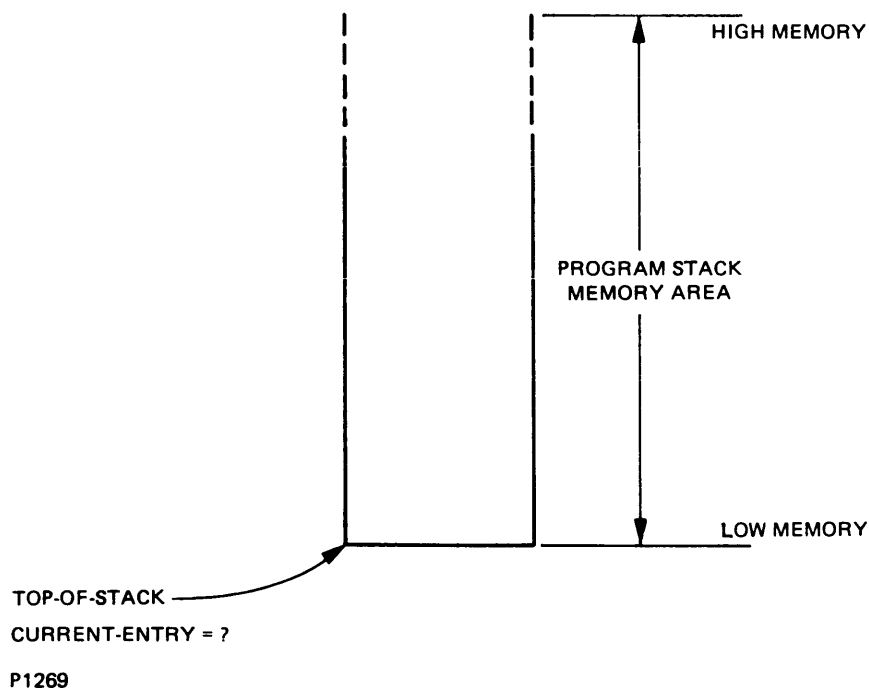
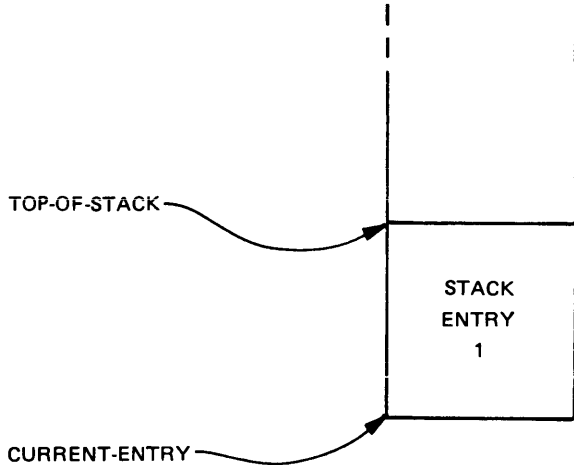
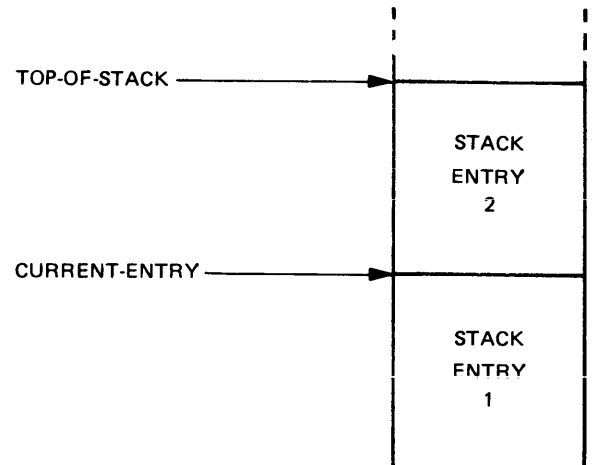


Figure 7-1. A Stack Containing No Entries



P1270

Figure 7-2. A Stack Containing One Entry

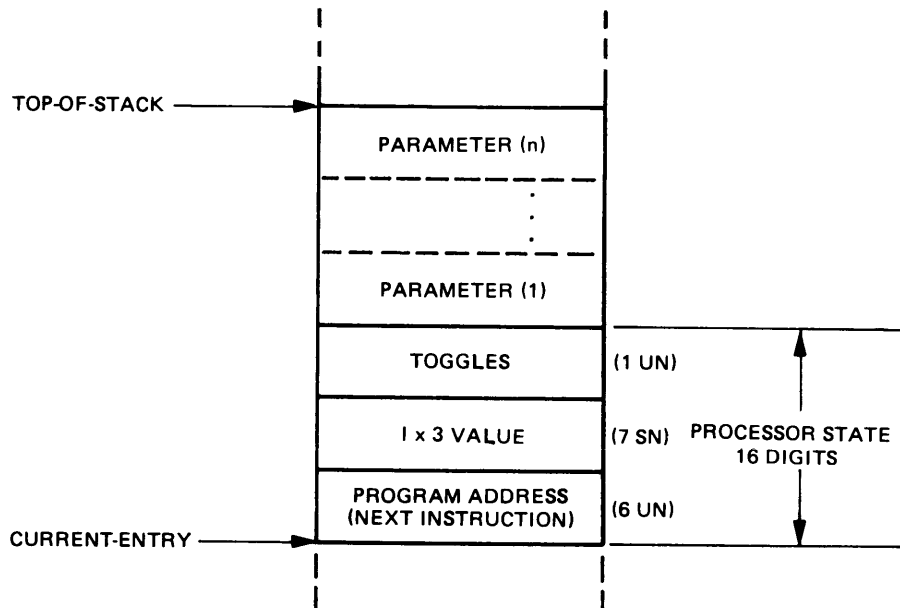


P1271

Figure 7-3. A Stack Containing Two Entries

If stack entries continue to be created, the stack soon reaches its limit. To prevent this, old or unneeded stack entries can be discarded, but must be removed from TOP-OF-STACK. With a stack of plates, only the top plate is readily accessible; the bottom plates cannot be reached unless all upper ones have been removed. This action of popping or cutting the stack is accomplished with the EXT (OP = 32) instruction. Cutting the stack is illustrated by reviewing figures 7-1, 7-2, and 7-3 in reverse sequence.

Programmatically, a stack entry is created when an NTR instruction is executed. Since an NTR is a branch-type instruction, but has an eventual return, the current processor state (program address, comparison toggles, and other information) must be saved. The processor state information constitutes a basic stack entry. If parameters are passed with the NTR, these parameters also go into the stack entry. A typical stack entry, appears as shown in figure 7-4.



P1272

Figure 7-4. Typical Stack Entry

B 2000/B 3000/B 4000 MCPVI Programmer's Guide  
How to Read a Dump

The processor state information requires 16 digits of memory. Program Address is the 6-digit base relative address of the instruction following the NTR. The value of IX3 at the time of the NTR is also saved. The comparison indicator, the overflow indicator, and the ASCII indicator are stored in the Toggles field. Toggles is defined as follows:

Toggles :8 (ASCII indicator)            1 = ASCII mode, 0 = EBCDIC  
          :4 (overflow indicator)        1 = Overflow  
          :21 (Comparison indicator) 1 = High, 2 = Low, 3 = Equal

This layout shows that to access the first parameter in a stack entry, the CURRENT-ENTRY value must be incremented by 16.

The pointers, called CURRENT-ENTRY and TOP-OF-STACK for illustrative purposes, are defined to be fixed locations in a program. The CURRENT-ENTRY pointer is located at BASE:+24:7:SN (IX3 location); TOP-OF-STACK is at BASE:+40:6:UN.

Since IX3 is saved in a stack entry, successive stack entries are linked together by the IX3 value in each stack entry. This is illustrated in the following example.

Assume the following instruction sequence:

```
L1: NTR  A                               IX3 : C0004892
      CNST 2 UA = XY                       BASE:+40 : 008000
L11: DISPLAY . . .
      .
      .
      .
A: NTR  B
      CNST 6 UN = 012345
A1: DISPLAY
      .
      .
      .
EX1: EXT
      .
      .
      .
B: WRITE
      MVN 3BASE:+15:IX3:6   PRINIT:UA:6
      WRITE PRINIT
EX2: EXT
```

Upon execution of the NTR A at label L1, the stack would appear as shown in figure 7-5.

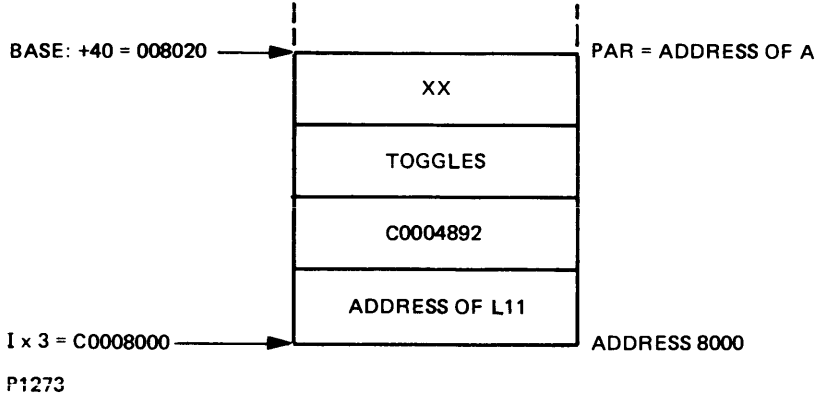


Figure 7-5. Stack After NTR A

Figure 7-6 shows the stack after NTR B at label A is executed.

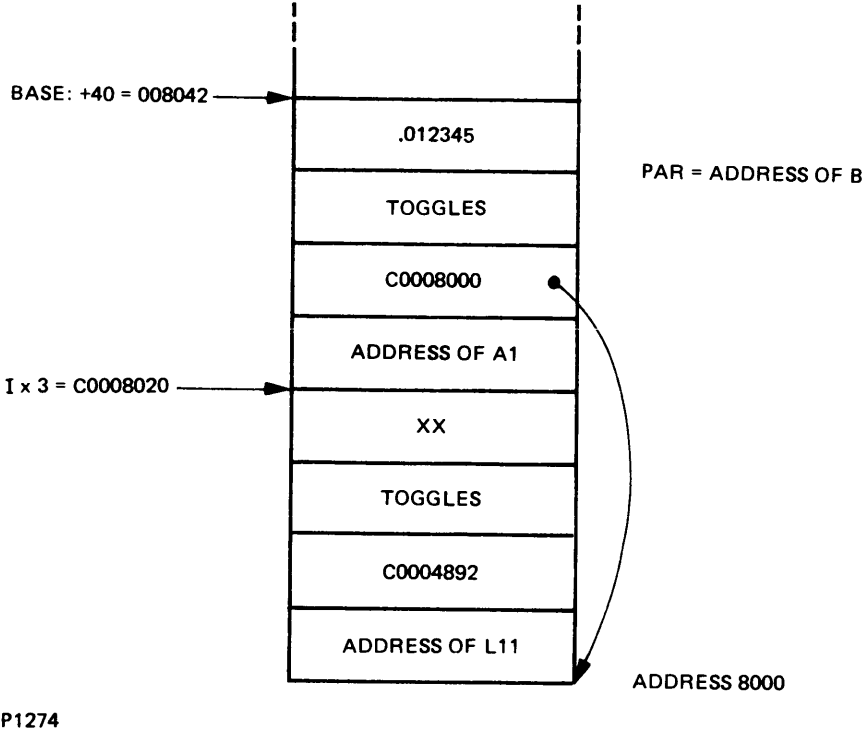
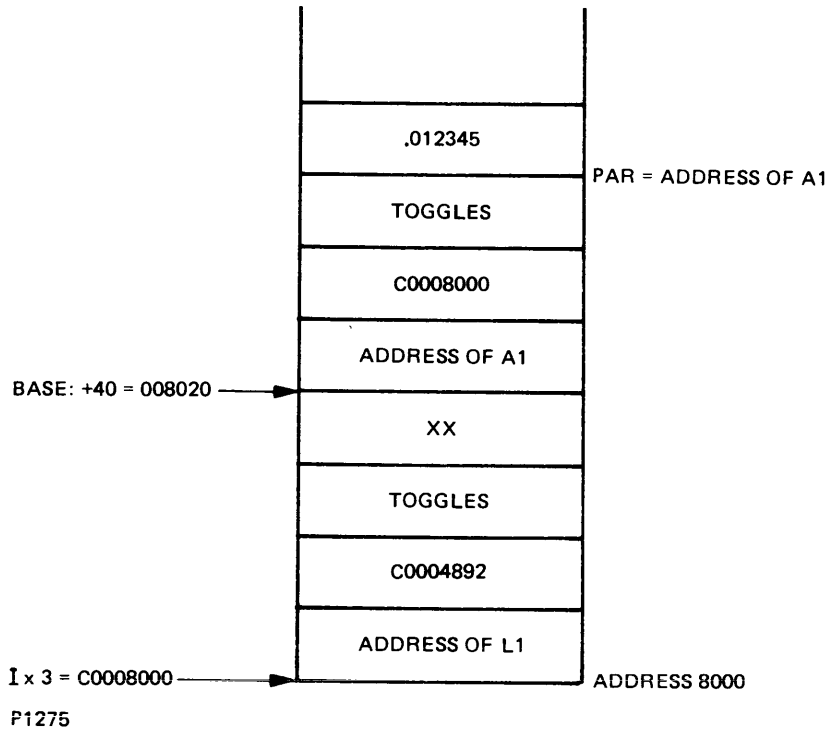


Figure 7-6. Stack After NTR B

After NTRing routine B and going through the EXT at EX2, the stack appears as shown in figure 7-7. Note that the information in the stack is not cleared; rather, the pointers are moved.

B 2000/B 3000/B 4000 MCPVI Programmer's Guide  
How to Read a Dump



**Figure 7-7. Stack After EXT, Routine B**

Since IX3 points at the current stack entry, the value cannot be modified if a proper EXT is to be performed. An EXT restores the processor state from the current stack entry. If IX3 does not point at the current stack entry, an EXT will give unpredictable results. Therefore, if IX3 is to be modified in an NTRed routine, the prior value must be SAVED and then restored before EXT is executed.

## MEMORY DUMP FILE STRUCTURE

The following information pertains to the format of the memory dump backup file created by MCPVI. The information is current as of the ASR 6.3 release.

### FILE NAMING

Memory dump files are identified by IDs of the form \$pnnnn where p = processor number and nnnn = sequential number starting from two. \$p0001 is a special ID reserved for MCP usage.

### FILE LAYOUT

A user dump file is a single area file containing unblocked 100-byte records. The number of records per area is calculated according to the following formula:

$$\begin{aligned} & (20 + \text{number of non-disk IOATs assigned to program}) \\ & + 5 * (\text{number of disk/pack IOATs declared} + \text{total size of device alternate blocks in KD}) \\ & + 5 * \text{memory assigned to program in KD} \\ & + 8 \end{aligned}$$

Each non-disk IOAT requires only 100-digits, so 100-digits of filler are added to make one record contain one non-disk IOAT. Space is reserved for an additional 20 non-disk IOATs. Three records are reserved for the dump control record, the JRT entry, and the MIX entry. Five records are set aside for a possible Time Sharing Process Stack.

The records indicated above are found in the dump file in the following sequence:

1. Control record. Two hundred digits containing miscellaneous information about the program and the dump file. (See definition below.)
2. JRT. Job Reference Table entry of the program (200-digits).
3. MIX. MIX table entry of the program. The size will be 200-digits for TSM jobs, 160-digits plus 40-digits of filler for others.
4. Memory information. Memory image of object program (in whole KD).
5. TSM information. TSM Process Stack or filler if not applicable (1KD).

#### NOTE

Consult the current MCP listing for descriptions of MIX and JRT records.

One or more of the following can be present depending on program requirements:

Large device alternate block (4KD).

IOAT (100-digits)  
File Header (124-digits)  
Buffer ( 2800 thru 3600-digits)

Small device alternate block (2KD).

IOAT (100-digits)  
File Header (124-digits)  
Buffer (1440 thru 1800-digits)

B 2000/B 3000/B 4000 MCPVI Programmer's Guide  
How to Read a Dump

Disk/Disk pack file entry.

IOAT (100-digits)

File Header (100-digits)

Address Blocks (from zero to four depending of the number of areas declared; each address block is 200-digits)

## CONTROL RECORD FORMAT

The first record of the dump file is a control record which is used during dump file creation. The information contained in the record pertains to the dump file and to the executing program. The format of the control record is:

Relative Location	Size	Function
0	6 UA	Program ID
12	6 UA	Multi-program ID
24	2 UN	Mix number
26	6 UA	ID of this file
38	3 UN	Snapshot dump base
41	3 UN	Snapshot dump limit
44	6 UN	File creation date
50	10 UN	File creation time
60	5 UN	File size in disk segments
65	4 UN	Program size in KD (base - limit)
69	2 UN	Number of 4KD device alternates
71	2 UN	Number of 2KD device alternates
73	2 UN	Number of disk/pack IOATs
75	2 UN	Number of disk/pack file headers
77	3 UN	Number of address blocks
80	2 UN	Number of hard IOATs
82	5 UN	Segment offset to 1st IOAT
87	4 UN	Program base address
91	7 UN	Total memory required
98	1 UN	CPU type
99	1 UN	Memory speed
100	2 UN	Filler
102	4 UA	MCP release number
110	6 UN	MCP release date
116	6 UN	Program compile date
122	6 UN	Segment dictionary memory address
128	6 UN	Overlay BCT address
134	1 UN	Label equate flag
135	1 UN	0 = Permanent file EXECUTE 1 = GO of CMP/GO 2 = Bound intrinsic execution
136	2 UN	Number of files declared
138	4 UN	Number of disk segments in program
142	58 UN	Filler



## SECTION 8

### RELATIVE AND INDEXED I/O

The relative and indexed input-output features provide random and sequential access to the records in a mass storage (DISK or DISKPACK) file, and prevent access to unused or deleted record positions.

A set of access routines is automatically bound into a program which declares any indexed or relative files.

Indexed and relative file formats differ from those of other files; once created with indexed or relative organization, a file must always be declared to have that organization.

#### RELATIVE I/O OVERVIEW

A relative file consists of records which are identified by relative record numbers.

The file can be thought of as composed of a serial string of areas, each capable of holding a logical record. Each of these record areas is identified by an integer value (relative record number) greater than zero which specifies the record area logical position in the file. Records are stored and retrieved based on this number.

For example, the tenth record is the one addressed by relative record number 10 and is in the tenth record area, whether or not records have been written in the first through ninth record areas. (Conceptually, RELATIVE organization is similar to RANDOM, but provides protection against the accessing of unused or deleted record positions.)

In the SEQUENTIAL access mode, the sequence in which records are accessed is the ascending order of the relative record numbers of all records which currently exist within the file.

In the RANDOM access mode, the sequence in which records are accessed is controlled by the programmer. The desired record is accessed by placing its relative record number in a RELATIVE KEY data item.

In the DYNAMIC access mode, the programmer can change at will from sequential access to random access using appropriate forms of input-output statements.

#### INDEXED I/O OVERVIEW

An indexed file consists of records, each uniquely identified by the value of one or more keys within the record. The records may be accessed in either a random manner, by the value of the key, or in a sequential manner.

A record description may include one or more alphanumeric key data items, each associated with an index. Each index provides a logical path to the data records according to the contents of a data item within each record which is the record key for that index.

The data item named in the RECORD KEY clause of the FILE-CONTROL entry for a file is the prime record key for that file. For the purposes of inserting, updating, and deleting records in a file, each record is identified solely by the value of its prime record key. This value must, therefore, be unique, and must not be changed when updating the record.

A data item named in the ALTERNATE RECORD KEY clause of the FILE-CONTROL entry for a file is an alternate record key for that file. The value of an alternate record key can be non-unique if the DUPLICATES phrase is specified. These keys provide alternate access paths for retrieval of records from the file.

In the **SEQUENTIAL** access mode, the sequence in which records are accessed is the ascending order of the **RECORD KEY** values. The order of retrieval of records within a set having duplicate record key values is the order in which the records were written into the set.

In the **RANDOM** access mode, the sequence in which records are accessed is controlled by the programmer. The desired record is accessed by placing the value of its record key in a **RECORD KEY** data item.

In the **DYNAMIC** access mode, the programmer can change at will from sequential access to random access using appropriate forms of input-output statements.

## **RELATIVE FILE IMPLEMENTATION CONSIDERATIONS**

Relative files differ from conventional files in the B 4000/B 3000/B 2000 systems in that blocks contain additional information not present in the blocks of conventional files. In order to interpret this additional information, a set of compiler-provided intrinsics is bound into a program which declares a relative file. A user-program operation on a relative file generates a call on the appropriate intrinsic routine; the intrinsic accesses the file.

There are certain considerations for relative files:

1. A relative file can only be accessed meaningfully in a COBOL program in which the file has been declared with **ORGANIZATION RELATIVE** (unless the programmer is prepared to code routines capable of interpreting the relative file blocks).
2. The presence of the access routines will increase the program size by approximately 5000-bytes. All access routines are bound into a program which accesses a relative file in any way and are not overlayable.
3. The data block will be larger than the simple product of the record size and the blocking factor, since a variable amount of additional information is present at the end of the block.
4. The last block of the file is a control block, which contains additional information about the file other than that stored in the disk and pack file headers. This block is created when the file is created, accessed when the file is accessed, and updated when the file is updated. The presence of the control block leads to the following:
  - a. When a relative file is created, the access routines immediately reserve both the first and the last areas of the file. The control block is written into the last area.
  - b. The size of the control block is 64-digits (32-bytes). This imposes a minimum block size of 32-bytes for the relative file. Data blocks must be at least that size to avoid wasting space.
  - c. The number of areas and records-per-area specified in a program that creates a relative file must also be specified in any program attempting to access the file. The access routines read the data file control block when the file is **OPENed**, using information in the **FD** to determine the location of the control block. If the file specifications differ from those given when the file was created, the access routines may not locate the control block, and program errors can occur.

For example, if a relative file is created with 50 areas and 100 records per area, the data file control block will be the block containing record number 5000. A program attempting to access the file must declare those same attributes for the file. Specifying 60 areas of 100 records would cause the access routine to attempt to read a block with record number 6000, leading to an end-of-file condition. In this case, the program would be terminated with a **STATUS KEY** value of "FA".

