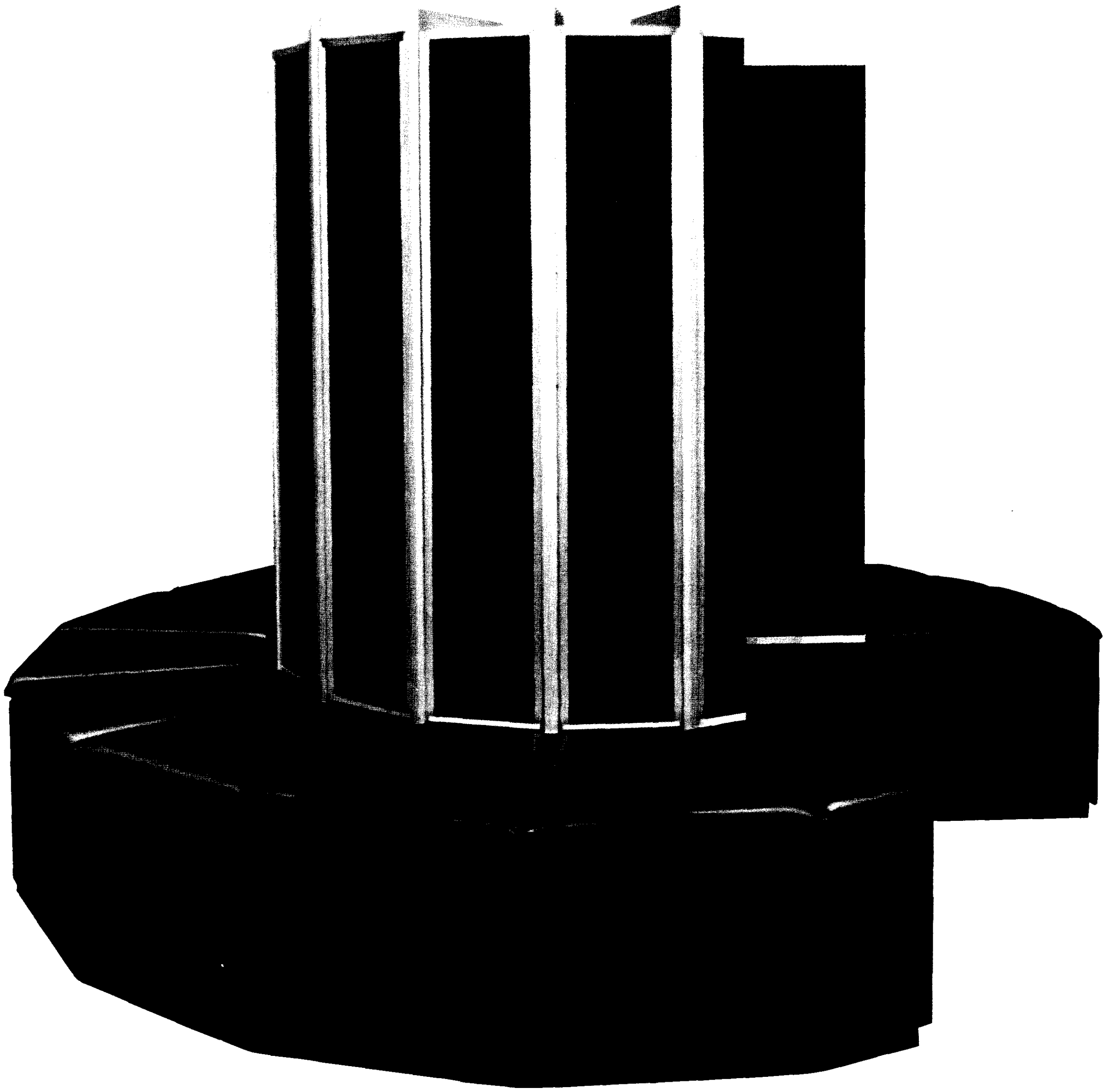
CRAY RESEARCH, INC.

CRAY-1S COMPUTER SYSTEM

I/O SUBSYSTEM SOFTWARE WORKBOOK

T-0201



FOR TRAINING PURPOSES ONLY

CRAY-1 S

COMPUTER SYSTEM

I/O SUBSYSTEM SOFTWARE

WORKBOOK

T-0201

RECORD OF REVISION CRAY RESEARCH, INC. PUBLICATION NUMBER

Each time this manual is revised and reprinted, all changes issued against the previous version in the form of change packets are incorporated into the new version and the new version is assigned an alphabetic level. Between reprints, changes may be issued against the current version in the form of change packets. Each change packet is assigned a numeric designator, starting with 01 for the first change packet of each revision level.

Every page changed by a reprint or by a change packet has the revision level and change packet number in the lower righthand corner. Changes to part of a page are noted by a change bar along the margin of the page. A change bar in the margin opposite the page number indicates that the entire page is new; a dot in the same place indicates that information has been moved from one page to another, but has not otherwise changed.

Requests for copies of Cray Research, Inc. publications and comments about these publications should be directed to:

CRAY RESEARCH, INC.,
1440 Northland Drive,
Mendota Heights, Minnesota 55120

Revision	Description
	September, 1980 - Original printing
A	November, 1980 - Reprint with revision. Changes include addition of detail on concentrator software.
B	January 1981 - Reprint with revision. Changes to add more detailed flow diagrams.
C	March 1981 - Reprint with revision. Changes include addition of interactive station software and deadstart procedures.
D	September 1981 - Reprint with revision. Changes to add more detail on Disk I/O and IOS Chassis Layouts.

TABLE OF CONTENTS

PART 1

I/O SUBSYSTEM HARDWARE

CHAPTER 1

SYSTEM OVERVIEW

INPUT/OUTPUT SUBSYSTEM

INCREASES CRAY-1 S CPU THROUGHPUT BY REDUCING ITS I/O AND FRONT-END RESPONSIBILITIES.

STREAMS DATA TO CENTRAL MEMORY OVER HIGH SPEED CHANNEL.

PROVIDES ACCESS TO ADDITIONAL PERIPHERALS. (TAPES)

FUNCTIONS AS A MAINTENANCE CONTROL UNIT.

DRIVES UP TO 48 DD-29 DISK DRIVES FOR MASS STORAGE.

ALLOWS OPERATOR CONTROL OF COS.

COLLECTS AND CONCENTRATES DATA FROM FRONT ENDS.

PROVIDES FOR JOB AND DATA ENTRY.

DISTRIBUTES CPU OUTPUT TO SLOWER PERIPHERAL EXPANDER DEVICES.

CONSISTS OF TWO TO FOUR I/O PROCESSORS WITH A SHARED BUFFER MEMORY.

PHYSICAL CHARACTERISTICS

4 COLUMN CHASSIS CONTAINS I/O PROCESSORS, BUFFER MEMORY, CONTROLLERS AND INTERFACES.

4 COLUMNS PLUS 2 POWER SUPPLIES WEIGHTS 3775 LB.

IOS HAS ITS' OWN POWER DISTRIBUTION UNIT (PDU)

COOLING AND POWER SHARED WITH CPU

CRAY-1 S/4X00 REQUIRES AN ADDITIONAL MOTOR GENERATOR (3) AND AN ADDITIONAL COMPRESSOR (3)

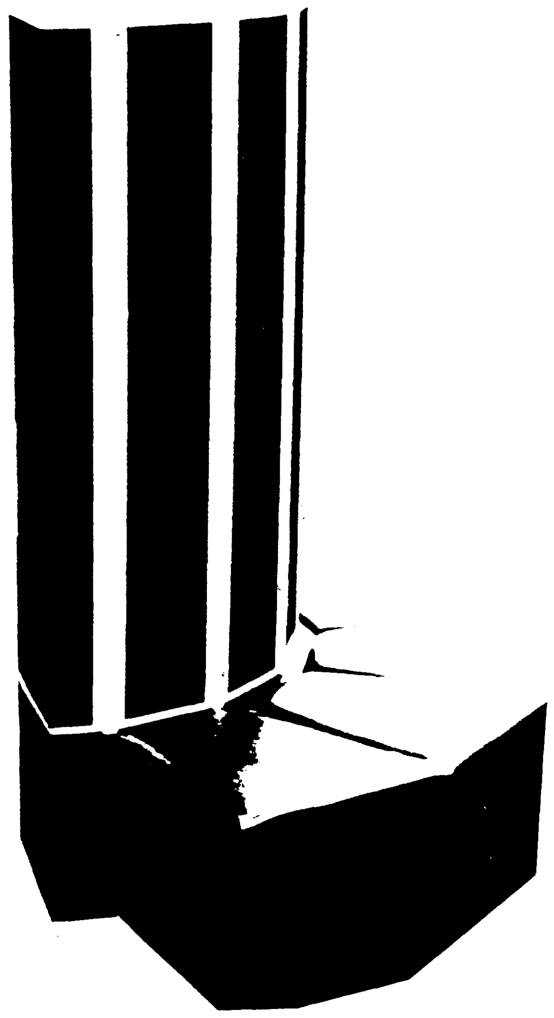
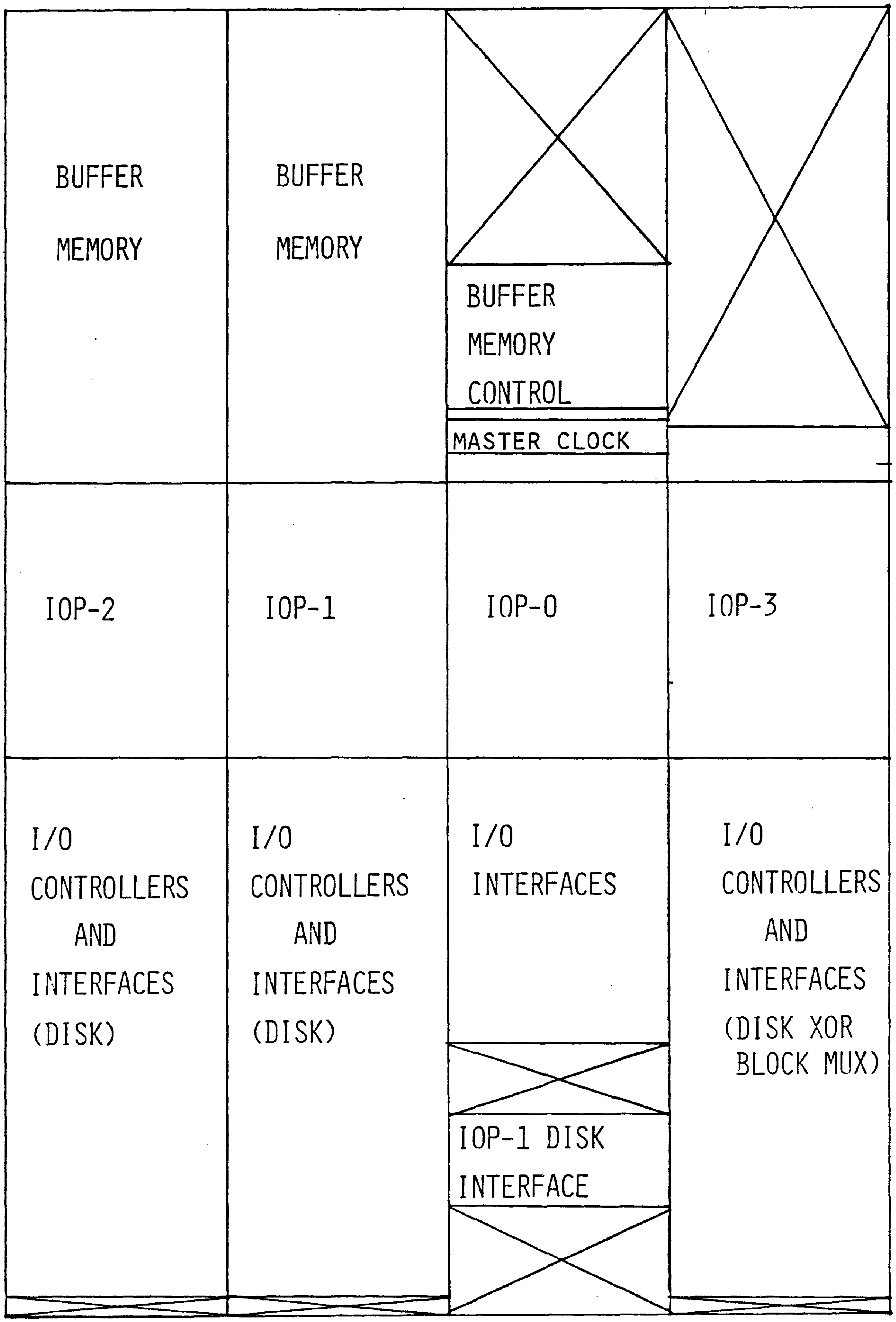
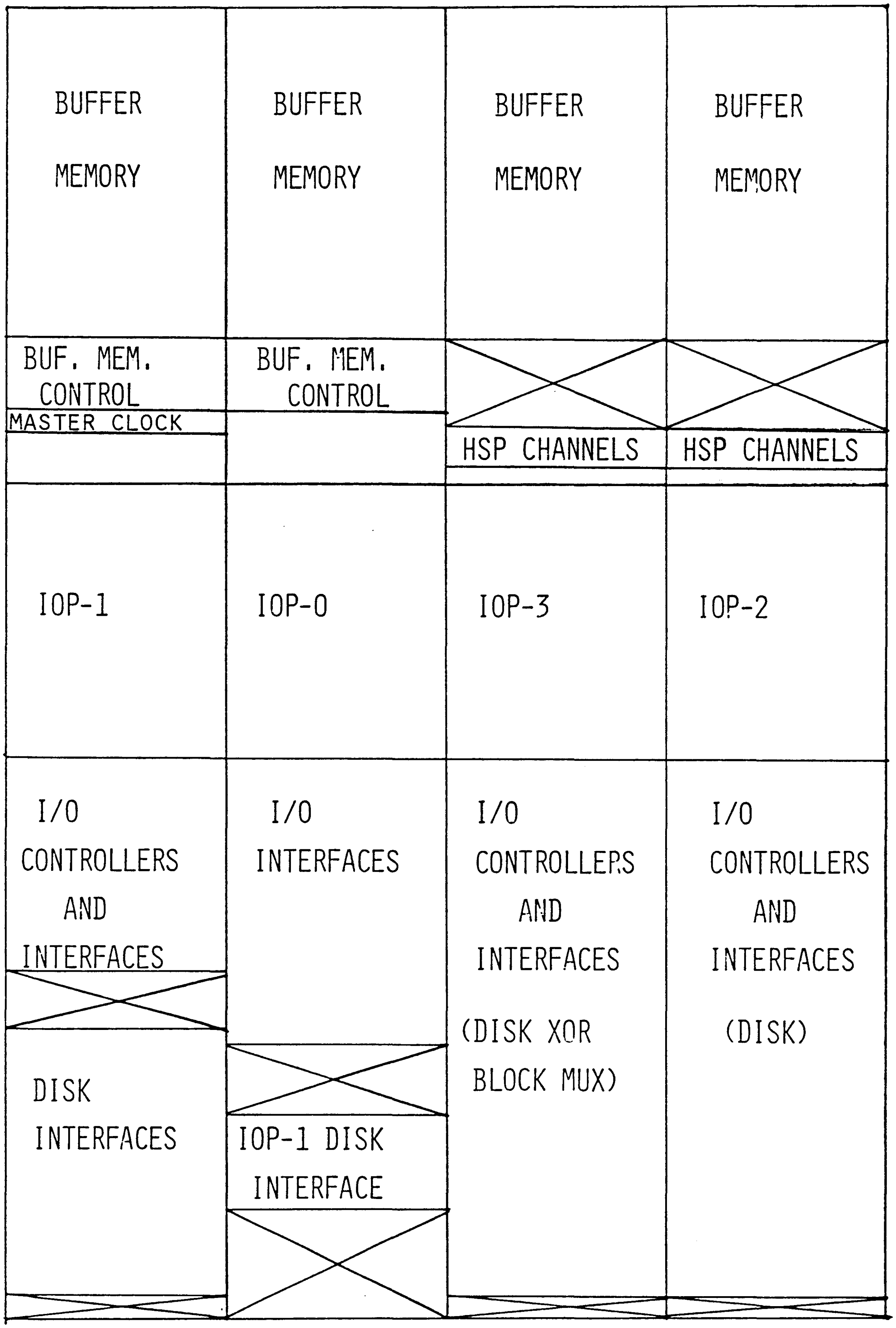


FIGURE 1-1. I/O SUBSYSTEM


BUFFER MEMORY
BUFFER MEMORY
BUFFER MEMORY CONTROL
MASTER CLOCK
IOP-2
IOP-1
IOP-0
IOP-3
I/O CONTROLLERS AND INTERFACES (DISK)
I/O CONTROLLERS AND INTERFACES (DISK)
I/O INTERFACES
IOP-1 DISK INTERFACE
I/O CONTROLLERS AND INTERFACES (DISK XOR BLOCK MUX)

I/O SUBSYSTEM, MODEL A
(SN 3-5, 7-10)



I/O SUBSYSTEM MODEL B
(SN 6, 11 +)

INDIVIDUAL I/O PROCESSOR

LOCAL MEMORY:

- 65,536 WORDS
- 16 BITS/WORD

COMPUTATION SECTION:

- INSTRUCTION CONTROL NETWORK
- 2 FUNCTIONAL UNITS (ADDER AND SHIFTER)
- LOGICAL 'AND' OPERATION
- 512 OPERAND REGISTERS
- SINGLE-ADDRESS MODE

I/O SECTION:

A = 810 MB/S

B = 850 MB/S

- 6 DIRECT MEMORY ACCESS (DMA) PORTS
- 4 PARCELS EVERY 6 CLOCK PERIODS MAXIMUM TRANSFER RATE PER PORT
- SEVERAL CHANNELS MAY MULTIPLEX INTO ONE PORT

MODEL NUMBERING CONVENTION

CRAY-1 S/1200 THROUGH /4400 CONTAIN AN I/O SUBSYSTEM

FIRST DIGIT INDICATES SIZE OF CENTRAL MEMORY IN MEGAWORDS

CRAY 1 mem

SECOND DIGIT INDICATES NUMBER OF I/O PROCESSORS

EXAMPLE:

CRAY-1 S/2400 HAS 2 MILLION WORDS OF CENTRAL MEMORY AND 4 I/O PROCESSORS

CRAY-1 S/x200

MINIMUM CONFIGURATION

TWO PROCESSOR SYSTEM

1. MASTER I/O PROCESSOR (MIOP)

   CONTROLS FRONT-END INTERFACES.

   HAS UP TO 4 DISPLAY CONSOLES.

   HAS AN EXPANDER CHANNEL WHICH MULTIPLEXES A PRINTER AND A MAG TAPE UNIT.

   CONNECTS TO BUFFER MEMORY THROUGH A DMA CHANNEL.

   EXCHANGES CONTROL SIGNALS WITH CPU OVER A LOW-SPEED CRAY-1 S CHANNEL PAIR.

   COMMUNICATES WITH OTHER IOPS OVER ACCUMULATOR CHANNELS.

   MAINTAINS SYSTEM INFORMATION ERROR LOG.

   COORDINATES ACTIONS OF CPU AND OTHER IOPS.

2. BUFFER I/O PROCESSOR (BIOP)

   HANDLES DATA TRANSFERS BETWEEN CPU AND I/O SUBSYSTEM.

   CONNECTS DIRECTLY TO CENTRAL MEMORY.

   CONNECTS TO BUFFER MEMORY THROUGH A DMA CHANNEL.

   COMMUNICATES WITH OTHER IOPS OVER ACCUMULATOR CHANNELS.

   DRIVES UP TO 16 DD-29 DISK DRIVES.

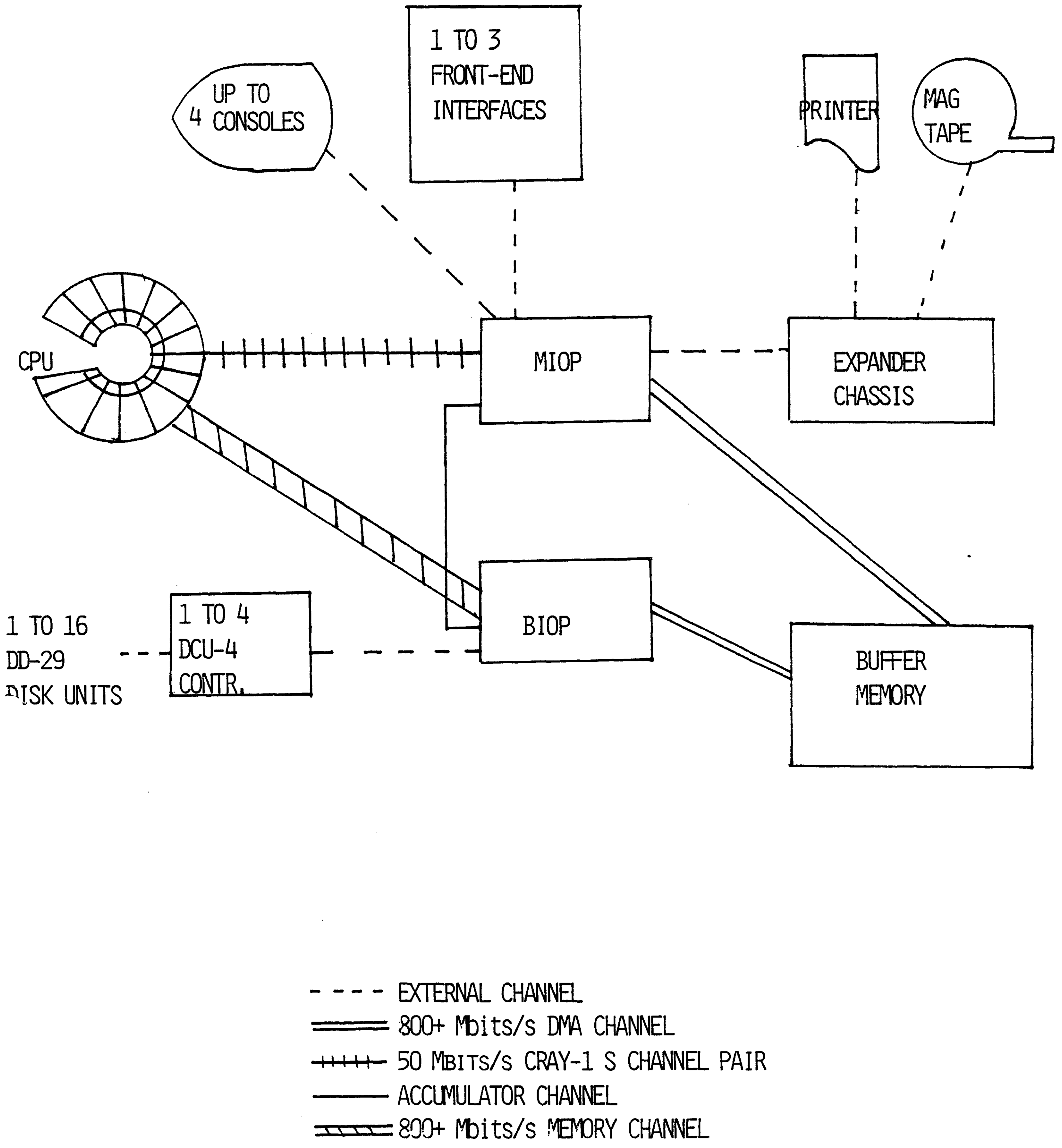


FIGURE 1-2. BLOCK DIAGRAM OF S/1200, S/2200 AND S/4200 SYSTEMS

CRAY-1 S/x300

THREE PROCESSOR SYSTEM

1. MIOP

2. BIOP

3. DISK I/O PROCESSOR (DIOP)

   CONNECTS TO BUFFER MEMORY THROUGH A DMA CHANNEL.

   COMMUNICATES WITH OTHER IOPS OVER ACCUMULATOR CHANNELS.

   DRIVES UP TO 16 DD-29 DISK DRIVES.

OR

3. BLOCK MULTIPLEXER I/O PROCESSOR (XIOP)

   CONNECTS TO BUFFER MEMORY THROUGH A DMA CHANNEL.

   COMMUNICATES WITH OTHER IOPS OVER ACCUMULATOR CHANNELS.

   HANDLES 1 TO 16 BLOCK MUX (IBM COMPATIBLE) CHANNELS.

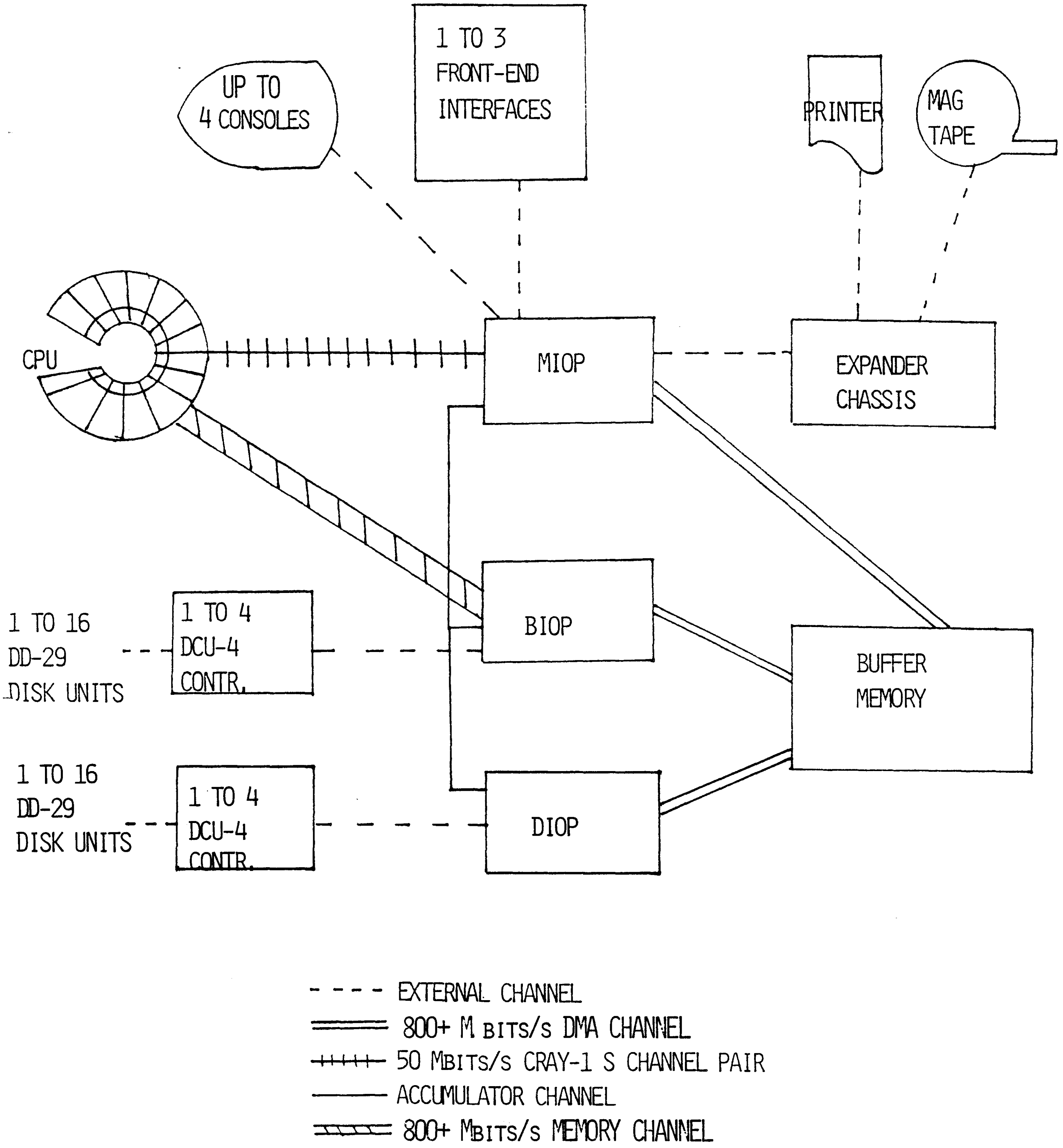


FIGURE 1-3. BLOCK DIAGRAM OF S/1300, S/2300 AND S/4300 SYSTEMS WITH INCREASED DISK CAPACITY.

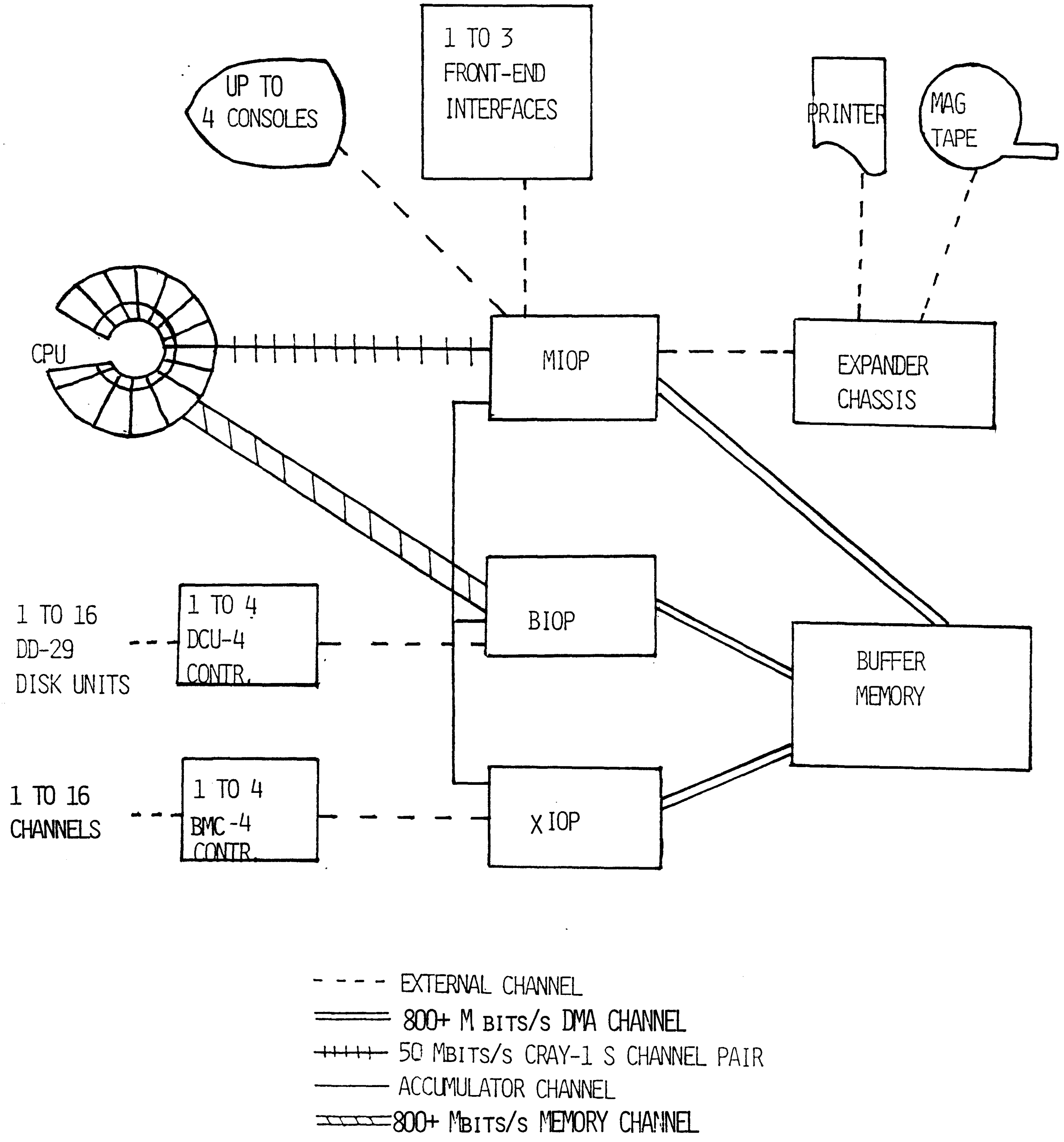


FIGURE 1-4. BLOCK DIAGRAM OF S/1300, S/2300 AND S/4300 SYSTEMS WITH BLOCK MULTIPLEXER CHANNELS.

CRAY-1 S/x400

FOUR PROCESSOR SYSTEM

MAXIMUM OF 48 DD-29 DISK DRIVES

TWO POSSIBLE CONFIGURATIONS

1. MIOP
2. BIOP
3. DIOP
4. DIOP

OR

1. MIOP
2. BIOP
3. DIOP
4. XIOP

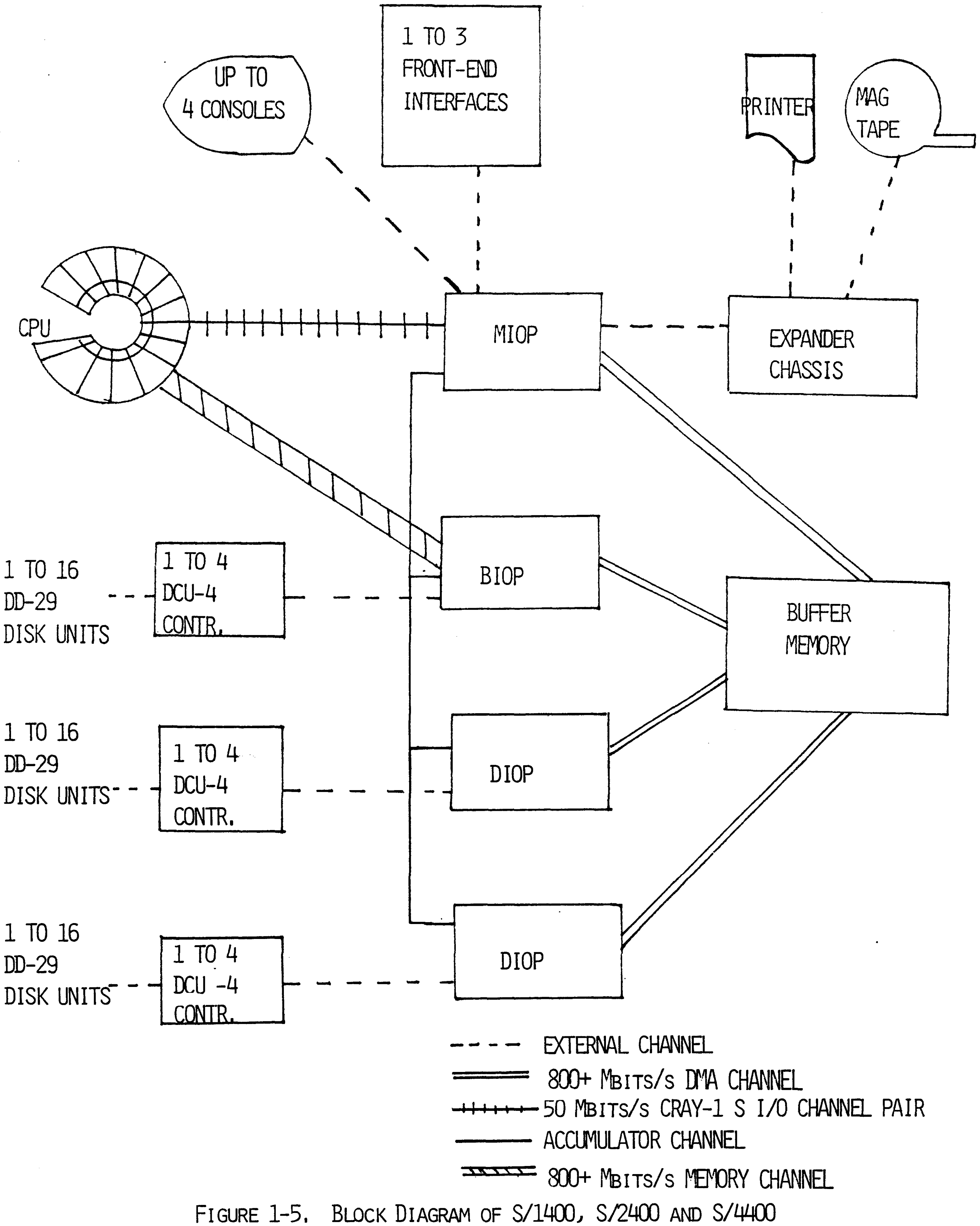


Figure 1-5. Block Diagram of S/1400, S/2400 and S/4400 Systems With Increased Disk Capacity.

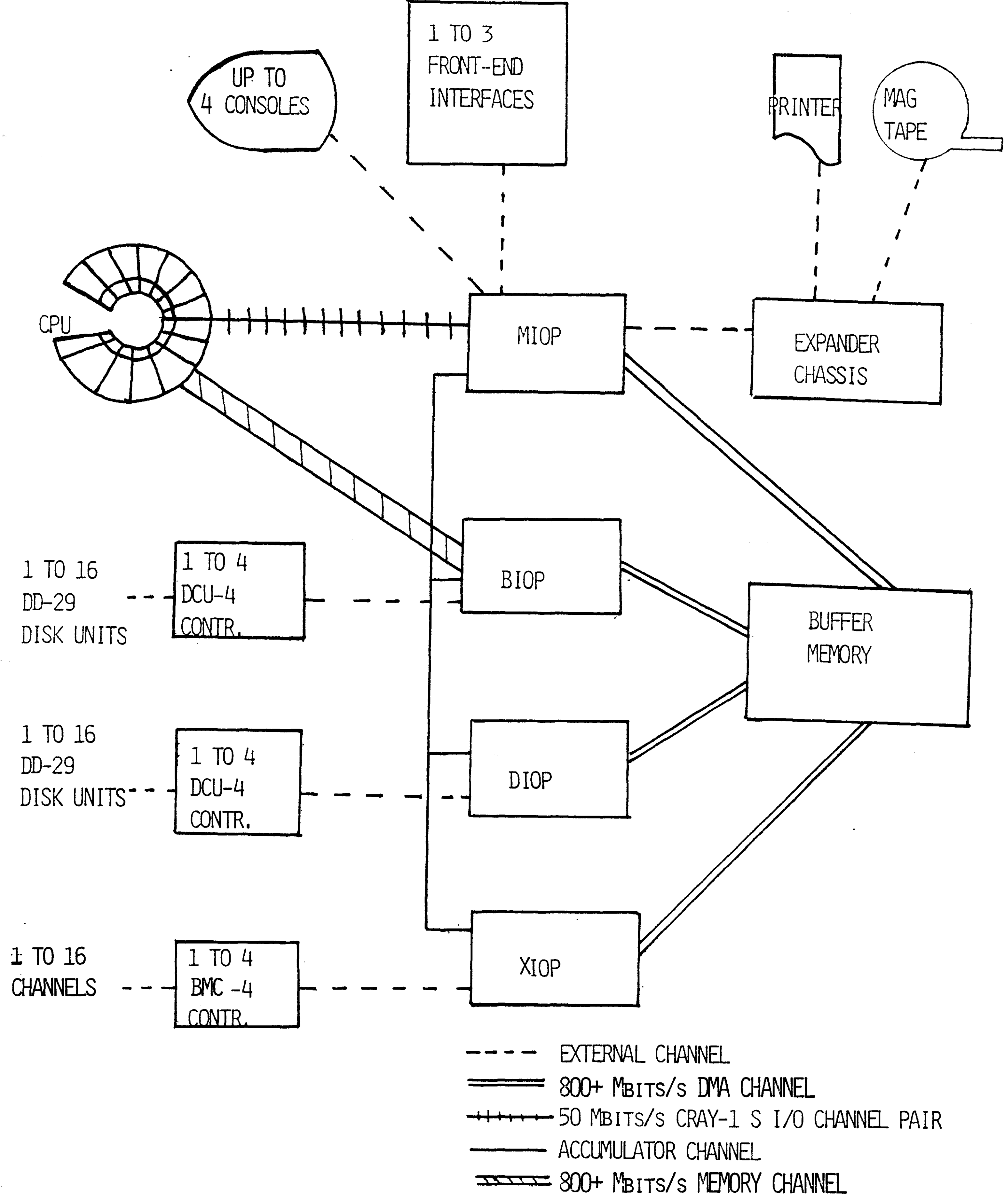


FIGURE 1-6. BLOCK DIAGRAM OF S/1400, S/2400 AND S/4400 SYSTEMS WITH BLOCK MULTIPLEXER CHANNELS.

CHAPTER 2

I/O PROCESSOR LOCAL MEMORY

FUNCTIONS

PROVIDES BUFFERS FOR BLOCK TRANSFERS.

HOLDS NUCLEUS OF OPERATING SYSTEM.

PROVIDES SPACE FOR EXECUTION OF IOS OVERLAY CODE.

CHARACTERISTICS

65,536 16 BIT WORDS IN 4 SECTIONS OF 4 BANKS

4 CP BANK BUSY TIME ON READ

6 CP BANK BUSY TIME ON WRITE

WHOLE SECTION GOES BUSY, NOT JUST BANK

BIPOLAR CIRCUITRY

7 CP READ TO ACCUMULATOR

INSTRUCTION FETCH DONE IN 4 CP BURSTS, 1 PARCEL/CP

OPERAND REFERENCE MOVES 1 PARCEL TO/FROM ACCUMULATOR

I/O REFERENCE MOVES 4 SEQUENTIAL PARCELS TO/FROM I/O CHANNEL

ODD PARITY; 1 PARITY BIT PER BYTE

NO ERROR CORRECTION

6 DIRECT MEMORY ACCESS PORTS

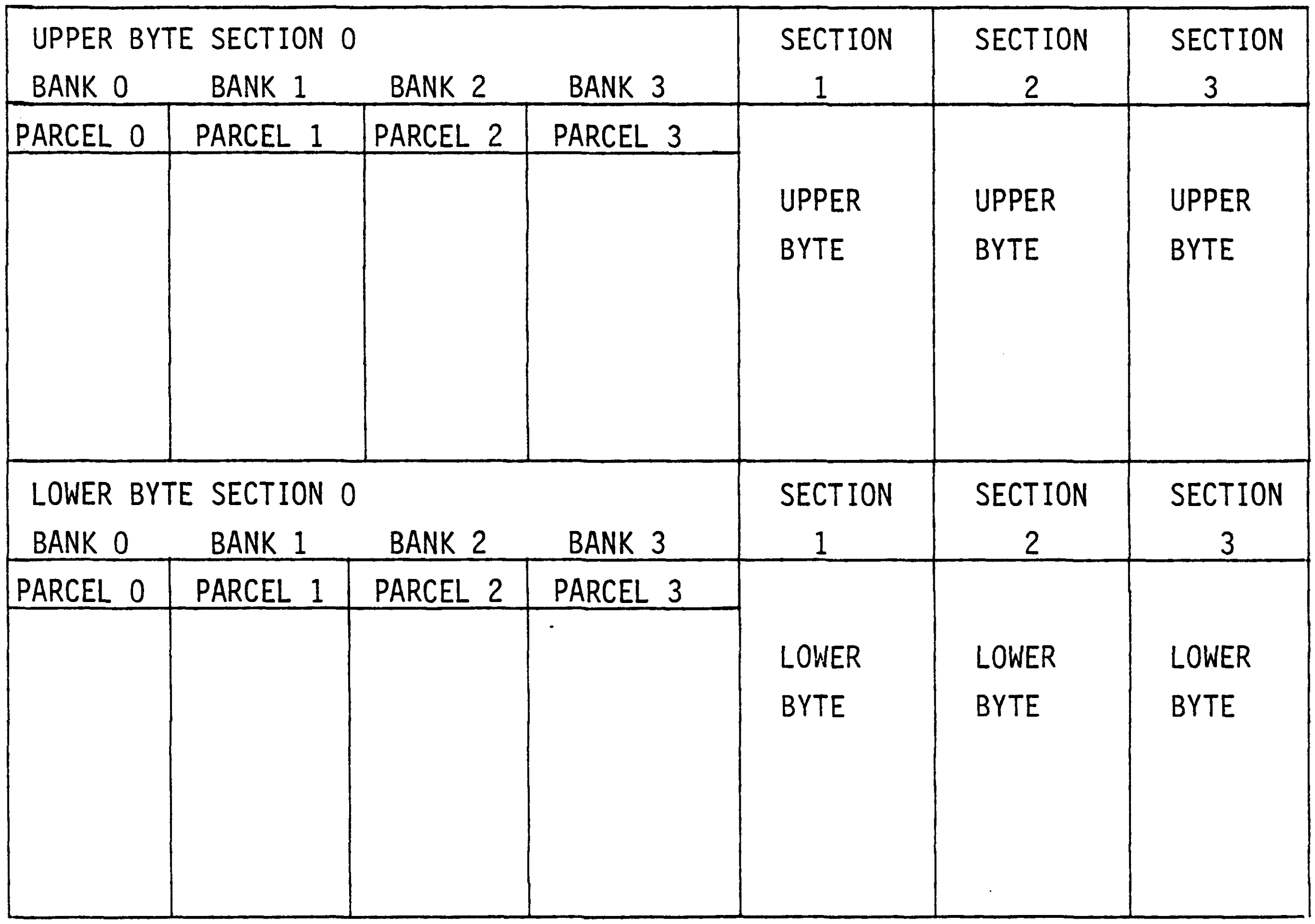


FIGURE 2-2. LOCAL MEMORY LAYOUT

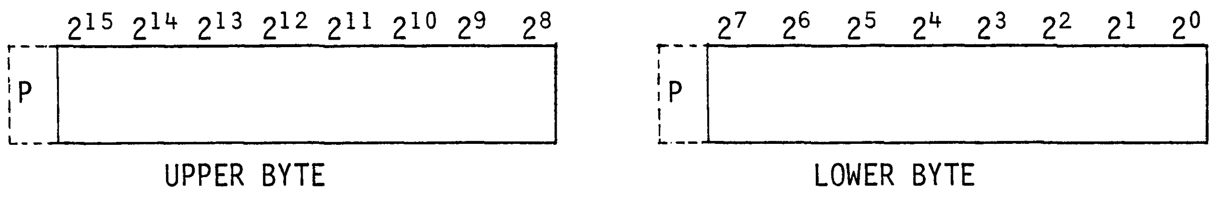


FIGURE 2-3. DATA WORD FORMAT

ADDRESSING SCHEME

LOWER 4 BITS SELECT SECTION AND BANK

NEXT 10 BITS SELECT ADDRESS IN CHIP

UPPER 2 BITS SELECT CHIP

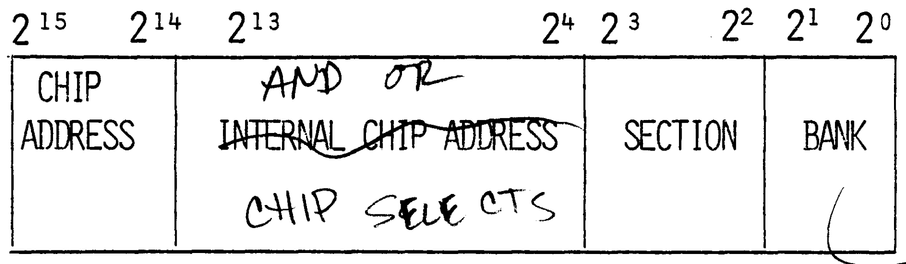


FIGURE 2-1. LOCAL MEMORY ADDRESS FORMAT

= Ø's On I/O and Fetch

THREE ADDRESS PATHS TO EACH LOCAL MEMORY SECTION FROM:

- I/O SECTION
- COMPUTATION SECTION
- FETCH REGISTER

ACCESS

3 READ PATHS AND 2 WRITE PATHS PER LOCAL MEMORY SECTION.

1 OF EACH TO ACCUMULATOR.

1 OF EACH TO I/O SECTION TO SERVICE DMA PORTS.

LAST READ PATH TO INSTRUCTION STACK FOR FETCH.

CHAPTER 3

COMPUTATION SECTION

BASIC COMPONENTS

INSTRUCTION CONTROL NETWORK

512 OPERAND REGISTERS

2 FUNCTIONAL UNITS (ADDER AND SHIFTER)

1 PROGRAMMER-VISIBLE ACCUMULATOR

LOGICAL 'AND' OPERATION

128 INSTRUCTION CODES

[illegible]

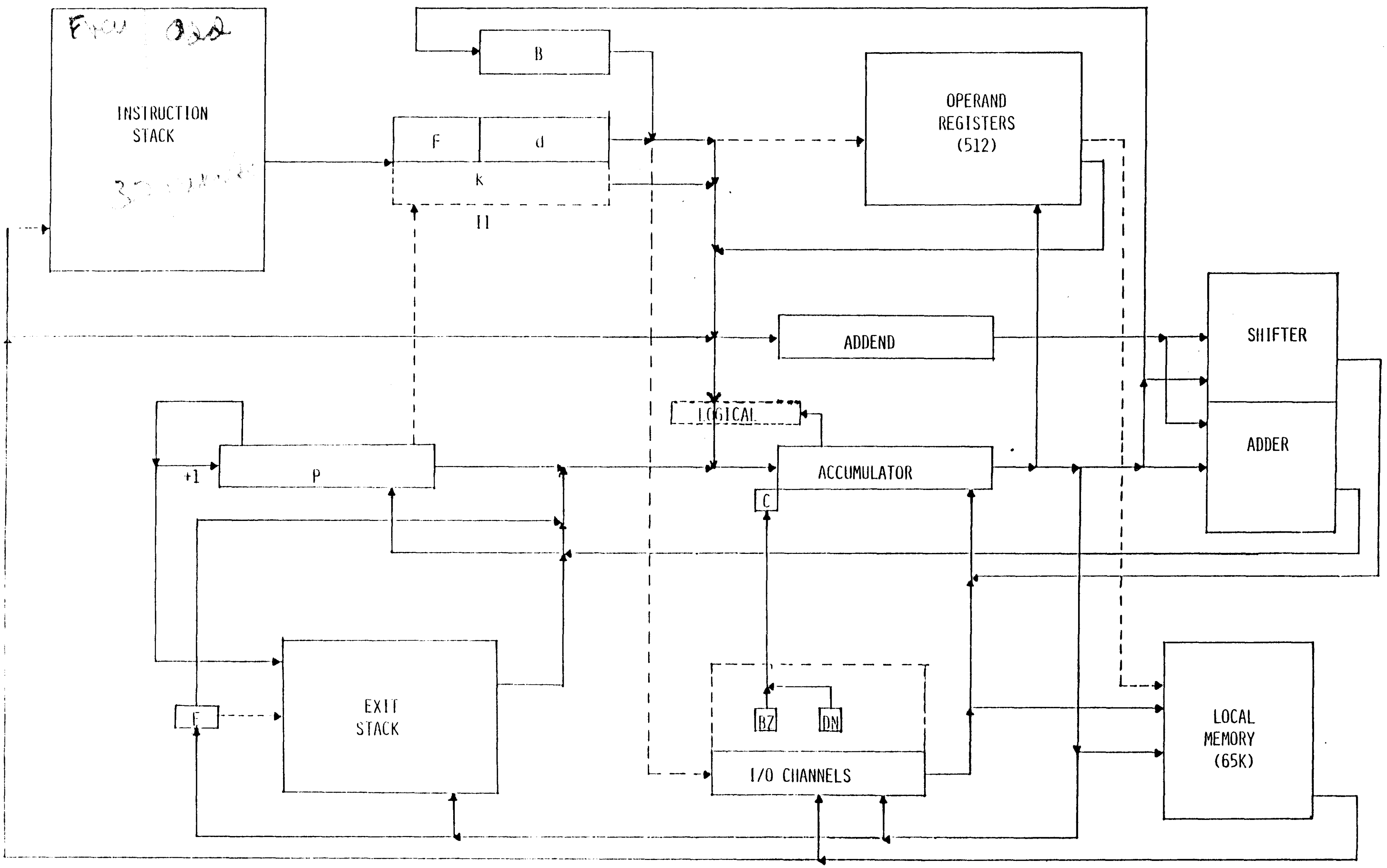


FIGURE 3-1. I/O PROCESSOR BLOCK DIAGRAM

INSTRUCTION CONTROL NETWORK

RESPONSIBLE FOR CONTROLLING ISSUE AND EXECUTION OF INSTRUCTIONS.

MAIN COMPONENTS ARE:

- INSTRUCTION STACK
- II (INSTRUCTION ISSUE) REGISTER
- B REGISTER
- P (PROGRAM ADDRESS) REGISTER
- PROGRAM EXIT STACK
- PROGRAM FETCH REQUEST FLAG

INSTRUCTION STACK

32 PARCELS IN TWO, 16 PARCEL STACKS

INSTRUCTIONS ARE FULLY INTERLEAVED

BACKGROUND FETCHES OCCUR IN BURSTS OF 4 SEQUENTIAL PARCELS

CIRCULAR

INSTRUCTIONS CONSIST OF 1 OR 2 PARCELS

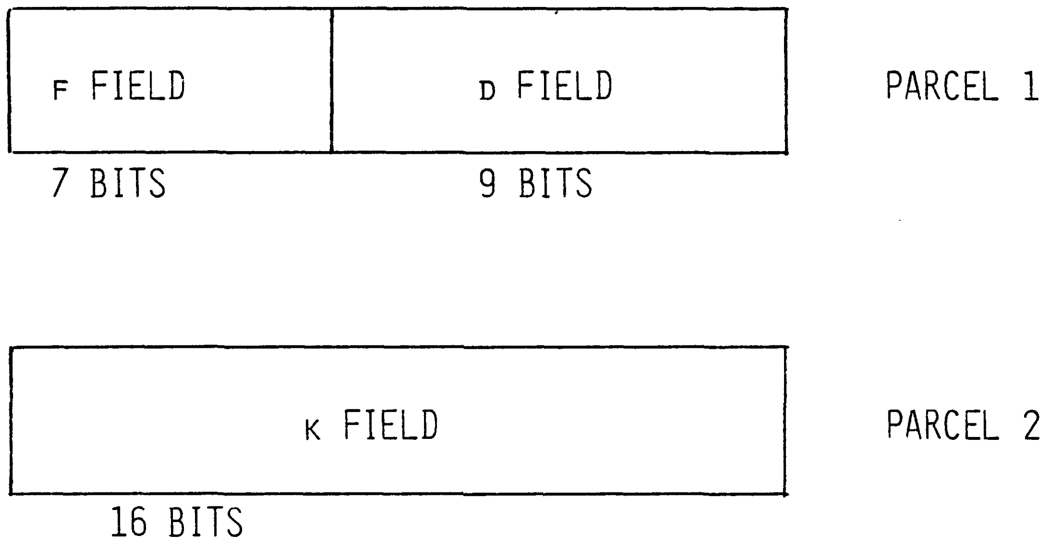


FIGURE 3-2. INSTRUCTION FORMAT

INSTRUCTION ISSUE (II) REGISTER

16 BITS WIDE

RECEIVES INSTRUCTION PARCEL TO ISSUE FROM INSTRUCTION STACK

INSTRUCTION MAY "WAIT" HERE IF ISSUE DELAYED

D FIELD MAY GO TO ADDEND REGISTER OR ACCUMULATOR

D FIELD MAY DESIGNATE AN OPERAND REGISTER OR I/O CHANNEL

K FIELD MAY GO TO ADDEND REGISTER OR ACCUMULATOR

B REGISTER

9 BITS WIDE

ALTERNATE TO D FIELD

LOADED FROM LOWER 9 BITS OF ACCUMULATOR

PROGRAM MODIFIABLE

MAY GO TO ADDEND REGISTER OR ACCUMULATOR

MAY DESIGNATE AN OPERAND REGISTER OR I/O CHANNEL

P REGISTER

16 BITS WIDE

HOLDS LOCAL MEMORY ADDRESS OF INSTRUCTION IN II REGISTER

INCREMENTS BY 1 AS EXECUTION OCCURS

BRANCHES OCCUR BY INCREMENTING OR DECREMENTING P, OR ENTERING NEW VALUE.

LOADED FROM ADD FUNCTIONAL UNIT OR EXIT STACK

CONTENTS MAY GO TO ACCUMULATOR OR EXIT STACK

PROGRAM EXIT STACK

16, 16 BIT REGISTERS

STORES RETURN ADDRESS ON SUBROUTINE CALL OR INTERRUPT

ADDRESSED BY 4 BIT E REGISTER

E INCREMENTED ON CALL AND DECREMENTED ON EXIT

E AND EXIT STACK MODIFIED BY ACCUMULATOR THROUGH CHANNEL 2

5 CP DELAY AFTER MODIFICATION NECESSARY BEFORE AN EXIT,
RETURN JUMP, OR INTERRUPT.

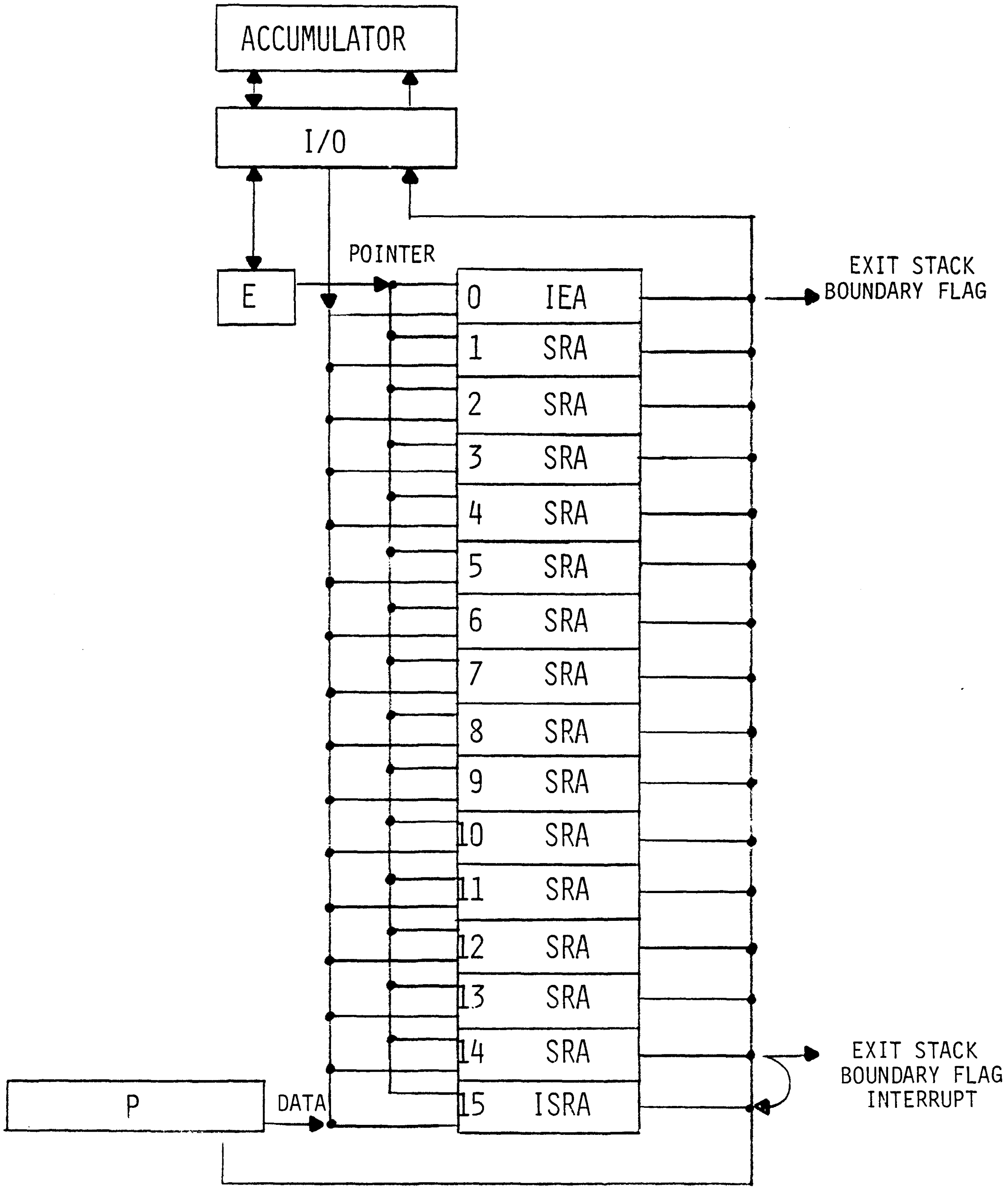
LOCATION ZERO RESERVED FOR INTERRUPT HANDLER

SOFTWARE MUST RECONFIGURE STACK WHEN FULL

CONTENTS GO TO P REGISTER ON EXIT

CONTENTS MAY GO TO ACCUMULATOR THROUGH CHANNEL 2

E MAY GO TO ACCUMULATOR THROUGH CHANNEL 2



IEA = Interrupt Entrance Address
SRA = Subroutine Return Address
ISRA = Interrupted Subroutine Return Jump Destination Address

FIGURE 3-3. PROGRAM EXIT STACK

PROGRAM FETCH REQUEST FLAG

SETS DURING EXECUTION OF JUMP INSTRUCTIONS 074-077 AND 120-137 IF THE CONTENT OF DD IS 0.