5. A Relative File Data Block is present in the compiled program for each relative file declared in the program. This block contains certain information needed by the access routines to maintain the file.
6. One buffer is allocated for each relative file. This buffer is controlled by the access routines via a Buffer Control Structure embedded in the Relative File Data Block.
7. Since the block format differs from that of conventional files by the addition of control information in each block, utility programs (such as DMPALL) used to list the file must be directed to list entire blocks.

File format, the File Information Block (FIB), the Relative File Data Block, and access routine interfaces are discussed in detail later in this section.

## RELATIVE FILE FORMATS

Each Relative File declared in a COBOL-74 program results in a single physical file of data and control information. However, the blocks of this file differ from conventional file blocks by having control information appended to them. The file is built and updated by the access routines. The MCP is told that the file is an unblocked random file. The new Relative File type is understood by the access routines with its two new block formats.

The Relative File type consists of one or more blocks of data records and one block of control information. The two block formats present in a Relative File are the data block and the control block.

Each data block contains valid and empty data records. The number of data records that a block can hold is specified by the **BLOCK CONTAINS** clause in the user program file description, unless overridden by the space requirements of the control block. Record slots are the same size as the user work area.

Following the last data record slot in the block is control information used by the access routines to store and deblock the data records. This block control information contains a reserved area and presence flags.

The reserved area (14 UN) is not used, but is present for compatibility with the Indexed File Data Block format. This field should contain zeros.

Each record slot (1 UN) in the block has a corresponding presence flag to indicate whether the slot contains a valid data record. A value of zero indicates the record is not used or deleted; one means the record is present.

Each Relative File has control information (control block) that must be saved when the file is not in use. The last block in the file has been reserved for this purpose. Users should note that this reduces the maximum number of data records that can be stored in the file and must add an extra block of records to the file size calculation, if this is significant. At file creation time the control block number is computed from the number of areas times the blocks per area contained in the FIB. Subsequent use of the file always retrieves and returns the control information to and from this block.

The size of the control block is 24-digits plus 40-digits for the data file description entry. This imposes a minimum block size on the data blocks in the file and can result in unused space if the record size times blocking factor is less than the control block size. The COBOL-74 compiler will generate a warning describing this condition and pad out the data block size. Note that the control block does not contain block control information, since its format is always known.

The control block contains two fields. the first field (24 UN) is not used, but is present for compatibility with the Indexed File Control Block format. This field must contain zeros.

The second field (40 UN) contains control information for the relative file from the Relative File Data Block in the user program.

## **RELATIVE FILE FIB**

There is one FIB created for the Relative File. This FIB is generated by using the information provided by the user in the File Control and FD sections of the COBOL 74 program. This FIB is created in the same manner as a FIB for a normal sequential file, with a few additions. These additions are as follows:

1. There is always a user work area assigned to this FIB. Its address is set in FIB-WA. This is the only interface the user has to the Relative File. The size of this user work area is the size specified in the user record description of the Relative File, padded to MOD 4 length.
2. Block size must be computed by using the user work area size times the blocking factor declared in the BLOCK CONTAINS clause. To this total, the COBOL-74 compiler adds 14-digits for control information along with 1-digit per record in the block. See Data Block Format in this section. The block size is then padded to MOD 4 length.
3. The Relative File FIB must reflect the record size equal to the block size as computed in step 2 above. This value is set in FIBMBS.
4. There is only one buffer attached to this FIB. The size of this buffer will be equal to the block size computed in step 2 above. FIB-AA and FIB-BB must be set to the beginning and ending addresses of the buffer.
5. The FIB must always specify buffer access mode, FIB-BA:1 set. This prevents MCP access to the user work area. The access routines will handle all movement of data to and from the user work area.
6. The FIB has an access mode of random specified in FIB-DA:1, regardless of the mode specified in the program. This prevents MCP OPEN buffer fills. The actual access mode declared by the user will be contained in the Relative File Data Block.
7. The FIB has the address of the ACTUAL KEY field in the Relative File Data Block set in FIBKEY. See Relative File Access Routines in this section.
8. The FIB also has FIBIX2:1 reset, and FIBRAD:8 set to force physical I/O, since the access routines determine when I/Os are to be done.
9. The FIB has FIBORG:8 set to indicate that this is a Relative File.

## **RELATIVE FILE DATA BLOCK (RF BLOCK)**

Each Relative File has one Relative File Data Block in the user program. This block contains all the information not in the FIB which the access routines require to handle a Relative File. This block is the focal point throughout the user program when trying to access a Relative File. The address of the RF Block will be the address passed on each function call to an access routine. The RF Block contains a number of substructures which are used by the access routines. Table 8-1 is a description of the contents of the Relative File Data Block (RF Block). This block is created by the COBOL-74 compiler in the exact order presented here.

B 2000/B 3000/B 4000 MCPVI Programmer's Guide  
Relative and Indexed I/O

Fields that are preceded by an \* are filled in by the compiler.

**Table 8-1. RF Block Format**

	Usage	Size	Function
*	File Status Pointer	8 UN	Pointer to file status word = all E means non-present
*	Open Type	1 UN	Relative File open state 0 = input 1 = output 2 = input-output 9 = not open
*	Access Mode	1 UN	Declared access mode 0 = sequential 1 = random 2 = dynamic
	Last Function	1 UN	Last function on Relative File 0 = unsuccessful 1 = open 2 = start 3 = read 4 = write 5 = rewrite 6 = delete 7 = close
	Current Record Area	8 UN 8 UN 4 SN 1 UN	Relative record number Data block number Data slot number State of current record 0 = undefined state 1 = beginning of file 2 = defined state
	Actual Key	8 UN	Read/Write key for Relative file
*	Buffer Control Structure	8 UN 1 UN 1 UN 6 UN	Block presently in buffer Buffer changed flag 0 = not changed 1 = changed Reserved Buffer beginning address
	Relative Key	1 UN 2 UN 6 UN	Type: 0 = UN, 2 = UA Length Address

	Usage	Size	Function
	File Description		
*		8 UN	Pointer to associated FIB
		1 UN	Reserved
*		6 UN	Digit offset to block info
*		4 UN	Records per block
*		5 UN	User record size (digits)
		8 UN	End-of-file data block number
		8 UN	Control block number

## RELATIVE FILE ACCESS ROUTINE INTERFACE

All of the routines required to access a Relative File are bound into the user program by the COBOL 74 compiler. Each program will have at most one set of routines regardless of the number of Relative Files declared. All of the routines are reentrant and are capable of handling any number of Relative Files. The access routines use the normal program stack.

An access routine is called from the user program when the user requires a function to be performed on a Relative File. These calls replace the normal READ, WRITE, OPEN, CLOSE BCTs used for conventional files. These calls are of the NTR-EXT type and pass call-by-address parameters. The routines, and specific parameters required by each routine are outlined in the following paragraphs.

### CLOSE

The CLOSE routine is called when a Relative File is to be closed. The parameters required for the CLOSE function are:

ACON address of pointer to RF Data Block

ACON address of 1 UN Close type

- 0 = file
- 4 = release
- 6 = lock
- 8 = purge
- C = remove

### DELETE

The DELETE routine is called when a Relative File data record is to be deleted. The parameters required for the DELETE function are:

ACON address of pointer to RF Data Block

ACON address of exception branch address

## OPEN

The OPEN routine is called when an open is required for a Relative File. The parameters required for the OPEN function are:

ACON address of pointer to RF Data Block

ACON address of 1 UN OPEN type

0 = input

1 = output

2 = input-output

ACON = address of 1 UN lock type

0 = no lock

4 = lock access

8 = lock

## READ

The READ routine is called when a data record of a Relative File is to be read. The parameters required for the READ function are:

ACON address of pointer to RF Data Block

ACON address of exception branch address

ACON address of 1 UN access mode

0 = sequential, no key supplied

This field is used to state whether a key phrase was present on the READ statement.

## REWRITE

The REWRITE routine is called when a Relative File data record is to be replaced by the contents of the user work area. The parameters required for the REWRITE function are:

ACON address of pointer to RF Data Block

ACON address of exception branch address

## START

The START routine is called when the Relative File is to be positioned to a particular data record for subsequent access. The parameters required for the START function are:

ACON address of pointer to RF Data Block  
ACON address of exception branch address  
ACON address of 1 UN start condition

2 = equal  
4 = greater  
6 = greater than or equal

This field contains the condition that the START will attempt to satisfy. This is the condition contained in the key phrase. If no key phrase was used, this field contains 2 (equal).

## WRITE

The WRITE routine is called when a data record is to be placed into a Relative File. The parameters required for the WRITE function are:

ACON address of pointer to RF Data Block  
ACON address of exception branch address

## INDEXED FILE IMPLEMENTATION CONSIDERATIONS

Indexed files differ from conventional files in the B 4000/B 3000/B 2000 systems in that one or more associated key files and data blocks contain additional information not present in the blocks of conventional files. In order to interpret this additional information, compiler-provided intrinsics are bound into a program which declares an indexed file. A user-program operation on a indexed file generates a call on the appropriate intrinsic routine; the intrinsic accesses the file.

There are certain considerations for indexed files:

1. An indexed file can only be accessed meaningfully in a COBOL program in which the file has been declared with ORGANIZATION INDEXED (unless the programmer is prepared to code routines capable of interpreting the indexed file blocks and key files).
2. The presence of the access routines will increase the program size by approximately 5000-bytes. All access routines are bound into a program which accesses an indexed file in any way and are not overlayable.
3. Three buffers are allocated to be shared among the key files of each indexed file. The buffers are approximately 5000-digits each in size, so the effect is to increase program size by approximately 15000-digits per indexed file.
4. The data block will be larger than the simple product of the record size and the blocking factor, since a variable amount of additional information is present at the end of the block.
5. The last block of the file is a control block, which contains additional information about the file beyond that stored in the disk and pack file headers. This block is created when the file is created, accessed when the file is accessed, and updated when the file is updated. The presence of the control block leads to the following:
  - a. When an indexed file is created, the access routines immediately reserve both the first and the last areas of the file. The control block is written into the last area.



- b. The control block has a minimum size of 184 digits plus 80 digits for each ALTERNATE KEY declared for the file. This imposes a minimum block size for the indexed file. Data blocks must be at least that size to avoid wasting space.
- c. The number of areas and records per area specified in a program which creates an indexed file must also be specified in any program attempting to access the file. The access routines read the data file control block when the file is OPENed, using information in the FD to determine the location of the control block. If the file specifications differ from those given when the file was created, the access routines may not locate the control block, and program errors can occur.

For example, if an indexed file is created with 50 areas and 100 records per area, the data file control block will be the block containing record number 5000. A program attempting to access the file must declare those same attributes for the file. Specifying 60 areas of 100 records would cause the access routine to attempt to read a block with record number 6000, leading to an end-of-file condition. In this case, the program would be terminated with a STATUS KEY value of "FA".

6. An Indexed File Data Block is present in the compiled program for each indexed file declared in the program. This block contains certain information needed by the access routines to maintain the file.
7. Four buffers are allocated for each indexed file. These buffers are controlled by the access routines via a Buffer Control Structure (BCS) embedded in the Indexed File Data Block (IF Block).
8. Since the block format differs from that of conventional files by the addition of control information in each block, utility programs (such as DMPALL) used to list the file must be directed to list entire blocks.
9. A block is physically written only when it becomes necessary to access a new block or to close the file, and the current block has been modified. Blocks are not necessarily written immediately after every record modification.
10. Program or system failure during a file update can result in unmatching key and data files. An in-use flag in the data file control block indicates such a condition; when it is set, the access routines prohibit access to the file. Users must have a file recovery procedure, such as one of the following:
  - a. Save copies of indexed data and key files after an update.
  - b. An indexed data file can be opened INPUT with ORGANIZATION RELATIVE. This is an exception to the general rule that relative and indexed files must always be declared to have the organization with which they were created. A recovery program can reconstruct the indexed file by reading the corrupted file in RELATIVE mode and writing it in IN-DEXED mode. At most, the data block being manipulated at the time of the failure will be incorrect.
11. The indexed file declared in a COBOL program is really a group of files: a data file and one or more key files. Outside of COBOL, these files are not recognized as being related. It is user responsibility to specify all files in dumps, loads, and so forth.

File formats, the File Information Blocks (FIBs), the Indexed File Data Block, and access routine interfaces are discussed in detail in the following paragraphs.

## INDEXED FILE FORMATS

Each Indexed File declared in a COBOL 74 program results in multiple physical files for data and key information. Furthermore, the blocks of these files differ from existing file blocks by having control information appended to them. The files are built and updated by the access routines. The MCP is told that each file is an unblocked random file. The new file types are understood by the access routines with two new block formats for each file type.

### INDEXED DATA FILE FORMAT

The Indexed Data File type consists of one or more blocks of data records and one block of control information.

Each data block contains valid and empty data records. The number of data records that a block can hold is specified by the BLOCK CONTAINS clause in the user program file description, unless overridden by the space requirements of the control block. Record slots are the same size as the user record area (work area).

Following the last data record slot in the block is control information used by the access routines to store and deblock the data records. This block control information contains:

1. Type (1 UN). This field is used to distinguish between full and partially filled data blocks. Allowable values are:
  - 2 = available space in block
  - 3 = block is full
2. Next (8 UN). Blocks containing available space are linked together in the data file using a LIFO scheme. Each block with available space points to the block number of the next such block in this field. The last block contains zeros, as do full data blocks.
3. Count (4 UN). The access routines keep a count of the number of valid records in the block as an easy way of telling when full blocks become partially filled and vice versa.
4. Presence Flags (1 UN each). Each record slot in the block has a corresponding presence flag to indicate whether the slot contains a valid data record. Allowable values are:
  - 0 = record not used or deleted
  - 1 = record is present

Each Indexed File group has control information that must be saved when the file is not in use. The last block in the data file has been reserved for this purpose. Users should note that this reduces the maximum number of data records that can be stored in the file and must add an extra block of records to file size calculation, if this is significant. At file creation time the control block number is computed from the number of areas times the blocks per area contained in the data file FIB. Subsequent use of the file always retrieves and returns the control information to and from this block.

The size of the control block is 24 digits plus 80 digits for the data file description entry plus 80 digits for each key file description entry. This imposes a minimum block size on the data blocks in the file and can result in unused space if the record size times blocking factor is less than the control block size. The COBOL-74 compiler will generate a warning describing this condition and pad out the data block size. Note that the control block does not contain block control information, since its format is always known. The format is:

1. In-use Flag (1 UN). Used to insure file integrity in case of halt/load condition when programs are updating the files. It is set to one by the OPEN access routine whenever the indexed file is OPENed OUTPUT or I-O; it is reset to zero when the file is closed by the CLOSE access routine. The OPEN routine will not permit access to an indexed file whose in-use flag is already set, unless the file is opened input in relative mode. Allowable values are:  
0 = file close successful  
1 = file close unsuccessful
2. Version Date Stamp (5 UN). Creation date of the Indexed File.
3. Last Update Date-Time Stamp (15 UN). Last Date and Time Indexed File was opened output or I/O.
4. Number of Keys (2 UN). Number of keys declared when the Indexed File was created.
5. Filler (1 UN).
6. File Description Entries (80 UN each). These fields contain control information for the data and key files from the Indexed File Data Block in the user program.

## INDEXED KEY FILE FORMAT

The Indexed Key File type consists of one or more blocks of key entries and one block of control information. The two block types present in an Indexed Key File are the key block and the control block.

The Indexed Key File conceptually can be described as an inverted tree-like structure with different levels within the tree. This tree structure contains one block at the top level called a ROOT block. It also contains one or more blocks at the lowest level called FINE blocks. All levels in between contain blocks called COARSE blocks.

Each block contains valid and empty key entries. The maximum number of key entries that a block can hold is computed by the COBOL-74 compiler.

A key entry consists of a key value whose length is described by the corresponding key data item in the data file record, and a pointer of 8 digits. The pointer portion of the entries in ROOT and COARSE blocks contain the number of a block at the next lowest level. The key value of ROOT and COARSE entries contain the highest key value of all entries in the lower block. The pointer portion of FINE table entries contains the number of the record slot in the data file containing the record whose key value matches the key portion of the entry. The last key value in the rightmost block on all levels contains a special entry called the omega entry. The key portion of the omega entry is greater than the largest user specified key value. The omega value is used as an upper bound for search routines.

Following the last key entry slot in the block is control information used by the access routines to access and to search the key entries. This block control information contains:

1. Type (1 UN). This field is used to indicate the level of the block within the tree. Allowable values are:
  - 2 = empty key block
  - 4 = root block
  - 5 = coarse block
  - 6 = fine block
  - 7 = root-fine block
2. Next (8 UN). All blocks on a level of the tree structure point to the next block to the immediate right within the level. The right-most block on a level contains zeros in this field.
3. Last (4 UN). Entries in each key block are packed toward the beginning of the block. The LAST field is the position of the last valid entry in the block (zero-relative to the origin entry). This eliminates the need for presence flags in key blocks.

Each Indexed Key File has control information that must be saved when the file is not in use. The last block in the key file has been reserved for this purpose. At file creation time, the control block number is computed from the number of areas times the blocks per area contained in the key file FIB. Subsequent use of the file always retrieves and returns the control information to and from this block.

The size of the control block is 16 digits. This imposes a minimum block size on the key blocks in the file. The COBOL-74 compiler insures that the entry size times entries per block is at least as large as the control block size. Note that the control block does not contain block control information, since its format is always known. The format is:

1. Last Update Date-Time Stamp (15 UN). Last Date and Time the Indexed File was opened for update. This field is checked against the corresponding field in the Indexed Data File control block at open time to insure that all files of the group are at the same update level.
2. Reserved (1 UN).

## **INDEXED DATA FILE FIB**

There is one FIB created for the data file of each Indexed File group. This FIB is generated by using the information provided by the user in the File Control, and FD sections of the COBOL-74 program. This FIB is created in the same manner as a FIB for a normal sequential file, with a few additions. These additions are as follows:

1. There is always a user work area assigned to this FIB. The address is set in FIB-WA. This is the only interface the user has to the Indexed File. The size of this user work area is the size specified in the user record description of the Indexed File, padded to MOD 4 length.
2. Block size must be computed by using the user work area size times the blocking factor declared in the BLOCK CONTAINS clause. To this total, the COBOL-74 compiler adds 14-digits for control information along with one presence digit per record in the block. The block size is then padded to MOD 4 length.
3. The Indexed File Data FIB must reflect the record size equal to the block size as computed in step 2 above. This value is set in FIBMBS.

4. There is only one buffer attached to this FIB. The size of this buffer will be equal to the block size computed in step 2 above. FIB-AA and FIB-BB are set to the beginning and ending addresses of the buffer.
5. The FIB must always specify buffer access mode, FIB-BA:1 set. This prevents MCP access to the user work area. The access routines will handle all movement of data to and from the user work area.
6. The FIB has an access mode of random specified in FIB-DA:1, regardless of what the program specifies. This prevents MCP OPEN buffer fills. The access mode declared by the user will be contained in the Indexed File Data Block.
7. The FIB must have the address of the ACTUAL KEY field in the Indexed File Data Block set in FIBKEY.
8. The FIB has FIBIX2:1 reset and FIBRAD:8 set to force physical I/O, since the access routines determine when I/Os are to be done.
9. The FIB must have FIBORG:4 set to indicate that this is the data file of the Indexed File group.

## INDEXED KEY FILE FIBS

The COBOL-74 compiler creates one FIB for each key that is declared for an Indexed File. This is necessary because each set of key values is contained in a separate physical file. These FIBs are generated based on the following requirements:

1. There is never a user work area assigned to key file FIBs. Key values to be used in I/O functions are always placed in the user work area of the data file prior to keyed function calls.
2. Block size is computed by the COBOL-74 compiler based on the number of key entries which will fit into a block size of 5000-digits. The 5000 number is a target block size to prevent excessive memory requirements for the user program. If the number of key entries per block exceeds 300, then the block will be limited to 300 entries to prevent search times on key blocks from becoming excessive. Likewise, if the number of key entries per block is less than 10, the block will be forced to 10 entries to prevent the tree structure from having too many levels. The block size is then computed by taking the number of key entries per block times the key entry size. To this total the COBOL-74 compiler then adds 14 digits of control information. The block is then padded to MOD 4 length.
3. The Indexed File Key FIBs must reflect the record size equal to the block size computed in step 2 above. This value is set in FIBMBS.
4. The size of each key file area is computed by the COBOL-74 compiler by taking the maximum number of records in the data file and dividing by the number of key entries per block computed in step 2 above. This number represents the number of FINE tables needed in the key file. The number of blocks in the lowest COARSE level is obtained by dividing the number of FINE blocks by the entries per block value. This process continues for each level of the tree structure until the divide results in one block on a level. The number of blocks on each level are added to give the total number of blocks in the key file. One more is added for the control block, and the result divided by the number of areas allocated to the key file. This final value is then placed in FIBRPA.
5. There are three key file buffers for each Indexed File declared in the user program. These buffers are NOT attached to the key file FIBs but are shared among the key files in an Indexed File group. This implies that the buffer size must be the maximum among all key file block sizes. The beginning address of each of the three key buffers is set in the Buffer Control Structure of the Indexed File Data Block.
6. Key file FIBs always specify buffer access mode (FIB-BA:1 set), since key files do not have work areas assigned.

7. Key file FIBs must have an access mode of random in FIB-DA:1. This prevents MCP OPEN buffer fills.
8. Key file FIBs must have the address of the ACTUAL KEY field in the Indexed File Data Block set in FIBKEY.
9. Key file FIBs have FIBIX2:1 reset and FIBRAD:8 set to force physical I/O, since the access routines determine when I/Os are to be done.
10. Key file FIBs have FIBORG:2 set to indicate that they are key files of the Indexed File group.

## INDEXED FILE DATA BLOCK (IF BLOCK)

Each Indexed File has one Indexed File Data Block in the user program. This block will contain all the information required by the access routines to handle an Indexed File that is not in the FIB. This block is the focal point throughout the user program when trying to access an Indexed File. The address of the IF Block will be the address passed on each function call to an access routine. The IF Block contains a number of substructures which are used by the access routines. The fields in the IF Block are listed in table 8-2. Fields preceded with an \* are filled in by the compiler.

**Table 8-2. Indexed File Data Block Format**

	Function	Size	Description
*	File Status Pointer	8 UN	Pointer to file status word
*	OPEN Type	1 UN	Indexed File open state 0 = input 1 = output 2 = input-output 9 = not open
*	Access Mode	1 UN	Declared access mode 0 = sequential 1 = random 2 = dynamic
	Last Function	1 UN	Last function 0 = unsuccessful 1 = open 2 = start 3 = read 4 = write 5 = rewrite 6 = delete 7 = close
	Current Record Area	8 UN 8 UN 4 SN 1 UN  1 UN 8 UN 4 SN	Relative record number Data block number Data slot number State of current record 0 = undefined state 1 = beginning of file 2 = defined state Key of reference Key block number Key entry number

**Table 8-2. Indexed File Data Block Format (Cont)**

	<b>Function</b>	<b>Size</b>	<b>Description</b>
	Actual Key	8 UN	Read/Write key for all files
	Buffer Control Structure	2 UN 8 UN 1 UN  3 UN 6 UN 8 UN	(One entry per buffer) File number, number in FD table Block presently in buffer Buffer changed flag 0 = not changed 1 = changed Reserved Buffer beginning address Reserved
*	Date Stamps	5 UN 5 UN 10 UN	File creation date Last update date Last update time
*	Number of Keys	2 UN	Number of Keys Defined
*	File Descriptions	8 UN	(One entry per file) Pointer to associated FIB
*		1 UN	File-declared-in-program flag 0 = not declared 1 = declared
*		6 UN	Digit offset to block info
*		4 UN	Records/entries per block
*		5 UN	Record/entry size
		8 UN	End-of-file data block number
		8 UN	Control block number
		8 UN	First available space block
		8 UN	ROOT block number
		8 UN	First FINE block number
*		4 UN	Entries kept on block split
*		1 UN	Type of key 0 = UN 1 = SN 2 = UA
*		3 UN	Length of key
*		5 UN	Digit offset to key data item
*	1 UN	Duplicates allowed 0 = duplicates not allowed 1 = duplicates allowed	
	4 UN	Reserved	

## INDEXED FILE BUFFERS

There are four buffers required for an Indexed File. These buffers are used for all data and key file I/Os. The buffers are managed by the access routines. The user program is not able to access these buffers directly.

The buffers are created by the COBOL-74 compiler. The beginning address of each is placed in the BCS table elements in the IF Block. The first buffer is used exclusively for data file I/Os, with the other three buffers used for key file I/Os. The buffers will never be used for any other file in the user program and are likely to have different sizes due to varying key and record lengths. (That is, the data buffer size can differ from the key buffer size.) Each key file buffer is the largest value computed for the individual key block sizes within the Indexed File.

The beginning address of the data buffer is placed in the element 0 of the BCS table. The key buffer beginning addresses are placed in the elements 1, 2, and 3 of the BCS table.

## **INDEXED FILES ACCESS ROUTINE INTERFACE**

All of the routines required to access an Indexed File will be bound into the user program by the COBOL-74 compiler. Each program will have, at most, one set of routines regardless of the number of Indexed Files declared. All of the routines are reentrant and are capable of handling any number of Indexed Files. The access routines use the normal program stack.

An access routine is called from the user program when the user requires a function to be performed on an Indexed File. These calls replace the normal READ, WRITE, OPEN, and CLOSE BCTs used for conventional files. These calls are of the NTR-EXT type and pass call-by-address parameters. The routines and specific parameters required by each routine are outlined in the following paragraphs.

### **CLOSE**

The CLOSE routine is called when an indexed file is to be closed. It completes the control blocks for the data file and all key files and closes the files. The parameters required for the CLOSE function are:

ACON address of pointer to IF Data Block

ACON address of 1 UN CLOSE type

- 0 = file
- 4 = release
- 6 = lock
- 8 = purge
- C = remove

### **DELETE**

The DELETE routine is called when an Indexed File data record is to be deleted. The parameters required for the DELETE function are:

ACON address of pointer to IF Data Block

ACON address of exception branch address

### **OPEN**

The OPEN routine is called when an open is required for an Indexed File. It opens the data file and key files, and checks date-time stamps and the in-use flag. The parameters required for the OPEN function are:

ACON address of pointer to IF Data Block



ACON address of 1 UN OPEN type

- 0 = input
- 1 = output
- 2 = input-output

ACON address of 1 UN LOCK type

- 0 = no lock
- 4 = lock access
- 8 = lock

## READ

The READ routine is called when a data record of an Indexed File is to be read. The parameters required for the READ function are:

ACON address of pointer to IF Data Block

ACON address of exception branch address

ACON address of 1 UN access mode

- 0 = sequential, no key supplied
- 1 = random, key supplied

This field is used to state whether a KEY phrase was present on the READ statement. If a KEY phrase was used, this field contains a one and the beginning address of the key field is placed in the last parameter. If a key phrase was not used, this field contains a zero.

ACON address of 2 UN key to use (FD element number)

This field is used to state which key must be used for the READ function. The number specified here is the number of the element in the FD table that corresponds to the key desired. The prime key = 01, alternate key number 1 = 02, and so forth. A value of 00 indicates that the current key of reference is not to be used.

ACON address of key value address

This field contains the actual starting address of the value in the key data item of the user work area.

## REWRITE

The REWRITE routine is called when an Indexed File data record is to be replaced by the contents of the user work area. The parameters required for the REWRITE function are:

ACON address of pointer to IF Data Block

ACON address of exception branch address

## START

The START routine is called when the Indexed File is to be positioned to a particular data record for subsequent access. The parameters required for the START function are:

ACON address of pointer to IF Data Block

ACON address of exception branch address

**ACON address of 2 UN key to use (FD element number)**

This field is used to state which key will be used for the START operation. The number specified here is the number of the element in the FD table that corresponds to the key desired. The prime key = 01, alternate key number 1 = 02, and so forth. A value of 00 indicates that the current key of reference is not to be used.

**ACON address of key value address**

This field will contain the actual starting address of the value in the key data item of the user work area.

**ACON address of 3 UN length of key**

This is the number of characters or digits, depending upon the key type, that the START will use to compare for the specified condition. This number can be less than or equal to the actual defined size of the key. The START will do a key-to-record compare on only the number of positions specified by this parameter.

**ACON address of 1 UN start condition**

- 2 = equal
- 4 = greater
- 6 = greater than or equal

This field contains the condition that the START will attempt to satisfy. This is the condition contained in the key phrase. If no key phrase was used, this field contains a 2 (equal).

## **WRITE**

The WRITE routine is called when a data record is to be placed into an Indexed File. The WRITE will update all key files associated with the data file. The parameters for the WRITE function are:

ACON address of pointer to IF Data Block

ACON address of exception branch address

## SECTION 9

# READER-SORTER DLP INTERFACE

### DLP OVERVIEW

On a 900 series Medium System, all I/O activity is controlled by an Input/Output Translator (IOT), that is located in the processor, and by a group of DLPs, that are installed in DLP bases. The IOT can communicate with from one to four DLP bases each of which can contain up to eight DLPs.

The IOT is basically a switching processor. When an Initiate I/O request is received the IOT works in conjunction with the DLP base to establish connection to the appropriate DLP. During the time that data is being transferred between the DLP and the system the IOT provides the DLP with a path to and from memory.

The DLP is the device which, upon receipt of a translated I/O descriptor from the IOT, establishes a communication path to the peripheral device. Once the path is established, the DLP accepts data from, or passes data to, the peripheral device. Unique DLPs are used to interface the different types of peripheral devices.

Because each DLP has a data buffer, data can be transferred to and from the peripheral device at the comparatively low rate of the device; then, when the buffer is full (or empty, depending on the descriptor), data can be transferred to the system at the highest rate allowed by memory.

### WRITE-FLIP-READ OPERATION

In order to increase system throughput while running a Reader-Sorter, the Write-Flip-Read operation was implemented in the R-S DLP. This operation reduces contention in the system by reducing the number of memory accesses by the DLP. A Write-Flip-Read operation entails a Write and a Read of the Reader-Sorter file during a single I/O. There are three Reader-Sorter BCTs that utilize the Write-Flip-Read operation. They are

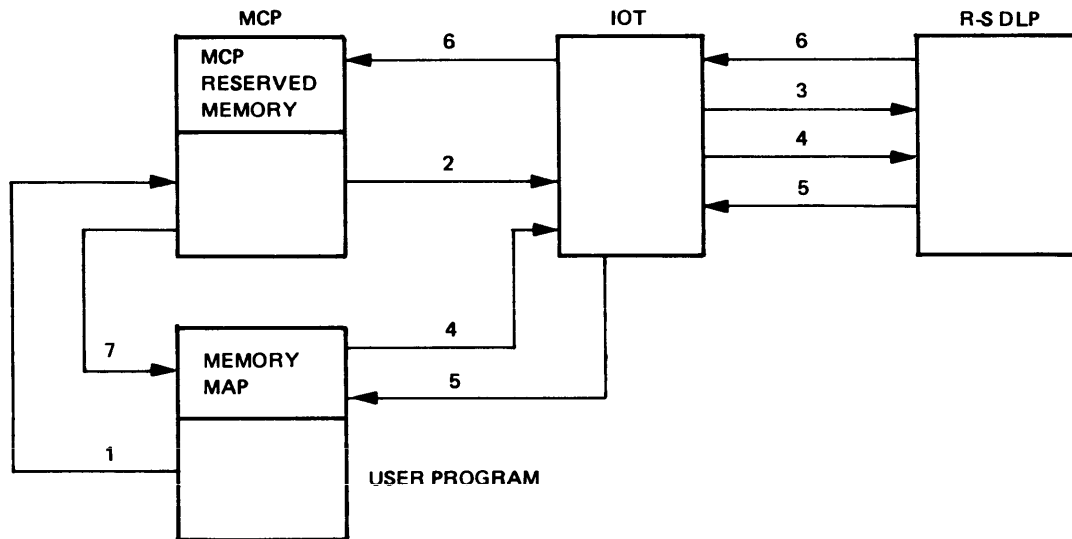
1. Start Flow Read.
2. Pocket Select Read.
3. Demand Feed Read.

These BCTs are described in this section under "Reader-Sorter BCTs". In general, a Write-Flip-Read operation consists of the following steps:

1. The user program executes one of the Reader-Sorter BCTs.
2. The MCP builds the appropriate DLP instruction (I/O descriptor and descriptor link), and the processor initiates the instruction.
3. The IOT transfers the descriptor to the R-S DLP.
4. The R-S DLP and the IOT transfer data from the area of the user program pointed to by the descriptor link to the Reader-Sorter. This information includes endorsement and control information.
5. After the control information is processed (step 4 above) the interface "flips", and the R-S DLP and the IOT transfer data from the Reader-Sorter to the user program. Included in this data is extended status information, called the Result Status. Exceptions identified in the Result Status are flagged either by the Reader-Sorter or by the R-S DLP.
6. The R-S DLP builds a result descriptor from information received from the Reader-Sorter, stores the result descriptor in the reserved memory location for the channel that was used, and the Interrupt flip-flop is set on. This result descriptor is referred to as the IOT/DLP R/D.

- The MCP places both the IOT/DLP R/D and a reformatted version of the IOT/DLP R/D in the user program and then reinstates the user program in the Control State User routine. The reformatted result descriptor is called the Soft Result Descriptor or the Soft R/D.

Figure 9-1 illustrates the paths of communication between the processor and the R-S DLP for I/O descriptor initiate, data flow, and result descriptor generation. The numbers used in the diagram are keyed to the preceding description of a Write-Flip-Read operation.



P5491

**Figure 9-1. Processor/R-S DLP Communications Paths**

## CONTROL STATE USER ROUTINE

A user program utilizing a Reader-Sorter DLP is composed of two routines, known as the Control State User routine and the Normal State routine. These routines are required to operate asynchronously. The main interface between these routines is a wrap-around, in-memory queue known as the soft tank.

The Control State User routine handles time-critical processing requirements. When this routine is invoked, the user program operates in control state, meaning that only certain processor error conditions are allowed to interrupt its processing. These error conditions are:

- Invalid instructions.
- Address errors.
- Instruction timeouts.
- Irrecoverable memory errors.

In addition, timer interrupts are also permitted to interrupt this routine.

B 2000/B 3000/B 4000 MCPVI Programmer's Guide  
Reader/Sorter DLP Interface

The Control State User routine is invoked and placed in control state by the MCP whenever the character recognition data for a document is transferred from the R-S DLP to the user program (I/O Complete). If the Normal State routine has not finished processing a previous item when this occurs, it will be interrupted. The MCP analyzes the IOT/DLP R/D and invokes the Control State User routine at one of three locations as specified in the user statement for the Reader-Sorter file. The three locations, in priority of reinstatement, are

1. Soft Result Descriptor attention. Soft R/D exceptions are present. There may also be Result Status exceptions. For a description of the Soft R/D see the subsection entitled "Notes on the Soft Result Descriptor".
2. Result Status attention. Result Status exceptions are present. For a description of the Result Status see the subsection entitled "Notes on the Result Status".
3. No exceptions.

The Control State User routine is exited by issuing a Pocket Select Read BCT. After this, the Normal State routine is eligible to be reinstated. The Control State User routine performs the following functions:

1. Analyze and format the current document.
2. Recognize and perform error handling either for exceptions reported with their associated document or for exceptions that cannot be associated with a document.
3. Make a pocket select decision on the document.
4. Cause the document to be endorsed if required. If endorsement is required, the Control State User routine must also make sure that the proper endorsements are loaded from memory into the appropriate location in the Memory Map prior to issuing the Pocket Select Read BCT.
5. Cause the document to be microfilmed, if required, and set up the pocket select number field in the Memory Map by setting the camera control bit and the endorser band control bits in the Memory Map.
6. Request flow be stopped by setting the flow stop bit in the Memory Map.

Although the Control State User routine has complete control of the processor, it is extremely limited in the functions it may perform because of its time-critical nature. Once the routine is invoked it must issue the Pocket Select Read BCT before the document in process reaches the endorser. Otherwise that document will be too late to pocket select, endorse, or film and will be rejected. The interval of time available to the Control State User Routine to process a document is pocket select time (PST). The interval of time after a document is read and before the next document is read is document cycle time (DCT). Document cycle time is 36.9 milliseconds for a B 9137 and 23 milliseconds for a B 9138. MCP time (MCPT) is the time used by the MCP to interrogate the IOT/DLP R/D at I/O Complete, reinstate the user program in the Control State User routine, and to recognize the Pocket Select Read BCT and issue a Pocket Select I/O descriptor. Hardware contention time (HCT) is the time used by the IOT and DLP to gain a path to memory and time spent waiting for the MCP to handle I/O Completes.

Pocket Select time is computed as follows:

$$\text{PST} = (\text{DCT} - \text{MCPT} - \text{HCT}) / \# \text{ OF READER-SORTERS}$$

Because the Control State User routine is so time-dependent it is not allowed to issue any BCT other than the Pocket Select Read BCT.

## NORMAL STATE ROUTINE

The Normal State routine must complement the Control State User routine by performing functions that cannot be handled in the Control State routine. The Normal State routine is divided into two routines: the Normal State Processing routine and the Flow Stop routine.

### Normal State Processing Routine

The Normal State Processing routine is invoked for each item processed by the Control State routine. The primary purposes of this routine are:

1. Exception handling.
2. Transferring formatted document images from the soft tank to a storage media.
3. Maintaining a recovery file.

### Flow Stop Routine

The user program is reinstated at the flow stop label after the Reader-Sorter stops flow and after all documents in the transport have been processed by the Control State User routine and the Normal State Processing routine. The user program requests a Stop Flow by setting the Stop Flow bit in the Memory Map. The Flow Stop routine performs the following functions:

1. Exception handling and operator notification of exception conditions.
2. Operator notification of events.
3. Validations that require access to a storage media.

## SOFT TANK

As mentioned above, the main interface between the Control State User routine and the Normal State routine is a wrap-around, in-memory queue called the soft tank. The tank is used as a document queue to facilitate asynchronous processing by the two routines. Each element of the tank should contain, at a minimum, a document image and exception information reported with the document (Soft R/D and Result Status). Other fields may be added to each entry as a convenience. The Control State routine builds the soft tank and maintains the index to the next entry. The Normal State routine pops the tank and maintains the index to the next entry to be retrieved from the tank. A counter of the number of items currently in the tank should be maintained. To ensure against overflow the tank must be large enough to hold the sum of the following:

1. The maximum number of items that the Normal State routine is to be allowed to fall behind the Control State User routine.
2. The maximum number of items in motion between the feeder and the read heads when a Stop Flow occurs.
3. The maximum number of items that the DLP can buffer (2 items).

Certain Result Status and Soft R/D exceptions, such as Can't Read are reported with the document; however, other conditions such as Real Time Too Late are reported with the following I/O Complete. For this reason it is suggested that documents not be processed in the Normal State Processing routine until the next I/O is complete and the Result Status and the Soft R/D for that document are available.

## FLOW AND DEMAND MODES OF PROCESSING

Reading of documents is accomplished either in flow or in Demand mode. In Flow mode, the Reader-Sorter reads continually until a Stop Flow is programmed or until the Reader-Sorter or the R-S DLP causes a Stop Flow to occur. In Demand mode the user program requests a single document to be fed and read.

## MEMORY MAP

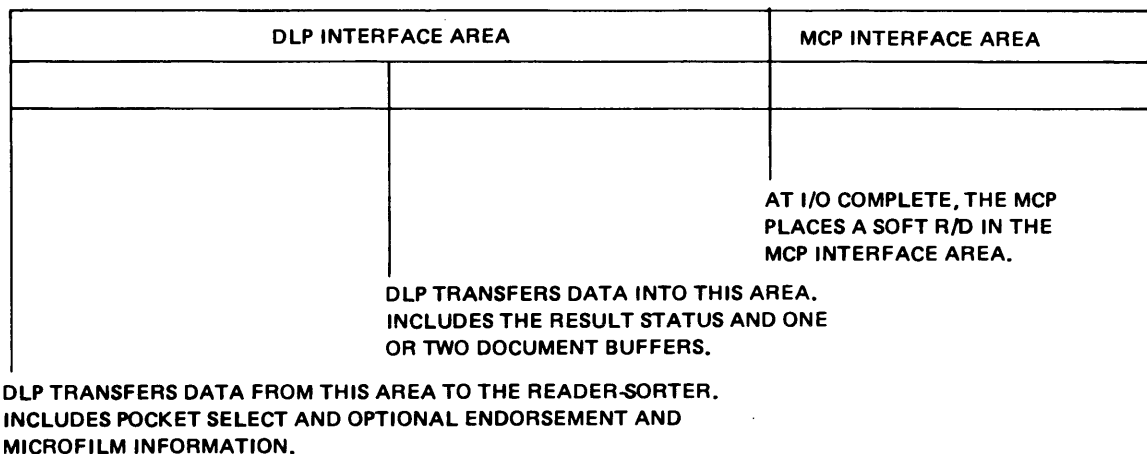
The user's program must define two interface areas in the Reader-Sorter file. They are collectively known as the Memory Map. The first area is called the DLP Interface area. This area is 744 digits in length and is used for two-way communications between the user program and the DLP. The second area is called the MCP Interface area. This area is 36 digits in length and is used for two-way communications between the user program and the MCP. The total size of the Memory Map is 780 digits.

The DLP Interface area is divided into two areas. During the write portion of a Write-Flip-Read operation, the DLP transfers positions 0 – 347 from the Memory Map to the Reader-Sorter. Then during the read portion of the I/O, the DLP transfers positions 348 – 744 from its buffers to the Memory Map. Positions 0 – 347 contain pocket select, microfilm, and endorsement information. Positions 348 – 744 contain document and Result Status information.

When a Stop Flow condition occurs after a Write-Flip-Read operation is completed, the DLP will place the last used microfilm ID number in the document area (beginning at position 358) instead of document information. If the Stop Flow was caused by a jam or a missort, the DLP will also store a 2-digit code identifying the fault location of the condition on the Reader-Sorter (beginning at position 356).

The MCP Interface area is used for two-way communications with the user program. After a Reader-Sorter I/O complete, the MCP builds and stores the Soft Result Descriptor before invoking the Control State user routine. In addition, The MCP will also store status and characteristics information in the MCP Interface area when requested by a Status or Characteristics BCT. Finally, this area contains information needed by the MCP to build DLP instructions.

Figure 9-2 describes the functions of the Memory Map during a Pocket Select Read BCT. The diagram should be read from left to right.



P5492

Figure 9-2. Memory Map Functions

The user program should zero out all reserved fields in the Memory Map before starting flow or before issuing a Demand Feed Read BCT.

The Memory Map is described Table 9-1.

**Table 9-1. Memory Map Description**

Position	Size	Description
0	780D	Memory Map.
0	744D	DLP Interface area.
0	64D	Reserved.
64	272D	Endorser bands. Refer to Non-Impact Endorsement, following, for further explanation of endorser fields.
64	3D	Reserved.
67	1D	Band identifier number for field 4. Must be a non-zero value (1 – 4) if this band is being loaded.
68	64D	Endorsement text for field 4. 32 bytes of alphanumeric endorsement text are provided.
132	3D	Reserved.
135	1D	Band identifier number for field 3. Must be a non-zero value (1 – 4) if this band is being loaded.
136	64D	Endorsement text for field 3. Thirty-two bytes of alphanumeric endorsement text are provided.
200	3D	Reserved.
203	1D	Band identifier number for field 2. Must be a non-zero value (1 – 4) if this band is being loaded.
204	64D	Endorsement text for field 2. 32 bytes of alphanumeric endorsement text are provided.
268	3D	Reserved.
271	1D	Band identifier number for field 1. Must be a non-zero value (1 – 4) if this band is being loaded.
272	64D	Endorsement text for field 1. 32 bytes of alphanumeric endorsement text are provided.
336	4D	Reserved. See Non-Impact Endorsement for details. of this field.
340	2D	Pocket select number. The pocket number to which the item is to be sent. If rejecting a document this number must be greater than 47. Used for Pocket Select/Read BCT (op = 29).