SETTING OF THIS FLAG CAUSES AN INTERRUPT AND LOADS THE REGISTER <u>NUMBER</u> OF DD INTO A 9 BIT INTERFACE REGISTER.

MONITOR PROGRAM IS ENTERED DUE TO INTERRUPT AND MAY USE THIS REGISTER NUMBER TO LOAD A SEGMENT OF CODE FOR EXECUTION.

OPERAND REGISTERS

ONLY TYPE OF ADDRESSING

512, 16 BIT REGISTERS

USED FOR TEMPORARY STORAGE, INDIRECT ADDRESSING AND AS INDEX REGISTERS.

REDUCE LOCAL MEMORY REFERENCES BY COMPUTATION SECTION

ADDRESSED BY D FIELD OR B REGISTER

DATA ENTERED ONLY FROM ACCUMULATOR

CONTENTS GO TO ACCUMULATOR OR ADDEND REGISTER

FUNCTIONAL UNITS

ADDER:

ADDS AND SUBTRACTS IN 2's COMPLEMENT MODE NO NEGATIVE #'s

OPERANDS FROM ACCUMULATOR AND ADDEND REGISTER

RESULTS TO ACCUMULATOR OR P REGISTER

CARRY BIT COMPLEMENTED IF A CARRY IS GENERATED

1 CP TO COMPLETE OPERATION

1 CP TO STORE RESULT

SHIFTER:

SHIFTS ACCUMULATOR AND CARRY BIT

LEFT, RIGHT, CIRCULAR OR END-OFF ZERO FILL

LOWER 5 BITS OF ADDEND REGISTER USED AS SHIFT COUNT

1 CP TO COMPLETE OPERATION

1 CP TO STORE RESULT IN ACCUMULATOR

'AND':

FORMS LOGICAL PRODUCT OF ACCUMULATOR AND OPERANDS AT INPUT TO ACCUMULATOR.

CARRY BIT IS CLEARED

OPERANDS SUPPLIED BY D OR K FIELD; AN OPERAND REGISTER; OR A LOCAL MEMORY LOCATION.

ACCUMULATOR

16 BITS WIDE PLUS 1 BIT CARRY REGISTER

ALWAYS SUPPLIES ONE OPERAND IF TWO REQUIRED

BRANCH INSTRUCTIONS USE BRANCH ACCUMULATOR

CARRY BIT IS CLEARED WHEN LOADING ACCUMULATOR

CARRY BIT CAN BE MODIFIED BY READING I/O CHANNEL FLAGS

SOURCES	DESTINATIONS
B REGISTER	B REGISTER
II, D FIELD	
II, K FIELD	
OPERAND REGISTERS	OPERAND REGISTERS
ADDER/SHIFTER	ADDER/SHIFTER
LOCAL MEMORY	LOCAL MEMORY
I/O CHANNELS	I/O CHANNELS
EXIT STACK	EXIT STACK
E REGISTER	E REGISTER

TABLE 3-1. ACCUMULATOR SOURCES AND DESTINATIONS

ADDEND REGISTER

16 BITS WIDE

SUPPLIES ONE OPERAND TO ADDER AND SHIFT COUNT TO SHIFTER

RECEIVES DATA FROM:

- B REGISTER
- II, D FIELD
- II, K FIELD
- OPERAND REGISTERS
- LOCAL MEMORY

INSTRUCTIONS

1 OR 2 PARCELS

UPPER 7 BITS OF FIRST PARCEL IS FUNCTION CODE

LOWER 9 BITS OF FIRST PARCEL IS POSITIVE DESIGNATOR D.

SECOND PARCEL IS A 16 BIT POSITIVE CONSTANT K

D FIELD USED AS:

- OPERAND REGISTER DESIGNATOR
- SHIFT COUNT (LOWER 5 BITS)
- DISPLACEMENT FOR BRANCH INSTRUCTION
- AN OPERAND VALUE

K FIELD USED AS:

- DISPLACEMENT FOR BRANCH INSTRUCTION
- AN OPERAND FOR THE ADDER AND LOGICAL 'AND' OPERATION

NOTATION

A	-	ACCUMULATOR
C	-	CARRY BIT
D	-	II, D FIELD
K	-	II, K FIELD
DD	-	CONTENT OF OPERAND REGISTER ADDRESSED BY D
(DD)	-	CONTENT OF MEMORY ADDRESSED BY OPERAND REGISTER DD
B	-	B REGISTER
(B)	-	CONTENT OF OPERAND REGISTER ADDRESSED BY B
IOD	-	3 CHARACTER MNEMONIC FOR CHANNEL ADDRESSED BY D FIELD
IOB	-	CHANNEL ADDRESSED BY B
,	-	IF

→ ONLY WHEN ∅, 1, 3 → 37, 44 → 67

SYS CONTROL

Arith

Branch

I/O

SYS CONTROL

Arith

Arith

000	PASS	054	B = A	124	P = dd + k, C = 0
001	EXIT	055	B = A + B	125	P = dd + k, C # 0
002	I = 0	056	B = B + 1	126	P = dd + k, A = 0
003	I = 1	057	B = B - 1	127	P = dd + k, A # 0
004	A = A > d	060	A = (B)	130	R = dd, C = 0
005	A = A < d	061	A = A & (B)	131	R = dd, C # 0
006	A = A >> d	062	A = A + (B)	132	R = dd, A = 0
007	A = A << d	063	A = A - (B)	133	R = dd, A # 0
010	A = d	064	(B) = A	134	R = dd + k, C = 0
011	A = A & d	065	(B) = A + (B)	135	R = dd + k, C # 0
012	A = A + d	066	(B) = (B) + 1	136	R = dd + k, A = 0
013	A = A - d	067	(B) = (B) - 1	137	R = dd + k, A # 0
014	A = k	070	P = P + d	140	iod : 0
015	A = A & k	071	P = P - d	141	iod : 1
016	A = A + k	072	R = P + d	142	iod : 2
017	A = A - k	073	R = P - d	143	iod : 3
020	A = dd	074	P = dd	144	iod : 4
021	A = A & dd	075	P = dd + k	145	iod : 5
022	A = A + dd	076	R = dd	146	iod : 6
023	A = A - dd	077	R = dd + k	147	iod : 7
024	dd = A	100	P = P + d, C = 0	150	iod : 10
025	dd = A + dd	101	P = P + d, C # 0	151	iod : 11
026	dd = dd + 1	102	P = P + d, A = 0	152	iod : 12
027	dd = dd - 1	103	P = P + d, A # 0	153	iod : 13
030	A = (dd)	104	P = P - d, C = 0	154	iod : 14
031	A = A & (dd)	105	P = P - d, C # 0	155	iod : 15
032	A = A + (dd)	106	P = P - d, A = 0	156	iod : 16
033	A = A - (dd)	107	P = P - d, A # 0	157	iod : 17
034	(dd) = A	110	R = P + d, C = 0	160	IOB : 0
035	(dd) = A + (dd)	111	R = P + d, C # 0	161	IOB : 1
036	(dd) = (dd) + 1	112	R = P + d, A = 0	162	IOB : 2
037	(dd) = (dd) - 1	113	R = P + d, A # 0	163	IOB : 3
040	C = 1, iod = DN	114	R = P - d, C = 0	164	IOB : 4
041	C = 1, iod = BZ	115	R = P - d, C # 0	165	IOB : 5
042	C = 1, IOB = DN	116	R = P - d, A = 0	166	IOB : 6
043	C = 1, IOB = BZ	117	R = P - d, A # 0	167	IOB : 7
044	A = A > B	120	P = dd, C = 0	170	IOB : 10
045	A = A < B	121	P = dd, C # 0	171	IOB : 11
046	A = A >> B	122	P = dd, A = 0	172	IOB : 12
047	A = A << B	123	P = dd, A # 0	173	IOB : 13
050	A = B			174	IOB : 14
051	A = A & B			175	IOB : 15
052	A = A + B			176	IOB : 16
053	A = A - B			177	IOB : 17

4 classes

TABLE 3-2. I/O PROCESSOR INSTRUCTION SUMMARY

SYSTEM CONTROL INSTRUCTIONS

000 - PASS

ACTS AS A NO-OP

001 - EXIT

RETURNS CONTROL TO SUBROUTINE CALLER OR INTERRUPTED ROUTINE.

002 - I=0

CLEARS SYSTEM INTERRUPT ENABLE FLAG, LOCKING OUT INTERRUPTS.

003 - I=1

SETS SYSTEM INTERRUPT FLAG, ALLOWING INTERRUPTS. DELAYED UNTIL COMPLETION OF A 000,001,003 TO 037 OR 044 TO 067 INSTRUCTION.

040 - C=1, IOD=DN

041 - C=1, IOD=BZ

042 - C=1, IOB=DN

043 - C=1, IOB=BZ

FORCES CARRY BIT TO SAME STATE AS SPECIFIED CHANNEL'S DONE (DN) OR BUSY (BZ) FLAG.

ARITHMETIC INSTRUCTIONS

THE FOLLOWING OPERATIONS ARE AVAILABLE:

- ADD
- SUBTRACT
- SHIFT
- LOGICAL PRODUCT
- LOAD
- STORE
- INCREMENT
- DECREMENT

WHEN ANY ARITHMETIC INSTRUCTION COMPLETES THE RESULT IS ALSO IN THE ACCUMULATOR.

JUMP INSTRUCTIONS

8 UNCONDITIONAL JUMPS 070-077

32 CONDITIONAL JUMPS 100-137 FORMED BY APPENDING THE FOLLOWING CONDITIONS:

,C=0
,C#0
,A=0
,A#0

6 BASIC TYPES OF JUMPS:

1. RELATIVE JUMPS WITH D AS OFFSET

070	P=P+D
071	P=P-D

2. RELATIVE RETURN JUMPS WITH D AS OFFSET

072	R=P+D
073	R=P-D

3. ABSOLUTE JUMP TO ADDRESS IN OPERAND REGISTER

074	P=DD

4. ABSOLUTE JUMP TO SUM OF ADDRESS IN OPERAND REGISTER AND K.

075	P=DD+K

5. ABSOLUTE RETURN JUMP TO ADDRESS IN OPERAND REGISTER

076	R=DD

6. ABSOLUTE RETURN JUMP TO SUM OF ADDRESS IN OPERAND REGISTER AND K.

077	R=DD+K

CHANNEL CONTROL INSTRUCTIONS

16 POSSIBLE PER CHANNEL

EACH CHANNEL INTERPRETS INSTRUCTION IN UNIQUE WAY

CHANNELS MAY RECOGNIZE SUBFUNCTIONS SPECIFIED IN ACCUMULATOR

CHANNEL SELECTED BY D FIELD, 140-157; OR B REGISTER 160-177

CHAPTER 4

I/O SECTION

CONFIGURATION

12 DEDICATED CHANNELS REQUIRED BY EACH IOP

$0 .. 13_8$ = STANDARD CHANNELS

28 OPTIONAL CHANNELS WHICH MAY BE IMPLEMENTED DIFFERENTLY BY EACH IOP.

THESE MAY USE UP TO 5 DMA PORTS

CHANNELS NUMBERED OCTALLY $0 .. 47_8$

INPUT CHANNELS EVEN; OUTPUT CHANNELS ODD

ACCUM CHANNELS 16 BIT Control Between A reg and Some I/O device. (ie. cause interrupt)

DMA PORTS ALLOW BLOCK TRANSFERS

MAY MULTIPLEX SEVERAL DEVICES THROUGH ONE PORT

ACCUM CHAN + DATA PATH TO LOCAL MEM.

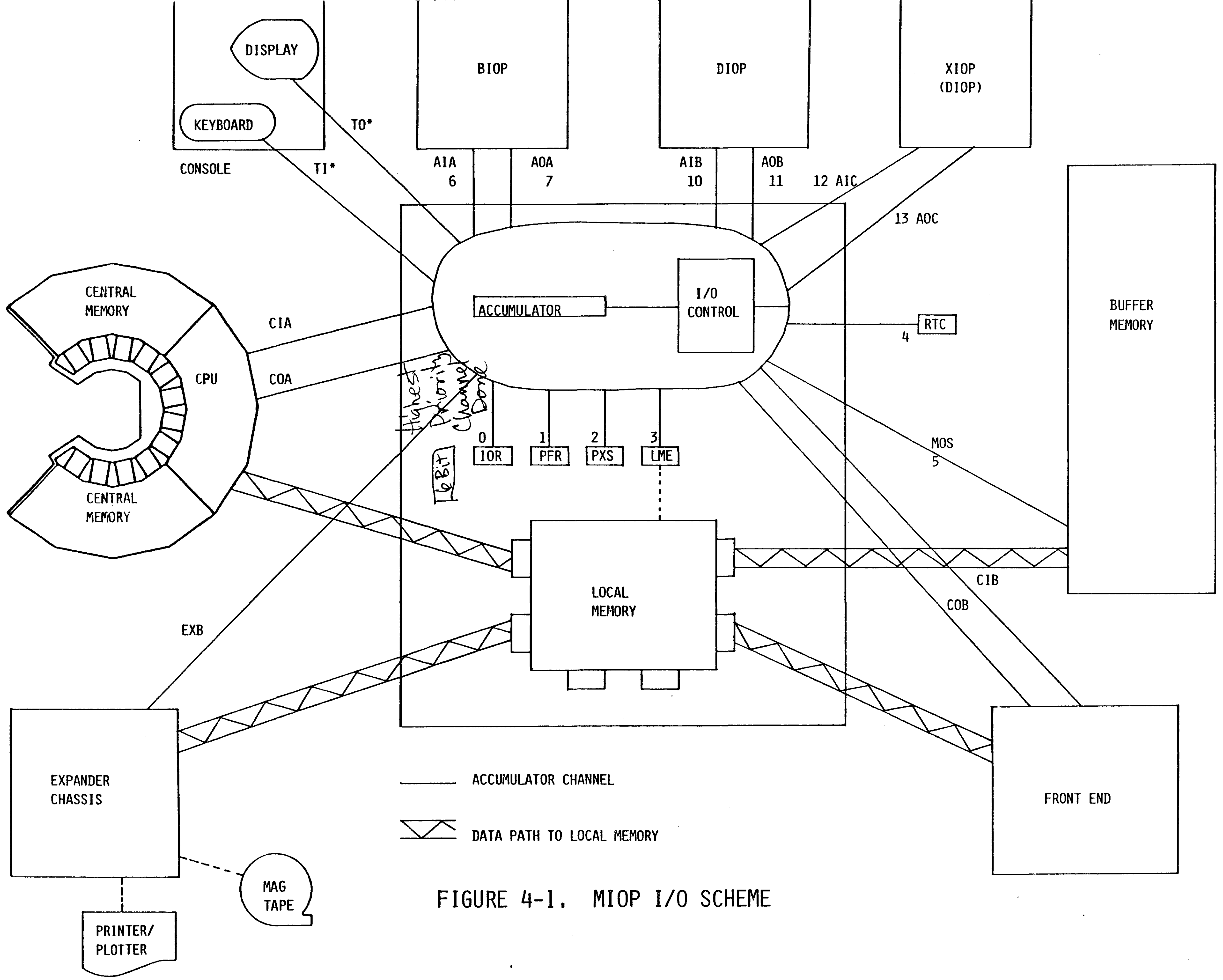


FIGURE 4-1. MIOP I/O SCHEME

DEVICE	CHANNEL	MNEMONIC
I/O INTERRUPT REQUEST	0	IOR
PROGRAM FETCH REQUEST	1	PFR (No Busy Flag)
PROGRAM EXIT STACK	2	PXS
LOCAL MEMORY ERROR	3	LME
REAL TIME CLOCK	4	RTC
BUFFER MEMORY	5	MOS
IOP INPUT	6,10,12	AI*
IOP OUTPUT	7,11,13	AO*
INPUT FROM CPU (FRONT END)	OPTIONAL	CI*
OUTPUT TO CPU (FRONT END)	OPTIONAL	CO*
INPUT FROM CENTRAL MEMORY	OPTIONAL	HIA
OUTPUT TO CENTRAL MEMORY	OPTIONAL	HOA
ERROR LOG	OPTIONAL	ERA
CONSOLE KEYBOARD	OPTIONAL	TI*
CONSOLE DISPLAY	OPTIONAL	TO*
DISK STORAGE UNIT	OPTIONAL	DK*
PERIPHERAL EXPANDER	OPTIONAL	EXB
BLOCK MULTIPLEXER	OPTIONAL	BM*

TABLE 4-1. CHANNEL ASSIGNMENTS

*A,B,C,...

Channel Priority

$1 .. 47_8, \phi$ [illegible]

Always DONE Never Busy

Channel Instruction

Returns Channel Addr of Highest Priori

Done!

Functions

:∅ CLR BZ & DN

:6 LOCAL DISABLE INT

:7 LOCAL ENABLE INT

ACCUMULATOR CHANNELS

USED MAINLY FOR CONTROL

TRANSFER ONE PARCEL OF DATA TO OR FROM ACCUMULATOR

DEDICATED CHANNELS 0-4 AND 6-13

DISPLAY, KEYBOARD AND ERROR LOGGING INTERFACE CHANNELS

OUTPUT FROM ACCUMULATOR TAKES 1 CP (IF ACCUMULATOR READY)

INPUT TO ACCUMULATOR TAKES 4-6 CP.

DMA CHANNELS

HAVE A PATH (ACCUMULATOR CHANNEL) TO COMPUTATION SECTION FOR PASSING CONTROL SIGNALS.

HAVE DATA PATHS (THROUGH DMA PORT) TO LOCAL MEMORY.

TRANSFER 4 PARCELS PER READ OR WRITE REQUEST.

MAXIMUM TRANSFER RATE OF 4 PARCELS IN 6 CP.

SIMULTANEOUS INPUT AND OUTPUT VIA SEPARATE PORTS.

DEDICATED CHANNEL 5.

CPU HIGH AND LOW SPEED, DISK, BLOCK MULTIPLEXER AND PERIPHERAL EXPANDER CHANNEL.

OVERVIEW OF I/O

CHANNEL INSTRUCTION SENT TO INTERFACE SPECIFIED IN D OR B FOR INTERPRETATION.

ALL CONTROL INFORMATION PASSED IN AN I/O INSTRUCTION GOES THROUGH THE ACCUMULATOR.

DATA TRANSFERS MAY BE SINGLE PARCELS OR BLOCKS OF DATA.

AN INTERFACE MAY REQUIRE SEVERAL I/O INSTRUCTIONS TO ACCOMPLISH A DATA TRANSFER.

CHANNEL STATE MONITORED THROUGH INTERFACE BUSY (BZ) AND DONE (DN) FLAGS.

DEDICATED CHANNELS

I/O REQUEST CHANNEL 0

READS HIGHEST PRIORITY INTERRUPTING CHANNEL NUMBER.

IOR:10   READ INTERRUPTING CHANNEL NUMBER

-LOADS LOWER 6 BITS OF ACCUMULATOR WITH HIGHEST PRIORITY INTERRUPTING CHANNEL.

-DN ALWAYS SET, BZ ALWAYS CLEAR

PROGRAM FETCH REQUEST CHANNEL 1

READS NUMBER OF OPERAND REGISTER WHOSE CONTENT WAS ZERO IN AN 074-077 AND 120-137 INSTRUCTION. MONITOR MAY THEN FETCH APPROPRIATE SEGMENT OF CODE FOR EXECUTION.

PFR:0	CLEAR PFR FLAG. THERE IS NO BUSY FLAG.
PFR:6	CLEAR CHANNEL INTERRUPT ENABLE FLAG (IEF).
PFR:7	SET IEF
PFR:10	LOAD ACCUMULATOR WITH OPERAND REGISTER NUMBER AND CLEAR PFR FLAG.

Device	Mnemonic	Function
I/O REQUEST CH. 0	IOR : 10	Read interrupt channel number
PROGRAM FETCH REQUEST CH. 1	PFR : 0	Clear the program fetch request flag
	PFR : 6	Clear the channel interrupt enable flag
	PFR : 7	Set the channel interrupt enable flag
	PFR : 10	Read the operand register number
PROGRAM EXIT STACK CH. 2	PXS : 0	Clear the exit stack boundary flag
	PXS : 6	Clear the channel interrupt flag
	PXS : 7	Set the channel interrupt enable flag
	PXS : 10	Read exit stack pointer, E
	PXS : 11	Read exit stack address, (E)
	PXS : 14	Enter exit stack pointer, E
	PXS : 15	Enter exit stack address, (E)
I/O MEMORY ERROR CH. 3	LME : 0	Clear the I/O Memory parity error flag
	LME : 6	Clear the channel interrupt enable flag
Bank Section 4 Byte	LME : 7	Set the channel interrupt enable flag
	LME : 10	Read error information
REAL-TIME CLOCK CH. 4	RTC : 0	Clear the channel done flag
	RTC : 6	Clear the channel interrupt enable flag
	RTC : 7	Set the channel interrupt enable flag
BUFFER MEMORY CH. 5	MOS : 0	Clear the channel busy and done flags
	MOS : 1	Enter the I/O Memory address for next transfer
	MOS : 2	Enter upper portion of Buffer Memory address
	MOS : 3	Enter lower portion of Buffer Memory address
	MOS : 4	Read Buffer Memory to I/O Memory
	MOS : 5	Write Buffer Memory from I/O Memory
	MOS : 6	Clear the channel interrupt enable flag
	MOS : 7	Set the channel enable interrupt flag
	MOS : 14	Set the control flags
I/O PROCESSOR INPUT (AIA-AIC) CH. 6, 10, 12	AI* : 0	Clear the channel done flag
	AI* : 6	Clear the channel interrupt enable flag
	AI* : 7	Set the channel interrupt enable flag
	AI* : 10	Read input to accumulator and resume channel
I/O PROCESSOR OUTPUT (AOA-AOC) CH. 7, 11, 13	AO* : 0	Clear the channel busy and done flags
	AO* : 1	Enter control bits from accumulator
	AO* : 6	Clear the channel interrupt enable flag
	AO* : 7	Set the channel interrupt enable flag
	AO* : 14	Set the channel busy flag and output accumulator data.

TABLE 4-2. DEDICATED CHANNEL FUNCTIONS

PROGRAM EXIT STACK CHANNEL 2

PROVIDES INFORMATION NECESSARY TO RESTRUCTURE STACK.

PXS:0	CLEAR EXIT STACK BOUNDARY FLAG. NO BUSY FLAG.
PXS:6	CLEAR IEF
PXS:7	SET IEF
PXS:10	E TO A, CLEAR C
PXS:11	(E) TO A, CLEAR C
PXS:14	A TO E
PXS:15	A TO (E)

I/O MEMORY ERROR CHANNEL 3

CONNECTED TO I/O MEMORY ERROR DETECTION CIRCUITS.
PROVIDES ERROR INFORMATION FOR MAINTENANCE.

LME:0	CLEAR PARITY ERROR FLAG
LME:6	CLEAR IEF
LME:7	SET IEF
LME:10	LOAD LOWER 5 BITS OF ACCUMULATOR WITH ADDRESS OF MEMORY ERROR. THIS GIVES BANK, SECTION AND BYTE OF ERROR.

REAL TIME CLOCK CHANNEL 4

CONNECTED TO RTC WITH 1MS INTERRUPT INTERVAL.
NO BUSY FLAG OR INTERFACE REGISTERS.
DONE FLAG SETS EVERY MILLISECOND.
READABLE RTC ~~OPTION~~ ON MODEL B (SN 6, 11+)

RTC:0	CLEAR DN FLAG
RTC:6	CLEAR IEF
RTC:7	SET IEF
RTC:10	READ ~~LOW~~ HIGH ORDER BITS OF RTC (2^{1} TO 2^{16})

BUFFER MEMORY CHANNEL 5

PERFORMS BLOCK TRANSFERS THROUGH A DEDICATED DMA PORT.
THREE INTERFACE REGISTERS: (HALF DUPLEX)

A) 24 BIT BUFFER MEMORY ADDRESS REGISTER
B) 14 BIT LOCAL MEMORY ADDRESS REGISTER
C) 14 BIT BUFFER MEMORY BLOCK LENGTH

MOS:0	CLEAR DN AND BZ FLAGS (BOTH SET ON ERROR). MUST BE DONE AFTER EVERY DOUBLE BIT ERROR BEFORE NEXT TRANSFER.
MOS:1	LOAD B) WITH UPPER 14 BITS OF ACCUMULATOR
MOS:2	LOAD UPPER 15 BITS OF A) WITH LOWER 15 BITS OF ACCUMULATOR
MOS:3	LOAD LOWER 9 BITS OF A) WITH LOWER 9 BITS OF ACCUMULATOR
MOS:4	LOAD C) WITH LOWER 14 BITS OF ACCUMULATOR. START BUFFER TO LOCAL BLOCK TRANSFER.
MOS:5	LOAD C) WITH LOWER 14 BITS OF ACCUMULATOR. START LOCAL TO BUFFER BLOCK TRANSFER.
MOS:6	CLEAR IEF
MOS:7	SET IEF
MOS:14	LOAD INTERFACE CONTROL REGISTER WITH LOWER 3 BITS OF ACCUMULATOR (DIAGNOSTICS ONLY).

I/O PROCESSOR INPUT CHANNEL 6, 10, 12

16 BIT INTERFACE REGISTER HOLDS DATA FROM ANOTHER IOP'S ACCUMULATOR.

AI*:0	CLEAR DN FLAG. NO BUSY FLAG.
AI*:6	CLEAR IEF
AI*:7	SET IEF
AI*:10	READ INTERFACE TO ACCUMULATOR. THIS CLEARS INTERFACE REGISTER.

I/O PROCESSOR OUTPUT CHANNEL 7, 11, 13

ALLOWS IOP TO MASTER CLEAR, DEADSTART AND DEAD DUMP ANOTHER IOP THROUGH A 3 BIT CONTROL REGISTER.
16 BIT REGISTER HOLDS DATA FOR ANOTHER IOP.

AO*:0	CLEAR BZ AND DN.
AO*:1	LOAD CONTROL REGISTER WITH LOWER 3 BITS OF ACCUMULATOR. 2^0=MASTER CLEAR: 2^1=DEADSTART: 2^2=DEAD DUMP.
AO*:6	CLEAR IEF
AO*:7	SET IEF
AO*:14	LOAD INTERFACE REGISTER WITH ACCUMULATOR. DN FLAG SETS WHEN TARGET IOP PERFORMS AN AI*:10.

INTERFACES

MAIN PURPOSES ARE:

BUFFERING DATA

GENERATING CONTROL SIGNALS

MULTIPLEXING SEVERAL DEVICES INTO ONE CHANNEL

INTERPRET THE 4 BIT FUNCTION CODE SENT BY COMPUTATION SECTION.

USE BZ AND DN FLAGS FOR CONTROL.