**Table 9-1. Memory Map Description (Cont)**

<b>Position</b>	<b>Size</b>	<b>Description</b>
342	1D	Endorser band control. 8 bit = 1 spray endorser band no. 1 4 bit = 1 spray endorser band no. 2 2 bit = 1 spray endorser band no. 3 1 bit = 1 spray endorser band no. 4
343	1D	Camera/Stop Flow control. 8 bit = 1 microfilm this document. 4 bit Reserved. 2 bit = 1 impact endorse this document 1 bit = 1 Stop Flow.
344	4D	Reserved (all zero).
348	8D	Result Status.
348	1D	8 bit = 1 Station A buffer overflow (BIT 1). 4 bit = 1 Station A parity error (R-S → DLP) (BIT 2). 2 bit = 1 Station A can't read (BIT 3). 1 bit = 1 Too-late-to-read (BIT 4).
349	1D	8 bit Reserved (BIT 5). 4 bit Reserved (BIT 6). 2 bit Reserved (BIT 7). 1 bit = 1 Document tracking error (BIT 8).
350	1D	8 bit = 1 Station B buffer overflow. (BIT 9). 4 bit = 1 Station B parity error (R-S → DLP) (BIT 10). 2 bit = 1 Station B can't read (BIT 11). 1 bit Reserved (BIT 12).
351	1D	8 bit Reserved (BIT 13). 4 bit = 1 Interface error (BIT 14). 2 bit = 1 Internal DLP error (BIT 15). 1 bit Reserved (BIT 16).
352	1D	8 bit = 1 Flow stopped (BIT 17). 4 bit = 1 Not ready (BIT 18). 2 bit = 1 Black band (BIT 19). 1 bit = 1 Endorser parity error (BIT 20).
353	1D	8 bit = 1 Real time too late (physical too late to pocket, endorse, or microfilm) (BIT 21). 4 bit = 1 Multiple documents (BIT 22). 2 bit = 1 Overlength document (BIT 23). 1 bit = 1 Underspaced document (BIT 24).
354	1D	8 bit = 1 Missort (BIT 25).

**Table 9-1. Memory Map Description (Cont)**

<b>Position</b>	<b>Size</b>	<b>Description</b>
		4 bit = 1 Feeder jam (BIT 26).
		2 bit = 1 Jam (BIT 27).
		1 bit = 1 Film advance (BIT 28).
355	1D	8 bit Reserved (BIT 29).
		4 bit Post read document error (mech) (BIT 30).
		2 bit Post read document error (elec) (BIT 31).
		1 bit = 1 Parity error from the R-S DLP. (BIT 32).
356	20D	Flow stop information. Stored with each flow stop.
356	2D	Fault location for jam or missort. 2-digit number specifies beam-of-light or pocket location.
358	18D	Microfilm ID number(9 bytes). The last film ID number used by the sorter. This field is not meaningful unless the camera is powered on.
356	388D	Document information.
356	2D	Read station A Total document length counter. This field gives the total length of the document read at station A. Total document length is given in bytes.
358	2D	Read station B total document length counter. Same as for station A.
360	384D	Read Station A and B document buffers. There are two buffers each of which may contain up to 96 bytes of document information. The read station B document buffer immediately follows the last character of document information from the read station A buffer. Unless the "report feed error" option is set (B9138 utility mode) a B9138 feed check will not cause the Reader-Sorter to stop flow. The user program is notified of a feed check by a single can't read character in the document buffer. (document buffer = 01003F).
744	36D	MCP Interface area.

**Table 9-1. Memory Map Description (Cont)**

<b>Position</b>	<b>Size</b>	<b>Description</b>
744	8D	IOT/DLP Result Descriptor. Eight digits of R/D are always reported. The first four digits of this R/D are generated by the IOT and are referred to as the IOT R/D. The second four digits are generated by the DLP and are referred to as the DLP R/D. The MCP stores the IOT/DLP R/D in this location at each I/O Complete.
752	4D	Reserved.
756	1D	Number of endorser text bands to be loaded. This number must be zero through four. The DLP is capable of accepting from zero through four bands of endorsement text on each pocket select operation. (See the Non-Impact endorsement section of this document). The MCP uses this field to compute the beginning address of the "write" portion of the write flip read operation based on the number of bands to load.
757	1D	Status. The MCP places the status of the Reader-Sorter in this area in response to a Status BCT. Formatting is as follows: 8 bit = 1 Slewing microfilm. = 0 Not slewing microfilm. 4 bit = 1 Camera not ready. = 0 Camera ready, not present, or not powered on. 2 bit = 1 Endorser not ready. = 0 Endorser ready, not present, or not powered on. 1 bit = 1 Reader-Sorter not ready. = 0 Reader-Sorter ready.
758	3D	Characteristics. The MCP places the characteristics of its Reader-Sorter in this area in response to a Characteristics BCT. Formatting is as follows:
758	1D	8 bit = 1 Endorser band one present. = 0 Endorser band one not present. 4 bit = 1 Endorser band two present. = 0 Endorser band two not present. 2 bit = 1 Endorser band three present. = 0 Endorser band three not present.

**Table 9-1. Memory Map Description (Cont)**

<b>Position</b>	<b>Size</b>	<b>Description</b>
		1 bit = 1 Endorser band four present. = 0 Endorser band four not present.
759	1D	8 bit = 1 Reader-Sorter is a B9137. = 0 Reader-Sorter is a B9138.
		4 bit = 1 Camera present. = 0 Camera not present or powered on.
		2 bit = 1 R-S DLP interface. = 0 4A control interface.
		1 bit = 1 Reader-Sorter power failure.
760	1D	8 bit = 1 Impact endorser present
761	3D	Reserved
764	4D	MCP Soft Result Descriptor. The MCP places a reformatted IOT/DLP result descriptor in this field when an I/O Complete of a reader-sorter BCT occurs.

**NOTE**

See the "Notes on the MCP Soft Result Descriptor" for further explanation of Soft R/D error conditions.

764	1D	8 bit = 1 I/O invalid to the DLP (BIT 1). 4 bit = 1 BCT invalid to the MCP (BIT 2). 2 bit = 1 Flow condition error (BIT 3). 1 bit = 1 System interface parity error or descriptor error (BIT 4).
765	1D	8 bit = 1 Microfilm operation not completed. (BIT 5). 4 bit = 1 Non-present option required. (BIT 6). 2 bit = 1 Reserved (BIT 7). 1 bit = 1 Pocket select error (BIT 8).
766	1D	8 bit = 1 Bad interface information (BIT 9). 4 bit = 1 Timeout (BIT 10). 2 bit = 1 Camera not ready (BIT 11). 1 bit = 1 Parity error (R-S → DLP) (BIT 12).
767	1D	8 bit = 1 Power failure (BIT 13). 4 bit = 1 Memory overflow (BIT 14). 2 bit = 1 DLP error (BIT 15). 1 bit IOT R/D error (BIT 16).
768	4D	Pocket light/generate image count mark parameters (NNRB).

**Table 9-1. Memory Map Description (Cont)**

<b>Position</b>	<b>Size</b>	<b>Description</b>
768	2D	NN = The pocket number for which the light should be illuminated if B = 0. If B = 1, then NN equals the number of ICMs which should be generated.
770	1D	Reserved.
771	1D	B = 0 Pocket light illumination. On the B9138, if the cutslips/pocket light option is set on (cutslips), a cutslip will be fed from the secondary hopper and pocketed to pocket NN. The document will not be read.  B = 1 ICM operation. B = 2 Advance batch number for impact endorser
772	4D	Start flow parameters (RHFF).
772	1D	Reserved.
773	1D	H = 1 Data in the low order nine bytes of endorser band 1 is microfilm header data.  H = 0 No microfilm header data present.
774	1D	F = 1 Read data from read station A.
775	1D	F = 1 Read data from read station B.
776	4D	Demand read parameters (RHFF). Each digit is used in the same manner as the corresponding digit in the Start Flow parameters. Separate areas are used for Start Flow and demand read parameters in order to simplify the programming effort required to use a mixture of the two techniques.

## **MCP READER-SORTER EXTENSIONS**

The MCP Reader-Sorter extentions include Open BCT, Close BCT, and the MCP MICR Module.

### **OPEN BCT**

When opening a DLP type Reader-Sorter file, the external file identification (device name) value in the label area is compared to the device names of the unassigned DLP units. When a match is found, the unit is assigned to the program by the MCP. If a match is not found, the MCP displays NO FILE and an IL is required.

For Assembler coded programs prior to issuing the Open BCT, the user program must set the 8 bit in the first digit of the FIB (referred to as FIBST1) to declare use of this DLP interface and require selection of a B 9137/B 9138 Reader-Sorter.

## CLOSE BCT

A DLP Reader-Sorter file must be closed with Release.

## MCP MICR MODULE

The MCP requires that the MICR module be in memory when running Reader-Sorters in an on-line mode. The MICR module is automatically loaded into memory along with the standard version of the MCP at Halt/Load time provided the MICR option is set.

The MICR module has primary responsibility for the following:

1. Reader-Sorter BCT validations.
2. Building I/O descriptors to be sent to the R-S DLP.
3. Interrogating the IOT/DLP R/D at I/O Complete.
4. Formatting the Soft R/D.
5. Reinstating the user program at the proper location after a Reader-Sorter I/O Complete or after handling a Reader-Sorter BCT.

## Reader-Sorter BCTs

All MICR BCTs conform to the following general format:

BCT 374 (absolute memory address 374)  
BUN around (next instruction)  
P1 = ACON FIB  
P2 = NN  
P3 = ACON error  
P4 = ACON flow stopped

P1 is a pointer to the user program sorter file FIB.

P2 is a 2-digit number which uniquely identifies the MICR BCT type (see Table 9-2).

P3 is not included in all MICR BCTs. It is a pointer to the error label in case the BCT does not pass the standard MICR BCT verification (see Table 9-2).

P4 is not included in all MICR BCTs. It is a pointer to the flow stop label.

**Table 9-2. Reader-Sorter BCT Types and Operations**

<b>Reader-Sorter BCT Type</b>	<b>Operation</b>
42	Start Flow Read.
43	Demand Feed Read.
44	Pocket Light/Generate ICMs.
45	Microfilm Slew.
47	Status.
48	Characteristics.
46	Logical Read.
29	Pocket Select Read.

B 2000/B 3000/B 4000 MCPVI Programmer's Guide  
Reader/Sorter DLP Interface

There is a BPL syntax for the Reader-Sorter BCTs (see B 2000/B 3000/B 4000 Series BPL Reference Manual, form number 1113735.)

The MICR module has two routines for validating Reader-Sorter BCTs. The first routine validates all Reader-Sorter BCTs other than the Pocket Select Read BCT and the second routine validates the Pocket Select Read BCT. The MCP predetermines which of the two routines will be called by modifying the branch address at absolute memory address 374. The MCP enables the Pocket Select Read BCT validation routine when a Reader-Sorter I/O goes complete and just prior to reinstating the user program in the Control State User routine. Otherwise, the standard Reader-Sorter BCT validation routine is called.

With the exception of the Pocket Select Read BCT, the Reader-Sorter BCTs are verified by the MCP in the following manner:

1. P2 is a valid MICR BCT type. If not, results in a DS or DP condition.
2. The open sorter file table is searched by MIX number and FIB address. If an entry is not found, results in a DS or DP condition.
3. P3 is non-zero, contains no undigits, is within program base/limit, and is mod 2. If not, results in a DS or DP condition.
4. Verify that the flow condition is proper. The Logical Read BCT is the only valid MICR BCT that can be issued in the Normal State routine while flow is in progress. If the flow condition is not proper the MCP sets the Flow condition error (BIT 3) in the Soft R/D and reinstates the program at the error label associated with the BCT.

Pocket Select Read BCTs are verified as follows:

1. P2 is Pocket Select Read BCT type. If not, the MCP marks the file as if a Stop Flow condition had occurred, sets the BCT invalid to the MCP bit (BIT 1) in the Soft R/D, and permits the processing portion of the user program to run in order to complete the handling of any tanked items. The user program is then reinstated at the flow stopped label for recognition of the error condition.
2. The number of endorser text bands to load or to spray must be 0 thru 4. If not, the MCP marks the file as if a Stop Flow condition had occurred, sets the BCT invalid to the MCP bit (BIT 1) in the Soft R/D and permits the Normal State Processing routine to run in order to complete the handling of any tanked items. The user program is then reinstated at the flow stopped label for recognition of the error condition.

### *START FLOW READ BCT*

The Start Flow Read BCT initiates flow feed. Just prior to flow feed, the DLP Will optionally read the band information from the Memory Map to allow the user program to place a microfilm header on the microfilm. While in Flow mode, only the Pocket Select Read and Logical Read BCTs are valid.

The format of the Start Flow Read BCT is:

BCT 374  
BUN around  
P1 = ACON FIB  
P2 = 42  
P3 = ACON error  
P4 = ACON flow stopped

The Reader-Sorter file must be opened and flow must be stopped prior to issuing a Start Flow.

The program is reinstated at one of three locations:

1. Once flow is physically started, the program is reinstated at the instruction following the Start Flow BCT (BUN around). Since Start Flow is not treated as a logical read, the user's program must proceed to a Logical Read BCT when it has been reinstated.
2. If flow cannot be successfully started because of an error condition, the user's program is reinstated at the error label as specified in the Start Flow BCT (P3) with the appropriate Soft R/D bits set.
3. If flow cannot be started because of a feeder jam, the user program will be reinstated at the flow stop label (P4) with the appropriate Result Status bit set.

The MCP performs the following functions when a Start Flow Read BCT is issued:

1. Performs standard BCT verification.
2. Constructs I/O descriptor which is composed of op code, Start Flow parameters, and A & B address.
3. Initializes MCP tanking counter.

Prior to issuing the Start Flow Read BCT, it is the user program's responsibility to:

1. Set RHFF (position 772, MCP Interface area).
2. Move the microfilm header data into the last nine positions of the endorsement text for endorser band 1, if microfilm data is to be loaded. For more information on Microfilm Header data see the subsection entitled "Microfilming".
3. Open the Reader-Sorter file.

### *DEMAND FEED READ BCT*

The Demand Feed Read BCT results in one document per BCT being fed to a read station and read. A Pocket Select Read BCT must follow each read completion.

Continuous use of Demand Read is not recommended since it causes excessive wear on the mechanical portions of the Reader-Sorter.

The format of the Demand Feed Read BCT is as follows:

BCT 374  
BUN around  
P1 = ACON FIB  
P2 = 43  
P3 = ACON error

Flow must be stopped prior to issuing a Demand Feed Read BCT. This BCT may be issued even though the Reader-Sorter has been previously opened flow.



The user program is reinstated at one of two locations:

1. When an item has been fed, read, stored, and pocket selected by the Control State User routine, the program is reinstated at the instruction following the BCT (BUN around). The program should not proceed to a Logical Read BCT because the Demand Feed Read BCT implies a logical read. After all processing is completed for that document, the program may proceed to any other valid BCT as required.
2. If flow cannot be successfully started because of an error condition, the user's program is reinstated at the error label as specified in the BCT parameter (P3) with the appropriate Soft R/D bits set.

The user's program should set up RHFF (position 772, MCP Interface area) for the Demand Feed Read BCT.

*POCKET LIGHT/BATCH COUNT ADVANCE/GENERATE IMAGE COUNT MARKS  
BCT*

This BCT illuminates the specified pocket light or generates Image Count Marks on the microfilm, or advances batch count on the Impact Endorser, depending on the parameter settings.

The format of the BCT is as follows:

BCT 374  
BUN around  
P1 = ACON FIB  
P2 = 44  
P3 = ACON error

Flow must be stopped prior to issuing this BCT. The program is reinstated at one of two locations:

1. At the instruction following the BCT (BUN around) when the operation is complete and there are no exceptions.
2. Any exception results in the program being reinstated at the error label as specified in the BCT parameter (P3) with the appropriate Soft R/D bits set.

When this BCT is given, the MCP:

1. Performs standard BCT verification.
2. Constructs an I/O descriptor which is composed of an OP code and pocket light/ICM parameters.

Prior to issuing this BCT, the user must set the Pocket Light/Generate Image Count Mark parameters (NNRB) in the MCP Interface area as required.

ICM BCTs should be issued one ICM at a time so that if the operation is not completed the user program can determine how many ICMs were successfully placed on the microfilm. However, this is not a requirement.

On a B 9138 there is an option called the Cutslips/Pocket Light option. If this option is set to cutslips then a cutslip will be sent from the secondary feeder to the specified pocket instead of lighting the indicated pocket light. The item that is fed from the secondary feeder is not read and therefore is not reported to the user program. See also the B 9138 Merge On Text option.

### *MICROFILM SLEW BCT*

This BCT is issued when it is desired to advance the microfilm to the beginning of the next 100 or 200-foot reel within a cassette. There are no parameters in the Memory Map for this BCT.

The format of the BCT is as follows:

BCT 374  
BUN around  
P1 = ACON FIB  
P2 = 45  
P3 = ACON error

Flow must be stopped prior to issuing this BCT. The program is reinstated at one of two locations:

1. At the instruction following the BCT (BUN around) when the operation is complete and there are no exceptions. This BCT goes I/O Complete when the slew is successfully started. It is the user program's responsibility to recognize any subsequent malfunction and/or completion of the slew operation. (See Microfilming for more information on detecting the completion of the slew.
2. Any exception results in the program being reinstated at the error label as specified in the BCT parameter (P3) with the appropriate Soft R/D bits set.

When this BCT is given, the MCP performs the standard BCT verification and constructs an I/O descriptor which is composed of an op code only.

For more information on microfilming see "Microfilming".

### *STATUS BCT*

If the user program requires information regarding the status of the Reader-Sorter, it may issue a Status BCT. This results in the status of the Reader-Sorter being placed in the MCP Interface area. There are no parameters in the Memory Map for this BCT.

The format of the Status BCT is as follows:

BCT 374  
BUN around  
P1 = ACON FIB  
P2 = 47  
P3 = ACON error

Flow must be stopped prior to issuing this BCT.

The program is reinstated at the instruction following the BCT (BUN around) when the operation is complete and there are no exceptions. Any exception results in the program being reinstated at the error label as specified in the BCT parameter (P3) with the appropriate Soft R/D bits set.

When this BCT is given, the MCP performs the standard BCT verification, and determines the status of the Reader-Sorter and sets up the MCP Interface area.

### *CHARACTERISTICS BCT*

If the program requires information regarding the characteristics of the Reader-Sorter, it may issue this BCT. This results in the characteristics of the Reader-Sorter being placed in the MCP Interface area. There are no parameters associated with this BCT.

The format of the Characteristics BCT is as follows:

```
BCT 374
BUN around
P1 = ACON FIB
P2 = 48
P3 = ACON error
```

Flow must be stopped prior to issuing this BCT.

The program is reinstated at the instruction following the BCT (BUN around) when the operation is complete and there are no exceptions. Any exception results in the program being reinstated at the error label as specified in the BCT parameter (P3) with the appropriate Soft R/D bits set.

When this BCT is given, the MCP performs the standard BCT verification, determines the characteristics of the Reader-Sorter, and sets up the MCP Interface area.

### *LOGICAL READ BCT*

The Logical Read BCT does not result in the initiation of a physical I/O operation. It merely serves as the synchronization mechanism between the Control State User routine and the Normal State Processing routine.

The format of the Logical Read BCT is as follows:

```
BCT 374
BUN around
P1 = ACON FIB
P2 = 46
P3 = ACON error
P4 = ACON flow stopped
```

The Logical Read BCT is only valid if flow is in process.

The program is reinstated at one of three locations:

1. At the instruction following the Logical Read BCT (BUN around) when there is an item to process.
2. At the error label if the standard BCT verification is not passed.
3. At the flow stopped label when a Stop Flow condition occurs and all items have been processed.

The MCP performs the following functions when a Logical Read BCT is given:

1. Standard BCT verification.
2. Control synchronization. When a Logical Read BCT is issued and there is an outstanding pocket selected item (soft tank not empty), the program is reinstated at the instruction following the Logical Read BCT (BUN around). If there is not an outstanding pocket selected item (soft tank empty) and flow has not stopped, the program is marked waiting I/O Complete and is not reinstated until the next pocket select is issued.

The MICR Module maintains a tanking counter that is incremented by one every time a pocket select is issued, and is decremented by one each time a Logical Read BCT is issued. By using this counter, the MCP knows if there is an item to process in the Normal State Processing routine.

Processing continues as outlined above until flow stops. The program continues to be reinstated at the instruction following the Logical Read BCT until all pocket selected items have been logically read (soft tank empty). At the next Logical Read BCT the program is reinstated at the flow stop label.

During the processing portion of the program, the user must untank the items along with the Result Status and Soft R/D and determine the necessary action to be taken on that item based on these two result indicators. The program should return to a Logical Read BCT when it has completed the processing of that item.

When the user program is reinstated at the flow stopped label, it must recognize and handle any exception conditions and determine the cause of the flow stop so that appropriate action can be taken based on the information reported in the Result Status and Soft Result Descriptor. After a Stop Flow, the user program must issue a Start Flow Read BCT before returning to a Logical Read BCT.

### *POCKET SELECT READ BCT*

The Pocket Select Read BCT is the only acceptable exit from the Control State User routine either when flow is in progress or after a Demand Feed Read BCT has been issued.

A Pocket Select Read BCT must be issued by the Control State User routine for each item. This BCT implies that the user routine is ready to read and pocket select the next item as soon as the DLP makes it available (I/O complete).

The format of the Pocket Select Read BCT is:

```
BCT 374  
BUN around  
P1 = ACON FIB  
P2 = 29
```

The MCP performs the following functions when the Control State User routine issues the Pocket Select Read BCT:

1. Updates tanking counter for synchronization. See commentary under Logical Read BCT for more on the tanking counter.
2. Performs Pocket Select Read BCT verification.
3. Constructs an I/O descriptor which is composed of an op code and an A address which is based on the number of bands to be loaded.
4. If in Flow mode, marks the program ready to run. If in Demand mode, marks the program waiting I/O complete (pocket select).

The Control State User routine must:

1. Capture all pertinent information required. The Pocket Select Read BCT implies that the Control State User routine is ready to accept another document from the interface.
2. Load the desired pocket number in the DLP Interface area.
3. Set the desired impact and non-impact endorser control bits.
4. Load the desired text bands and band numbers if new texts are required.
5. Set the number of text bands to be loaded.
6. Set the camera control bit.
7. Set the Stop Flow bit if required.
8. Recognize and handle the exception conditions as reported in the Soft R/D.
9. Recognize and handle the exception conditions as reported in the Result Status.
10. Issue the Pocket Select Read BCT.

## **FLOW MODE PROCESSING SUMMARY AND USER PROGRAM OUTLINE**

The R-S DLP Interface can perhaps best be summarized and illustrated by describing the document flow of four items through the user program while in Flow mode. The document flow discussion is somewhat idealistic since the timing of I/O Completes for the Reader-Sorter file is arbitrary, and since the discussion assumes no exception conditions.

This description refers to the diagram in figure 9-3 which is a pseudo-coded outline of the two user program routines: the Control State User routine and the Normal State routine. Each line of the diagram is divided into two areas: the six leftmost upper case characters are labels and the comments within brackets ( < > ) are descriptions of each of the major steps within one of the two routines.

## **DOCUMENT FLOW DESCRIPTION**

Once all BOJ processing is complete, including the opening of the Reader-Sorter file, the user program branches to the beginning of the Normal State routine. At the label START, the user program issues the Start Flow Read BCT. This BCT causes document flow to begin on the Reader-Sorter.

After the MCP reinstates the user program from the Start Flow Read BCT, the user program executes the Logical Read BCT at the label READ (see Figure 9-3). The user program will not be reinstated in the Normal State routine until the first document is processed by the Control State User routine unless certain exceptions exist, such as feeder jam. If one of these exceptions exists, the user program is reinstated at the flow stop action label (specified in the Logical Read BCT).

At I/O Complete, the user program is reinstated by the MCP at the beginning of the Control State User routine. Assuming that there are no exceptions the document is processed and placed in the soft tank along with the Result Status and Soft R/D. Then, after all processing is complete, the user program issues the Pocket Select Read BCT.

After completion of the Pocket Select Read BCT, the MCP reinstates the user program in the Normal State routine at the instruction following the Logical Read BCT. The first document in the soft tank after a start flow is treated differently from subsequent items. An item should not be processed by the Normal State routine until all errors affecting that document have been reported. Some Result Status and Soft R/D errors will not be reported until the next I/O Complete (for example, real time too late). Therefore, Normal State processing of an item is delayed until the next Result Status and Soft R/D are available. However, the document data is untanked and placed in the Normal State work area.

The user program branches back to the Logical Read BCT indicating that the Normal State routine is ready to accept another item from the soft tank.

B 2000/B 3000/B 4000 MCPVI Programmer's Guide  
Reader/Sorter DLP Interface

When another I/O Complete occurs, the Control State User routine is invoked. After all processing in this routine the Pocket Select Read BCT is again issued. Document 2 and its Result Status and Soft R/D is in the soft tank.

The user program is reinstated at the instruction following the Logical Read BCT.

This time through the Normal State routine, the user program pops the second Result Status and Soft R/D from the soft tank and processes the first document. Before issuing the third Logical Read BCT, the second item is popped and placed in the Normal State work area. At any time during this processing, another I/O Complete could occur and the user program would be interrupted and placed in the Control State User routine. Then when that document (number 3) is pocket selected and placed in the soft tank, the MCP reinstates the user program at the next Normal State instruction to be executed before the interrupt occurred.

After Normal State processing of document 1 is complete, the program branches to the Logical Read BCT. Note that the soft tank now contains document 3 with its associated Result Status and soft R/D, and the Normal State work area contains document 2.

The user program is reinstated at the instruction following the Logical Read BCT and begins processing the second document in the Normal State routine.

Once again an I/O goes complete, and the user program is reinstated in the Control State User routine. Flow through control state continues as outlined above, but this time assumes that the user program requests a Stop Flow.

The user program is reinstated in Normal State and continues processing, winding up by issuing the Logical Read BCT again.

Processing in Normal State continues as outlined above. All items that were fed but not read at the time the flow stop was issued must be processed. When the Logical Read BCT is issued for the last item, the user program will be reinstated at the flow stop action label. The user program must reconcile any Result Status and Soft R/D errors and handle the last item. Then the program must decide whether to continue the task by starting flow again or ending the task.

B 2000/B 3000/B 4000 MCPVI Programmer's Guide  
 Reader/Sorter DLP Interface

```

LABEL <pseudo code>
*****
*           control state user routine           *
*****

RDATTN <Soft R/D exception entry point>
      :
      :
RSATTN <Result Status exception entry point>
      :
      :
NOEXC  <no exception entry point>
      :
      :
      :
      <place formatted document, Result Status, &
      Soft R/D in soft tank>
      <bump control state soft tank index>
      <Pocket select read BCT>

*****
*           normal state routine           *
*****

START <Start flow read BCT>
READ  <Logical read BCT. Flow stop action label is
      FLWSTP>
      <if first document after start flow go to UNTANK>
      <retrieve Result Status, & Soft R/D>
      :
      <process document in Normal State routine>
      :
UNTANK <retrieve first/next document from soft tank>
      <bump normal state soft tank index>
      <go to READ>
FLWSTP      :
           :
           <if end of job go to EOJ>
           :
           <go to START>

EOJ      :
      <stop>
P5493

```

Figure 9-3. Document Flow in Control State User and Normal State Routines

## NOTES ON THE RESULT STATUS

Information regarding the execution of a Write-Flip-Read operation is sent to the DLP from the Reader-Sorter as 32-bit Result Status information. Then the DLP updates and transfers the Result Status to the user program (DLP interface area) along with the document information. Table 9-3 shows which Result Status exception conditions are applicable to each of the three BCTs that use the Write-Flip-Read operation. An "X" indicates that the Result Status exception may be reported after the execution of the BCT.

**Table 9-3. Result Status Exception Conditions for Write-Flip-Read**

Bit	Exception	Start Flow-Read	Demand Feed-Read	Pocket Select-Read
1,9	buffer overflow	X	X	X
2,10	parity error (R-S → DLP)	X	X	X
3,11	can't read	X	X	X
4	too late to read			X
8	document tracking error	X	X	X
14	interface error	X	X	X
15	internal DLP error	X	X	X
17	flow stopped	X	X	X
18	sorter not ready	X	X	X
19	black band	X	X	X
20	endorser parity error	X	X	X
21	real time too late			X
22	multiple documents	X	X	X
23	overlength document	X	X	X
24	underspaced document	X	X	X
25	missort			X
26	feeder jam	X	X	X
27	jam	X	X	X
28	film advance	X	X	X
32	parity error (DLP → R-S)	X	X	X

As mentioned previously, exception conditions identified in the Result Status are reported with the affected document at the next I/O Complete (subsequent document or flow stop) or at flow stop only. The way these exception conditions are reported is outlined in Table 9-4. For missorts, multiple documents, overlength document, or underspaced document exceptions see additional comments under the exception description. A description of each type of Result Status exception follows the table.

**Table 9-4. Reporting Result Status Exception Conditions**

Bit	Exception	Affected Doc.	Next I/O	Flow Stop
1,9	buffer overflow	X		



**Table 9-4. Reporting Result Status Exception Conditions (Cont)**

Bit	Exception	Affected Doc.	Next I/O	Flow Stop
2,10	parity error (R-S → DLP)	X		
3,11	can't read	X		
4	too late to read			X
8	document tracking error			X
14	interface error	X		
15	internal DLP error	X		
17	flow stop			X
18	not ready	X		X
19	black band			X
20	endorser parity error		X	
21	real time too late		X	X
22	multiple documents	X	X	X
23	overlength document	X	X	X
24	underspaced document	X	X	X
25	missort		X	X
26	feeder jam			X
27	jam			X
28	film advance			X
32	parity error (DLP → R-S)		X	

**Buffer overflow (Bits 1 & 9).**

The Reader-Sorter read more than 96 characters from the document.

**Parity error (R-S → DLP) (Bits 2 & 10).**

A parity error was detected on the Reader-Sorter to DLP interface by the DLP. A "SUB" character is used by the DLP to replace the characters in error.

**Can't read (Bits 3 & 11).**

The Reader-Sorter was unable to read one or more characters from the document. The Reader-Sorter will replace unreadable characters with a "SUB" character.

**Too late to read (Bit 4).**

The DLP buffers are full and there is insufficient space to store all of the data from the Reader-Sorter for another document. The DLP stops the feeder and sets the flow stopped bit. The real time too late bit will also be set. This bit is also reported on dual read situations when read station B is not present.

**Document tracking error (Bit 8).**

The DLP lost track of the document at the read heads. This exception only occurs during a dual read when read station B finishes with a document before read station 1. The DLP reports this bit with the Stop Flow bit and no further document data is sent to the user program after the current I/O. Any document information should be ignored.

**Interface error (Bit 14).**

The DLP detected a parity error on information from the Reader-Sorter. The document data that is reported with this condition is questionable. The document should be rejected.

**Internal DLP error (Bit 15).**

The DLP detected a condition where the integrity of any further data sent to the user program cannot be guaranteed. The user program should stop flow and reject the current and all subsequent documents in the transport.

**Flow stopped (Bit 17).**

The Reader-Sorter transport is empty and there are no items remaining in the DLP buffers. If a jam or missort caused the flow stop, then two digits of fault location information are stored immediately following the Result Status. All flow stop conditions cause the last microfilm identification number used by the Reader-Sorter to be stored beginning at position 356 in the Memory Map. This number is meaningless unless the camera is in use.

**Not ready (Bit 18).**

The Reader-Sorter is not ready.

**Black band (Bit 19).**

A black-banded item was detected exiting the feeder. The Reader-Sorter will stop flow immediately following the black-banded document. The flow stopped bit will also be set.

**Endorser parity error (Bit 20).**

The Reader-Sorter detected a parity error during the transfer of endorser data from the DLP. For each parity error detected on endorser band data, either two "DLE" characters will be substituted for the character pair, or the two characters will be deleted.

**Real time too late (Bit 21).**

Pocket Select information was not received by the Reader-Sorter before the document reached the time-critical area. A B 9137 Reader-Sorter turns off the feeder, rejects the late item and all of the subsequent items in the transport. The DLP will report flow stop after it processes the last item. A B 9138 Reader-Sorter turns off the feeder, rejects the late item, and processes all subsequent items in the transport normally. The DLP will report the error with the fault item.

**Multiple, overlength, or underspaced documents (Bits 22, 23, or 24).**

Multiple, overlength, or underspaced documents are collectively referred to as feed checks.

Multiple, overlength, or underspaced documents were detected by the Reader-Sorter before the items reached the first read head. A B 9137 Reader-Sorter will report these conditions with the affected documents and stop the feeder. The bit will not be reported at flow stop. On the B 9138 Reader-Sorter these conditions will not be reported unless the report feed error option is set in the B 9138 Utility Mode. If not set, a feed check will be reported as an item with a single "can't read" character. A B 9138 with the report feed error option set will report this condition for the fault item only and will not stop flow.

Each item with one of these bits set must be pocket selected to the reject pocket. Failure to pocket select the item will result in a real time too late exception. Pocket selection of a fault item to a pocket other than the reject pocket will cause a missort to be reported since such an item always rejects.

**Missort (Bit 25).**

The Reader-Sorter detected a missort. A B 9137 Reader-Sorter turns off the feeder and attempts to pocket all subsequent documents correctly. The DLP will report the missort on the next I/O Complete and on each subsequent I/O Complete up to and including flow stop. A B 9138 Reader-Sorter turns off the feeder and attempts to pocket all subsequent documents correctly. The DLP will report the missort at flow stop since the B 9138 controls its own terminal displays and recovery.

At flow stop the fault location is specified as a 2-digit code immediately following the Result Status. This code represents the number of the pocket assigned to the item by the user program.

**Feeder jam (Bit 26).**

The feeder has stopped due to a jam. The Reader-Sorter will attempt to read and pocket select all items in the transport. The DLP will report this condition with the flow stop bit set.

**Jam (Bit 27).**

A jam has been detected at an area other than the feeder. The feeder and the transport are turned off. The jam must be manually cleared. The DLP reports this condition with the flow stop bit set. Two digits of fault location information are stored immediately following the Result Status. The B 9138 generates its own terminal displays for the jam.

A B 9137 Reader-Sorter will report the jam prior to the transport being cleared. However, a B 9138 Reader-Sorter will not report the jam or flow stop to the user program until the transport has been cleared because of the possibility of further jams and missorts during the B 9138 restart.

**Film advance (Bit 28).**

The end of a 100 or 200-foot section of film was reached during document flow. This condition is reported along with the flow stopped and not ready bits. For more information, see Microfilming, following.

**Post-read document error – mechanical (Bit 30).**

Document slippage was detected. This exception is reported with all fault documents. The Reader-Sorter sends all fault items to the reject pocket. Failure to pocket select a fault item to the reject pocket will result in a missort. The Reader-Sorter will stop flow. This condition is only applicable to B 9137 Reader-Sorters.

**Post-read document error – electrical (Bit 31)**

The document tracking logic in the Reader-Sorter was in error. This exception is reported with all fault documents. The Reader-Sorter sends all fault items to the reject pocket. Failure to pocket select a fault item to the reject pocket will result in a missort. The Reader-Sorter will stop flow. This condition is applicable only to B 9137 Reader-Sorters.

**Parity error (DLP → R-S) (Bit 32).**

A parity error was detected by the Reader-Sorter on information sent from the DLP.

## **NOTES ON THE SOFT RESULT DESCRIPTOR**

Upon completion of an operation, the DLP returns an IOT/DLP result descriptor to the processor. This result descriptor contains the information necessary for the MCP to determine if the operation was completed properly. The MCP uses this result descriptor to build the Soft R/D that it places in the MCP Interface area at I/O complete.

During flow, Soft R/D errors are reported to the user program either with the affected document, on the next I/O (subsequent document or flow stop), at flow stop, or at the BCT error label. In Table 9-5 an "X" identifies where the condition may be reported.

**Table 9-5. Soft Result Descriptor Error Locations**

Bit	Exception	Affected Doc.	Next I/O	Flow Stop	Error Label
1	I/O invalid to DLP (pocket select)			X	
	I/O invalid to DLP (non-pocket select)				X
2	BCT invalid to MCP				X
3	flow condition error				X
4	system interface error	X			
6	non-present option		X		
8	pocket select error		X		
9	bad interface information			X	
10	timeout			X	
12	parity error (R/S → DLP)	X			
13	power failure	X			X
14	memory overflow	X			X
15	DLP error	X			X

In Table 9-6, an “X” indicates that the Soft R/D exception condition corresponding to a bit position in the table is a possible exception for the MICR BCT.

**Table 9-6. Bit Positions for Soft Result Descriptor Exception Conditions**

MICR BCT Type	Soft R/D Bit Position															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
ICM	x	x	x	x	x	x		x	x	x	x	x	x	x	x	x
pkt lght	x	x	x	x	x			x	x			x	x	x	x	x
slew	x	x	x	x	x	x		x	x	x	x	x	x	x	x	x
charact	x	x	x	x												
status	x	x	x	x												
pkt sel	x	x	x	x	x		x	x	x			x	x	x	x	x
strt flw	x	x	x	x	x			x	x			x	x	x	x	x
demand	x	x	x	x	x			x	x			x	x	x	x	x
feed																

A description of each type of Soft R/D error follows:

### **I/O INVALID TO THE DLP (BIT 1)**

One of the following conditions occurred:

1. Parity error on initiate I/O.
2. The MCP received an invalid IOT/DLP result descriptor from the DLP.
3. The four digits in the Memory Map beginning at position 344 were not zero.
4. Write data was too long or too short. The Memory Overflow bit (bit 14 in the Soft R/D) will also be set.

These conditions are handled as if an immediate I/O Complete has occurred. The user program is either reinstated at the error label specified in the BCT or, if the error occurred as the result of a Pocket Select Read BCT, the program will continue to retrieve items from the soft tank in the Normal State routine and the bit will be reported at flow stop. This exception will cause an immediate flow stop.

### **BCT INVALID TO THE MCP (BIT 2)**

One of the following conditions occurred:

1. The user program issued a BCT other than the Pocket Select-Read BCT while in the Control State User routine.
2. The number of endorser bands to load or to spray was not 0 – 4.
3. The Start Flow parameters (location 772) or the demand read parameters (location 776) were not valid.

This condition is handled as if an immediate I/O Complete has occurred. This bit is set by the MCP and is not reported in the IOT/DLP result descriptor. The user program is reinstated at the error label specified in the BCT if the error occurred as the result of a Start Flow Read BCT. However, if the error occurred as the result of a Pocket Select Read BCT, the user program will continue to retrieve items from the soft tank in the Normal State processing routine and the bit will be reported at flow stop. This condition will cause an immediate flow stop.

### **FLOW CONDITION ERROR (BIT 3)**

One of the following conditions occurred:

1. Flow was in progress when a BCT requiring flow not to be in progress was issued.
2. A Soft R/D was generated but not a Result Status. The Result Status will be all hexadecimal Es. This is the normal situation when certain exceptions are reported.

This condition is handled as if an immediate I/O Complete occurred. The bit is set by the MCP and is not reported in IOT/DLP result descriptor. For condition 1, the user program is reinstated at the error label specified in the BCT. This condition will cause an immediate flow stop.

### **SYSTEM INTERFACE PARITY ERROR (BIT 4)**

A parity error was detected on the data lines between the processor and the DLP during the initial I/O of a Reader-Sorter BCT.

## **MICROFILM OPERATION NOT COMPLETED (BIT 5)**

One of the following conditions occurred:

1. Microfilm or Slew operation not completed properly.
2. Pocket light operation not completed properly.

## **NON-PRESENT OPTION (BIT 6)**

A non-present or not ready option was requested. If this exception is reported during flow, it will be reported on the next I/O Complete after the option went non-present. The interface cannot tell the user program which documents were not filmed or endorsed; all it can do is report when the user program tries to film/endorse and is not able to. The pocket select will not be aborted and flow will not be stopped.

## **POCKET SELECT ERROR (BIT 8)**

One of the following conditions occurred:

1. The Reader-Sorter did not respond to a pocket select request from the DLP within the proper time interval.
2. The Reader-Sorter responded to a pocket select request although none was issued.

## **BAD INTERFACE INFORMATION (BIT 9)**

The DLP detected an error in the interface.

## **TIMEOUT (BIT 10)**

One of the following conditions occurred:

1. The Reader-Sorter did not go not ready within the proper time frame for an ICM operation.
2. The Reader-Sorter did not respond within the proper time frame for a microfilm slew operation.
3. The Reader-Sorter did not go not ready within the proper time frame for a pocket light operation.
4. The Reader-Sorter did not respond within the proper time frame to a Start Flow or demand read operation. This condition may be caused by a parity error (DLP → R-S) on the Start Flow or Demand Read descriptor.

This condition will be reported with the Flow Condition Error bit (bit 3) set indicating that a Result Status was not reported to the user program.

## **CAMERA NOT READY (BIT 11)**

Camera option is available, power is applied, and a not ready condition exists. This bit is not set if the camera option is not available.

## **PARITY ERROR (R-S → DLP) (BIT 12)**

A parity error was detected on the Reader-Sorter to DLP transfer.

## READER-SORTER POWER FAILURE (BIT 13)

One of the following conditions occurred:

1. The interface cable is not connected.
2. The Reader-Sorter is powered off or is off-line.

If one of these conditions occurs during a Pocket Select Read, an IOT/DLP result descriptor is sent to the MCP immediately, and any information in the DLP buffers is not valid. The user program will be reinstated at flow stop. This condition will cause an immediate flow stop.

Whenever this condition occurs the DLP will reset itself. Therefore, any further pocket select requests will be invalid.

## MEMORY OVERFLOW (BIT 14)

One of the following conditions occurred:

1. The system requested termination of the read portion of a data transfer before the DLP had transferred all of the available data. This condition is also known as B-address exceeded.
2. This bit will also be set if the Write data is too long or too short.

If this condition occurs as the result of a Pocket Select Read BCT, the user program will continue to retrieve items from the soft tank in the Normal State Processing routine. This condition causes an immediate flow stop.

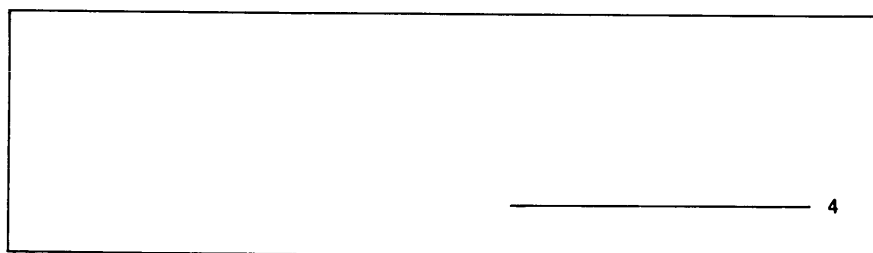
## DLP ERROR (BIT 15)

An ambiguous state exists in the DLP. The pending operations should be terminated and the channel result descriptor evaluated to discover the problem. The DLP will stop flow immediately.

If this condition occurs as the result of a Pocket Select Read BCT, the user program will continue to retrieve items from the soft tank in the Normal State Processing routine. This condition causes an immediate flow stop.

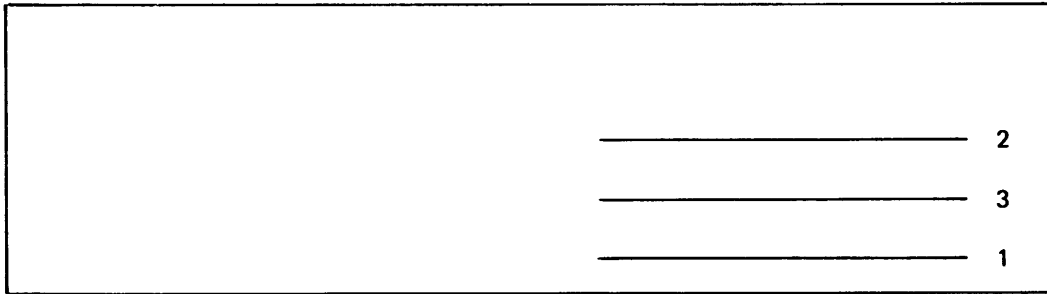
## NON-IMPACT ENDORSEMENT

In the DLP Interface area (locations 64 – 339) there are up to four fields, one for each ink jet on the endorser. These fields are used to load text into the endorser. Each field is composed of one word, with the low-order digit identifying the ink jet on the endorser, and 32 bytes of endorsement text. A line of endorsement text is called a band. Each ink jet is identified by a band identification number. In figures 9-4 and 9-5, each line represents a band. The number to the right of each line is the band identification number for that band.



P5494

Figure 9-4. Single Jet Endorser (Usually the Front of the Document)



P5495

**Figure 9-5. Three-Jet Endorser (Usually the Back of the Document)**

Bands 2, 3 and 4 are each composed of 32 bytes of user-defined alphanumeric text. Band 1 has 23 bytes of user-defined alphanumeric text. The rightmost byte of the 23 may be used as the merge on text recognition character (see the B 9138 Merge On Text option). The final 9 bytes in band 1 contain the document ID number. This number is programmatically assigned. When microfilming, it is placed on the film with each document picture.