THE FOLLOWING FUNCTIONS ARE COMMON TO MOST INTERFACES:

IOD:0 OR IOB:0	CLEAR DN AND BZ, READY CHANNEL
IOD:6 OR IOB:6	CLEAR IEF
IOD:7 OR IOB:7	SET IEF

Device	Mnemonic	Function
DISK STORAGE UNIT (DKA-DKP)	DK* : 0	Clear the channel control
	DK* : 1	Select mode or request status
	DK* : 2	Read data into I/O Memory
	DK* : 3	Write data from I/O Memory
	DK* : 4	Select a new head group
	DK* : 5	Select a new cylinder
	DK* : 6	Clear the channel interrupt enable flag
	DK* : 7	Set the channel interrupt enable flag
	DK* : 10	Read I/O Memory current address
	DK* : 11	Read status response
	DK* : 14	Enter I/O Memory beginning address
	DK* : 15	Status response register diagnostic
CONSOLE KEYBOARD (TIA,TIB,TIC,...)	TI* : 0	Clear the channel done flag
	TI* : 6	Clear the channel interrupt enable flag
	TI* : 7	Set the channel interrupt enable flag
	TI* : 10	Read data into accumulator and clear done flag
CONSOLE DISPLAY (TOA,TOB,TOC,...)	TO* : 0	Clear the channel busy and done flags
	TO * : 6	Clear the channel interrupt enable flag
	TO * : 7	Set the channel interrupt enable flag
	TO * : 14	Send accumulator data to display
EXPANDER CHASSIS	EXB : 0	Idle the channel
	EXB : 1	Data input from A register (DIA)
	EXB : 2	Data input from B register (DIB)
	EXB : 3	Data input from C register (DIC)
	EXB : 4	Read busy/done flag, interrupt number
	EXB : 5	Load device address
	EXB : 6	Send interface mask (MSKO)
	EXB : 7	Set interrupt mode
	EXB : 10	Read data bus status
	EXB : 11	Read status 1
	EXB : 13	Read status 2
	EXB : 14	Data output to A register (DOA)
	EXB : 15	Data output to B register (DOB)
	EXB : 16	Data output to C register (DOC)
	EXB : 17	Send control

TABLE 4-3. INTERFACE FUNCTIONS

Device	Mnemonic	Function
INPUT FROM CPU TYPE I/O CHANNEL (CIA,CIB,CIC...)	CI* : 0	Clear channel
	CI* : 1	Enter I/O Memory address, start input
	CI* : 2	Enter parcel count
	CI* : 3	Clear channel parity error flags
	CI* : 4	Clear ready waiting flag
	CI* : 6	Clear interrupt enable flag
	CI* : 7	Set interrupt enable flag
	CI* : 10	Real I/O Memory address
	CI* : 11	Read status (ready waiting, parity errors)
OUTPUT TO CPU TYPE I/O CHANNEL (COA,COB,COC...)	CO* : 0	Clear channel
	CO* : 1	Enter I/O Memory address
	CO* : 2	Enter parcel count
	CO* : 3	Clear error flag
	CO* : 4	Set/clear external control signals
	CO* : 6	Clear interrupt enable flag
	CO* : 7	Set interrupt enable flag
	CO* : 10	Read I/O Memory address
	CO* : 11	Read status (4-bit channel data, error)
INPUT FROM CPU MEMORY CHANNEL	HIA : 0	Clear channel busy, done flags
	HIA : 1	Enter I/O Memory address
	HIA : 2	Enter upper CP memory address
	HIA : 3	Enter lower CP memory address
	HIA : 4	Read CP memory, enter block length
	HIA : 6	Clear interrupt enable flag
	HIA : 7	Set interrupt enable flag
	HIA : 14	Enter diagnostic mode
OUTPUT TO CPU MEMORY CHANNEL	HOA : 0	Clear channel busy, done flags
	HOA : 1	Enter I/O Memory address
	HOA : 2	Enter upper CP memory address
	HOA : 3	Enter lower CP memory address
	HOA : 5	Write CP memory, enter block length
	HOA : 6	Clear interrupt enable flag
	HOA : 7	Set interrupt enable flag
	HOA : 14	Enter diagnostic mode

TABLE 4-3. INTERFACE FUNCTIONS (CONTINUED)

Device	Mnemonic	Function
ERROR LOGGING CHANNEL	ERA : 0	Idle channel
	ERA : 6	Clear interrupt enable flag
	ERA : 7	Set interrupt enable flag
	ERA : 10	Read error status
	ERA : 11	Read error information (first parameter)
	ERA : 12	Read error information (second parameter)
	ERA : 13	Read error information (third parameter)
BLOCK MULTIPLEXER CHANNEL (BMA,BMB,BMC...)	BM* : 0	Clear channel control
	BM* : 1	Send reset functions
	BM* : 2	Channel command
	BM* : 3	Read request in address (wait request in)
	BM* : 4	Single byte I/O (wait service in/status in)
	BM* : 5	Interface disconnect
	BM* : 6	Clear channel interrupt enable flag
	BM* : 7	Set channel interrupt enable flag
	BM* : 10	Read I/O Memory address
	BM* : 11	Read byte count
	BM* : 12	Read status
	BM* : 13	Read input tags
	BM* : 14	Enter I/O Memory address
	BM* : 15	Enter byte count
	BM* : 16	Enter address
	BM* : 17	Enter output tags

TABLE 4-3. INTERFACE FUNCTIONS (CONTINUED)

INTERFACE CHANNELS

DISK CHANNEL DKA → DKP

TRANSFER DATA TO/FROM DISK STORAGE UNITS
4 CHANNELS PER DCU-4 CONTROLLER
UP TO 16 DISK CHANNELS ON EACH BIOP AND DIOP

CONSOLE KEYBOARD CHANNEL TI*

ACCEPTS INPUT FROM KEYBOARD, ONE CHARACTER AT A TIME.
1 CHANNEL PER CONSOLE

CONSOLE DISPLAY CHANNEL TO*

SENDS OUTPUT TO DISPLAY, ONE CHARACTER AT A TIME
1 CHANNEL PER CONSOLE

EXPANDER CHASSIS CHANNEL EXB

TRANSFERS DATA TO/FROM MAG TAPE AND TO PRINTER
1 CHANNEL ON MIOP

INPUT FROM CPU I/O CHANNEL CIA → CID

ACCEPTS INPUT FROM CPU AND FRONT ENDS.
UP TO 4 CHANNELS ON MIOP

OUTPUT TO CPU I/O CHANNEL COA $\rightarrow$ COD

OUTPUTS DATA TO CPU AND FRONT ENDS.
PROVIDES IOS WITH CPU DEADSTART CAPABILITY
UP TO 4 CHANNELS ON MIOP

INPUT FROM CPU MEMORY CHANNEL HIA

ACCEPTS DATA DIRECTLY FROM CENTRAL MEMORY INTO
BIOP LOCAL MEMORY.
1 CHANNEL ON BIOP
CAPABLE OF TRANSFER RATES IN EXCESS OF 800 MBIT/S

OUTPUT TO CPU MEMORY CHANNEL HOA

OUTPUTS DATA DIRECTLY TO CENTRAL MEMORY FROM BIOP
LOCAL MEMORY.
1 CHANNEL ON BIOP
CAPABLE OF TRANSFER RATES IN EXCESS OF 800 MBIT/S

ERROR LOGGING CHANNEL ERA

REPORTS ERRORS FROM THE FOLLOWING SOURCES:
- OTHER LOCAL MEMORIES
- BUFFER MEMORY
- CENTRAL MEMORY
- CPU MEMORY CHANNELS

1 CHANNEL ON MIOP

BLOCK MULTIPLEXER CHANNEL BMA → BMP

PROVIDES ACCESS TO IBM PLUG-COMPATIBLE PERIPHERALS
UP TO 16 CHANNELS ON XIOP

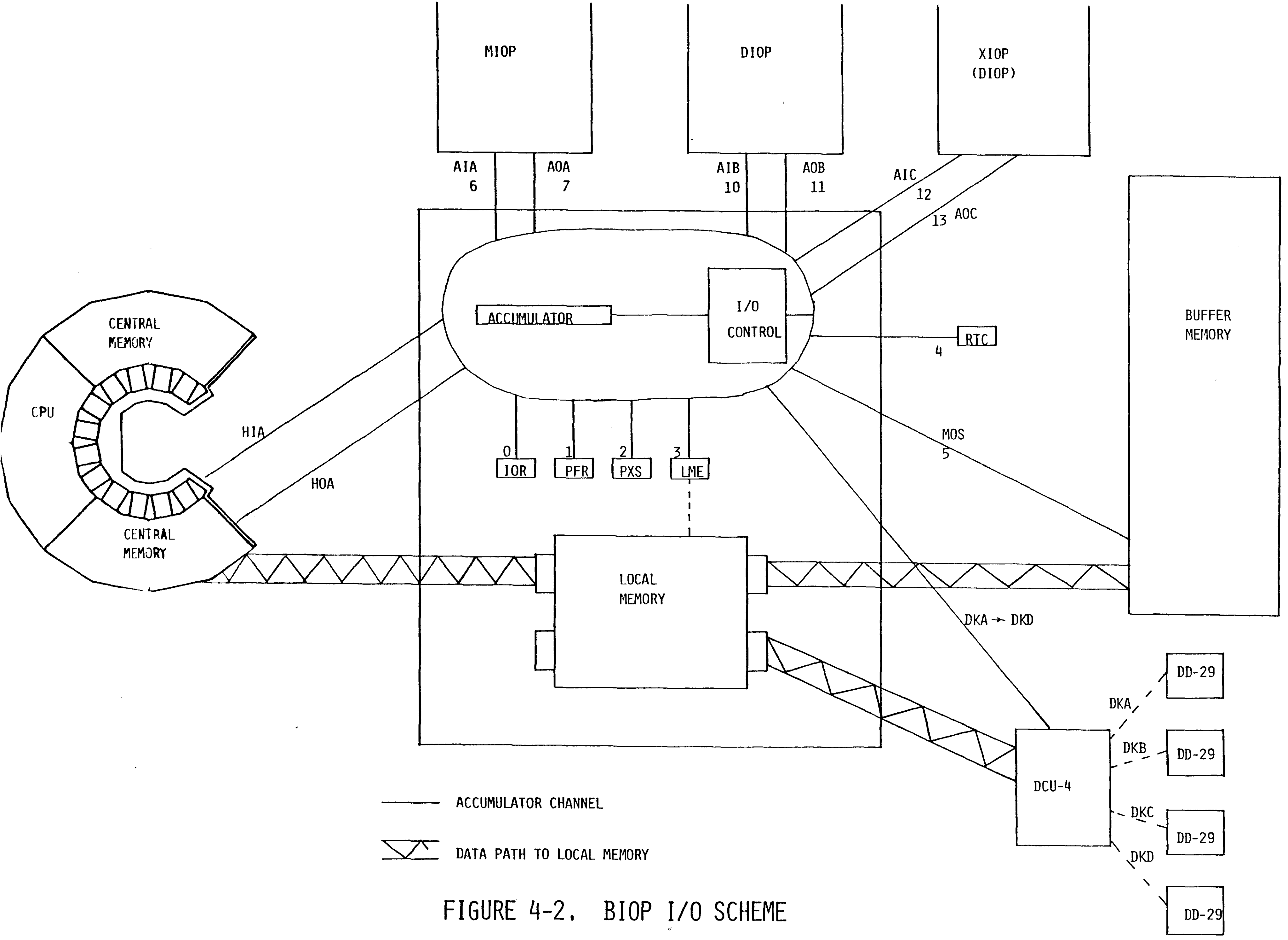


FIGURE 4-2. BIOP I/O SCHEME

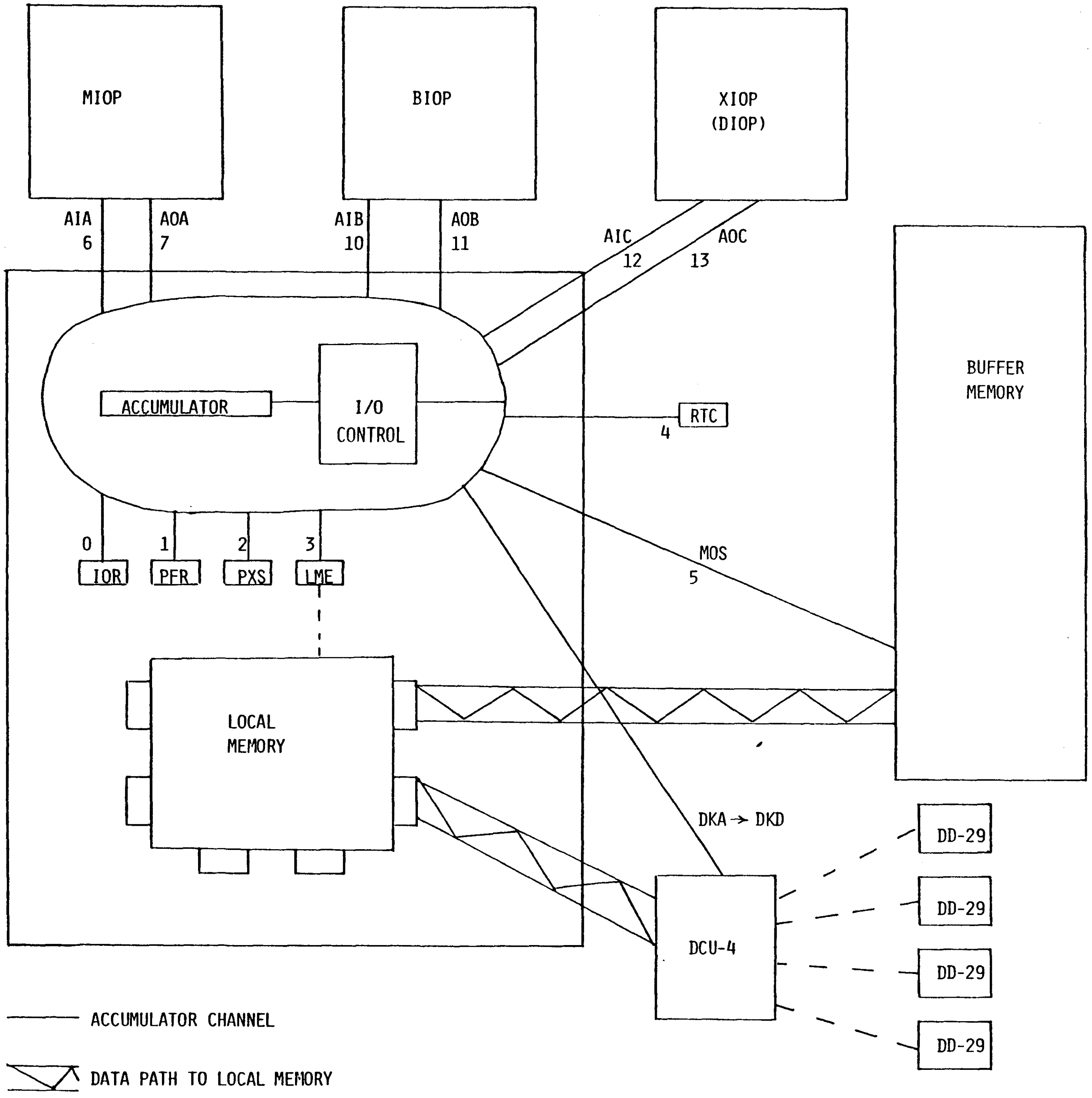
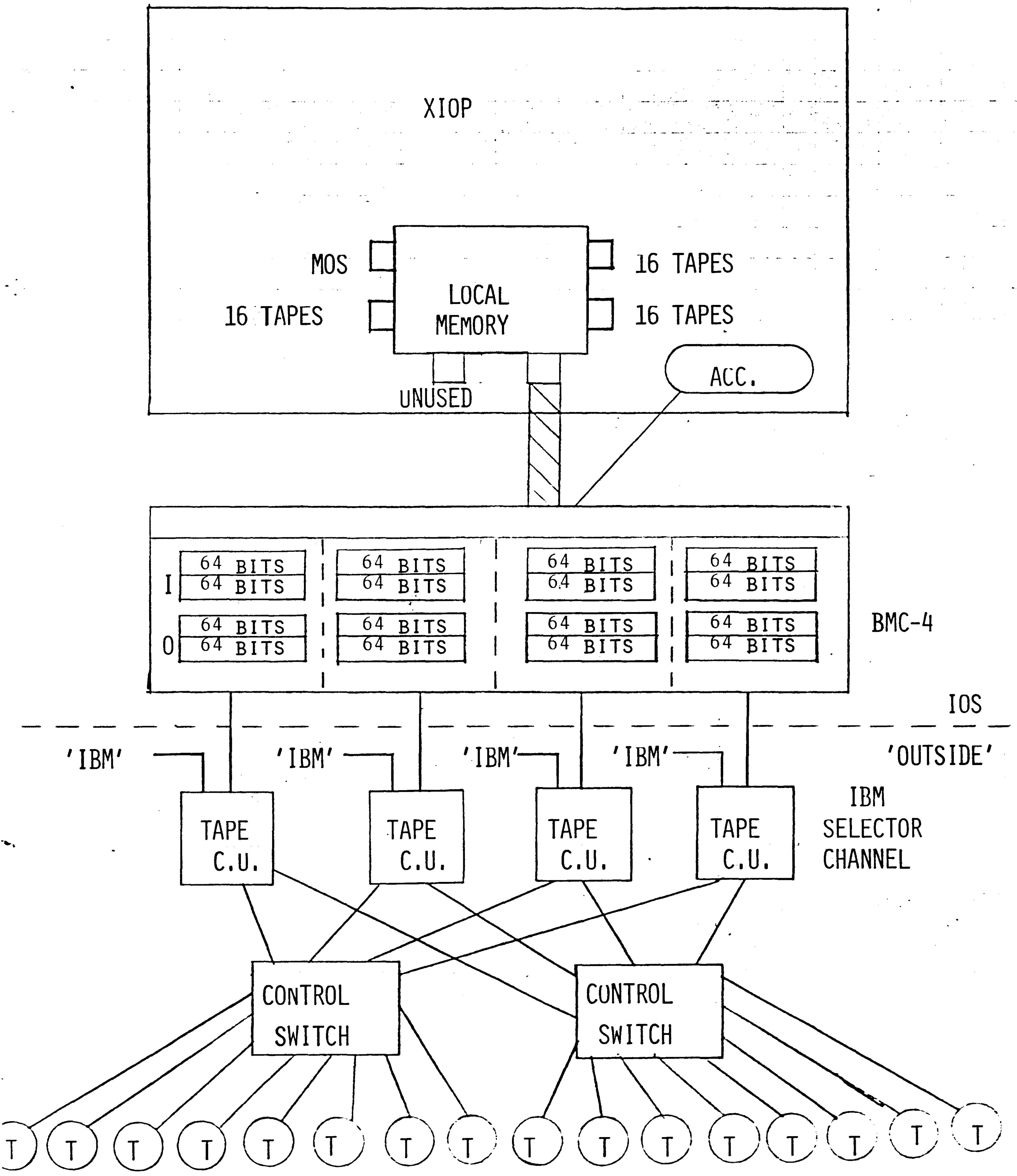


FIGURE 4-3. DIOP I/O SCHEME



4 x 16 TAPE CONFIGURATION

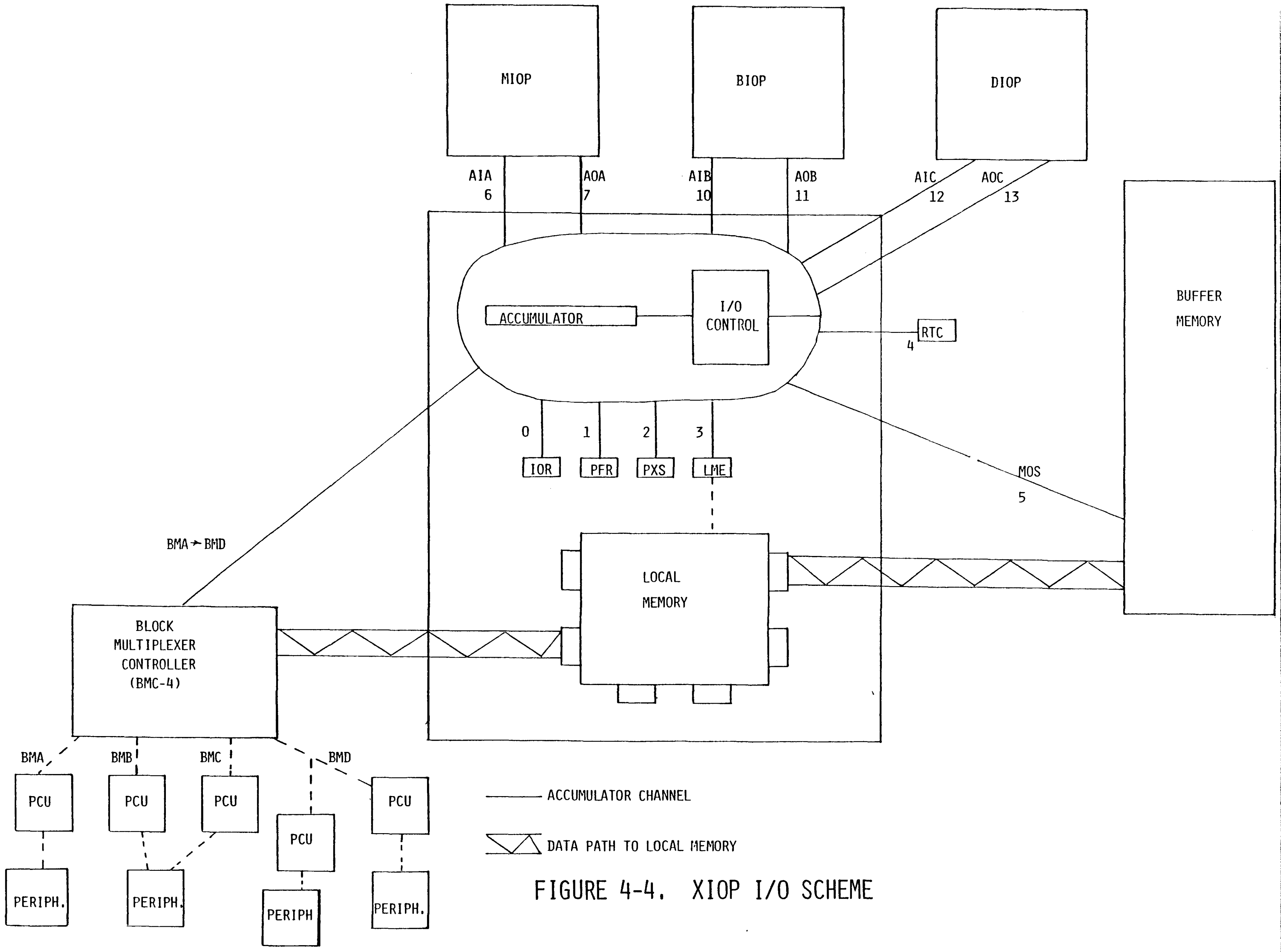


FIGURE 4-4. XIOP I/O SCHEME

CHAPTER 5

BUFFER MEMORY

FUNCTIONS

BUFFERS DATA TO/FROM DISK, TAPE AND FRONT ENDS.

USED AS A DISK CACHE.

PROVIDES SPACE FOR OVERLAYS USED BY I/O SUBSYSTEM OPERATING SYSTEM.

PROVIDES SPACE FOR PASSING LARGE MESSAGES BETWEEN IOPs.

USER BUFFER MEMORY RESIDENT DATASETS

? COS OVERLAYS, ROLL JOB MEMORY

CHARACTERISTICS

1/2 OR 1 MILLION 64 BIT WORDS IN 8 OR 16 BANK MODE
(1,4, 8 MW model B) (1-4 8 bank only)

NEGATIVE CHANNEL METAL OXIDE SEMICONDUCTOR (NMOS) CIRCUITRY

REQUIRES REFRESH EVERY 2 MS 5-20% Degradation

LARGE SCALE INTEGRATION (16K CHIPS ON MODEL A; 64 K CHIPS ON MODEL B)

RESIDES IN I/O SUBSYSTEM CHASSIS

12 CP ACCESS TIME

30 CP BANK BUSY TIME (32 CP Bank Busy time (Model B))

4 ACCESS PORTS

TWO FUNCTIONS NECESSARY TO PASS 24 BIT ADDRESS

PROTECTED BY SECDED LOGIC

Model A	.5 Mw	1280	Mbits/sec	all ports
"	1 mw	2560	Mbits/sec	
Model B	1 mw	1024	M bits/sec	
"	8 mw	2048	"	

PART 2

APML

CHAPTER 6

APML SYNTAX

IN AN ASSEMBLY LANGUAGE THERE IS A ONE-TO-ONE CORRESPONDENCE BETWEEN INSTRUCTIONS AND MACHINE CODE.

IN A MACRO LANGUAGE ONE INSTRUCTION MAY GENERATE SEVERAL MACHINE INSTRUCTIONS.

ASSEMBLY LANGUAGE vs MACRO LANGUAGE

EXAMPLE OF ASSEMBLED CAL CODE:

```
                                                IDENT      CAL
                                                START      BEGIN
0   0000000000000000000012      NUM             CON        10
1                        1      SUM             BSS        1
                       2a+      BEGIN           =          *
2a  1001 00000000+                              A1         NUM,0
 c  022201                                      A2         1
 d  022302                                      A3         2
3a  022400                                      A4         0
 b  031110                      LOC             A1         A1-1
 c  030442                                      A4         A4+A2
 d  030223                                      A2         A2+A3
4a  030001                                      A0         A1
 b  011 00000003b+                              JAN        LOC
 d  1104 00000001+                              SUM,0      A4
                                                ENDP
                                                END
```

EXAMPLE OF ASSEMBLED APML CODE:

```
                                                IDENT          APML
                                      6         REGISTER       (R1,R2,R3)
                                                SCRATCH        R3
 0    010012   024006                           R1=12    R3 = 0
 2    010001   024007                           R2=1
 4    027006                          LOC       R1=R1-1
 5    010002   025007                           R2=R2+2  (R3 = R3 + R2)
 7    020006   107004                           P=LOC,R1#0
11    014000  /000016   024010                  (SUM)=R2
      020007   034010
16                                    SUM       <1>
                                                END
```

APML FEATURES

1) EXTREMELY FLEXIBLE ASSIGNMENT AND CONDITION SYNTAX

2) USES MOST CAL PSEUDOS

3) KEY SYMBOL DEPENDENT

APML NOTATION

SYMBOL	MEANING
A	ACCUMULATOR
B	B REGISTER (OPERAND REGISTER INDEX)
(B)	CONTENT OF OPERAND REGISTER ADDRESSED BY B
C	CARRY BIT
DD	CONTENT OF OPERAND REGISTER (TWO CHARACTER SYMBOL).
R!	PREFIXES A SYMBOL TO REFER TO AN OPERAND REGISTER.
[DD] → NOT R!.	OPERAND REGISTER NUMBER OF DD
(DD)	CONTENT OF MEMORY ADDRESSED BY THE CONTENT OF OPERAND REGISTER DD.
(DD+K)	CONTENT OF MEMORY ADDRESSED BY THE SUM OF THE CONTENT OF OPERAND REGISTER DD AND K.
E	EXIT STACK POINTER
(E)	EXIT STACK ENTRY ADDRESSED BY E
I	INTERRUPT ENABLE FLAG
IOB	I/O CHANNEL DEFINED BY THE CONTENT OF B
IOD	I/O CHANNEL MNEMONIC DEFINED BY CHANNEL PSEUDO INSTRUCTION.
K	POSITIVE NUMERIC OR CHARACTER CONSTANT OR SYMBOL.
(K)	CONTENT OF MEMORY ADDRESSED BY THE VALUE K
P	P REGISTER (PROGRAM ADDRESS REGISTER)
R	INDICATES RETURN JUMP

APML FORMAT

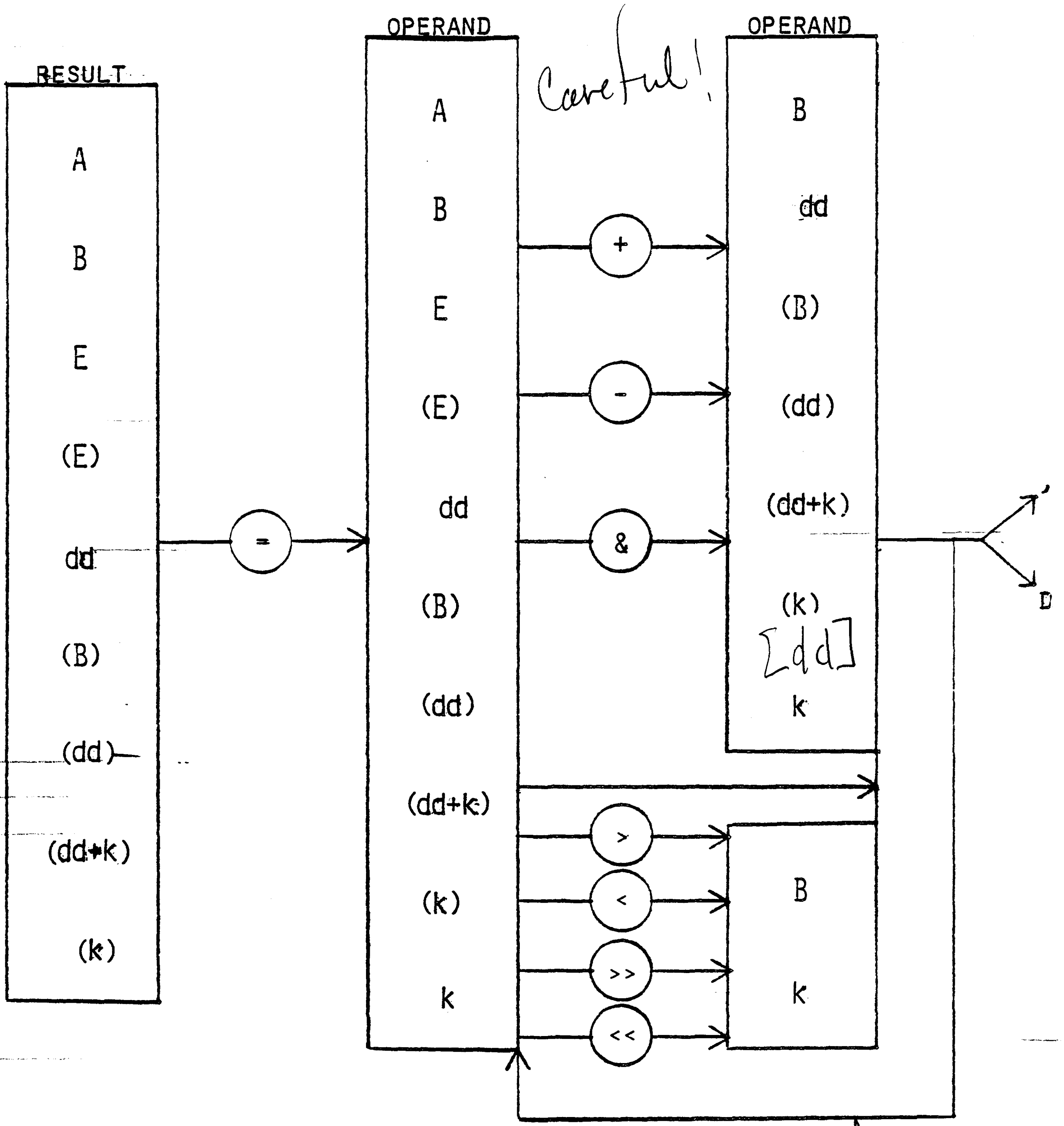
L	ASSIGN	.COMMENT
L	ASSIGN,COND	.COMMENT
L	$DAT_1, DAT_2, \ldots$	.COMMENT
L	NAME $OP_1, OP_2, \ldots$	.COMMENT
*	COMMENT	
L	*	.COMMENT

L	OPTIONAL STATEMENT LABEL MUST BEGIN IN COLUMN 1
ASSIGN	ASSIGNMENT STATEMENT ALWAYS HAS = OR :
COND	ASSIGNMENT CONDITION
DAT_I	DATA ITEM (SEE PDATA PSEUDO INSTRUCTION)
NAME	PSEUDO NAME
OP_I	OPERANDS
.COMMENT	ALWAYS PRECEDED BY A PERIOD FOLLOWING A BLANK
,	MEAN 'IF' WHEN USED TO DELIMIT CONDITION
*	INDICATES ENTIRE LINE IS A COMMENT WHEN PLACED IN FIRST NON-BLANK COLUMN, OR ASSIGNS CURRENT LOCATION COUNTER TO L.