A band and a band ID number may be loaded into any one of the four fields in the DLP Interface area. Also, from zero to four bands may be loaded into the endorser. However, it is beneficial to include only those text bands which are different from those currently stored in the endorser band memory. Elimination of these repeated bands lessens the data transfer time during the time critical pocket select process. The endorsement texts to be changed should be placed in the low order text locations. For example, if only a single band text is to be changed, it should be placed in buffer location 272 with the appropriate memory band number in buffer location 271. If two texts are to be changed, they should be placed in buffer locations 204 and 272 with corresponding memory band numbers in buffer locations 203 and 271.

Digit 756 in the MCP Interface area specifies the number of bands to be loaded into the endorser. This number may be zero through four.

Digit 342 in the DLP Interface area specifies which ink jets to fire.

Digits 336 – 339 in the DLP Interface area tell the DLP that it is at the end of endorsement text. These digits must always be zero.

## **MICROFILMING**

The camera module on the Reader-Sorter is a precision optical/mechanical instrument that records images on microfilm at high speed and to a high level of resolution. The document transport mechanism is connected to the film transport mechanism. The film moves during exposure so there is no relative movement between the film and the image of the document. The film is started when the leading edge of the document enters the optical gate of the microfilmer, and is stopped when the trailing edge leaves the gate.

A 9-digit number is photographed alongside the document image. This is the same number that was specified in the low nine bytes of endorser band 1 when the document was pocket selected. An image count mark (ICM) is also included with each document image. These ICMs serve as locators on the film. The various microfilm viewing devices key off these ICMs during their scanning functions.



B 2000/B 3000/B 4000 MCPVI Programmer's Guide  
Reader/Sorter DLP Interface

The Reader-Sorter camera uses Duo-Duplex film. Both the front and the back of the document are photographed simultaneously with both sides of the document using one half of the film width. After the first half of the film is exposed, the film cassette is taken out of the camera, flipped and reinserted into the camera. Then the second half of the film is exposed. Either "thin base" (0.0025 inch) or "thick base" (0.0055 inch) film may be used. Thin base film allows the loading of longer film lengths in the camera during photography. Use of thin base film is standard. The use of thick base film requires a minor wiring change by a Field Engineer. Film may be supplied or loaded in either disposable or reusable cassettes which have a nominal capacity of 500 feet for thick base film and 1000 feet for thin base film. Additional capacity is provided for footage required for leaders and film advances.

The user program specifies whether a document is to be microfilmed in the Control State user routine. If the document is to be filmed the 8 bit of the Camera/Stop flow control digit (position 343) in the DLP Interface area is set to 1. Otherwise, the bit is set to zero.

The microfilmer automatically causes interruption of document feed and microfilm operations at intervals of 100 feet (for thick base film ) or 200 feet (for thin base film). This permits a film advance of approximately 3 feet to provide a clear band of film. The clear section permits cutting of the film without losing any of the filmed images. The film advance works in the following way.

The microfilmer informs the Reader-Sorter that it is at the end of the 100-foot or 200-foot section of film. The sorter stops feeding documents, but continues reading, transporting and pocket selecting documents still in the feed mechanism. The microfilmer, in turn, microfilms these documents. At flow stop, the film advance bit is in the Result Status.

When all documents in the transport are pocket selected, the microfilmer goes not ready and advances three feet of film. These three feet of film are not counted on the film exposed indicator.

When the film advance is completed, the microfilmer is again ready to microfilm documents under Reader-Sorter control, and the user program starts flow.

The film advance procedure takes about four seconds and is reported in the Result Status. The User program can determine when the advance has finished by performing a Status BCT. It is suggested that this BCT be performed at four or five-second intervals until the advance is complete or the camera goes not ready (exception condition).

It is the user's program responsibility to keep track of how many 100-foot or 200-foot sections of unexposed film are left on a cassette side. The user program must also remember which side of the cassette is being filmed.

A 9-character microfilm header number may be recorded on the film to identify the film section for subsequent storage and retrieval. Document and ICM imaging are inhibited during this sequence. The microfilm header number is placed in the position normally occupied by the document identification number.

A microfilm header number may be put on the microfilm:

1. At the beginning of the first 100-foot or 200-foot section of film after the camera has been set up and the film has been advanced three feet.
2. At the beginning of any other 100-foot or 200-foot usable section of film.
3. Within a section of film, between documents.
4. Within a section of film, after a film slew. In this case, film has been slewed to a point within a 100-foot or 200-foot section and the header is to be entered before another batch of documents is microfilmed.
5. At the end of a side of film.

In cases 2 and 3, if the Reader-Sorter is in Flow mode, it must be stopped before the header number can be entered.

Once flow is stopped, the microfilm header number is placed in the low order nine bytes of endorsement band 1; digit 773, in the MCP Interface area (Start Flow parameters), is set to 1; and flow is started.

Film may be programmatically slewed to the beginning of the next 100-foot or 200-foot section of film using the Microfilm slew BCT. Flow on the Reader-Sorter must be stopped prior to issuing this BCT. The user program can also tell when a microfilm slew is in progress thru use of the Status BCT. After the Status BCT is issued, the 8 bit of the second digit in the MCP Interface area (location 757) will be turned on if the slew is in progress.

## **MISCELLANEOUS**

### **B 9138 MERGE ON TEXT OPTION**

The Merge Feed On Text feature allows documents to be fed from the secondary feeder under user program control. The merge items are "merge fed on demand" from the secondary feeder. Merge feed works as follows:

1. The B 9138 recognizes the merge request.
2. The Sorter momentarily stops feed from the primary feeder.
3. When the last item from the primary feeder clears the secondary feeder transport entry point, demand feeds for the current number of merge requests are issued.
4. As soon as the last merge feed item has entered the transport, the Sorter will re-initiate flow from the primary feeder.

If any fault conditions occur on a merge feed item prior to acknowledgement of a successful pocket select it is the user program's responsibility to re-issue the request. If flow is stopped before a merge can be performed, then the appropriate number of secondary hopper demand feeds will be issued at the next Start Flow by the Reader-Sorter.

The Merge Feed On Text option must be enabled in the Utility mode of the B 9138.

The merge request is recognized by interpreting the tenth from the last byte in the endorser band number 1 text. The last nine bytes of this band are reserved as microfilmer ID numbers when performing microfilmer camera operations. If the byte contains a blank, then merge feed is not recognized. If the byte is any other character, then a merge feed request is recognized.

After a merge feed request is issued, the user program must replace the byte text with a blank. If the option is not enabled coincident with endorsing, the character represented by this byte will be sprayed by the endorser.

## NOTES

1. There is no prohibition of the number or frequency of Merge Feed On Text requests.
2. Merge feed requests can be made on the disposition of previously fed merge items.
3. Due to the band text character data substitution function performed when there is a parity problem on transmission of band text from the control to the sorter, it is possible for a merge feed on text operation to be performed when the user program is not expecting one. All user programs should be able to handle unexpected items from the secondary feeder.
4. The user should use this option with care. It is possible to feed one item from the primary feeder, forget to "blank out" the merge field byte, and continue to feed only from the secondary feeder.
5. If the endorser option is not present, the user can still load band 1 data for merge feeds only, but it is the user's responsibility not to send the endorse spray bits set with the band load/pocket select command. To do so will result in the non-present option bit in the Result Status being set on the next I/O Complete.
6. If the microfilmer option is not present the user can still load band 1 data for merge feeds only, but it is the user's responsibility not to send the microfilm click bit set for microfilming along with the band load/pocket select command.
7. Transmitting band text only via the DLP on any pocket select is legitimate at any time.
8. This method of feeding cutslips from the secondary hopper is preferred over the "pocket-light" method since the data from the cutslip is available to the user program and the cutslip's final disposition is under programmatic control with complete error handling ability. In addition, merge feed requests do not require flow to be stopped.

## DLP DELIMITER CHARACTER SET

The valid delimiter characters for the DLP are as follows:

### MICR Font E13B

Symbol	Delimiter	EBCDIC
Transit	;	5E
Amount	:	7A
On-us	<	4C
Dash	-	7E
Can't read	SUB	3F
Parity error	DLE	10

## ASSEMBLER CONSTRUCTS

Assembler provides a User statement, a File statement, and a Unit card, described in the following paragraphs.

### USER Statement

The Assembler syntax for the USER construct, which specifies the locations for reinstating the Control State User routine, is as follows:

1. A label = Result Status attention.
2. B label = Soft Result Descriptor attention.
3. D label (column 58) = no exceptions.

## FILE Statement

In Assembler, the FILE statement must:

1. Specify label convention by a blank or S in column 46 of the C address field.
2. Specify the external file identification (device name in the A address field).
3. Contain "SOR" in columns 34 – 36.
4. Specify work area and buffer with a blank, 0, or W in column 55.
5. Be labeled in column 58. Although not used by the MCP or the DLP, the Assembler compiler will give a syntax error without the label.

## UNIT CARD

Unit card syntax is:

```
UNIT cc/uu <device name> [=] <hardware mnemonic>
```

The device name is handled in the same manner as a data-comm adapter-id, that is, as a 1- to 6-character data name. The first character must be alphabetic. The following five characters must be alphanumeric.

The hardware mnemonic for the B 9137/B 9138 Reader-Sorter utilizing a DLP is:

<b>Mnemonic</b>	<b>Reader-Sorter</b>
S4A	B9137-3
S4B	B9138

For example:

```
UNIT 7/0 SRTR11 S4A  
UNIT 12/0 SRTR12 S4B
```

## SECTION 10

### 4A CONTROL APPLICATION PROGRAM INTERFACE

#### INTRODUCTION

A B 9137-3 or B 9138 Reader/Sorter (hereafter referred to as Sorter) is used with B 2000/B 3000/B 4000 series systems through the 4A Reader/Sorter Control (hereafter referred to as Control). The Control interfaces to both the Sorter and the system through a set of bidirectional lines. Each of the lines has two possible logic levels: HIGH or LOW. Communication between the units is accomplished by changes in the level on a combination of lines. The logic levels of the lines can be viewed by placing a display monitor on the Control.

#### COLDSTART/WARMSTART UNIT CARD

Unit card syntax is

UNIT cc/uu <device name> [=] <hardware mnemonic>

The device name is handled in the same manner as a data comm adapter-id, that is, a 1- through 6-character data name the first character of which must be alphabetic. The following five characters must be alphanumeric.

Hardware mnemonics for Sorters using a 4A Control are:

Sorter	Mnemonic
B 9137-3	S4A
B 9138	S4B

#### USER FILE STATEMENT

In Assembler, the User File statement must conform to the following procedures:

1. Specify label convention by a blank or S in column 46 of the C address field.
2. Specify the external file identification (device name) in the A address field.
3. Contain SOR in columns 34 – 36.
4. Specify the work area and buffer with a blank, 0 or W in column 55.
5. Contain a label in column 58. Although the label is not used by the MCP or the 4A Control, the ASMBLR compiler will give a syntax error without the label.

For BPL syntax see B 2000/B 3000/B 4000 Series BPL Reference Manual form no. 1113735.

#### MEMORY MAP

The user program has two interface areas for the Sorter file. They are collectively known as the Memory Map (see Table 10-1). The first area is called the Control Interface area. This area is 756 digits in length and is used by the Control to communicate with the user program. The second area is called the MCP Interface area. This area is 24 digits in length and is used by the MCP to pass information to the user program. The total size of the Memory Map is 780 digits.

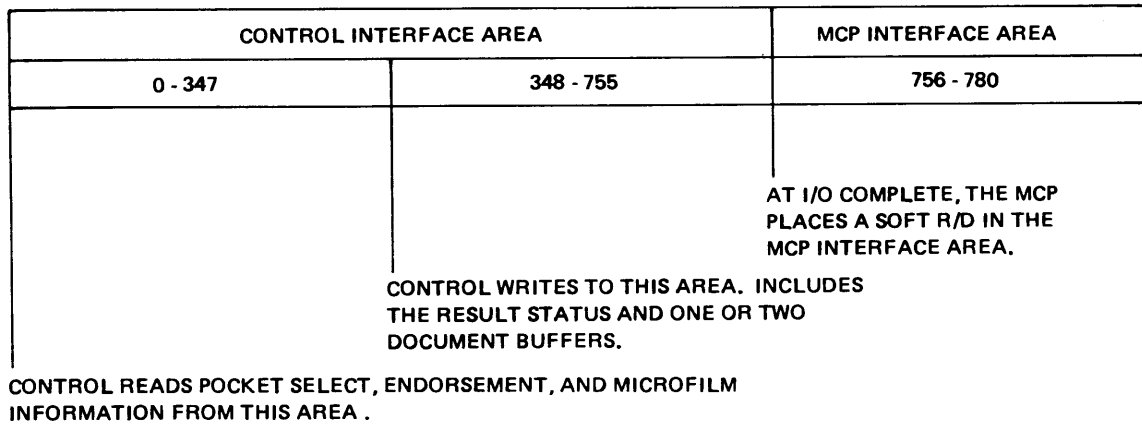
The 4A Control is different from most other device controls in that it reads from the user program and writes to the user program during the same I/O operation. The Control Interface area is divided into two areas. The Control reads from the first area at locations 0 – 347; then during the same I/O

B 2000/B 3000/B 4000 MCPVI Programmer's Guide  
4A Control Application Program Interface

operation, writes into the second area at locations 348 – 755. As an example, consider N to be the current document passing before the Sorter read station. The Control reads the Pocket Select, Microfilm, and Endorser information for the previous document N – 1, writes the Result Status containing information for both documents N and N + 1, and writes the current document, N.

When the I/O operation is complete, the MCP places a Soft Result Descriptor in the MCP Interface area.

Figure 10-1 describes the functions of the Memory Map areas for a Pocket Select/Read BCT.



P5496

**Figure 10-1. Memory Map Functions**

NOTE

Construct the user program so as to zero out all reserved fields in the Memory Map before Start Flow begins.

The Memory Map is described in Table 10-1.

**Table 10-1. Memory Map**

Position	Size	Description
0	780D	Memory Map.
0	756D	Control Interface area.
0	64D	Reserved.
64	272D	Endorser Bands (Note 1).
64	3D	Reserved.
67	1D	Band Identifier number for field 4. Must be a value of 1,2,3 or 4 if this band is being loaded.
68	64D	Endorsement Text for field 4. 32 bytes of alphanumeric endorsement text are provided.

B 2000/B 3000/B 4000 MCPVI Programmer's Guide  
4A Control Application Program Interface

**Table 10-1. Memory Map (Cont)**

Position	Size	Description
132	3D	Reserved.
135	1D	Band Identifier number for field 3. Must be a value of 1,2,3 or 4 if this band is being loaded.
136	64D	Endorsement Text for field 3. 32 bytes of alphanumeric endorsement text are provided.
200	3D	Reserved.
203	1D	Band Identifier number for field 2. Must be a value of 1,2,3 or 4 if this band is being loaded.
204	64D	Endorsement Text for field 2. 32 bytes of alphanumeric endorsement text are provided.
268	3D	Reserved.
271	1D	Band identifier number for field 1. Must be a value of 1,2,3 or 4 if this band is being loaded.
272	64D	Endorsement text for field 1. 32 bytes of alphanumeric endorsement text are provided.
336	4D	All zero. Indicates to Control that it is at the end of endorsement text.
340	2D	Pocket Select number. The pocket number to which the item is to be sent. If rejecting a document this number must be greater than 47. Used for Pocket Select/Read BCT (OP = 29).
342	1D	Endorser band control. 8 bit = 1 Spray Endorser Band #1. 4 bit = 1 Spray Endorser Band #2. 2 bit = 1 Spray Endorser Band #3. 1 bit = 1 Spray Endorser Band #4.
343	1D	Camera/Stop Flow control. 8 bit = 1 microfilm this document.

**Table 10-1. Memory Map (Cont)**

<b>Position</b>	<b>Size</b>	<b>Description</b>
		4 bit Reserved.
		2 bit Reserved.
		1 bit = 1 Stop Flow.
344	4D	Reserved.
348	8D	Result Status.
348	1D	8 bit = 1 Station A buffer overflow.
		4 bit = 1 Station A Parity Error from Sorter.
		2 bit = 1 Station A Can't Read.
		1 bit = 1 Station A Too Late To Read.
349	1D	8 bit Reserved.
		4 bit = 1 Pocket Select Error (B 9138 only).
		2 bit Reserved.
		1 bit Reserved.
350	1D	8 bit = 1 Station B Buffer Overflow. Same as for station A.
		4 bit = 1 Station B Parity Error from Sorter. Same as for station A.
		2 bit = 1 Station B Can't Read. Same as for station A.
		1 bit = 1 Station B Too Late To Read. Same as for station A.
351	1D	8 bit Reserved.
		4 bit Reserved.
		2 bit Reserved.
		1 bit Reserved.
352	1D	8 bit = 1 Flow Stopped.
		4 bit = 1 Not Ready.
		2 bit = 1 Black Band.
		1 bit = 1 Failsoft Error.
353	1D	8 bit = 1 Real Time Too Late (physically too late to pocket, endorse, or microfilm).



**Table 10-1. Memory Map (Cont)**

<b>Position</b>	<b>Size</b>	<b>Description</b>
		4 bit = 1 Double Document (Note 2).
		2 bit = 1 Overlength Document (Note 2).
		1 bit = 1 Underspaced Document (Note 2).
354	1D	8 bit = 1 Missort.
		4 bit = 1 Feeder Jam (no feed) (Note 3).
		2 bit = 1 Jam (Note 3).
		1 bit = 1 Film Advance.
355	1D	8 bit = 1 Reserved.
		4 bit = 1 Post Read Document Error (mechanical).
		2 bit = 1 Post Read Document Error (electrical).
		1 bit = 1 Parity Error from control.
356	2D	Fault location for jam or missort. 2 digit number specifies beam-of-light or pocket location.
358	18D	Microfilm ID number(9 bytes). The last film ID number used by the Sorter. This field is not meaningful unless the camera is powered on.
356	200D	Station A 100-byte document buffer.
356	2D	Total document length counter. This field gives the total length of the document read at station A. The length includes ETX characters that were inserted by the Control.
		Total document length is given in bytes.
358	2D	Address pointer to first character. This field gives the offset, in digits, from the beginning of the document buffer to the beginning of text. This pointer is always 04.

**Table 10-1. Memory Map (Cont)**

<b>Position</b>	<b>Size</b>	<b>Description</b>
360	196D	Station A buffer. Control inserts one "ETX" character at the beginning and one "ETX" at the end of the document. If the document is an odd number of bytes, another "ETX" will be entered at the beginning of the document. Unless the "report feed error" option is set (B 9138 utility mode) a B 9138 feed check does not cause the Sorter to stop flow. The program is notified of a feed check by a single Can't Read character item (Station A Total Document buffer = 040403033F03).
556	200D	Station B 100 byte document buffer. The format for station B is the same as for station A (Note 4).
756	24D	MCP Interface area.
756	1D	Number of endorser text bands to be loaded. This number must be 0,1,2,3 or 4. The 4A Control is capable of accepting from zero through four bands of endorsement text on each Pocket Select operation. (See the Non-Impact endorsement section of this document).
757	1D	Status. The MCP places the status of the Sorter in this area in response to a Status BCT. Formatting is as follows: 8 bit = 1 Slewing Microfilm. 4 bit = 1 Camera Not Ready. = 0 Camera ready, not present, or not powered on. 2 bit = 1 Endorser Not Ready. = 0 Endorser ready, not present, or not powered on. 1 bit = 1 Sorter Not Ready.
758	2D	Characteristics. The MCP places the characteristics of the Sorter in

**Table 10-1. Memory Map (Cont)**

<b>Position</b>	<b>Size</b>	<b>Description</b>
		this area in response to a Characteristics BCT. Formatting is as follows:
758	1D	8 bit = 1 Endorser Band one present. 4 bit = 1 Endorser Band two present. 2 bit = 1 Endorser Band three present. 1 bit = 1 Endorser Band four present.
759	1D	8 bit = 1 Sorter is a B 9137. = 0 Sorter is a B 9138. 4 bit = 1 Camera present. 2 bit Reserved. 1 bit = 1 Read station B present.
760	4D	Reserved.
764	4D	MCP Soft Result Descriptor.
764	1D	8 bit = 1 I/O invalid to the Control. 4 bit = 1 BCT invalid to the MCP. 2 bit = 1 Flow condition error. 1 bit = 1 System interface parity error encountered on I/O initiate.
765	1D	8 bit = 1 Microfilm operation not completed. 4 bit = 1 Non-present option required. 2 bit = 1 Failsoft error encountered on control data. 1 bit = 1 Internal control Data Ram Parity error on Read Station A.
766	1D	8 bit = 1 Internal control Data Ram parity error on Read Station B. 4 bit = 1 Memory Parity error.

**Table 10-1. Memory Map (Cont)**

<b>Position</b>	<b>Size</b>	<b>Description</b>
		2 bit = 1 Failsoft error on Control to memory transfer.
		1 bit = 1 Parity error detected on Sorter to control transfer.
767	1D	8 bit = 1 Power failure.
		4 bit = 1 B – address exceeded.
		2 bit Reserved.
		1 bit Reserved.
768	4D	Pocket light/generate image count mark parameters (NNRB).
768	2D	If B=0, NN is the pocket number for which the light will be illuminated. If B=1, NN equals the number of ICM that will be generated.
770	1D	Reserved.
771	1D	B = 0 Pocket light illumination. On the B 9138, if the cutslips/pocket light option is set on (cutslips), a cutslip will be fed from the secondary hopper and pocketed to pocket NN. The document will not be read. B = 1 ICM operation.
772	4D	Start Flow parameters (RHFF).
772	1D	Reserved.
773	1D	H = 1 Data in the low order nine positions of endorser band 1 is microfilm header data. H = 0 No microfilm header data present.

**Table 10-1. Memory Map (Cont)**

<b>Position</b>	<b>Size</b>	<b>Description</b>
774	1D	F = 1 Read data from Read Station A.
775	1D	F = 1 Read data from Read Station B.
776	4D	Demand Read parameters (RHFF). Each digit is used in the same manner as the corresponding digit in the Start Flow parameters. Separate areas are used for Start Flow and Demand Read parameters in order to simplify the programming effort required to use a mixture of the two techniques.