SYMBOL MEANING

SYMBOL	ASSIGNMENT	CONDITION
=	EQUAL	EQUAL
#		NOT EQUAL
+	ADD	ADD
-	SUBTRACT	SUBTRACT
&	LOGICAL PRODUCT	LOGICAL PRODUCT
:	CHANNEL FUNCTION	
<	SHIFT LEFT	LESS THAN
>	SHIFT RIGHT	GREATER THAN
<<	CIRCULAR LEFT SHIFT	
>>	CIRCULAR RIGHT SHIFT	
<=		LESS THAN OR EQUAL
>=		GREATER THAN OR EQUAL

ASSIGNMENT SYNTAX

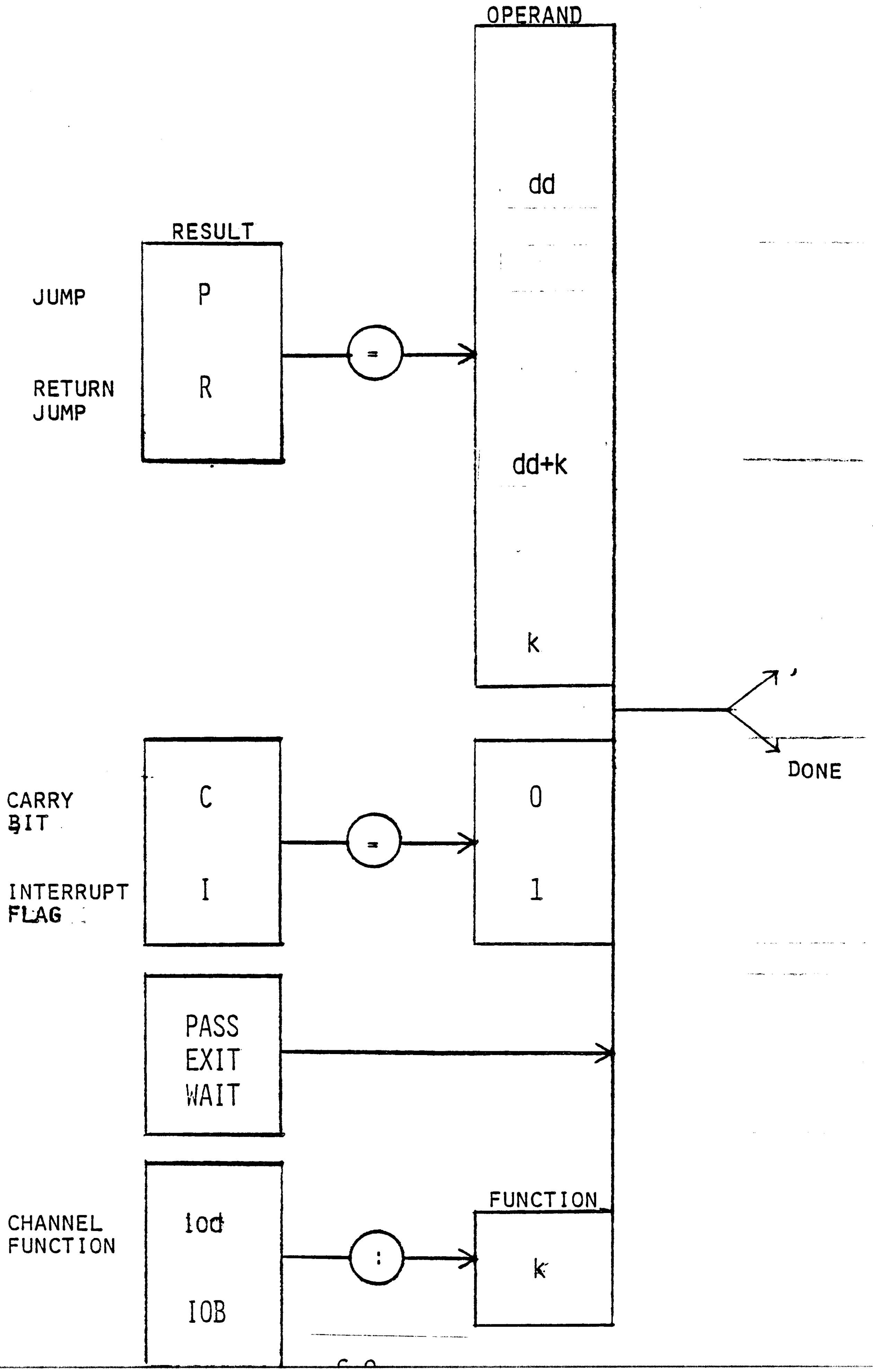


See 4.14 in APML

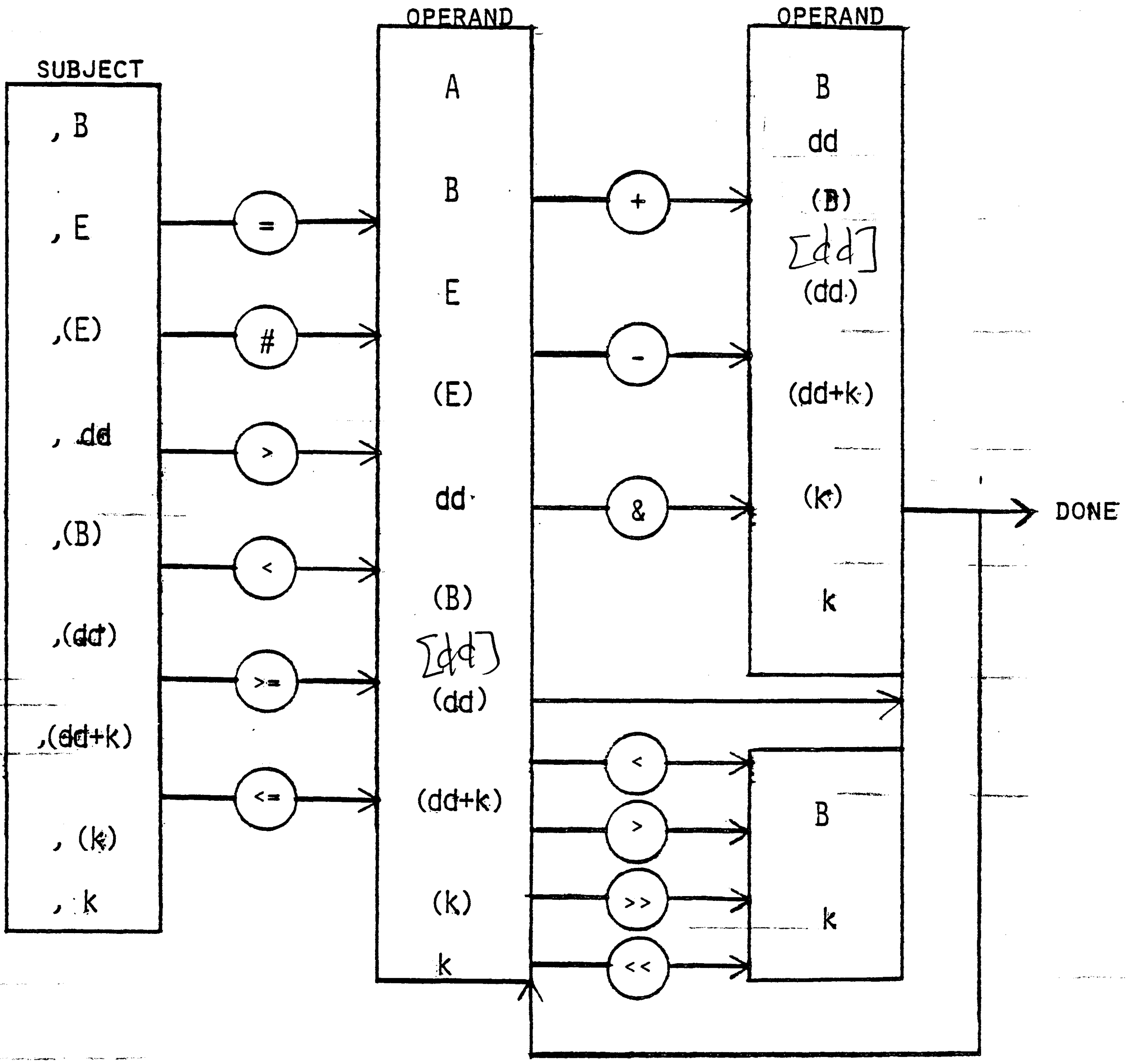
18 max operands

Scratch Register usage for (dd+k) (k) forms

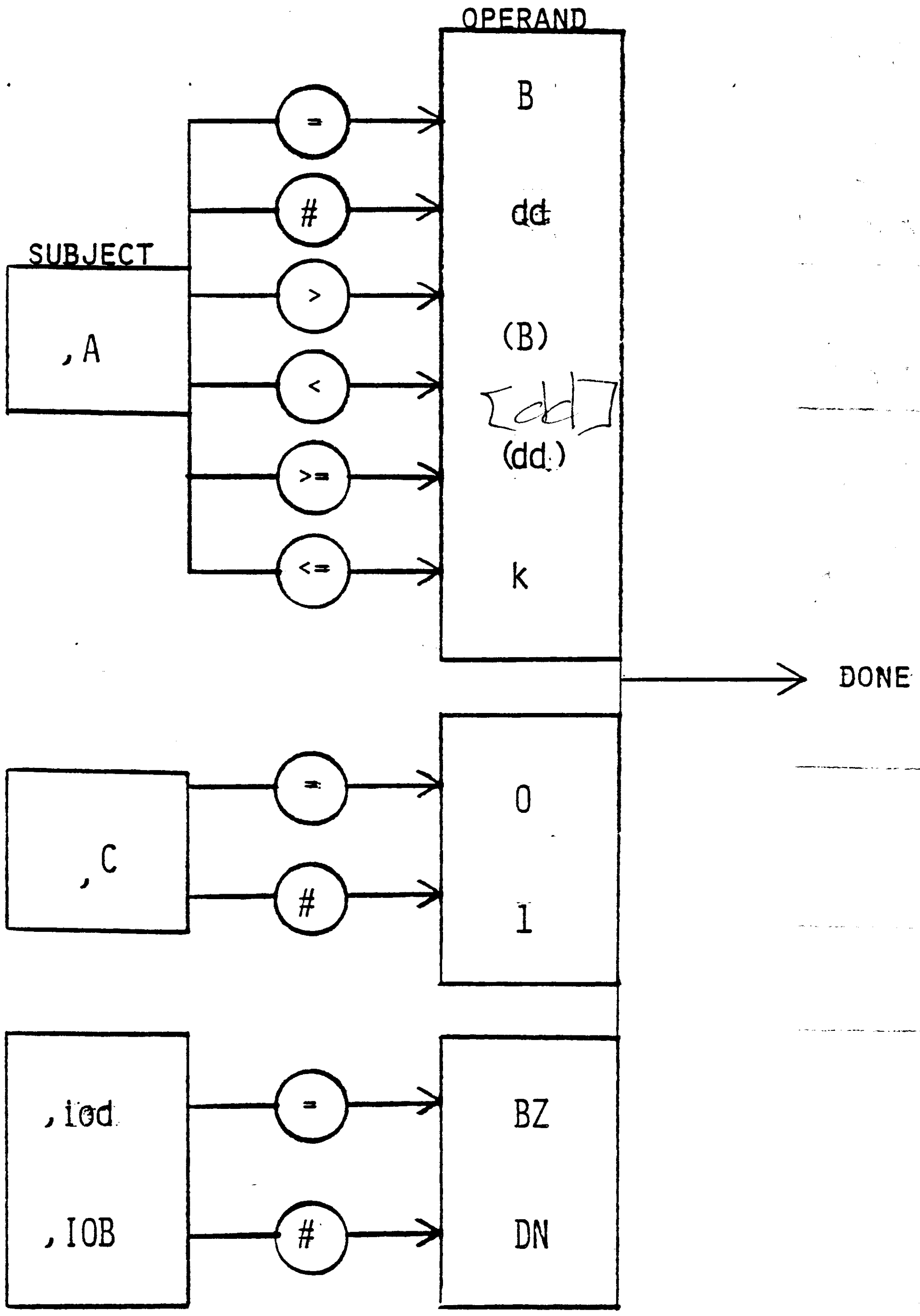
ASSIGNMENT SYNTAX



CONDITION SYNTAX



CONDITION SYNTAX



SAMPLE ASSIGNMENT STATEMENTS:

```
21      000334                                LOC       334
22      020013   062000   004010                        A=R5+(B)>10&B
        051000
26      014000  /000021   024231                        (LOC)=E+R3-(BOG)
        014000  /000044   024235
        150002   022011   033235
        034231
40      074012                                          P=R4

41      040000                                          C=1
42      070000                                          WAIT
43      174000                                          IOB:14
44      000223                                BOG       223
```

SAMPLE CONDITIONAL STATEMENTS:

```
45      ~~15~~240002  052000   006010                   ~~A~~R4=B,R3<E+B>>10
        024231   020011   023231
        101002   050000
55      053000   103003   010007                        E=7,A=B
        154002
61      101003   010000   054000                        B=0,C=0
64      040005   100002   010000                        A=0,MOS=ON
```

EXAMPLES OF THE DANGERS OF USING A AS AN OPERAND:

1) IN ASSIGNMENT STATEMENT:

```
0    010007                                                  A=7
1    020007   012025   024231                                B=A+(R1+25)
     032231   054000
```

2) IN CONDITIONAL STATEMENT:

```
6    010010                                                  A=10
7    020011   013037   102002                                B=A,R3#37
     054000
```

P MAY NOT BE USED AS OPERAND BY PROGRAMMER

EXAMPLE:

```
S7          13   070000                          P=P+2
            14   050000                          A=B
            15   054000               CAT        B=A
```

P IS USED AS OPERAND BY ASSEMBLER

EXAMPLE:

```
~~16   070002~~                              P=DOG
  17   050000                                A=B
  20   054000                    DOG         B=A
```

APML CONTROL STATEMENT

X = XREF

```
APML,I=IDN,L=LDN,B=BDN,E=EDN,ABORT,DEBUG,OPTIONS,LIST=NAME,S=SDN,SYM=SYM,T=BST.
```

I	OMITTED	SOURCE ON $IN
	I=IDN	SOURCE ON IDN
L	OMITTED	LIST OUTPUT ON $OUT
	L=0	NO LIST OUTPUT
	L=LDN	LIST OUTPUT ON LDN
B	OMITTED	BINARY ON $BLD
	B=0	NO BINARY
	B=BDN	BINARY ON BDN
E	OMITTED	NO ERROR LISTING
	E	ERROR LIST ON $OUT
	E=EDN	ERROR LIST ON EDN, IF EDN=LDN, THEN NO LDN.
ABORT	OMITTED	DO NOT ABORT
	ABORT	ABORT ON FATAL ASSEMBLY ERROR
DEBUG	OMITTED	IF FATAL ERROR OCCURS, WRITES BINARY RECORD AND SETS FATAL ERROR FLAG.
	DEBUG	WRITES BINARY RECORD WITH FATAL ERROR FLAG CLEAR.
OPTIONS	SEE LIST PSEUDO	
LIST	OMITTED	"NAMED" LIST PSEUDOS IGNORED
	LIST=NAME	MATCHING NAME NOT IGNORED
	LIST	ALL LIST PSEUDOS ACTIVATED
S	OMITTED	$APTEXT
	S=0	NO SYSTEM TEXT
	S=SDN	SYSTEM TEXT ON SDN
SYM	OMITTED	NO SYMBOL TABLE
	SYM	SYMBOL TABLE ON BDN ($BLD)
	SYM=SYM	SYMBOL TABLE ON SYM
T	OMITTED	NO BINARY SYSTEM TEXT
	T	BINARY SYSTEM TEXT IS $BST
	T=BST	BINARY SYSTEM TEXT IS BST

EXAMPLE OF CONTROL STATEMENT FORMAT NECESSARY TO ASSEMBLE AN APML PROGRAM:

```
JOB,JN=EXAMPLE.
ACCOUNT,AC=CRT.
APML.
/EOF
                IDENT         APML
6               REGISTER      (R1,R2,R3)
                SCRATCH       R3
                R1=12
                R2=1
LOC             R1=R1-1
                R2=R2+2
                P=LOC,R1#0
                (SUM)=R2
SUM             <1>
                END
/EOF
```

CHAPTER 7

SELECTED APML PSEUDOS

ALL THE CAL PSEUDO INSTRUCTIONS ARE AVAILABLE EXCEPT:

COMMON

OPDEF

THE = PSEUDO INSTRUCTION BECOMES EQUALS

THE FOLLOWING PSEUDO INSTRUCTIONS ARE UNIQUE TO APML:

PDATA

BASEREG

NEWPAGE

GLOBAL

SCRATCH

CHANNEL

IDENT & END

REQUIRED

IDENT IDENTIFIES PROGRAM MODULE

IDENT IS PHYSICALLY THE FIRST STATEMENT OF EACH MODULE

END IS PHYSICALLY THE LAST STATEMENT OF EACH MODULE

LOCATION	RESULT	OPERAND
IGNORED	IDENT	NAME
IGNORED	END	IGNORED

NAME - NAME OF PROGRAM MODULE

EXAMPLE:

```
                                          IDENT           PSEUDO
0    050000                               A=B
                                          END
```

EQUALS AND SET

DEFINES A SYMBOL WITH THE VALUE AND ATTRIBUTES DETERMINED BY THE EXPRESSION.

SYMBOL IS NOT REDEFINABLE FOR EQUALS.

SYMBOL IS REDEFINABLE FOR SET.

LOCATION	RESULT	OPERAND
SYMBOL	EQUALS	EXP,ATTRIBUTE
SYMBOL	SET	EXP,ATTRIBUTE

SYMBOL	-	UNQUALIFIED SYMBOL
EXP	-	ANY EXPRESSION
ATTRIBUTE	-	OPTIONAL. OVERRIDES ATTRIBUTE OF EXP P - PARCEL W - WORD V - VALUE

EXAMPLE:

```
                                              IDENT          EQUSET
                            2       R1        EQUALS          2
                                              BASEREG         R1
                         1024       GEORGE    EQUALS         1024
                           17       CAT       SET            17,P
0     075002 /000017                          P=CAT
                         1031       CAT       SET            GEORGE+5
                                              END
```

SCRATCH

USED TO DECLARE SCRATCH REGISTERS FOR GENERATING CODE FROM COMPLEX STATEMENTS.

LOCATION	RESULT	OPERAND
IGNORED	SCRATCH	R_1, R_2, R_3, R_4, R_5

R_I	UP TO 5 PREVIOUSLY DEFINED OR NON-DEFINABLE SYMBOLS. SYMBOLS MUST BE DEFINED ELSEWHERE.

EXAMPLE:

```
                                                IDENT      SCRATCH
                             7      SHARK       EQUALS     7
                             3      DO          SET        3
                                                SCRATCH      SHARK,DO,DA
                             6      DA          EQUALS     6
0                                   LOC         <1>
1    014000 /000000  024007                     (LOC)=(1067)
     014000 /001067  024003
     030003  034007
                                                END
```

BSS - BSSZ

RESERVES 64 BIT WORDS IN LOCAL MEMORY, STARTING AT CURRENT LOCATION COUNTER. FORCES WORD BOUNDARY IN DOING SO.

LOCATION	RESULT	OPERAND
SYMBOL	BSS	COUNT
SYMBOL	BSSZ	COUNT

SYMBOL	OPTIONAL, IS ASSIGNED WORD ADDRESS OF LOCATION COUNTER
COUNT	NUMBER OF WORDS

EXAMPLE:

```
                                              IDENT       BSSBSSZ
 0    050000                                  A=B
 1w                             12    NON     BSS         12
13w                              4    ZERO    BSSZ        4
74                                    HERE    *
                                              END
```

BASE

ALLOWS SPECIFICATION OF NUMERIC DATA BEING OCTAL, DECIMAL, OR MIXED. DEFAULT IS OCTAL.

LOCATION	RESULT	OPERAND
IGNORED	BASE	DBASE

DBASE	DESIRED BASE. O-OCTAL, D-DECIMAL, M-MIXED * REVERTS TO PREVIOUS BASE

EXAMPLE:

```
                                   IDENT        BASE
0    010012                        A=12                      .BASE O
                                   BASE         *
1    010012                        A=12                      .BASE O
                                   BASE         D
2    010014                        A=12                      .BASE D
                                   BASE         *
3    010012                        A=12                      .BASE O
                                   END
```

MACRO

A SEQUENCE OF SOURCE PROGRAM INSTRUCTIONS THAT ARE SAVED BY THE ASSEMBLER FOR INCLUSION IN A PROGRAM WHEN CALLED FOR BY THE MACRO NAME. THE MACRO CALL RESEMBLES A PSEUDO INSTRUCTION.

LOCATION	RESULT	OPERAND
IGNORED	MACRO	
SYMBOL	NAME	$P_1, P_2, \ldots, KW_1=D_1, KW_2=D_2, \ldots$
	.	
	.	
NAME	ENDM	

SYMBOL	1-8 CHARACTER OPTIONAL SYMBOL - IF PRESENT IT IS A POSITIONAL PARAMETER.
NAME	MACRO NAME TO BE USED WHEN ASSEMBLING INTO A PROGRAM. THIS NAME WILL REDEFINE ANY CURRENTLY ACTIVE PSEUDO INSTRUCTION.
P	POSITIONAL PARAMETER. MAY BE NONE, ONE OR MORE. WHEN USED, POSITION OF PARAMETER MUST BE ADHERED TO.
KW	KEYWORD PARAMETER. MAY BE NONE, ONE, OR MORE. WHEN USED THE KW NAME IN THE MACRO HEADING MUST BE USED. THE KW NAMES MAY BE USED IN ANY ORDER IN THE MACRO CALL.
D	DEFAULT VALUE OF A KW NAME. WHEN A BLANK OR COMMA FOLLOW THE = SIGN THE DEFAULT IS A NULL VALUE.
ENDM	DEFINITION END. THIS TERMINATES THE MACRO DEFINITION. THE NAME IN THE LOCATION FIELD MUST MATCH THE NAME IN THE MACRO HEADING.

EXAMPLE:

```
                                     MACRO
<PROTOTYPE>                 BAG      IDLE           COUNT,CAT=6
                                     LOCAL          XXXXXXXX
<DEFINITION>                BAG      A=COUNT
<DEFINITION>                XXXXXXXX A=A-1
<DEFINITION>                         P=XXXXXXXX,A#0
<DEFINITION>                         B=CAT
                            IDLE     ENDM
```

MACRO CALLS

LOCATION	RESULT	OPERAND
SYMBOL	NAME	PARG_1,PARG_2,...,KWARG_1=A_1,KWARG_2=A_2,...

SYMBOL	OPTIONAL IF SYMBOL USED ON THE MACRO DEFINITION. SUBSTITUTED WHENEVER SYMBOL APPEARS IN THE MACRO DEFINITION - IF SYMBOL DOES NOT APPEAR IN THE MACRO DEFINITION THE FIELD MUST BE EMPTY.
NAME	MUST MATCH THE NAME OF THE MACRO DEFINITION.
PARG_I	POSITIONAL ARGUMENT TO BE SUBSTITUTED IN THE MACRO PROTOTYPE. TWO CONSECUTIVE COMMAS INDICATE A NULL POSITIONAL ARGUMENT.
$\mathrm{KWARG}_\mathrm{I}$	KEYWORD PARAMETER TO BE USED IN THE MACRO PROTOTYPE. KEYWORD ARGUMENTS MAY APPEAR IN ANY ORDER.
A_I	KEYWORD ARGUMENT TO BE SUBSTITUTED FOR THE DEFAULT VALUE IN THE MACRO PROTOTYPE. IF KWARG IS USED WITH THE ABSENCE OF A_I THEN THE DEFAULT VALUE IS USED THROUGHOUT THE MACRO.

EXAMPLE:

```
                                              IDENT          CALL
                             7        NUM     SET            7
                                              IDLE           NUM,CAT=24
 0    010007                               A=NUM
 1    013001                          %%000000 A=A-1
 2    107001                                  P=%%000000,A#0
 3    010024    054000                        B=24
                                      SHRIMP  IDLE           NUM
 5    010007                          SHRIMP     A=NUM
 6    013001                          %%000001 A=A-1
 7    107001                                  P=%%000001,A#0
10    010006    054000                        B=6
                                              END
```

LOCAL

SPECIFIES SYMBOLS WHICH ARE DEFINED ONLY WITHIN A MACRO

LOCATION	RESULT	OPERAND
IGNORED	LOCAL	SYM_1,SYM_2,...

SYM_I SYMBOLS THAT ARE TO BE LOCAL

EXAMPLE:

```
                                      MACRO
<PROTOTYPE>                           LOC           TEST
                                      LOCAL         YYYYYYYYY
<DEFINITION>             YYYYYYYY     SET           TEST
<DEFINITION>                          A=YYYYYYYYY
                         LOC          END
```

ABS

DESIGNATES ABSOLUTE RATHER THAN RELOCATABLE ASSEMBLY

THE KERNEL USES ABSOLUTE ASSEMBLY

LOCATION	RESULT	OPERAND
IGNORED	ABS	IGNORED

EXAMPLE:

```
           IDENT          KERNEL
           ABS
*          CODE
           END
```

PDATA

LOGICALLY IDENTICAL TO DATA GENERATION.

ALLOWS UNRESTRICTED USE OF REGISTER SYMBOLS AS DATA.

LOCATION	RESULT	OPERAND
L	PDATA	$DATA_1$,$DATA_2$,...

L STATEMENT LABEL WITH PARCEL ATTRIBUTE

$DATA_I$ CAN BE ONE OF THE FOLLOWING:

1. NUMBER
2. SYMBOL
3. CHARACTER STRING
   USES AS MANY PARCELS AS NECESSARY
4. PARCEL STORAGE RESERVATION
5. * - ASSIGNS CURRENT PARCEL COUNTER TO L

EXAMPLE:

```
                                         IDENT        PDATA
                       2       R1        EQUALS       2
                     217       A         EQUALS       217
0   000217  000002  000007     DOG       PDATA        A,R1,"DATA ITEM",<10>,7
                                         END
```

BASEREG

USED FOR JUMPS TO A LABEL OUTSIDE OF CURRENT "PAGE."

A "PAGE" IS AT MOST 512 PARCELS.

LOCATION	RESULT	OPERAND
IGNORED	BASEREG	R,B

R	SYMBOL FOR DESIRED BASE REGISTER
B	BIAS (CONTENTS OF REGISTER)

EXAMPLE:

```
                                                     IDENT        BASEREG
                                 1         R1        EQUALS       1
                                                     BASEREG      R1
  0    075001 /001744                                P=NEXT
  2                                                  <1742>
1744                                       NEXT      <1>
                                                     END
```

— PSUEDO ~~NEWPAGE~~ VWD, BSS

— LABEL ON PDATA OR DATA GENERATION

— NEWPAGE

NEWPAGE

FORCES A NEWPAGE

LOCATION	RESULT	OPERAND
IGNORED	NEWPAGE	IGNORED

EXAMPLE:

```
                                                       IDENT          NEWPAGE
                                  1        R1          EQUALS         1
                                                       BASEREG        R1
0    075001 /000002                                    P=NEXT
                                                       NEWPAGE
2                                          NEXT        <1>
                                                       END
```

GLOBAL

DECLARE A SYMBOL TO BE GLOBAL SO IT CAN BE RETAINED ACROSS PROGRAM MODULES.

LOCATION	RESULT	OPERANDS
IGNORED	GLOBAL	$SYM_1, SYM_2, \ldots$

SYM_I	NON-REDEFINABLE SYMBOL SYMBOL MUST NOT BE RELOCATABLE

EXAMPLE:

```
                                                 IDENT          GLOBAL
                                                 ABS
                                                 GLOBAL         NON,BA
                                 0       BA      EQUALS         0
                                                 BASEREG        BA
  0    075000 /001767                            P=NON
  2                                              <1765>
1767                                     NON     <1>
                                                 END

                                                 IDENT          GLOBAL1
                                 3       BB      EQUALS         3
                                                 BASEREG        BB
  0    075003 /002351                            P=LOC
                                                 BASEREG        BA
  2    075000 /001767                            P=NON
                                                 BASEREG        *
  4                                              <2345>
2351                                     LOC     <1>
                                                 END
```

CHANNEL

USED TO DEFINE A CHANNEL MNEMONIC

LOCATION	RESULT	OPERAND
M	CHANNEL	N

M	MNEMONIC (CONVENTION IS 3 CHARACTERS)
N	CHANNEL NUMBER

EXAMPLE:

```
                                             IDENT           CHANNEL
                           5        BUF      CHANNEL         5
0   140005                                   BUF:0
1   140005                                   MOS:0
                                             END
```

BLOCK

CONTROLS THE ORDER IN WHICH SOURCE CODE IS ASSEMBLED.

THE SOURCE CODE IS DIVIDED INTO BLOCKS,
EACH OF WHICH HAS ITS OWN LOCATION COUNTER.

THE BLOCK PSEUDO INSTRUCTION IS USED IN THE KERNEL SO THAT THE TABLES DO NOT END UP IN THE FIRST 4000 PARCELS OF LOCAL MEMORY.

LOCATION	RESULT	OPERAND
IGNORED	BLOCK	NAME

NAME	NAME OF BLOCK.
	* - REVERT TO PREVIOUS BLOCK.
	BLANK - REVERT TO NOMINAL BLOCK (DEFAULT)

EXAMPLE:

```
                                          IDENT        BLOCK
  0    050000                     CAT     A=3
                                          BLOCK        ONE
  4    050000                     BAT     A=3
                                          BLOCK        TWO
 10    050000                     RAT     A=3
                                          BLOCK
  1    050000                     HAT     A=3
                                          BLOCK        *
 11    050000                     MAT     A=3
                                          BLOCK        ONE
  5    050000                     NAT     A=3
                                          END
```

QUAL

QUALIFIES SYMBOLS SO THAT THE SAME SYMBOL MAY BE USED MORE THAN ONCE IN A PROGRAM MODULE.

LOCATION	RESULT	OPERAND
IGNORED	QUAL	QUALIFIER

QUALIFIER	QUALIFIER TO BE APPLIED TO ALL SYMBOLS DEFINED UNTIL THE NEXT QUAL STATEMENT.
	* - REVERT TO PREVIOUS QUAL PSEUDO INSTRUCTION.
	BLANK - SYMBOLS ARE UNQUALIFIED (DEFAULT)
	REFERENCES TO SYMBOLS QUALIFIED BY A QUALIFIER OTHER THAN THAT CURRENTLY IN EFFECT ARE OF THE FORM: /QUALIFIER/SYMBOL

EXAMPLE:

```
                                                  IDENT            QUAL
  0                                      LOC      *
  0     014000 /000000                            A=LOC
  2     014000 /000004                            A=/CLOWN/LOC
                                                  QUAL             CLOWN
  4                                      LOC      *
  4     014000 /000004                            A=LOC
  6     014000 /000000   054000                   B=//LOC
                                                  QUAL             *
 11     014000 /000000                            A=LOC
                                                  END
```

CHAPTER 8

SELECTED $APTEXT MACROS

MACROS EXIST IN $APTEXT FOR A VARIETY OF APPLICATIONS, INCLUDING:

EXIT STACK CONTROL

EXECUTION CONTROL

FIELD AND TABLE ACCESS

OVERLAY AND REGISTER DEFINITIONS

OVERLAY COMMUNICATION

BRANCH CONTROL

ARITHMETIC AND LOGICAL OPERATIONS

HISTORY TRACE ENTRY

INTERRUPT STATE CONTROL

MEMORY REARRANGEMENT

PARAMETER DESCRIPTION

LINKING AND UNLINKING ITEMS IN A MEMORY CHAIN

REGISTER

LOCATION	RESULT	OPERAND
ORIGIN	REGISTER	$(\mathrm{SYM}_1, \mathrm{SYM}_2, \ldots)$

ORIGIN	STARTING OPERAND REGISTER NUMBER (OCTAL)
SYM_I	LIST OF SYMBOLS TO BE ASSIGNED TO OPERAND REGISTER

SAME AS THE FOLLOWING:

SYM_1 EQUALS ORIGIN

SYM_2 EQUALS ORIGIN + 1

.

.

.

SYM_I EQUALS ORIGIN + (I-1)

EXAMPLE:

```
                                              IDENT          REGISTER
                                       7      REGISTER       (R1,AA,CAT)
                                              SCRATCH        R1
0   030011   024010                           AA=(R!CAT)
                                              END
```

REGDEFS

ASSIGNS OPERAND REGISTERS TO REGISTER SYMBOLS

ALLOCATES SCRATCH REGISTERS

DEFINES TEMPORARY REGISTERS FOR USE BY OTHER MACROS CALLED WITHIN THIS PROGRAM MODULE.

LOCATION	RESULT	OPERAND
L	REGDEFS	GLOBAL, PARS, LOCAL, TEMP

L	OPTIONAL SYMBOL OR CONSTANT BETWEEN 0 AND 777 OCTAL SPECIFIES ORIGIN REGISTER (AVOID 300_8 TO 307_8). DEFAULT IS 310_8
GLOBAL	UP TO 8 REGISTER SYMBOLS TO BE ASSIGNED TO REGISTERS 300_8 TO 307_8.
PARS	LIST OF SYMBOLS TO BE ASSIGNED TO WORKING OPERAND REGISTERS.
LOCAL	LIST OF SYMBOLS TO BE ASSIGNED TO LOCAL REGISTERS.
TEMP	LIST OF SYMBOLS TO BE ASSIGNED TO TEMPORARY REGISTERS.

THE FOLLOWING REGISTERS ARE ALSO DEFINED:

%S1 TO %S5	SCRATCH REGISTERS
%T1 TO %T6	MACRO TEMPORARY REGISTERS
%W1 TO %W5	WORKING REGISTERS AVAILABLE TO OVERLAY.

EXAMPLE:

```
                                       IDENT          REGDEFS
                                       REGDEFS        (G1,G2),(R1,R2,R3),(L1,L2),T1
1    010006   024331                   R2=6
2    030332   034330                   (R1)=(R3)
4    020327                            A=R1%W5
                                       END
```

LOGICAL OPERATIONS

LOCATION	RESULT	OPERAND
L	NAND	OP_1, OP_2, R
L	OR	OP_1, OP_2, R
L	NOR	OP_1, OP_2, R
L	XOR	OP_1, OP_2, R

L	OPTIONAL STATEMENT LABEL
OP_I	OPERANDS
R	RESULT

DEFINITIONS

NAND	OR	NOR	XOR
1100	1100	1100	1100
<u>1010</u>	<u>1010</u>	<u>1010</u>	<u>1010</u>
0111	1110	0001	0110

EXAMPLE:

```
          IDENT       LOGICAL
          REGDEFS     ,(R1,R2),
LAB1      NOR         17,R1,R2
          END
```

NEXT

ASSIGN SUCCESSIVE VALUES

LOCATION	RESULT	OPERAND
SYM	NEXT	VALUE

SYM	OPTIONAL SYMBOL NAME
VALUE	OPTIONAL INITIAL VALUE

	VALUE PRESENT	VALUE BLANK
SYM PRESENT	SYM=VALUE $NEXT=VALUE+1	SYM=$NEXT $NEXT=$NEXT+1
SYM BLANK	$NEXT=VALUE+1	$NEXT=$NEXT+1

EXAMPLE:

```
          IDENT          NEXT
CAL       NEXT           14           .CAL=14, $NEXT=15
BIG       NEXT                        .BIG=15, $NEXT=16
          NEXT           12           .$NEXT=13
          NEXT                        .$NEXT=14
          END
```

FIELD

LOCATION	RESULT	OPERAND
SYM	FIELD	P,S,W

SYM	FIELD SYMBOL NAME
P	PARCEL OFFSET
S	STARTING BIT (DEFAULT 0)
W	WIDTH OF FIELD (DEFAULT 16)

THE FOLLOWING PARAMETERS ARE GENERATED

SYM@P	PARCEL OFFSET FROM BEGINNING OF TABLE
SYM@S	STARTING BIT OF FIELD (SOFTWARE NUMBERED)
SYM@N	WIDTH OF FIELD
SYM@M	MASK FOR FIELD, RIGHT JUSTIFIED
SYM@X	COMPLEMENT OF MASK IN PROPER POSITION IN FIELD

IF P=* SYM@P IS UNDEFINED
IF S=* SYM@S,SYM@N,SYM@M,SYM@X ARE UNDEFINED

EXAMPLE:

```
         IDENT          FIELD
TB1      FIELD          0,3,9
TB2      FIELD          1
TB3      FIELD          2,0,7
TB4      FIELD          3,,5
         END
```

FIELD GETS AND PUTS

LOCATION	RESULT	OPERAND
L	GET	DEST,FIELD,BASE
L	PUT	SOURCE,FIELD,BASE
L	RGET	DEST,FIELD,SOURCE
L	RPUT	SOURCE,FIELD,DEST

L	OPTIONAL STATEMENT LABEL
DEST	DESTINATION OPERAND REGISTER OR MEMORY LOCATION
SOURCE	OPERAND REGISTER OR MEMORY LOCATION CONTAINING DATA TO BE STORED.
FIELD	FIELD TO BE LOADED, DEFINED BY FIELD MACRO
BASE	TABLE BASE ADDRESS IN AN OPERAND REGISTER
GET	LOADS A FIELD FROM A TABLE INTO AN OPERAND REGISTER OR MEMORY LOCATION.
PUT	STORES DATA IN A FIELD IN A TABLE FROM AN OPERAND REGISTER OR MEMORY LOCATION.
RGET	LOADS AN OPERAND REGISTER OR MEMORY LOCATION FROM A FIELD IN AN OPERAND REGISTER OR MEMORY LOCATION.
RPUT	LOADS A FIELD IN AN OPERAND REGISTER OR MEMORY LOCATION FROM AN OPERAND REGISTER OR MEMORY LOCATION.

EXAMPLE:

```
                                            IDENT        FGETPUT
                                    0       REGISTER     (R1,R2,R3,TA)
                                            SCRATCH      R3
                                    FIELD1  FIELD        2,3,9
0    014000 /012340   024003                TA=12340
                                            GET          R1,FIELD1,TA
12   014000 /001024   024001                R2=1024
                                            PUT          R2,FIELD1,TA
                                            RGET         R1,FIELD1,(R2)
                                            RPUT         R1,FIELD1,(R2)
                                            END
```

CONDITIONAL BLOCK MACROS

$IF - $ELSEIF - $ELSE - $ENDIF

USED TO DELIMIT BLOCKS OF CODE; ONLY ONE OF WHICH IS EXECUTED.

LOCATION	RESULT	OPERAND
	$IF	$(COND_1)$,ANDOR,$(COND_2)$
	$ELSEIF	$(COND_1)$,ANDOR,$(COND_2)$
	$ELSE	
	$ENDIF	

$COND_I$	A VALID APML CONDITIONAL EXPRESSION
ANDOR	LOGICAL OPERATOR 'AND' OR 'OR'. IF BLANK, COND2 IGNORED.

$IF MUST BE FIRST CONDITIONAL BLOCK MACRO AND MUST HAVE A CORRESPONDING $ENDIF.

$ELSEIF OCCURS BETWEEN $IF AND $ENDIF WHEN DELIMITING MORE THAN TWO BLOCKS.

$ELSE IS OPTIONAL AND DELIMITS LAST BLOCK BEFORE $ENDIF.

EXAMPLE:

```
                                                  IDENT       IF
                          7        R1             EQUALS      7
                         10        R2             EQUALS      10
                                                  SCRATCH     R1
                          4        LOC            SET         4
                                                  $IF         (R2<LOC),AND,B=7
10    010000   024010                               R2=0
                                                  $ELSEIF     (LOC#4)
17    010001   024010                               R2=1
                                                  $ELSE
22    010002   024010                               R2=2
                                                  $ENDIF
                                                  END
```

RTCALL

GONE IN 1.11

USED BY KERNEL TO CALL SUBROUTINES WITHOUT USING EXIT STACK.
THESE CALLS MAY NOT BE NESTED.

LOCATION	RESULT	OPERAND
L	RTCALL	SUB,RTN

L	OPTIONAL STATEMENT LABEL
SUB	SUBROUTINE ENTRY POINT. THE LAST INSTRUCTION IN THIS SUBROUTINE MUST BE P=RT. REGISTER RT MUST BE DEFINED.
RTN	NEXT STATEMENT TO BE EXECUTED AFTER SUBROUTINE DONE. DEFAULT IS STATEMENT FOLLOWING RTCALL STATEMENT.

EXAMPLE:

```
                                                    IDENT            RTCALL
                            0         RT            EQUALS           0
                            4         RA            EQUALS           4
                                                    BASEREG          RA,1
                            7         R1            EQUALS           7
0    010013   024007                                R1=13
2    010001   054000                                B=1
                                                    RTCALL           ODDEVEN,RETURN
0    010001   154002                                E=1                                .NOT EXECUTE
2                                     RETURN        *
2    050000                                         A=B
3    070000                                         WAIT
4                                     ODDEVEN        *
4    020007   011001                                A=R1&1
6    103003   010000   054000                       B=0,A=0
1    074000                                         P=RT
                                                    END
```

PART 3

I/O SUBSYSTEM OPERATING SYSTEM

CHAPTER 9

OPERATING SYSTEM OVERVIEW

FUNCTIONS

PERFORMS I/O BETWEEN CPU AND PERIPHERALS

MANAGES FRONT-END COMMUNICATIONS

PERFORMS STATION FUNCTIONS

DEADSTARTS CPU

CHARACTERISTICS

MULTI-TASKING
    UP TO 32 TASKS ACTIVE AT A TIME /IOP

NONPRE-EMPTIVE SCHEDULING

SIMPLE 16 LEVEL TASK PRIORITY SCHEME

INTERRUPT DRIVEN

EXTENSIVE USE OF OVERLAYS

SYSTEM COMPONENTS

15K

KERNEL

NUCLEUS OF OPERATING SYSTEM.
LOCAL MEMORY RESIDENT.
EXECUTES IN EACH I/O PROCESSOR WITH MINOR MODIFICATIONS.
RESPONSIBLE FOR:

- ACTIVITY MANAGEMENT
- INTER-ACTIVITY COMMUNICATION
- RESOURCE MANAGEMENT
- INTERRUPT HANDLING
- INTER-PROCESSOR COMMUNICATION

OVERLAYS

RESIDE IN BUFFER MEMORY.
READ INTO LOCAL MEMORY WHEN NEEDED.
MAKE UP THE BULK OF THE SYSTEM.
NOT ALL USED BY ANY ONE PROCESSOR.

DISK SUBSYSTEM
RESIDES MOSTLY IN BUFFER MEMORY AS OVERLAYS.
EXECUTES IN BIOP OR DIOP.

STATION SUBSYSTEM
RESIDES IN BUFFER MEMORY AS OVERLAYS.
EXECUTES MOSTLY IN MIOP.

CONCENTRATOR SUBSYSTEM
RESIDES IN BUFFER MEMORY AS OVERLAYS.
EXECUTES MOSTLY IN MIOP.

INTERACTIVE CONCENTRATOR SUBSYSTEM
RESIDES IN BUFFER MEMORY AS OVERLAYS.
EXECUTES MOSTLY IN MIOP.

TAPE SUBSYSTEM
RESIDES IN BUFFER MEMORY AS OVERLAYS.
EXECUTES MOSTLY IN XIOP.

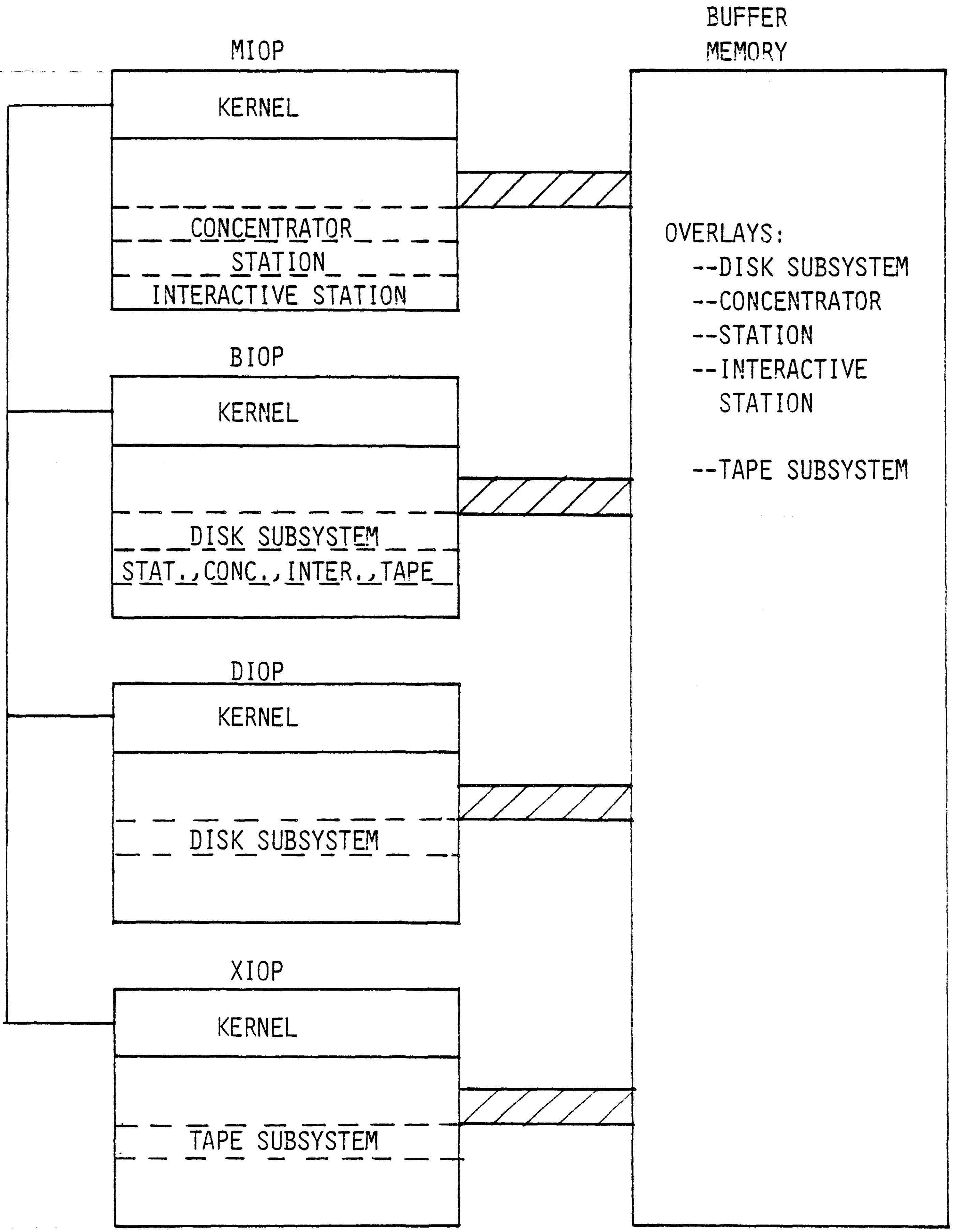


FIGURE 9-1. IOS SOFTWARE CONFIGURATION

CHAPTER 10

SOFTWARE STRUCTURE

&

RESOURCE IMPLEMENTATION

THE RESOURCES AVAILABLE TO THE OPERATING SYSTEM ARE LOCAL AND BUFFER MEMORY.

THE SOFTWARE STRUCTURE INCLUDES:

ACTIVITY DESCRIPTOR

STORAGE MODULE

SOFTWARE STACK

POPCELL

DISK ACTIVITY LINK

LOCAL MEMORY

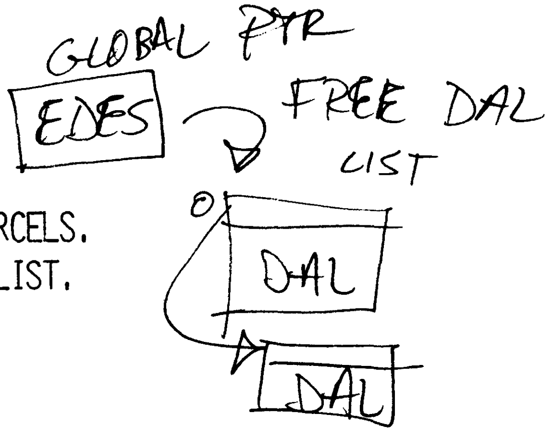
THE KERNEL RESIDES IN LOCAL MEMORY IN EACH IOP.

THE KERNEL MAINTAINS FOUR LOCAL MEMORY CHAINS.

OVERLAY MEMORY CHAIN:

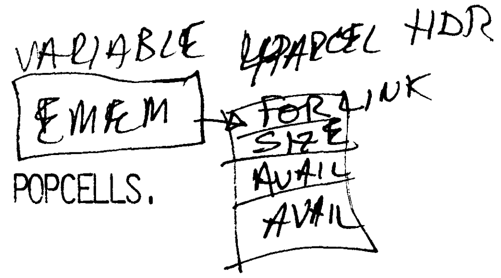
- LOCATED AFTER KERNEL.
- ALLOCATED IN MULTIPLES OF FOUR PARCELS.
- IMPLEMENTED AS A DOUBLY LINKED LIST.
- USED TO HOLD EXECUTING OVERLAYS.

DAL CHAIN:



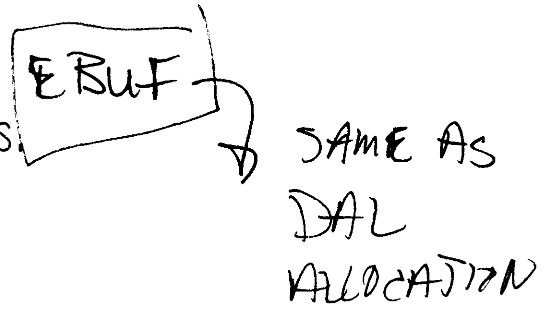
- LOCATED AFTER OVERLAY MEMORY.
- ALLOCATED IN MULTIPLES OF 40_8 PARCELS.
- IMPLEMENTED AS A FORWARD LINKED LIST.
- USED FOR MESSAGE SPACE.

FREE MEMORY CHAIN:



- LOCATED AFTER DALS.
- ALLOCATED IN MULTIPLES OF FOUR PARCELS.
- IMPLEMENTED AS A FORWARD LINKED LIST.
- USED FOR TABLES, ACTIVITY DESCRIPTORS AND POPCELLS.

LOCAL I/O BUFFER CHAIN:



- LOCATED IN UPPER MEMORY
- ALLOCATED IN MULTIPLES OF 4000_8 PARCELS.
- IMPLEMENTED AS A FORWARD LINKED LIST.
- USED MAINLY FOR I/O BUFFERS.

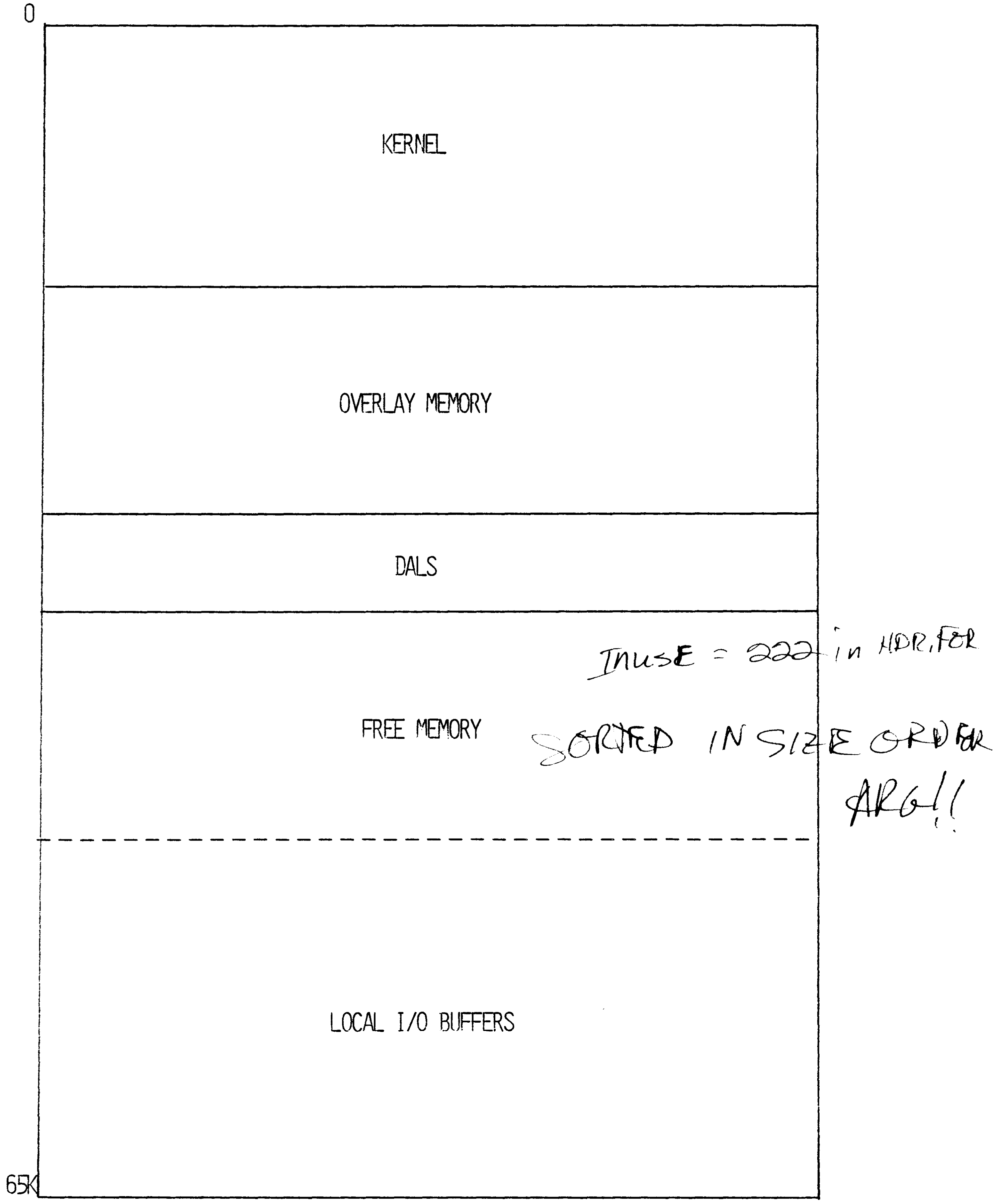


FIGURE 10-1. LOCAL MEMORY CONFIGURATION

BUFFER MEMORY

SHARED BY ALL IOPs

CONTAINS ALL THE OVERLAYS AVAILABLE TO IOPs

EACH IOP HAS ITS OWN KERNEL STORAGE AREA
    USED FOR TEMPORARY STORAGE AND I/O BUFFERS

EACH IOP HAS ITS OWN MESSAGE AREA
    MESSAGES ARE MAINTAINED BY SENDER

SYSTEM DIRECTORY CONTAINS INFORMATION ABOUT BUFFER MEMORY PARTITIONING.

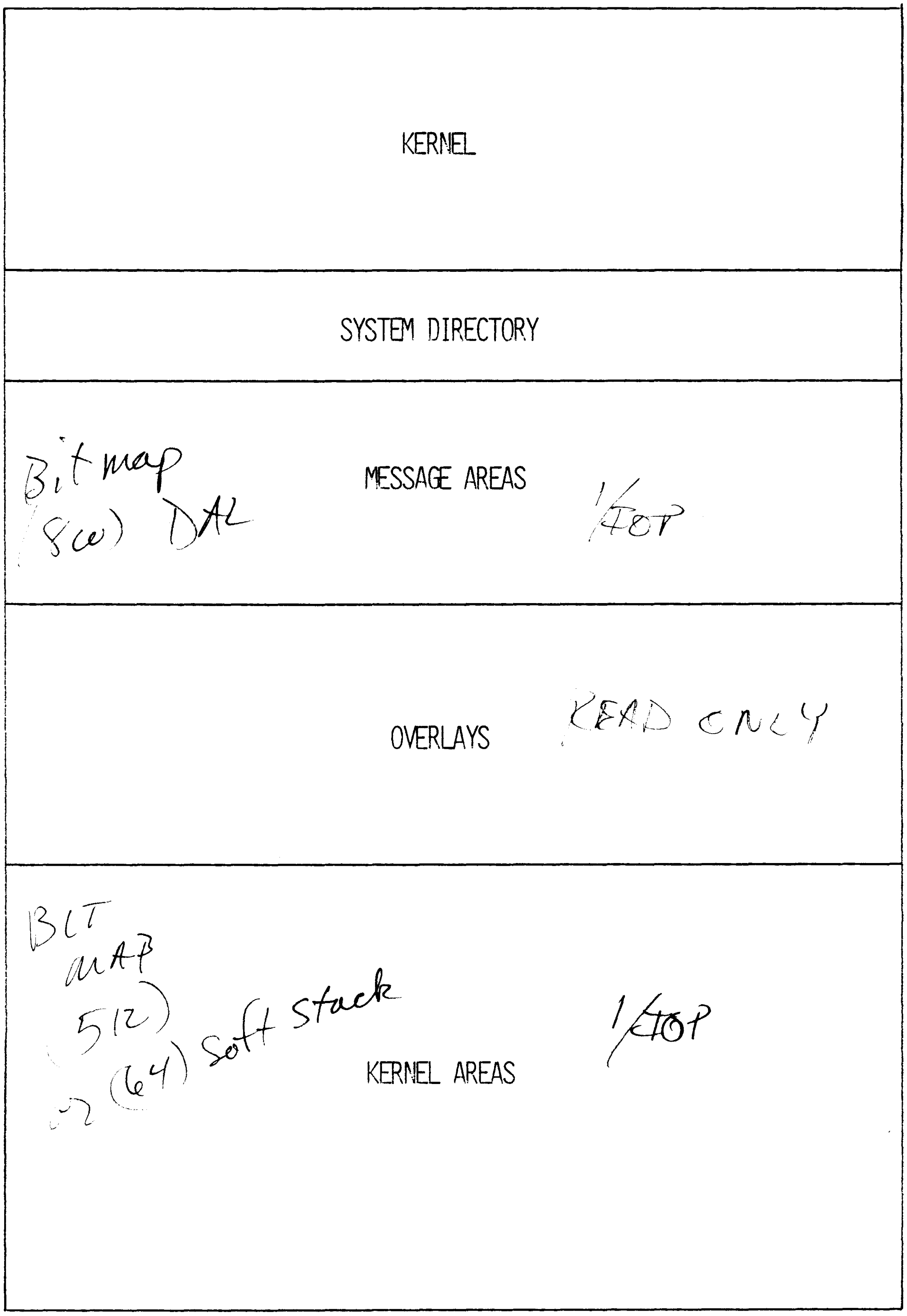


FIGURE 10-2. BUFFER MEMORY CONFIGURATION

PARCEL	WORD	
0	0	MIOP MESSAGE AREA ADDRESS IN MOS (2 PARCELS)
2	0	BIOP MESSAGE AREA ADDRESS
4	1	DIOP MESSAGE AREA ADDRESS
6	1	XIOP MESSAGE AREA ADDRESS
10	2	~~OVERLAY DESCRIPTOR TABLE ADDRESS IN MOS (2 PARCELS)~~ UNUSED
12	2	~~SIZE - # ENTRIES | RESERVED~~
14	3	1ST OVERLAY ADDRESS (2 PARCELS)
16	3	UNUSED
20	4	MIOP KERNEL AREA IN MOS
22	4	SIZE OF MIOP MOS MODULE (NUMBER OF 1000_8 WORD AREAS)
24	5	BIOP KERNEL AREA IN MOS
26	5	SIZE (IN 1000_8 WORD AREAS)
30	6	DIOP KERNEL AREA IN MOS
32	6	SIZE (1000_8 WORD AREAS)
34	7	XIOP KERNEL AREA IN MOS
36	7	SIZE (1000_8 WORD AREAS)
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
52	12	.