NOTES

1. See Non-Impact endorsement section for further explanation of endorser fields.
2. On the B 9138, double, overlength or underspaced bits will not be reported unless the Report Feed Error option is set (B 9138 Utility mode). If not set, a feed error will be reported in the document buffer as a single Can't Read item. It is recommended that this option not be set. A B 9137 will report this condition continuously until flow stops. A B 9138 with the Report Feed Error option set will report this condition for the affected document only and will not stop flow.
3. For both feeder jams and non-feeder jams, the Control supplies the last microfilm identification number after flow is stopped.
4. The document buffers are variable in format. If the MICR image for station A is less than 96 bytes, the document for station B is left justified against the document buffer for station A.

## MCP INTERFACE (MICR 4A CONTROL BCT)

All MICR BCTs conform to the following general format:

BCT 374 (absolute memory address 374)  
BUN around (next instruction)  
P1 = ACON FIB  
P2 = NN  
P3 = ACON error  
P4 = ACON flow stopped

### NOTES

P1 is a pointer to the user program Sorter file FIB.

P2 is a 2-digit number which uniquely identifies the MICR BCT type (refer to Verification). For BCT Type numbers and their operations, see Table 10-2.

P3 is not included in all MICR BCTs. It is a pointer to the error label in case the BCT does not pass the standard MICR BCT verification (refer to Verification).

P4 is not included in all MICR BCT. It is a pointer to the flow stop action label.

**Table 10-2. BCT Type Numbers and Operations**

BCT Type	Operation
42	Start Flow
43	Demand Feed and Read
44	Pocket Light/Generate ICM
45	Microfilm Slew
47	Status
48	Characteristics
46	Logical Read
29	Pocket Select/Read

A detailed description of each BCT follows. Comments on Open and Close BCTs for a 4A Control Sorter file are also included.

There is a BPL syntax for the 4A Control BCT (see B 2000/B 3000/B 4000 Series BPL Reference Manual, form no. 1113735).

## STANDARD NON-POCKET SELECT/READ BCT VERIFICATION

With the exception of the Pocket Select/Read BCT which is verified in a separate routine, the Sorter BCTs are verified by the MCP in the following manner:

1. P2 must be a valid MICR BCT type. If it is not, the result is a DS or DP condition with the error message INV BCT PARAM displayed.
2. The open Sorter file table is searched by MIX # and FIB address. If an entry is not found, this results in a DS or DP condition with INV (F-N) (ADR) FILE RSTRCTD OR NOT OPEN error message displayed.
3. P3 must be non-zero, contain no undigits, be within program base/limit, and be mod 2. If not, the result is a DS or DP condition with INV BCT PARAM error message displayed.

4. Verify that the flow condition is proper. With the Sorter in a flow condition, Logical Read is the only valid BCT. If not in a flow condition, the MCP causes the Soft R/D bit to be set and reinstates the user program at the error label associated with the BCT.

## STANDARD POCKET SELECT/READ BCT VERIFICATION

P2 must be Pocket Select/Read BCT type. If not, the MCP marks the file as if a Stop Flow condition had occurred, sets the Soft R/D bit, and permits the processing portion of the user program to run in order to complete the handling of any tanked items. The user program is then reinstated at the Flow Stopped label for recognition of the error condition.

The number of endorser text bands must be 0, 1, 2, 3, or 4. Otherwise, the error condition is handled as described in 1.

## OPEN BCT

When opening a 4A-type Sorter file, the external file identification (device name) value in the label area is compared to the device names of the unassigned 4A-type units. When a match is found, the unit is assigned to the program by the MCP. If a match is not found, a NO FILE message is displayed by the MCP and an IL is required. For Assembler-coded programs prior to issuing the Open BCT, the user program must set the 8-bit in FIBST1 to declare use of this 4A interface and require selection of a B 9137/B 9138 Sorter.

## CLOSE BCT

A 4A-type Sorter file must be closed with Release.

## START FLOW BCT

The Start Flow Read descriptor initiates Flow Feed. Just prior to Flow Feed, the Control reads the band information from system memory. As an option, the user program places a microfilm header on the microfilm. While in Flow mode, only the Pocket Select/Read and Logical Read BCT are valid.

The format of the Start Flow BCT is:

```
BCT 374
BUN around
P1 = ACON FIB
P2 = 42
P3 = ACON error
P4 = ACON flow stopped
```

### NOTE

Flow must be stopped prior to issuing a Start Flow.

The program is reinstated at one of two locations as follows:

1. Once flow is physically started, the program is reinstated at the instruction following the Start Flow BCT (BUN around). Since Start Flow is not treated as a Logical Read, the user program must proceed to a Logical Read BCT when it has been reinstated.
2. If flow cannot be successfully started because of an error condition, the user program is reinstated at the error label as specified in the Start Flow BCT (P3). If the Soft Result Descriptor shows a system interface parity error on I/O initiate or Failsafe error on control data, do

not attempt further I/O operations to the Control through the user program but cause the program to proceed to a file CLOSE. If the Non-Present Option Required bit is on, the program will re-evaluate its requirements and the Reader/Sorter options through the characteristics and status request. When the discrepancy is rectified, another Start Flow can be attempted.

3. If flow cannot be started because of a feeder jam, the user program will be reinstated at flow stop. If the I/O invalid, BCT invalid, or flow condition error bits are on, the program probably contains a logic error and will proceed to a file close.

When a Start Flow BCT is issued, the MCP:

1. Performs the standard BCT verification.
2. Constructs I/O descriptor which is composed of OP Code, Start Flow parameters, and A & B address.
3. Initializes tanking counter.
4. Sets up Soft R/D and reinstate program at exception or no exception address.

Prior to issuing the Start Flow BCT, the user program must:

1. Set RHFF (location 772, MCP Interface area).
2. Move the microfilm header data into the last nine positions of the endorsement text for endorser band 1, if microfilm data is to be loaded. For more information on Microfilm Header data see "Microfilming" in this section.

## DEMAND FEED AND READ BCT

The Demand Feed and Read BCT results in one document per BCT being fed to a read station and read. A Pocket Select/Read BCT must follow each read completion.

Use of Demand Read is not recommended since it causes excessive wear on the mechanical portions of the Sorter.

The format of the Demand Feed and Read BCT is as follows:

BCT 374  
BUN around  
P1 = ACON FIB  
P2 = 43  
P3 = ACON error

Flow must be stopped prior to issuing a Demand Feed and Read. The Demand Feed and Read BCT can be issued even though the Sorter has been previously opened flow.

The user program is reinstated at one of two locations as follows:

1. When an item has been fed, read, stored, and pocket selected by the Pocket Select user routine, the program is reinstated at the instruction following the Demand Read BCT (BUN around). The program will not proceed to a Logical Read BCT because the Demand Feed and Read BCT implies a logical read. After all processing is completed for that document, the program proceeds to any other valid BCT as required.
2. If it is impossible to successfully feed an item, the program is reinstated at the error label as specified in the Demand Read BCT (P3). The error conditions are handled as described under Start Flow.



Write the user program so as to set up RHFF for the Demand Read BCT.

## POCKET LIGHT/GENERATE IMAGE COUNT MARKS BCT

This BCT illuminates the specified pocket light or generates Image Count Marks on the microfilm, depending on the parameter settings.

The format of the BCT is as follows:

BCT 374  
BUN around  
P1 = ACON FIB  
P2 = 44  
P3 = ACON error

### NOTE

Flow must be stopped prior to issuing this BCT.

The program is reinstated at one of two locations:

1. At the instruction following the BCT (BUN around) when the operation is complete and there are no exceptions.
2. At the error label with the appropriate Soft R/D bits set if there are any exceptions.

When this BCT is given, the MCP:

1. Performs standard BCT verification.
2. Constructs an I/O descriptor which is composed of OP code and pocket light/ICM parameters.
3. Sets up Soft R/D and reinstates the program at exception or no exception address.

Prior to issuing this BCT, the user must set the Pocket Light/Generate Image Count Mark parameters (NNRB) in the MCP Interface area as required.

Write the user program so that ICM BCTs will be issued one ICM at a time. In this way, if the operation is not completed, the user program can determine how many ICMs were successfully placed on the microfilm.

With B 9138 operation, there is a cutslips/pocket light option. If this option is set to cutslips, a cutslip will be sent from the secondary feeder to the specified pocket instead of lighting the indicated pocket light. (See the discussion of the B 9138 Merge on Text Option in this section.)

## MICROFILM SLEW BCT

This BCT is issued when it is desired to advance the microfilm to the beginning of the next 100-or 200-foot reel within a cassette. There are no parameters associated with this BCT.

The format of the BCT is as follows:

BCT 374  
BUN around  
P1 = ACON FIB  
P2 = 45  
P3 = ACON error

Flow must be stopped prior to issuing this BCT.

The program is reinstated at one of two locations as follows:

1. At the instruction following the BCT (BUN around) when the operation is complete and there are no exceptions. This BCT goes I/O Complete when the slew is successfully started. It is the user program's responsibility to recognize any subsequent malfunction.
2. Any exception results in the program being reinstated at the error label with appropriate Soft R/D bits set.

When this BCT is given, the MCP:

1. Performs a standard BCT verification.
2. Constructs an I/O descriptor which is composed of an OP code only.
3. Sets up a Soft R/D and reinstates the program at exception or no exception address.

For more information on microfilming see Microfilming in this section.

## STATUS BCT

If the user program requires information regarding the status of the Sorter, it issues this status BCT. This results in the status of the Sorter being placed in the MCP Interface area. There are no parameters associated with this BCT.

The format of this BCT is as follows:

BCT 374  
BUN around  
P1 = ACON FIB  
P2 = 47  
P3 = ACON error

### NOTE

Flow must be stopped prior to issuing this BCT.

The program is reinstated at one of two locations:

1. At the instruction following the BCT (BUN around) when the operation is complete and there are no exceptions.
2. At the error label with the appropriate Soft R/D bits set when there are any exceptions.

When this BCT is given, the MCP:

1. Performs standard BCT verification.
2. Determines status of the Sorter and sets up MCP interface area.
3. Sets up Soft R/D and reinstates program at exception or no exception address.

## CHARACTERISTICS BCT

If the program requires information regarding the characteristics of the Sorter, it issues this BCT. This results in the characteristics of the Sorter being placed in the MCP Interface area associated with this BCT.

The format of this BCT is as follows:

BCT 374  
BUN around  
P1 = ACON FIB  
P2 = 48  
P3 = ACON error

NOTE

Flow must be stopped prior to issuing this BCT.

The program is reinstated at one of two locations:

1. At the instruction following the BCT (BUN around) when the operation is complete and there are no exceptions.
2. At the error label with the appropriate Soft R/D bits set when there are any exceptions.

When this BCT is given, the MCP

1. Performs standard BCT verification.
2. Determines characteristics of the Sorter and sets up MCP interface area.
3. Sets up Soft R/D and reinstates program at exception or no exception address.

## LOGICAL READ BCT

The Logical Read BCT does not result in the initiation of physical I/O operation. It merely serves as the synchronization mechanism between the Pocket Select user routine and the processing portion of the program.

The format of the Logical Read BCT is as follows:

BCT 374  
BUN around  
P1 = ACON FIB  
P2 = 46  
P3 = ACON error  
P4 = ACON flow stopped

NOTE

The Logical Read BCT is valid only if flow is in process.

The program is reinstated at one of three locations:

1. At the instruction following the Logical Read BCT (BUN around) when there is an item to process.
2. At the error label if the standard BCT verification is not passed.
3. At the flow stopped label when a Stop Flow condition occurs and all items have been processed.

When a Logical Read BCT is given, the MCP:

1. Performs standard BCT verification.
2. Controls synchronization.

When a Logical Read BCT is issued and there is an outstanding pocket selected item (storage tank not empty), the program is reinstated at the instruction following the Logical Read BCT (BUN around). If there is not an outstanding pocket selected item (storage tank empty) and flow has not stopped, the program is marked waiting IOC and is not reinstated until the next Pocket Select is issued.

Processing continues as outlined above until flow stops. The program continues to be reinstated at the instruction following the Logical Read BCT until all pocket selected items have been logically read (storage tank empty). At the next Logical Read BCT (that is, the N + 1 Logical Read after pocket selecting N items) the program is reinstated at the flow stopped label.

During the processing portion of the program, the user must untank the items along with the Result Status and Soft R/D and determine the necessary action to be taken on that item based on these two result indicators. The program returns to a Logical Read BCT when it has completed the processing of that item.

Note that certain conditions, such as Can't Read, are reported in the Result Status and the Soft R/D which are available at the time the data is loaded into memory. There are other conditions, such as "real time too late to Pocket Select", which are reported with the following I/O Complete (Control State User routine or flow stopped). For that reason it may be beneficial to postpone the processing of each document until the result data reported with the following I/O Complete is available.

When the user program is reinstated at the flow stopped label, it must recognize and handle any exception conditions and determine the cause of the Stop Flow so that appropriate action can be taken based on the information reported in the Result Status and Soft Result Descriptor.

## POCKET SELECT/READ BCT

The Pocket Select/Read BCT is the only acceptable exit from the Control State User routine and is the only acceptable hardware operation when flow is in progress on the Sorter, or after a Demand Read BCT has been given.

The Control State User routine is reinstated whenever the character recognition data for an item is transferred from the Control to memory (I/O Complete). The Result Status and Soft Result Descriptor for the last document read is also available to the user routine at this time.

A Pocket Select/Read BCT must be issued by the Control State User routine for each item. This BCT implies that the user routine is ready to read and pocket select the next item as soon as the Control makes it available (I/O complete).

The format of the Pocket Select/Read BCT is:

```
BCT 374  
BUN around  
P1 = ACON FIB  
P2 = 29
```

The Control State User routine is reinstated at one of three locations depending upon exceptions encountered. The three locations, in priority of reinstatement, are

1. Soft R/D attention. There may also be Result Status conditions.
2. Result Status attention.
3. No exceptions.

B 2000/B 3000/B 4000 MCPVI Programmer's Guide  
4A Control Application Program Interface

The Assembler syntax for the USER construct which specifies the locations for reinstating the Pocket Select user routine is as follows:

1. A label = Result Status attention.
2. B label = Soft R/D attention.
3. D label (remarks column 58) = no exceptions.

The MCP performs the following functions at Read I/O Complete:

1. If Flow mode, sets flow in process.
2. Sets up the Soft R/D.
3. Checks for two fatal error types:
  - a. System interface (initiate) parity error. Prohibits Control from knowing what the processor requested. Did not recognize OP code.
  - b. Failsoft error (control data). Prohibits Control from knowing what the processor requested. Recognized OP Code but did not recognize control information.

If the exception exists and it is the first time, the previous I/O (Start Flow, Demand Read, Pocket Select) is reinitiated in the event that the condition is transient.

If the exception exists and it is the second time (on the reinitiated I/O) and it is

- 1) Start Flow or Demand Read. The program is reinstated at the error label associated with the BCT.
  - 2) Pocket Select. The program is reinstated at the flow stopped label after all tanked items are processed.
4. Checks for flow stopped condition. If so, reinstate program at flow stopped label after all tanked items are processed and reset flow in process.
  5. Reinstates Control State User routine at appropriate label.

When the Control State User routine issues the Pocket Select/Read BCT, the MCP:

1. Marks the user program suspended if a timeout has occurred.
2. Updates tanking counter for synchronization.
3. Performs Pocket Select/Read BCT verification.
4. Constructs an I/O descriptor which is composed of an OP code, and an A address based on the number of bands to be loaded.
5. If the Pocket Select I/O is invalid, sets the Soft R/D and marks the file as if a Stop Flow condition had occurred. Also:
  - a. In Flow mode, the processing portion of the user program is allowed to run in order to complete the handling of any tanked items. The user program is then reinstated at the flow stopped label for recognition of the error condition.
  - b. In Demand mode, the user program is reinstated as if the Pocket Select has completed so that the error condition can be recognized.
6. Makes the program ready to run if in Flow mode. If Demand mode, marks the program waiting IOC (pocket select).

B 2000/B 3000/B 4000 MCPVI Programmer's Guide  
4A Control Application Program Interface

In Flow mode, because of the variations in document size and processing requirements, it is possible for the processing portion of the user program to fall behind the Sorter and the Control State User routine. Because of the same variations, it is also possible for the processing portion to catch up. In order to provide for this situation, it is recommended that the user program include a storage tank large enough to store the maximum number of items which the program is expected to fall behind, plus the items which have been fed but not read at the time a Stop Flow is requested.

The Control State User routine places each item image and result information in the tank during Pocket Select processing. It also retrieves each image and result information from the appropriate position during the processing portion of the program. In this manner the Control State User routine can determine when this storage tank is near capacity so that a Stop Flow can be given to avoid a tank overflow.

The Control State User routine must:

1. Capture all pertinent information required. The Pocket Select/Read BCT implies that the character recognition data and result information areas are available for the read complete information transfer of the next item.
2. Load the desired pocket number in the Control Interface area.
3. Set the desired endorser control bits.
4. Load the desired text bands and band number if new texts are required.
5. Set the number of text bands to be loaded.
6. Set the camera control bit.
7. Set the Stop Flow bit if required.
8. Recognize and handle the exception conditions as reported in the Soft R/D.
9. Recognize and handle the exception conditions as reported in the Result Status.
10. Issue Pocket Select/Read BCT.

## **B 9138 MERGE ON TEXT OPTION**

The Merge Feed On Text feature allows documents to be fed from the secondary feeder under user program control. The merge items are Merge Feed On Demand from the secondary feeder. Merge Feed works as follows:

The B 9138 recognizes the merge request and the Sorter momentarily stops feed from the primary feeder.

When the last item from the primary feeder clears the secondary feeder transport entry point, demand feeds for the current number of merge requests are issued.

As soon as the last Merge Feed item has entered the transport, the Sorter will reinitiate flow from the primary feeder.

If a merged item jams prior to transmission to the program, it is the user program's responsibility to reissue the request. If flow is stopped for any reason before a merge can be performed, then the Sorter will issue the appropriate number of secondary loader demand feeds at the next Start Flow.

The Merge Feed On Text option must be enabled in the Utility mode of the B 9138

The merge request is recognized by interpreting the tenth from the last byte in the endorser band 1 text of the 4A Control. (The last nine bytes of this band are reserved as microfilmer ID numbers.) If the byte contains any character except a blank, the Merge Feed request is recognized. If the byte contains a blank, then the Merge Feed is not recognized.

After a Merge Feed request is recognized, the byte text must be replaced by a blank. If the option is not enabled simultaneous with endorsing, the character represented by this byte will be sprayed by the endorser.

### **NOTES**

1. There is no prohibition of the number or frequency of Merge Feed On Text requests.
2. Merge Feed requests can be made on the disposition of previously fed merge items.
3. Because the band text character data substitution function is performed when there is a parity problem on transmission of band text from the Control to the Sorter, it is possible for a Merge Feed On Text operation to be performed when the user program is not expecting one. Write the user program so that it will be able to handle unexpected items from the secondary feeder.
4. Use this option with care. It is possible to feed one item from the primary feeder, forget to "blank out" the merge field byte, and continue to feed only from the secondary feeder.
5. If the Endorser option is not present the user can still load Band 1 data for Merge Feeds only, but it is the user's responsibility not to send the endorse spray bits set with the Band Load/Pocket Select command. Currently, the 4A Control sets the Non-Present Option bit in the Result Status when loading takes place without the endorser or the camera present.
6. If the Microfilmer option is not present the user can still load Band 1 data for Merge Feeds only, but it is the user's responsibility not to send the microfilm click bit set for microfilming along with the Band Load/Pocket Select command. Currently, the 4A Control sets the Non-Present option bit in the Result Status when loading takes place without the endorser or the camera present.

7. Transmitting only band text during any Pocket Select operation with the 4A Control is a legitimate operation.
8. This method of feeding cutslips from the secondary hopper is preferred over the Pocket Light method since it allows the cutslip to be read.

## CONTROL DELIMITERS

Valid control delimiters for the 4A Control are as follows:

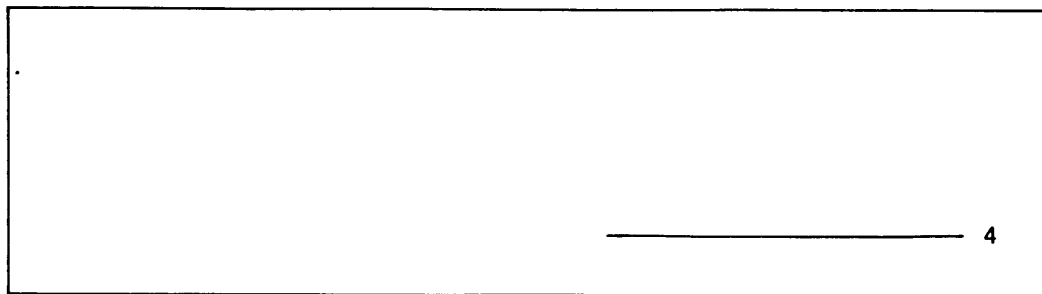
### MICR Font E13B

Symbol	4A Delimiter	EBCDIC
Transit	;	5E
Amount	:	7A
On-us	<	4C
Dash	-	7E
Can't Read	SUB	3F
Parity Error	DLE	10
Document Delimiter	ETX	03

## NON-IMPACT ENDORSEMENT

In the Control Interface area (locations 64-339) there are four fields: one for each ink jet on the endorser. These fields are used to load text into the endorser. Each field is composed of one word, with the low-order digit specifying the ink jet number, and 32 bytes of endorsement text. The endorsement text is called a band and the ink jet number is called the band identification number. In the following diagrams, each line represents a band. The number to the right of each line is the band identification number for that band.

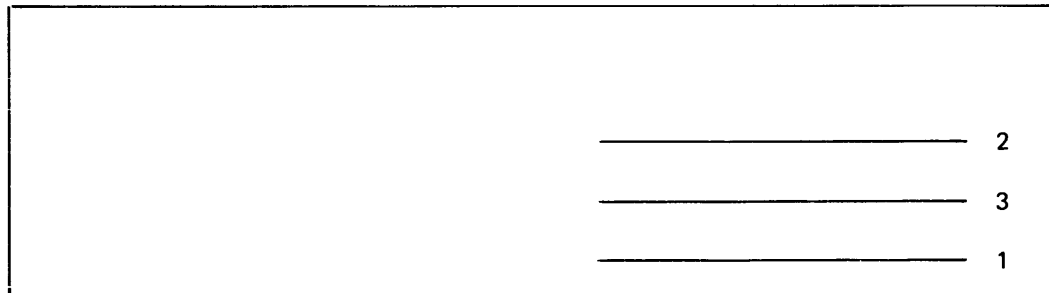
The single jet endorser (usually used on the front of a document) is represented in the following diagram.