FIGURE 10-3. SYSTEM DIRECTORY

TASK HANDLING

TASKS EXECUTING IN AN IOP ARE CALLED ACTIVITIES.

AN ACTIVITY IS AN INDEPENDENT PATH OF EXECUTION THROUGH THE CODE.

NORMALLY CONSISTS OF NESTED OVERLAY CALLS.

KERNEL MAINTAINS ACTIVITIES THROUGH THE USE OF ACTIVITY DESCRIPTORS, STORAGE MODULES (SMODS) AND POPCELLS.

ACTIVITY DESCRIPTOR

USED BY KERNEL TO SCHEDULE AND ACTIVATE ACTIVITIES.

ONE FOR EACH ACTIVITY.

BUILT BY A COMMON SUBROUTINE THROUGH A CREATE SERVICE REQUEST.

CONTAINS LINKS, ADDRESSES AND OTHER INFORMATION NECESSARY TO MANAGE AN ACTIVITY.

LOCAL MEMORY RESIDENT. IN FREE MEM

EXISTS UNTIL AN ACTIVITY IS TERMINATED.

ECPQ – CENTRAL PROCESSOR Q

PARCEL	
0	LINK FOR QUEUES
1	LINK TO EXISTING ACTIVITIES (FOR DEBUGGING)
2	PRIORITY ($0-17_8$)
3	MOS UPPER ADDRESS OF SOFTWARE STACK
4	MOS LOWER ADDRESS OF SOFTWARE STACK
5	UNUSED
6	UNUSED
7	UNUSED
10	LINK TO NEXT SMOD (OVERLAY) TO ACTIVATE IF IDLE LINK TO CURRENT IF ACTIVITY ACTIVE
11	40000=DEMON ACTIVITY
12	FUNCTION CODE OF CURRENT SERVICE REQUEST
13	KERNEL PARAMETERS FOR I/O REQUESTS AND PASSING STATUS TO OVERLAYS
14	" " " " " " " " "
15	" " " " " " " " "
16	" " " " " " " " "
17	" " " " " " " " "

FIGURE 10-4. ACTIVITY DESCRIPTOR

STORAGE MODULE (SMOD)

USED TO SAVE AN OVERLAY'S EXECUTING ENVIRONMENT.

ONE PER OVERLAY READ INTO LOCAL MEMORY.

SIZE VARIES DEPENDING ON HOW MANY REGISTERS NEED BE SAVED.

- MINIMALLY CONTAINS:
    - LINKS TO ACTIVITY DESCRIPTOR AND PREVIOUS SMOD.
    - OVERLAY INFORMATION.
    - A, B, C, E, AND P REGISTER CONTENTS.

- MAY CONTAIN:
    - OPERAND REGISTER CONTENTS ESSENTIAL TO ITS' OVERLAY.
    - PROGRAM EXIT STACK ENTRIES FOR ITS' OVERLAY.

SMOD IS PARTIALLY UPDATED WHEN AN OVERLAY DOES A KERNEL SERVICE REQUEST.

- IF CALL RESULTS IN LOSS OF CONTROL, SMOD IS COMPLETELY UPDATED.

REGISTERS ARE RE-LOADED FROM SMOD WHEN S.R. IS COMPLETED OR WHEN OVERLAY GETS CONTROL BACK.

INITIAL SMOD SET UP THROUGH CREATE SERVICE REQUEST AND IS WRITTEN TO BUFFER MEMORY AS A SOFTWARE STACK.

PARCEL	
0	ACTIVITY ADDRESS
1	LINK TO PREVIOUS SMOD (0 IF FIRST)
2	SIZE OF THIS SMOD
3	POINTER TO OVERLAY DESCRIPTOR TABLE ENTRY
4	UNUSED
5	UNUSED
6	A
7	B
10	C
11	E
12	~~B~~@(E) ~~ON INTERRUPTS~~
13	# OF OPERAND REGISTERS (7); STARTING REGISTER (9)
14	FIRST OPERAND REGISTER SAVED
.	
.	
.	
N	FIRST EXIT STACK ENTRY
.	
.	
.	
N+E	LAST EXIT STACK ENTRY

FIGURE 10-5. STORAGE MODULE

$SMD POINTS

SOFTWARE STACK

THERE IS A FIXED STACK IN LOCAL MEMORY WHERE THE SMODS FOR THE CURRENT ACTIVITY'S OVERLAYS RESIDE.

A SMOD IS 'PUSHED' ONTO THIS STACK WHEN AN OVERLAY CALLS ANOTHER OVERLAY.

> A 'PUSH' CONSISTS OF SAVING AN OVERLAY'S REGISTERS AND UPDATING THE SMOD POINTER IN THE ACTIVITY DESCRIPTOR.

THE CALLER'S SMOD IS 'POPPED' OFF THIS STACK WHEN THE CALLED OVERLAY DOES A RETURN SERVICE REQUEST.

> A 'POP' CONSISTS OF UPDATING THE RESTORING THE CALLING OVERLAY'S REGISTERS.

THIS SOFTWARE STACK IS WRITTEN OUT TO BUFFER MEMORY WHEN AN ACTIVITY RELINQUISHES CONTROL TO THE KERNEL AND OTHER ACTIVITIES ARE ON THE CENTRAL PROCESSOR QUEUE.

> THE LOCAL MEMORY SOFTWARE STACK IS NOW FREE FOR USE BY ANOTHER ACTIVITY.

WHEN AN ACTIVITY REGAINS CONTROL, IT'S SOFTWARE STACK WILL BE READ INTO THE LOCAL SOFTWARE STACK FROM BUFFER MEMORY.

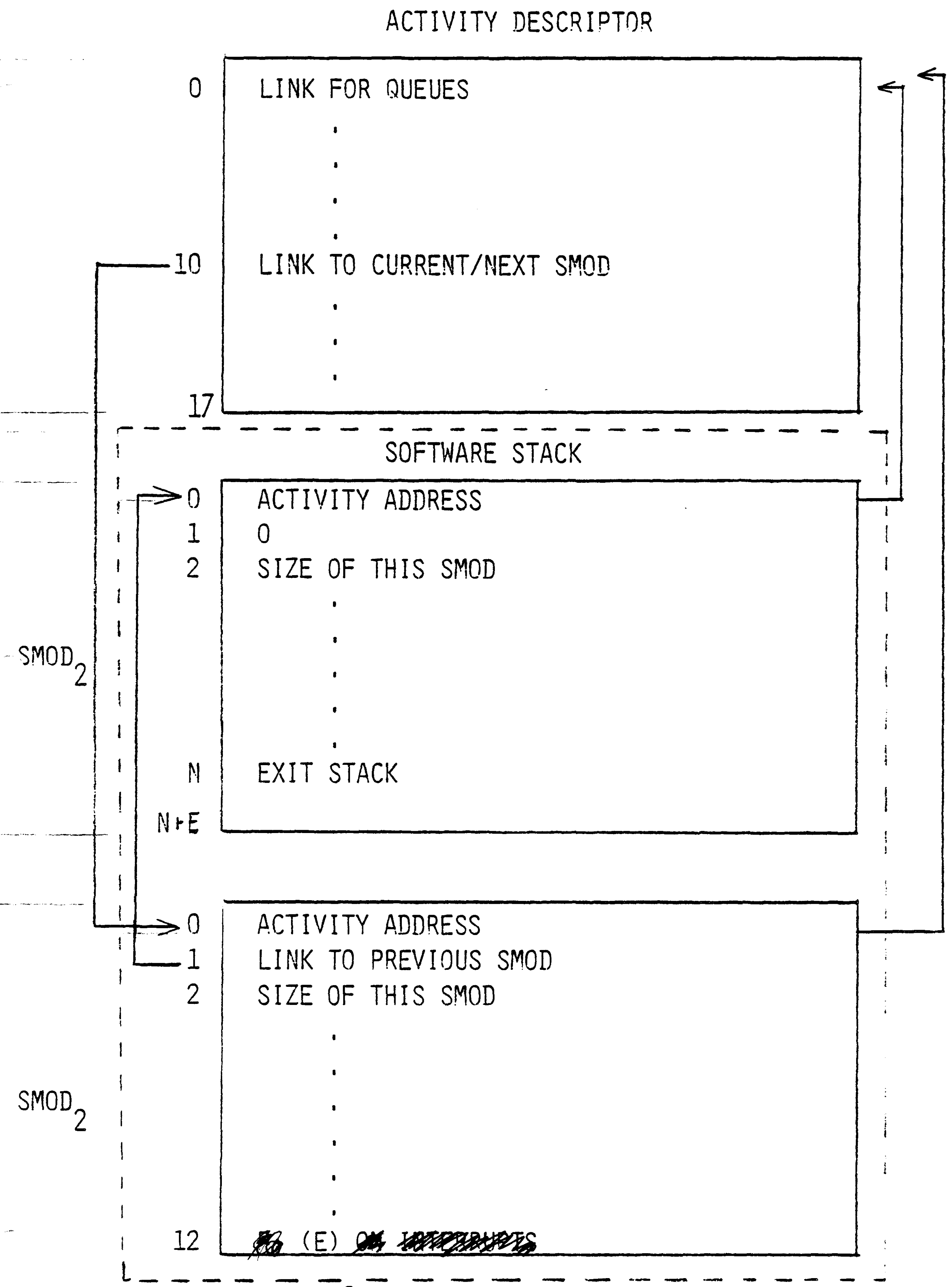


FIGURE 10-6. EXAMPLE OF SOFTWARE STACKING

POPCELL

ALLOWS ONE ACTIVITY TO CONTROL ANOTHER ACTIVITY IN A DIFFERENT IOP. ACTIVITY WILL PASS A MESSAGE THROUGH BUFFER MEMORY WHEN IT WANTS SOMETHING DONE BY THE SLAVE ACTIVITY.

BUILT BY THE KERNEL OF AN IOP WHEN AN ACTIVITY IN ANOTHER IOP DOES AN <u>ALERT</u> SERVICE REQUEST, CREATING AN ACTIVITY IN THE FIRST IOP.

IT IS REFERENCED BY THE KERNEL ON SUBSEQUENT <u>AWAKE</u> SERVICE REQUESTS FROM THE ORIGINAL ACTIVITY IN THE FIRST IOP.

PARCEL

0	LINK TO OTHER POPCELLS	
1	ACTIVITY ADDRESS	
2	PUSH/POPCELL	FIRST (ACTIVITY ADDRESS)
3	PUSH/POPCELL	LAST (ACTIVITY ADDRESS)
4	DAL QUEUE	FIRST
5	DAL QUEUE	LAST
6	UNUSED	
7	UNUSED	

FIGURE 10-7. POPCELL FORMAT

GLOBAL PTR

N!POPCELL 5203

INTER-IOP COMMUNICATION

OCCURS VIA ACCUMULATOR CHANNELS AND BUFFER MEMORY MESSAGE AREAS.

PARCEL PASSED VIA ACCUMULATOR CHANNEL MAY BE ENTIRE MESSAGE OR GIVE BUFFER MEMORY ADDRESS OF MESSAGE.

MESSAGES PASSED THROUGH BUFFER MEMORY HAVE A FIXED FORMAT AND ARE CALLED DISK ACTIVITY LINKS (DAL).

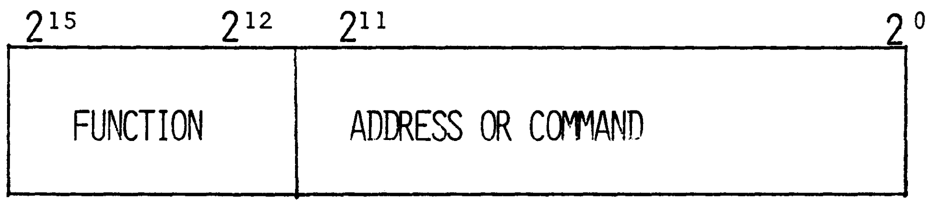


FIGURE 10-8. FORMAT OF ACCUMULATOR MESSAGE

FUNCTION CODES (2^{12} - 2^{15})	DEFINITION

0	COMMAND CODE IN BITS 2^0 - 2^{11}.
	0 = HALT THE IOP
	M$GO = INITIATE SYSDUMP
	AM$SYNC = SYNCHRONIZE IOP SOFTWARE CLOCK

1 - 7	UNUSED
10	A MESSAGE IS CONTAINED IN THE MOS MESSAGE AREA OF THE MIOP PROCESSOR AN AT ADDRESS WHICH IS CALCULATED USING THE LOWER ORDER 12 BITS OF THE ACCUMULATOR. EACH MESSAGE AREA IS OF SIZE 8 64-BIT WORDS. TO FIND THE MOS ADDRESS ONE MUST LEFT SHIFT THE ACCUMULATOR 3 BIT POSITIONS AND ADD THE BASE OF THE MOS MESSAGE AREA FOR PROCESSOR MIOP.
11	MESSAGE IS IN THE AREA CONTROLLED BY BIOP FOR MESSAGES TO THE OTHER PROCESSORS.
12	MESSAGE IS IN DIOP'S MESSAGE AREA. IOP - 2
13	MESSAGE IS IN XIOP'S MESSAGE AREA. IOP-3
14-15	UNUSED
16	USED BY CONCENTRATOR FOR ALLOCATING AND DEALLOCATING I/O BUFFERS IN ANOTHER IOP'S KERNEL STORAGE AREA.
17	THE ENTIRE COMMAND IS ENCODED IN THE LOWER 12 BITS, NO MOS DATA AREA IS ASSOCIATED WITH IT. 10 HEARTBEAT

TABLE 10-1. ACCUMULATOR MESSAGES

DISK ACTIVITY LINK

40_8 PARCELS IN LENGTH

SENT TO ANOTHER IOP TO REQUEST I/O BE PERFORMED.

USED BY DISK, TAPE, STATION, CONCENTRATOR AND INTERACTIVE STATION SUBSYSTEMS.

ALLOCATED FROM A CHAIN IN LOCAL MEMORY.

DE-ALLOCATED WHEN DONE RESPONSE RECEIVED.

PASSED THROUGH MESSAGE AREAS IN BUFFER MEMORY

SENDER MAINTAINS THE MESSAGE AREA.

PARCEL	
0	LINK FOR CHAINING DAL's
1	FUNCTION OF MESSAGE: 1=R/W DISK; 2=RELEASE MOS DAL; 3=MOVE CENTRAL TO MOS; 4=MOVE MOS TO CENTRAL; 5=SEND STATUS TO CPU; 6=CENTRAL TO MOS DONE; 7=MOS TO CENTRAL DONE; 20=ALERT; 21=ALERT DONE; 24=AWAKE; 25=RESPOND.
2	MOS UPPER OF DAL
3	MOS LOWER OF DAL
4	SENDER ACTIVITY DESCRIPTOR FOR RESPONSE
5	ACCUMULATOR MESSAGE
6	POPCELL ADDRESS
.	
.	
.	
37	

FIGURE 10-9. FORMAT OF DISK ACTIVITY LINK

MESSAGE (DAL) FLOW

0. ACTIVITY$_A$ BUILDS A DAL IN LOCAL MEMORY.

1. ACTIVITY$_A$ WRITES DAL TO IOP$_A$ MESSAGE AREA IN BUFFER MEMORY.

2. ACTIVITY$_A$ SENDS ACCUMULATOR MESSAGE TO IOP$_B$.

3. INPUT MESSAGE ACTIVITY (ACOM) IN IOP$_B$ READS IN DAL FROM BUFFER MEMORY.

   3A. ACTIVITY$_B$ PROCESSES MESSAGE.

4. ACTIVITY$_B$ UPDATES DAL FUNCTION CODE AND WRITES DAL TO ORIGINAL SPOT IN IOP$_A$ MESSAGE AREA.

   4A. ACTIVITY$_B$ DE-ALLOCATES LOCAL MEMORY DAL SPACE IN IOP$_B$.

5. ACTIVITY$_B$ SENDS ACCUMULATOR MESSAGE TO IOP$_A$.

6. ACOM IN IOP$_A$ READS IN DAL AND UPDATES THE DAL ALREADY IN LOCAL MEMORY.

   6A. ACTIVITY$_A$ CAN DE-ALLOCATE BUFFER MEMORY AND LOCAL MEMORY DAL SPACE IF DONE.

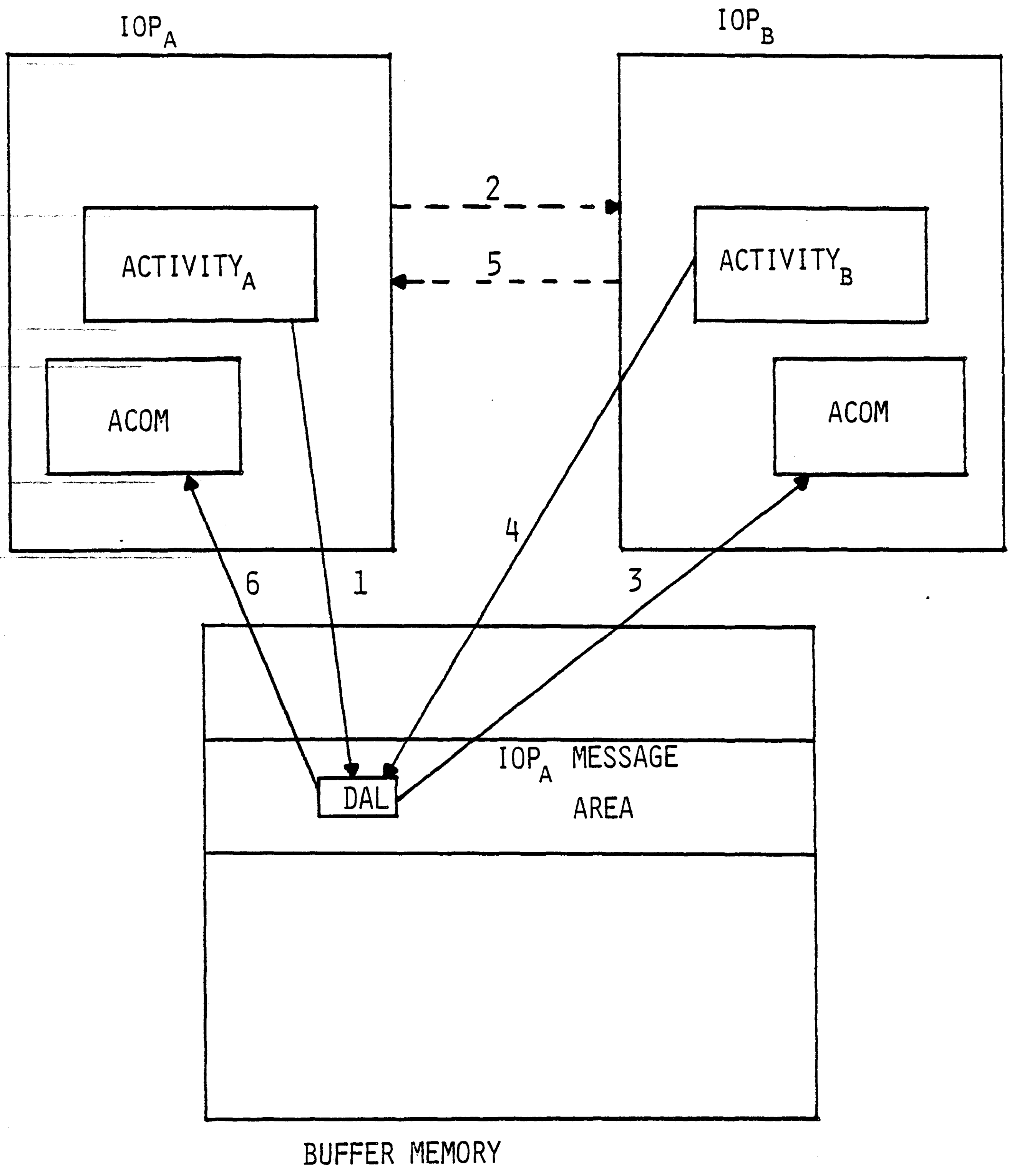


FIGURE 10-10. BASIC DAL FLOW

OPERAND REGISTER ASSIGNMENTS

REGISTER ASSIGNMENTS ARE MADE SO AS TO MAXIMIZE THE AMOUNT OF INTERRUPTIBLE CODE.

OVERLAYS USE DIFFERENT REGISTERS THAN THE KERNEL SO ON AN INTERRUPT, ONLY A, B AND C NEED BE SAVED.

ASSIGNMENTS ARE AS FOLLOWS:

0-177	KERNEL
200-277	DISK HANDLING
300-577	OVERLAYS
600-677	UNUSED
700-777	DEBUG ROUTINES

CHAPTER 11

OVERLAYS

GENERAL DESCRIPTION

EXECUTABLE PROGRAMS OR SUBROUTINES

RESIDE IN BUFFER MEMORY

READ INTO OVERLAY MEMORY IN LOCAL MEMORY FOR ACTIVATION

KERNEL MAINTAINS A BASE REGISTER CONTAINING THE OVERLAY'S BASE ADDRESS.

USUALLY SMALLER THAN 1024 PARCELS IN SIZE

COMPLETELY RE-ENTRANT

MAY REQUEST DATA AREAS FROM KERNEL FROM FREE MEM CHAIN

AN OVERLAY TABLE IS MAINTAINED TO PROVIDE INFORMATION ABOUT EACH OVERLAY.

OVERLAY TABLE

KERNEL RESIDENT

ONE ENTRY FOR EACH OVERLAY

FOUR PARCELS PER ENTRY

FIELD	PARCEL	
OT@WRO/OT@MUP	0	LENGTH IN WORDS (12 BITS);MOS UPPER (4 BITS)
OT@MLO	1	MOS LOWER ADDRESS OF OVERLAY
OT@PAR	2	# PARAMETERS (7 BITS); FIRST REGISTER (9 BITS)
OT@LOC	3	LOCAL MEMORY ADDRESS (0 IF NOT RESIDENT)

TABLE 11-1. OVERLAY TABLE

How large is OVLY?
Where is it?

How big should SMOD be?

OVERLAY MEMORY MANAGEMENT

THE KERNEL SETS UP AN AREA IN LOCAL MEMORY FOR OVERLAY MEMORY AT INITIALIZATION.

THE SIZE OF THIS AREA IS AN INSTALLATION PARAMETER.

THE AREA IS IMPLEMENTED AS TWO DOUBLY-LINKED LISTS:

THE ADJACENT BLOCK LIST IS ORDERED BY BLOCK ADDRESS AND IS USED TO MERGE PIECES AT RELEASE TIME.

THE MEMORY SEARCH LIST LINKS THE AVAILABLE BLOCKS FOLLOWED BY THE OVERLAY BLOCKS ORDERED IN A LEAST RECENTLY USED FASHION.

EACH BLOCK IN OVERLAY MEMORY HAS AN 8 PARCEL HEADER ASSOCIATED WITH IT.

THE ENTIRE LIST HAS AN 8 PARCEL HEADER AND A TRAILER ASSOCIATED WITH IT.

INITIALLY THE OVERLAY MEMORY CONSISTS OF A HEADER, TRAILER AND ONE BLOCK CONTAINING ALL THE AVAILABLE MEMORY.

THE KERNEL MAINTAINS A POINTER TO THE INITIAL HEADER IN REGISTER %MEMORY; A COUNT OF THE NUMBER OF OVERLAYS IN THE LIST IN REGISTER %OVCNT; AND A COUNT OF THE TOTAL NUMBER OF OVERLAY LOADS IN REGISTERS %OVLDS0 AND %OVLDS1.

LRU @ head

Best Fit @ Dealloc By Sorting

ONLY SIZE

IOP 0-3 = 30K

IOP 1-2 = 10K

FIELD	PARCEL	
MD@ID	0	HEADER IDENTIFIER:'MD'
MD@SUC	1	ADJACENT BLOCK LIST FORWARD POINTER
MD@PRE	2	ADJACENT BLOCK LIST BACKWARD POINTER
MD@TYP	3	BLOCK TYPE: MD$HEAD - HEADER OR TRAILER ENTRY = 0 MD$FREE - AVAILABLE = 1 MD$OLAY - CURRENTLY IN USE = 2 MD$BUF - FREE MEMORY BUFFER (DEFERRED) = 3
MD@FOR	4	MEMORY SEARCH LIST FORWARD POINTER
MD@BAK	5	MEMORY SEARCH LIST BACKWARD POINTER
MD@OVT	6	OVERLAY TABLE ENTRY ADDRESS IF MD@TYPE=MD$OLAY
	7	UNUSED

TABLE 11-2. OVERLAY MEMORY BLOCK HEADER

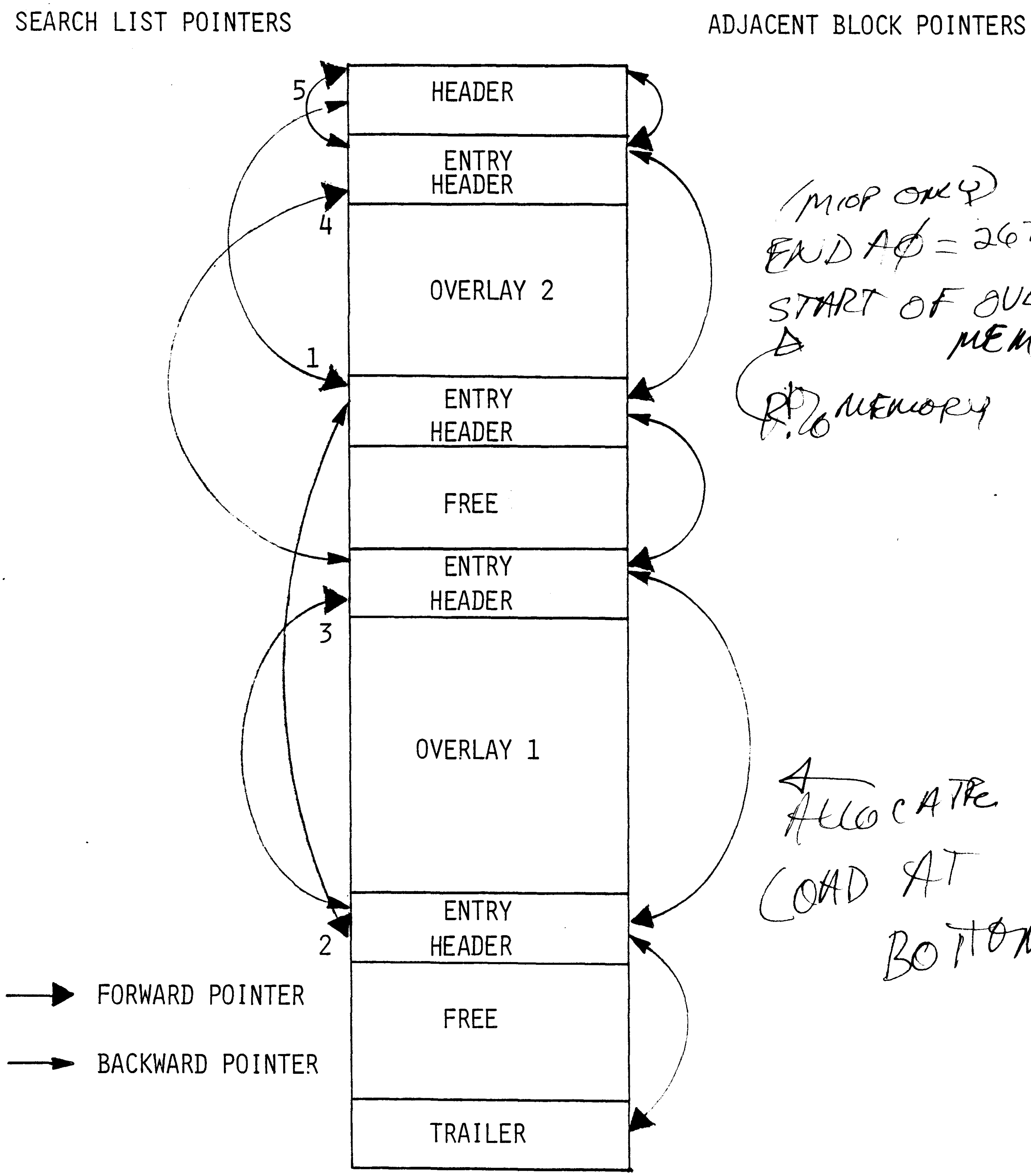
ONLY mem for IOP 1 ... = E~~222~~

First Fit

CRU marks as free even if not fit on allocation

Best Fit

~~FREE~~ mem sorted smallest to largest



(MIOP ONLY)
ENDAØ = 2672
START OF OVLY MEM
Δ
P.6 MEMORY

Δ ALLOCATE
LOAD AT
BOTTOM

FIGURE 11-1. EXAMPLE OF OVERLAY MEMORY LIST POINTERS

OVERLAY FORMAT

MAXIMUM OF 2048 PARCELS INSTALL PARM = OVL$LIM = 6K parcels

FIRST FOUR PARCELS (8 CHARACTERS) CONTAIN OVERLAY NAME

OVERLAY HAS ONE ENTRY POINT
    DEFAULT IS PARCEL 6

READ ONLY, SO ANY VARIABLE DATA AREAS MUST BE REQUESTED FROM THE KERNEL.
    AN ACTIVITY MUST RETURN MEMORY REQUESTED BY ONE OF ITS' OVERLAYS BEFORE TERMINATING.