P5494



The three jet endorser (usually used on the back of a document) is represented in the following diagram.



P5495

Bands 2, 3 and 4 are each composed of 32 bytes of user-defined alphanumeric text. Band 1 has 23 bytes of user-defined alphanumeric text. The rightmost byte can be used as the Merge On Text recognition character (see B 9138 Merge On Text option). The final nine bytes in Band 1 contain the Document ID number. This 9-byte number is programmatically assigned. If microfilming, this number is placed on the film with each document picture.

A band and band ID number can be loaded into any one of the four fields in the Control Interface area. Also, any number of bands, zero through four, can be loaded into the endorser. However, it is beneficial to include only those text bands which are different from those currently stored in the endorser band memory. Elimination of these repeated bands lessens the data transfer time during the time critical Pocket Select process. Place the endorsement texts to be changed in the low order text locations. For example, if only a single band text is to be changed, place it in buffer location 272 with the appropriate memory band number in buffer location 271. If two texts are to be changed, place them in buffer locations 204 and 272 with corresponding memory band numbers in buffer locations 203 and 271.

Digit 756 in the MCP Interface area specifies the number of bands to be loaded into the endorser. This number can be zero through four.

Digit 342 in the Control Interface area specifies which ink jets to fire. Digits 336 – 339 in the Control Interface area tell the Control that it is at the end of endorsement text. These digits should always be zero.

## NOTES ON RESULT STATUS

Error conditions identified in the Result Status (in the Control Interface area) are reported after an I/O Complete for the affected document, for the subsequent document, or after Stop Flow.

The MCP Soft Result Descriptor (located in the MCP Interface area) should be interrogated before the Result Status is interrogated.

B 2000/B 3000/B 4000 MCPVI Programmer's Guide  
4A Control Application Program Interface

Result Status bits are reported as given in Table 10-3.

**Table 10-3. Reporting Result Status Bits**

Description	Affected Document	Subsequent Document	Stop Flow
Buffer overflow	X		
Parity error from Sorter	X		
Can't read	X		
Too late to read		X	X
B 9138 Pocket Select error		X	X
Flow stopped			X
Not ready	X	X	
Black Band item (Flow Stopped bit also set)			X
Failsoft error		X	X
Real time too late to endorse, film, or pocket		X	X
Double document	X		
Overlength document	X		
Underspaced document	X		
Missort (Not Ready bit also set)		X	X
Feeder Jam (Stop Flow & Not Ready bits also set)			X
Jam (Stop Flow and Not Ready bits also set)			X
Film Advance			X
Post Read document error (Mech) (Double, Overlength, or Underspaced Document bits may also be set)	X		
Post Read document error (Electrical)		X	X
Parity error from Control (Too Late To Endorse, Film or Pocket bit may be set if unsuccessful retry took place and a parity error occurred in control information)		X	X

Process Sorter exceptions in the following order. On a B 9137, these errors can occur simultaneously. If multiple error conditions occur, treat the highest priority condition and ignore the others. This method of processing Sorter exceptions is not necessary for B 9138 Sorters.

1. Post read document error (Electrical)
2. Missort
3. Jam
4. Too late to endorse, film or pocket
5. Post read document error (Mechanical)
6. Feed Check
7. All others

## MICROFILMING

The camera module on the Sorter is a precise optical/mechanical instrument that records images on microfilm at high speed and to a high level of resolution. The document transport mechanism is connected to the film transport mechanism. The film moves during exposure so there is no relative movement between the film and the image of the document. The film is started when the leading edge of the document enters the optical gate of the microfilmer, and is stopped when the trailing edge leaves the gate.

A 9-character number is photographed along with the document image. This is the same as the number specified in the low 9 bytes of endorser band 1. An image count mark (ICM) is also included with each document image. The ICMs serve as locators on the film. Scanning functions of the various microfilm viewing devices "key off" these ICM.

The Sorter camera uses Duo-Duplex film. Both the front and the back of the document are photographed simultaneously using only one half of the film width. After the first half of the film is exposed, the film cassette is flipped and reinserted into the camera and the second half of the film is exposed. Either thin base or thick base film can be used. Thin base film allows loading of longer film lengths into the camera and is standard. The use of thick base film requires a minor wiring change. Film is supplied in either disposable or reusable cassettes having a nominal capacity of 500 feet for thick base film and 1000 feet for thin base film. Additional capacity is provided for footage required for leaders and film advances.

The user program specifies whether a document is to be microfilmed in the Control State User routine. If the document is to be filmed the 8-bit of the Camera/Stop Flow control digit (location 343) in the Control Interface area is set to 1. Otherwise, the bit is set to zero.

The microfilmer automatically causes interruption of document feed and microfilm operations at intervals of 100 or 200 feet (depending on film being used). This permits a film advance of approximately 3 feet to provide a clear band of film. The clear section permits cutting of the film without losing any of the filmed images. The film advance works as follows:

1. A signal is sent from the microfilmer to the Sorter indicating that the microfilmer is at the end of the 100-or 200-foot section of film. The Sorter stops feeding documents but continues reading, transporting and pocket selecting documents still in the feed mechanism. The microfilmer, in turn, microfilms these documents.
2. When all documents in the transport are pocket selected, the microfilmer goes not ready and advances three feet of film. These three feet of film are not counted on the film exposed indicator.
3. When the film advance is completed, the microfilmer is again ready to microfilm documents under Sorter control, and the Sorter starts flow.

B 2000/B 3000/B 4000 MCPVI Programmer's Guide  
4A Control Application Program Interface

The film advance procedure takes approximately four seconds and is reported in the Result Status.

It is the responsibility of the user program to keep track of how many 100-or 200-foot sections of unexposed film are left on a cassette side and which side of the cassette is being filmed.

A 9-character microfilm header number can be recorded on the film to identify the film section for subsequent storage and retrieval purposes. Document and ICM imaging are inhibited during this sequence. The microfilm header number is placed in the position normally occupied by the document identification number.

A microfilm header number is placed on the microfilm under the following conditions:

1. At the beginning of the first 100-or 200-foot section of film, after the camera has been set up and the film has been advanced three feet.
2. At the beginning of any other 100-or 200-foot usable section of film.
3. Within a section of film, between documents.
4. Within a section of film, after a film slew. In this case, film has been slewed to a point within a 100-or 200-foot section and the header is to be entered before another batch of documents is microfilmed.
5. At the end of a side of film.

In cases 2 and 3 the Sorter is in Flow mode and must be stopped before the header number can be entered.

When flow is stopped, the microfilm header number is placed in the low order 9 bytes of Endorsement Band 1; Digit 773 in the MCP Interface area (Start Flow parameters) is set to 1, and flow is started.

Film can be slewed programmatically to the beginning of the next 100-or 200-foot section of film using the Microfilm Slew BCT. Flow must be stopped prior to issuing this BCT. The Status BCT is used to confirm that a microfilm slew is in progress. After the Status BCT is issued, the 8 bit of the second digit in the MCP Interface area (location 757) is turned on if the slew is in progress.

## APPENDIX A

### EXAMPLE PROGRAM AND MEMORY DUMP

Figure A-1 shows a program which was executed to create the memory dump shown in figure A-2. The circled items in figure A-2 are further explained in notes which appear following the memory dump. Some of these circled items are also referred to in section 7.

```

000110 IDENTIFICATION DIVISION.
000120 PROGRAM-ID. DUMMY PROGRAM.
000130 * THIS PROGRAM PROGRAM IS GUARANTEED TO FAIL
000140 ENVIRONMENT DIVISION.
000150 CONFIGURATION SECTION.
000160 SOURCE-COMPUTER. B4700.
000170 OBJECT-COMPUTER. B2700.
000180 INPUT-OUTPUT SECTION.
000190 FILE-CONTROL.
000200     SELECT FILE001 ASSIGN TO TAPE.
000210     SELECT FILE002 ASSIGN TO DISKPACK.
000220     SELECT FILE003 ASSIGN TO PRINTER.
000230 DATA DIVISION.
000240 FILE SECTION.
000250 FD FILE001 VA OF ID "TAPEIN".
000260 01 FILE001-REC.
000270     CS TABLE-DATA PIC XX OCCURS 50 TIMES.
000280 FD FILE002 FILE CONTAINS 20 BY 100
000290     OBJECT CONTAINS 2 RECORDS
000300     VA OF ID "DISKIN".
000310 01 FILE002-REC.
000320     CS FILE002-DATA PIC X(50).
000330 FD FILE003 VA OF ID "PRNOUT".
000340 01 PRN-REC PIC X(132).
000350 WORKING-STORAGE SECTION.
000360 77 SUB1 PIC 9999 COMP VA 0.
000370 PROCEDURE DIVISION.
000380 0001.
000390     OPEN INPUT FILE002     OUTPUT FILE003.
000400 0002.
000410     READ FILE002 AT END STOP RUN.
000420     MOVE FILE002-REC TO PRN-REC.
000430     ADD 1 TO SUB1.
000440     MOVE SPACE TO TABLE-DATA (SUB1).
000450     WRITE PRN-REC.
000460     GO TO 0002.
000470 END-OF-JOB.
PROGRAM ID DUMMY .
COMPILE DATE 14:16 12/20/78 USING 348/73 COMPILER.
NO WARNINGS.
NO SEQUENCE ERRORS.
37 SYMBOLIC RECORDS COMPILED AT 277 RECORDS PER MINUTE.
8 SECONDS TOTAL ELAPSED CLOCK TIME.
15 DISK SEGMENTS REQUIRED FOR THIS PROGRAM.
1500 BYTES TOTAL CORE REQUIRED.
45000 BYTE COBOLV COMPILER. RELEASE NUMBER: ASX 6.2 .

```

	LOW ADDRESS	HIGH ADDRESS	LENGTH IN DIGITS
RESERVED MEMORY	000000	00138	00138
DATA DIVISION	00188	002136	001748
FIXED SEGMENT CONSTANTS	002136	002136	00000
FIXED SEGMENT INSTRUCTIONS	002136	002356	000220
INPUT OUTPUT BUFFERS	002356	002564	000208
STACK	002564	003000	000436
MAXIMUM DISK FILE HEADER SPACE			000200

P1276

**Figure A-1. Program Used to Produce a Dump**

```
-----
DUMP OF BULLdd.      MIX NUMBER: 08.      DUMP FILE ID: 800025.      DATE: DEC 20,78.      TIME: 14:15:32.      COMPILED: DEC 20,78.
PROCESSOR NUMBER 0.  CPU TYPE B4800.      MEMORY SPEED: 8 MHZ.      MCPVI: ASR 5.2      RELEASED: NOV 29,78.
-----
```

```
RUN CONTROL WORD:
P.A.R. = 2135 ← ①
BASE = 212
LIMIT = 215
HI ADP = 213
MODE = E30
OVERFLD = OFF
COMPARE = EQU
```

```
ACCUMULATOR: -99+00000000000000
```

```
RESERVED MEMORY:
 3 - 7 = 0000000 UNASSIGNED
 8 - 15 = 0000232 IX1
16 - 23 = 0000000 IX2
24 - 31 = 0000000 IX3
32 - 39 = 0000000 UNASSIGNED
40 - 47 = 002054 SCAN COUNT
      = 002054 STACK TOP
      = 0000000 HALT BREAKPOINT } ⑧
```

```
STACK: TOP = 2564.
```

```
PROGRAM STATUS:
TERMINATING. INVALID FILE READ ← ⑨
```

```
MIX:
MIX-ID = 30LL11 PROGRAM ID
MIX-JV = 1 OVERLAY WAIT COUNT
MIX-WA = 4 WAITING/STOPPED
MIX-WC = 4 SUSPENDED/WAITING
MIX-PP = 4 PROCESS PRIORITY
MIX-AP = 4 MEMORY PRIORITY
MIX-HK = 0 HARDWARE/TERM CODE
MIX-AD = 6 STOP/PUSH/SORT
MIX-IC = 02 NO. IOATS ASSIGNED
MIX-IR = 5 TERMINATE/BREAKOUT
MIX-KL = 0495 RUN LOG ID
MIX-PR = 00014950 PPS DISK ADDRESS } ⑩
```

```
JRT:
JRT-CC = 4 STATUS CODE
JRT-LQ = 4 RATIO SPILL INITIAL
JRT-SA = 4 EXTENSION TABLE
JRT-RZ = 14 REQUESTION CODE
JRT-RL = 0510 RUN LOG ID NO.
JRT-SP = 4 MEMORY PRIORITY
JRT-AP = 4 PROCESS PRIORITY
JRT-CP = 01 DICK FILE COUNT
JRT-FH = 00014950 PSM FILE DEF ADDR
JRT-FD = 004 PSM FILE DEF DURE
JRT-FX = 0705 PSM FILE DEF INDIR
JRT-TJ = 01522 SDD TIME
```

```
SEGMENT DICTIONARY:
LOCATION 124.
SEG IN_DURE 121-INSIR SEG-START 124 SEG-END 3335 REL-DISS 3335
CURRENT SEGMENT = 1.
```

Figure A-2. Dump Produced From Program in Figure A-1 (Sheet 1 of 5)

FILE: JISKIN

IOAT: IO-CHN = 05 CHANNEL
IO-UNT = 5 UNIT
IO-ID = JISKIN FILE ID
IO-MFD = 00000 MULTI-FILE ID
IO-ST1 = 7 OPEN IN
IO-HOW = 11 200 PACK-INTER
IO-FIB = 00000 FILE INFO BLOCK
IO-SIA = 04 STATUS DIGIT LINK
IO-ERT = 10 RETRY COUNT
IO-RCT = 00000000 CHECK COUNTER
IO-ADP = 04000000 NEXT I/O DISK ADDR
IO-AR# = 01 CURRENT AREA NUMBER
IO-HPT = 19300 DISK FILE HDM ADDR

IOAT 1145053404 0000000000 0000000010 0000000004 09E2D2D9D5 F0F0F0F0F0 F000700004 0038930100 0000025950 0000193300

JFHDR: JF-RSZ = 0010 RECORD SIZE, DIGITS
JF-RPB = 002 RECORDS PER BLOCK
JF-EOF = 00000000 EOF POINTER
JF-ORS = 0020 ORIGINAL RECORD SIZE
JF-ORR = 001 ORIGINAL RECD/BLK
JF-OSB = 0001 DISK SEGMENTS/BLOCK
JF-PAK = 4 PACK FLAGS
JF-ADR = 19300 CYCLE ADDR, BLOCK #1
JF-DIR = 04000000 DIRECTORY HDR ADJRS
JF-DFH = 04000036 HEADER BLOCK ADJRS
JF-DRX = 0210 DIRECTORY HDR INDEX
JF-BPA = 0002000 BLOCKS PER AREA
JF-SIZ = 104 JFHDR SIZE, DIGITS
JF-USR = 01 NO. USERS, JFHDR
JF-ST2 = 3 ADDR BLKS PRESENT

ADDRESS BLOCKS: 00 04000000 } 12

FIB: FIBST1 = 4 RECD BEFORE BRKJUT
FIBRKN = 40399 BREAKOUT SPACING
FIBRRC = 04040 SINGLE PACK FILE
FIB-BA = 4 NUMBER ALT. BUFFERS
FIB-AL1 = 4 SAVE FACTOR, DAYS
FIB-RRL = 04040 RECORD SIZE, DIGITS
FIB-RPB = 404 RECORDS PER BLOCK
FIB-RAB = 040404 BUFFER ADDRESS
FIB-WA = 001335 WORK AREA ADDRESS
FIB-HDM = 11 HARDWARE TYPE
FIB-BLK = 1 BLOCKING TECHNIQUE
FIB-FNM = 01 COMPILED FILE NO.
FIB-LBA = 20 BUFFER REL INDEX/10
FIB-CBS = 00000000 CURRENT BUFFER SIZE
FIB-RCT = 00000000 LOGICAL RECD COUNT
FIB-MBS = 0002000 BLOCK SIZE, DIGITS
FIB-NB = 200 NEXT BUFFER INDEX
FIB-IDA = 019320 ACTUAL IOAT ADDRESS
FIB-NAR = 20 NUMBER OF AREAS
FIB-RSA = 00025 RECORD SIZE, WORDS
FIB-RPA = 00000250 RECORDS PER AREA
FIB-CJD = 51 DESCRIPTOR OP CODE
FIB-LAB = 00118 LABEL BEGIN ADDRESS
FIB-LAE = 001320 LABEL END ADDRESS

FIB 404039904 404040404 404040404 404001335 110010100 000000000 058000000 000193200 000020000 002500000

BSW #1 000 01 014 002354 002564 004692 00000 000364

B 2000/B 3000/B 4000 MCPVI Programmer's Guide
Example Program and Memory Dump

Figure A-2. Dump Produced From Program in Figure A-1 (Sheet 2 of 5)









B 2000/B 3000/B 4000 MCPVI Programmer's Guide  
Example Program and Memory Dump

NOTES

1. Program failed at this address.
2. EOF pointer for the file DISKIN.
3. Number of records processed from file DISKIN.
4. IX3: pointer to current stack entry. Equal zero implies no stack or IX3 value changed.
5. Top-of-Stack pointer: next stack entry placed at this address.
6. Instruction pointed at by PAR, from 1.
7. Program executed this BCT and encountered a problem.
8. Reserved memory is same as address 0-48 in the memory dump portion.
9. Status interpreted by DMPOUT from MIX table values.
10. MIX table values interpreted by DMPOUT; only relevant fields are broken out.
11. Hardware code 11 means disk pack; DMPOUT further decodes IOAT entry to produce 206 PACK-INTER (interlace mode).
12. Address blocks are pointers to file areas (pages) on disk/disk pack.
13. Dump Control Segments include information pertaining to dump such as program-ID, date, and time of dump, JRT entry, and MIX entry. Information here intended for DMPOUT use; all information broken out elsewhere in the dump printout.

## APPENDIX B

### BRANCH COMMUNICATE CHART

Table B-1 lists the program Branch Communicates available with MCPVI. Where several functions use the same BCT, the parameter value which identifies its function is given in parentheses.

**Table B-1. Branch Communicates (BCTs) for MCPVI**

BCT	BCT Function
BCT 0114	READ, LOCK
BCT 0134	OPEN
BCT 0154	CLOSE
BCT 0174	OVERLAY
BCT 0194	STOP RUN
BCT 0214	ARM (5) CORE-SIZE (2) DATE (4) INTERROGATE DISK (3) INTERROGATE DISK PACK (9) MIXTBL (6) TIME (1) TIME-60 (8) TODAYS-DATE (0) TODAYS-NAME (7)
BCT 0234	WRITE, UNLOCK
BCT 0254	ACCEPT (0) BREAKOUT CONTROL (9) DISPLAY (1) DISPLAY-LINES (8) SORT (5) SORT RETURN (6) WAIT (2)
BCT 0274	ZIP
BCT 0294	EXITROUTINE
BCT 0314	SEEK
BCT 0334	TRACE, DUMP, BREAKOUT
BCT 0354	DCOM (all)
BCT 0374	MICR (all)
BCT 0394	SPACE
BCT 0414	CRCR
BCT 0434	DIRECT I/O
BCT 0454	QUICKTIME

B 2000/B 3000/B 4000 MCPVI Programmer's Guide  
Branch Communicate Chart

**Table B-1. Branch Communicates (BCTs) for MCPVI (Cont)**

<b>BCT</b>	<b>BCT Function</b>
BCT 0474	SPOMESSAGE
BCT 0494	STOQUE
BCT 0514	DEBUG
BCT 0534	USERCHANGE, USERCODE
BCT 0894	DATA BASE
BCT 0994	COMPLEX WAIT

## APPENDIX C

### PROGRAM CONSIDERATIONS FOR SHARED FILES

This section explains the programmatic operations available for use with shared files, and considerations for the use of shared files.

#### SHARED FILE OPERATIONS

The available SHARED operations and the corresponding compiler constructs are

Operation	COBOLV	BPL
LOCK	READ LOCK-ONLY	LOCK LOCK SEEK
UNLOCK	UNLOCK	UNLOCK
READ/SEEK	READ/SEEK	READ/SEEK
READ/SEEK LOCK	READ/SEEK LOCK	READ/SEEK LOCK
READ/SEEK RETRY	READ LOCK-WAIT	READ NO UNLOCK
WRITE	WRITE	WRITE
WRITE-NO-UNLOCK	WRITE LOCK	WRITE LOCK

These constructs work as follows:

#### LOCK

A LOCK will lock the address that contains the record. The user program is not reinstated until the lock operation is completed. The BPL LOCK SEEK permits the program to be reinstated without waiting for the LOCK to complete.

LOCK can affect an address in three ways:

1. Most commonly, the address is not locked at all. In this case, the address is stored in the Shared System Processor (SSP) or File Protect Memory (FPM) (if available), and the Record Lockout Table (RLT).
2. The address is locked by the current user. In this case, the address is already stored so there is nothing further to do.
3. The address is locked by another program either on the same processor or on another processor. In this case, the lock I/O descriptor is queued and the address is inserted into the RLT and marked either RLT contention or SSP/FPM contention as appropriate.

Only the third of these circumstances suspends the user until the LOCK is successful. If the operation was other than a plain Lock (for example, READ-LOCK), the other part of the operation will also be inhibited.

#### UNLOCK

An UNLOCK will unlock the address in the SSP/FPM and remove it from the RLT. Obviously, this cannot be done if the address is not already locked. Therefore, a program attempting to unlock a record which is not locked will cause a DS or DP condition.

#### READ and SEEK

A READ or SEEK will ignore a locked condition.



B 2000/B 3000/B 4000 MCPVI Programmer's Guide  
Branch Communicate Chart

When a file is being created (that is, opened output for the first time), much overhead can be saved if it is opened as a non-shared file, initialized and closed, and reopened as a shared file. This requires two file declaration statements.

A shared file must be declared with the same record and block sizes in all programs. A program with a mis-matched declaration is terminated.

There is no way to determine whether actual file is shared. This is controlled in the FIB. Therefore, if a program opens a file as not shared while another is updating it, there is no protection and data corruption can result.





2" BINDER

1½" BINDER

1" BINDER

**B 2000/B 3000/B 4000 Series**  
**MCPVI**  
SYSTEM SOFTWARE PROGRAMMER'S GUIDE

1090685

Printed in U.S.A.