PARCEL	
0	OVERLAY NAME
1	OVERLAY NAME
2	OVERLAY NAME
3	OVERLAY NAME
4	TYPE (1 BIT) 0=EXECUTABLE, 1=DATA; OVERLAY NUMBER (15 BITS)
5	PARAMETER DEFINITION: # (7 BITS); FIRST REG (9 BITS)
6	ENTRY POINT
7	
.	
.	
.	INSTRUCTIONS
3777	

FIGURE 11-2. TYPICAL OVERLAY FORMAT

OVERLAY DEFINITION

OVERLAY MACRO SETS UP PARAMETERS FOR AN OVERLAY.

SYM	OVERLAY	OVLNAME, FP=,NP=,BASEREG=, ENTRY=,TYPE=

don't use

SYM	OPTIONAL QUALIFIER FOR ALL SYMBOLS DEFINED IN OVERLAY. IF BLANK, OVLNAME IS USED AS QUALIFIER.
OVLNAME	NAME OF THIS OVERLAY
FP	FIRST REGISTER TO PASS EXPECTED PARAMETERS
NP	NUMBER OF PARAMETERS DEFAULT IS 0
BASEREG	BASE REGISTER TO USE FOR THIS OVERLAY DEFAULT IS %B (SET UP BY KERNEL)
ENTRY	ENTRY POINT OF OVERLAY DEFAULT IS PARCEL 6
TYPE	IF TYPE = DATA IS SPECIFIED THEN OVERLAY IS NON-EXECUTABLE.

O$XXX EQUALS (1) num in APTEXT

OVERLAY XXX

REGDEFS 330+

/ num

⋮

CODE

⋮

END

OVERLAY CALLS

CONTROL IS PASSED TO AN OVERLAY VIA THE CALL AND GOTO SERVICE REQUESTS.

CALL RESULTS IN A 'PUSH' OF THE CALLER'S SMOD ON THE SOFTWARE STACK.

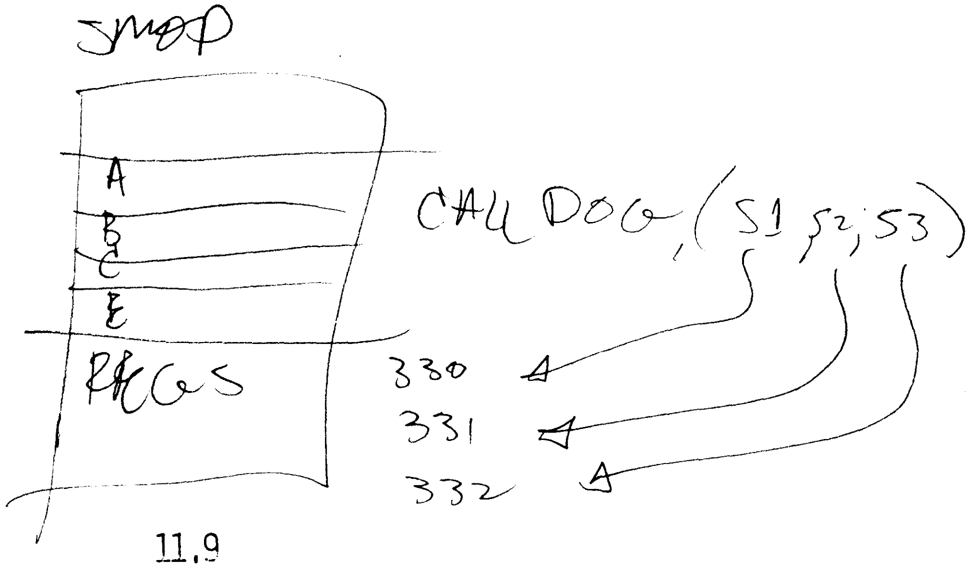
GOTO PASSES CONTROL DIRECTLY TO NEW OVERLAY. CALLERS SMOD IS NOT SAVED.

AN OVERLAY RETURNS CONTROL TO CALLER VIA THE RETURN SERVICE REQUEST.

RETURN RESULTS IN A 'POP' OF THE CALLER'S SMOD OFF THE SOFTWARE STACK.

PARAMETERS MAY BE PASSED TO A CALLED OVERLAY

OVERLAY MAY RETURN PARAMETERS IN CALLER'S SMOD AREA VIA THE RETREG MACRO.



EXAMPLE:

```
                                   OVERLAY        DON
        2       O$DON      EQUALS      O$DON

                * Copyright CRAY RESEARCH, INC., 1979, 1980, 1981.


                                   REGDEFS        ,(P1,P2,P3,P4),(T0,T1)
 6    020330   023331   024334     T0=P1-P2
11    020330   022331   024335     T1=P1+P2
                                   RETREG         T0,P3
                                   RETREG         T1,P4
                                   RETURN
                                   END
```

EMITTED BY MACRO

```
                  OVERLAY        PETE
     1   O$PETE   EQUALS     O$PETE

          * Copyright CRAY RESEARCH, INC., 1979, 1980, 1981.


                             REGDEFS     ,(AA,S1,S2,R1,R2,V1,N2)
  6    010030   024335       V1=30
 10    010012   024336       V2=12
 12    010027   024331       S1=27
 14    010010   024332       S2=10
                             CALL        DON,(S1,S2,R0=R1,R0=R2),A1=R1,A2=N2
 37    020333   021334       A=R1&R2
                             RETURN
                             END
```

R!%PSMOD +P3

Previous Storage MOD ptr

REG OFFSET

RESPONE IN R1

" " R2

KERNEL CONSOLE CALLABLE OVERLAYS

CERTAIN ACTIVITIES MAY BE CREATED BY KEYING IN OVERLAY NAME AT KERNEL CONSOLE.

CALL OVLY

KERNEL CREATES ACTIVITY AND PUTS IT ON IOP CENTRAL PROCESSOR QUEUE.

ACTIVITY THEN PROCEEDS AS ANY OTHER ACTIVITY.

OPERATOR MAY USE THIS FACILITY TO:

- DEADSTART CPU
- BRING UP THE STATION
- START A CONCENTRATOR
- ENTER THE INTERACTIVE CONCENTRATOR
- RUN TEST ROUTINES

CHAPTER 12

KERNEL

FUNCTIONS

ACTIVITY SCHEDULING

CENTRAL PROCESSOR ALLOCATION Activity Dispatching

LOCAL AND BUFFER MEMORY ALLOCATION

INTRA- AND INTER-ACTIVITY COMMUNICATION PROCESSING

INTER-PROCESSOR COMMUNICATION PROCESSING

INTERRUPT PROCESSING

CHARACTERISTICS

LOCAL MEMORY RESIDENT

EXECUTES IN EACH IOP WITH MINOR MODIFICATION

EXECUTES IN NON-INTERRUPTIBLE MODE

BASIC COMPONENTS

ACTIVITY DISPATCHER

INTERRUPT HANDLER

SERVICE REQUEST PROCESS

MEMORY CONTROL

COMMON SUBROUTINES

KERNEL TABLES

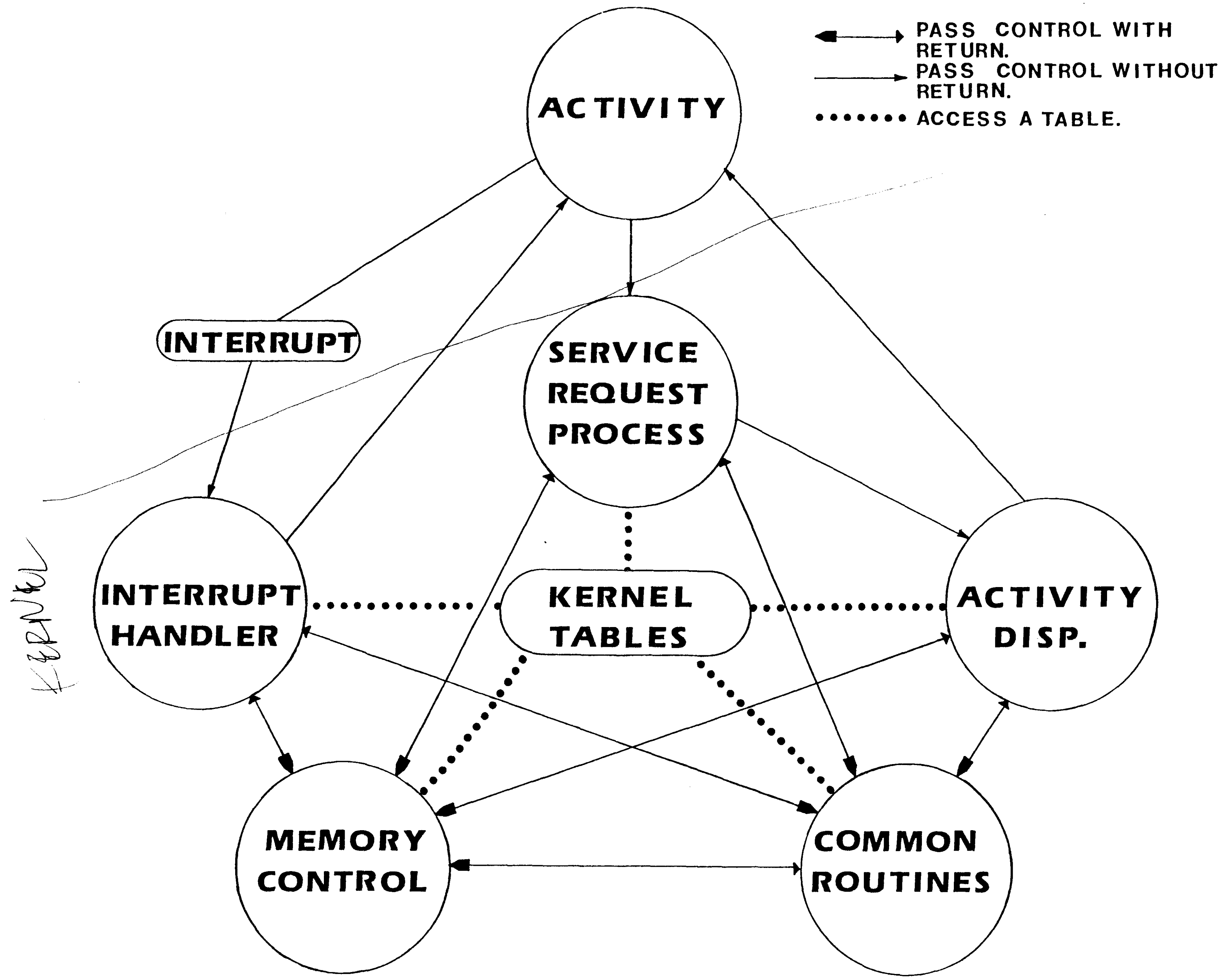


FIGURE 12-1. KERNEL/ACTIVITY INTERACTION

ACTIVITY DISPATCHER

MANAGES ACTIVITIES THROUGH USE OF ACTIVITY DESCRIPTORS AND STORAGE MODULES.

TRANSFERS CONTROL FROM ONE ACTIVITY TO ANOTHER

SWAPS SOFTWARE STACKS BETWEEN LOCAL AND BUFFER MEMORY.

MAINTAINS OVERLAY MEMORY

CONTAINS KERNEL IDLE LOOP

ENTERED FROM KERNEL SERVICE REQUEST PROCESS

DEMON ACTIVITIES

PERFORM HIGH PRIORITY TASKS, OFTEN IN NON-INTERRUPTIBLE MODE.

EACH CONSISTS OF ONE OVERLAY.

SOFTWARE STACKS (SMODS) ARE LOCAL MEMORY RESIDENT

NO REGISTERS SAVED

NOT SWAPPED TO BUFF MEM

ACTIVITY DESCRIPTORS NEVER DEALLOCATED

ASSEMBLED WITH KERNEL, SO MAY CALL KERNEL SUBROUTINES

ACOM DEMON

BCOM DEMON FOR TAPE

ACOM OVERLAY

HANDLES IOP TO IOP COMMUNICATION VIA BUFFER MEMORY.

READS IN DAL FROM BUFFER MEMORY AND PASSES CONTROL TO DISK DEMON, AMSG DEMON, OR CDEM DEMON.

ACTIVATED BY IOP TO IOP INTERRUPT ANSWERING.

AMSG DEMON
AMSG OVERLAY

PROCESSES DALS FOR ALERT, AWAKE AND RESPOND. POPCELLS

PROCESSES SOME ACCUMULATOR-ONLY MESSAGES.

ACTIVATED BY ACOM.

CDEM DEMON
CDEM OVERLAY

CRAY DEMON

HANDLES CPU TO MIOP COMMUNICATON AND STATION AND CONCENTRATOR TRANSFERS IN BIOP.

ACTIVATED BY CPU TO MIOP INTERRUPT ANSWERING OR ACOM.

DISK DEMON
DISK OVERLAY

NUCLEUS OF THE DISK SUBSYSTEM.

ACTIVATED BY ACOM OR DISK INTERRUPT ANSWERING.

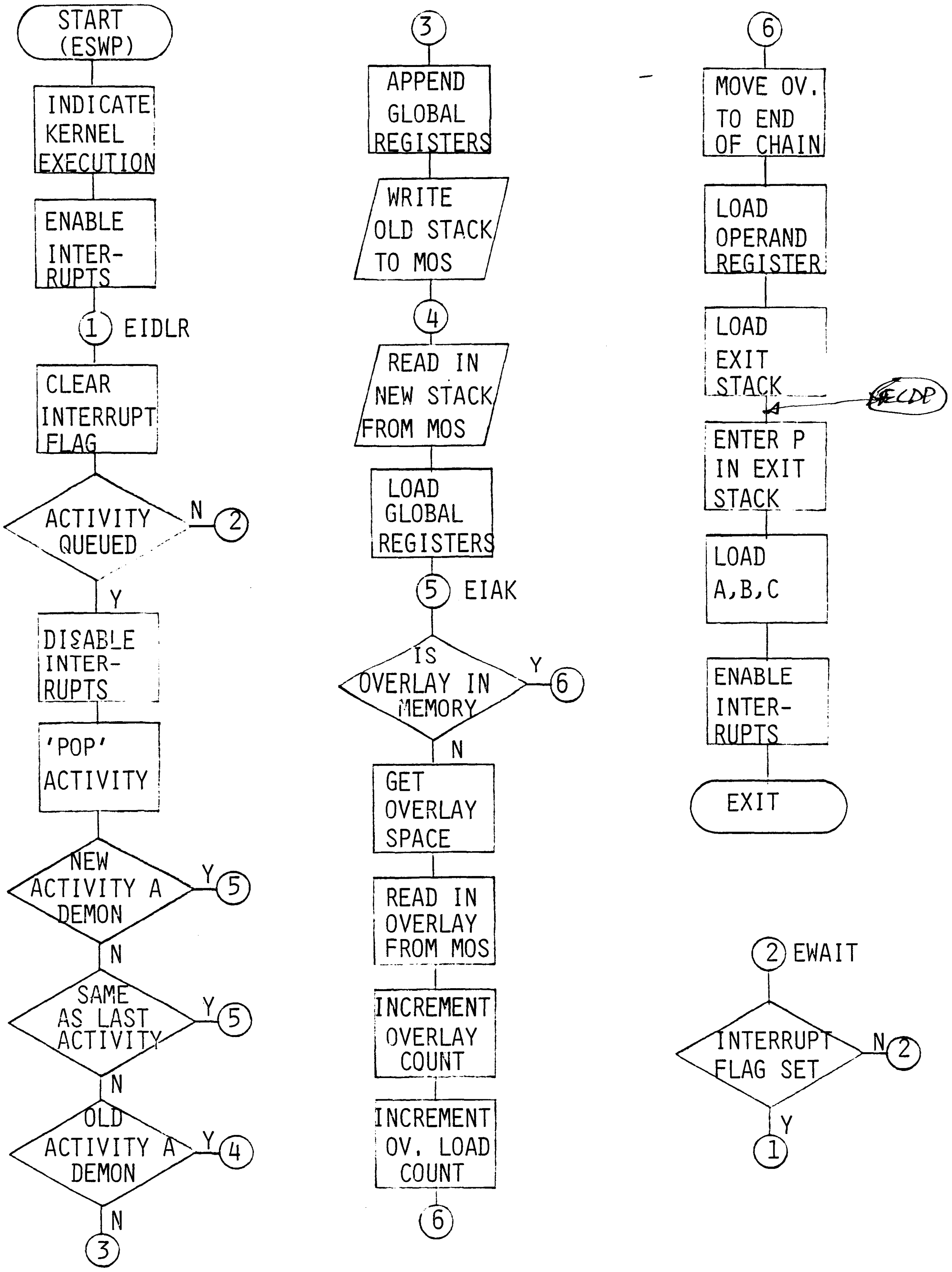


FIGURE 12-2. ACTIVITY DISPATCHER FLOW DIAGRAM.

INTERRUPT ANSWERING

ENTERED WHEN A DONE FLAG SETS ON A CHANNEL WHICH HAS INTERRUPTS ENABLED.

HARDWARE READS IAA OUT OF EXIT STACK INTERRUPT ANSWERING ADDR

CURRENT ACTIVITY'S A, B, C, E AND P REGISTERS ARE SAVED. FOR TRACE FUNC.

INTERRUPTING CHANNEL NUMBER IS READ FROM CHANNEL 0.

JUMP TO APPROPRIATE HANDLER IS DETERMINED FROM OFFSET INTO AN INTERRUPT JUMP TABLE (EITB).

RETURNS CONTROL TO INTERRUPTED ACTIVITY WHEN ALL INTERRUPTS PROCESSED.

STANDARD INTERRUPT HANDLERS ARE:

- IPFI - PROGRAM FETCH REQUEST INTERRUPTS
- IPXS - PROGRAM EXIT STACK INTERRUPTS
- ILMERR - LOCAL MEMORY ERROR INTERRUPTS
- IRTC - REAL-TIME CLOCK INTERRUPTS
- IIAP - IOP TO IOP INPUT INTERRUPTS
- IOAP - IOP TO IOP OUTPUT INTERRUPTS

OPTIONAL INTERRUPT HANDLES ARE:

- IREPORT - ERROR LOGGING CHANNEL MIOP ONLY
- IEXP - EXPANDER CHANNEL MIOP
- IDID - DISK CHANNELS BIOP/DIOP
- ICRI - CRAY-1 LOW SPEED INPUT CHANNEL MIOP
- ICRY - CRAY-1 LOW SPEED OUTPUT CHANNEL MIOP
- IBMX - BLOCK MULTIPLEXER CHANNELS XIOP
- ITIA - CRT INPUT AND OUTPUT CHANNELS

NO INTERRUPT HANDLER FOR CPU MEMORY CHANNEL OR BUFFER MEMORY CHANNEL.

KERNEL WAITS FOR CHANNEL TO FREE, ISSUES I/O REQUEST, THEN PROCEEDS; OR WAITS FOR CHANNEL TO FINISH.

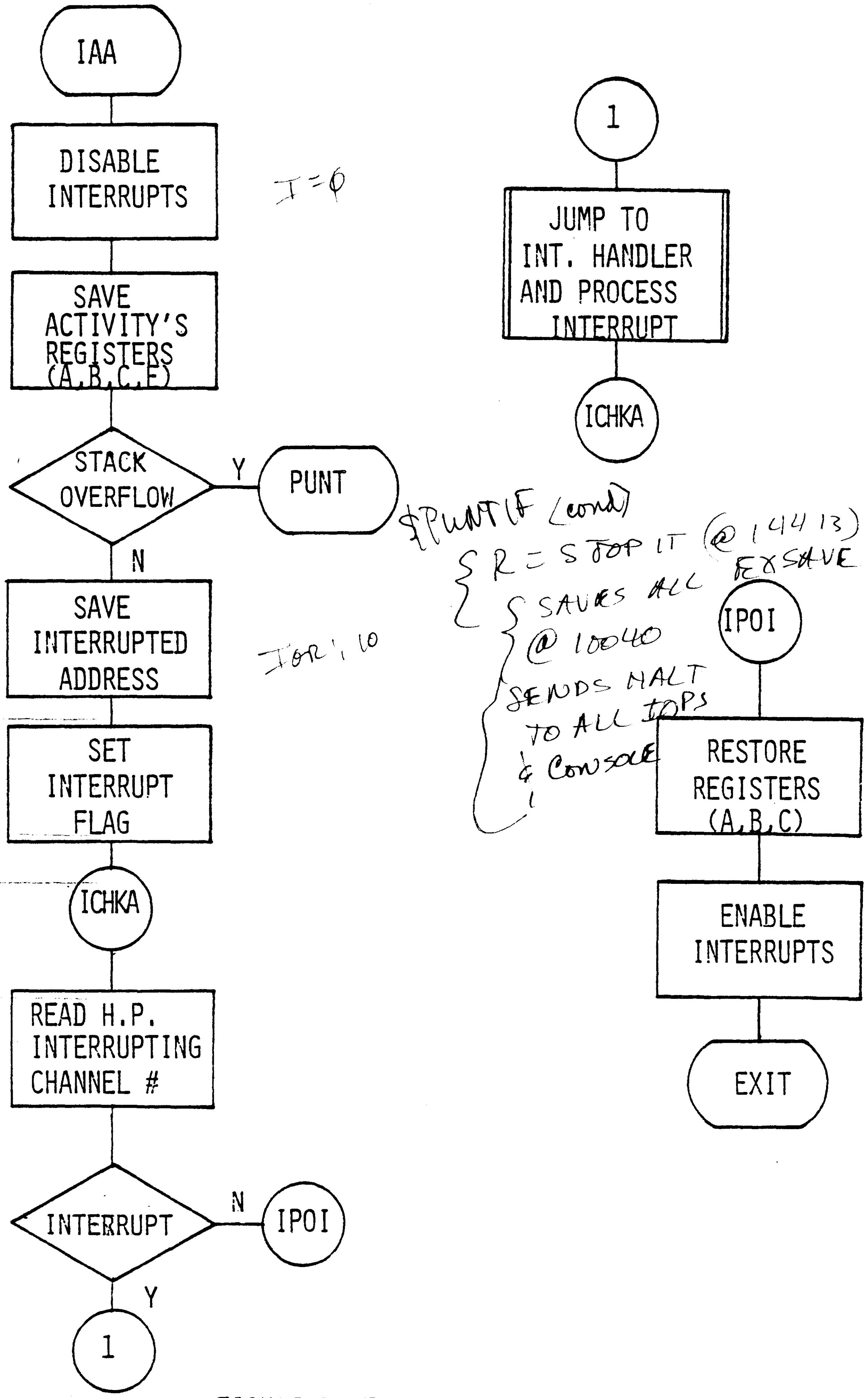


FIGURE 12-3. INTERRUPT ANSWERING FLOWCHART

KERNEL ERROR HALT PROCESS

ENTERED WHEN SOFTWARE DETECTS AN ERROR ($PUNTIF MACRO IS EXECUTED)

DISABLES INTERRUPTS

SAVES A, B, C, E REGISTERS, EXIT STACK, AND ALL CHANNEL BZ AND DN FLAGS.

SENDS ERROR HALT MESSAGE TO KERNEL CONSOLE

HALTS OTHER IOP'S

PASSES CONTROL TO SYSDUMP.

SERVICE REQUEST PROCESS

PERFORMS ESSENTIAL SERVICES FOR ACTIVITIES, IN NON-INTERRUPTIBLE MODE.

ACTIVITY CALLS A MACRO WHICH PASSES PARAMETERS TO ANOTHER MACRO, WHICH SETS UP PARAMETERS AND DOES A RETURN JUMP TO SERVICE REQUEST PROCESS.

SERVICE REQUEST PROCESS IS AS FOLLOWS:

1) LOCK OUT INTERRUPTS

2) SAVE A, B, E AND P IN SMOD

3) SAVE SPECIFIED OPERAND REGISTERS IN SMOD
   SAVE EXIT STACK IN SMOD IF ANY REGISTERS SAVED.

4) GET FUNCTION CODE FROM FUNREG

5) JUMP TO ADDRESS AT FCTABLE + FUNCTION CODE

DEPENDING ON FUNCTION, CONTROL IS PASSED TO REQUESTER, KERNEL, OR NEW OVERLAY ON COMPLETION.

SAVES A & B even though mashed by CALL macro

CALL PROCEDURE:

LOCATION	RESULT	OPERAND
L	SERVICE	PARAMS, B=FUNREG, A1=START, A2=LAST

L	OPTIONAL STATEMENT LABEL
SERVICE	DESIRED SERVICE FUNCTION NAME
PARAMS	NECESSARY PARAMETERS, OVERLAY NAME, ETC. THESE ARE PUT IN REGISTERS FOLLOWING FUNREG.
FUNREG	REGISTER FOR PASSING FUNCTION CODE
START	FIRST REGISTER TO SAVE
LAST	LAST REGISTER TO SAVE

EXAMPLE:

```
          IDENT          SERVICE
          REGDEFS        ,(CC,BB,R1,R2,R3,R4,R5,R6)
CAT       DELAY          1,B=BB,A1=R3,A2=R5
          PAUSE          1
          GETMEM         124,CC
          END
```

SERVICE REQUESTS EXIST FOR:

CREATING, RESCHEDULING AND TERMINATING ACTIVITIES

PASSING CONTROL BETWEEN OVERLAYS

LOCATING AN OVERLAY IN BUFFER MEMORY

CONTROLLING PUSH AND TIMER QUEUES

SENDING AND RECEIVING MESSAGES ON CRT CHANNELS

SENDING RESPONSES TO OTHER IOPS

REQUESTING ANOTHER IOP TO CREATE OR ACTIVATE AN ACTIVITY

INITIATING FRONT-END AND BLOCK MUX I/O

SENDING MESSAGES TO CPU AND RECEIVING A RESPONSE

ALLOCATING AND RELEASING LOCAL AND BUFFER MEMORY

MOVING DATA BETWEEN BUFFER MEMORY AND CENTRAL MEMORY

MOVING DATA BETWEEN LOCAL MEMORY AND BUFFER MEMORY.

FLUSHING THE OVERLAY MEMORY BUFFERS.

SELECTED SERVICE FUNCTIONS

CREATE

- SETS UP INDEPENDENT ACTIVITY AND PLACES IT ON CP QUEUE AT PRIORITY.
  - INITIALIZES ACTIVITY DESCRIPTOR AND SOFTWARE STACK
  - WRITES SOFTWARE STACK TO BUFFER MEMORY
  - RETURNS CONTROL TO REQUESTER

TERMINATE

- TERMINATES THIS ACTIVITY
  - RELEASES BUFFER MEMORY SOFTWARE STACK AREA
  - RELEASES ACTIVITY DESCRIPTOR AREA
  - RETURNS CONTROL TO KERNEL

CALL

- PASSES CONTROL TO ANOTHER OVERLAY
  - RESULTS IN A 'PUSH' ONTO THE SOFTWARE STACK
  - NEW OVERLAY GETS CONTROL DIRECTLY

GOTO

- PASSES CONTROL TO ANOTHER OVERLAY
  - CALLER'S SMOD IS NOT SAVED
  - NEW OVERLAY GETS CONTROL DIRECTLY

RETURN

- RETURNS CONTROL TO OVERLAY CALLER
  - RESULTS IN A 'POP' OFF THE SOFTWARE STACK
  - ACTIVITY IS THEN PLACED ON CP QUEUE.
  - RETURNS CONTROL TO KERNEL

RETURN TO NO ONE IS A TERM!

ALERT

- CREATES AN ACTIVITY IN A DIFFERENT IOP.
  - RETURNS CONTROL TO THE KERNEL.

AWAKE

- ACTIVATES AN ACTIVITY IN A DIFFERENT IOP.
  - ACTIVITY MUST HAVE BEEN PREVIOUSLY ALERTED.
  - RETURNS CONTROL TO REQUESTER OR KERNEL, DEPENDING ON WHETHER OR NOT A RESPONSE IS DESIRED FROM THE AWAKENED ACTIVITY.

RESPOND

- SENDS A RESPONSE TO THE ACTIVITY WHICH DID AN AWAKE.
  - RETURNS CONTROL TO REQUESTER.

ASLEEP — new 1.11

ALERT MECHANISM FLOW:

1. ACTIVITY$_A$ DOES AN <u>ALERT</u> SERVICE REQUEST SPECIFYING THE FIRST OVERLAY OF THE ACTIVITY AND THE IOP TO CREATE IT IN.

2. KERNEL$_A$ BUILDS A DAL FROM THIS INFORMATION AND SENDS IT TO KERNEL$_B$ THROUGH BUFFER MEMORY.

3. KERNEL$_A$ IDLES ACTIVITY$_A$.

4. AMSG$_B$ BUILDS A POPCELL, CREATES THE NEW ACTIVITY$_B$, PUTS POPCELL ADDRESS IN AD$_B$ AND PLACES AD$_B$ ON CP QUEUE.

5. AMSG$_B$ THEN PLACES ADDRESS OF POPCELL IN DAL AND KERNEL$_B$ RETURNS THE DAL TO KERNEL$_A$ THROUGH BUFFER MEMORY.

6. AMSG$_A$ THEN PLACES POPCELL ADDRESS IN ACTIVITY$_A$ DESCRIPTOR (PARCEL 6).

7. AMSG$_A$ THEN PLACES ACTIVITY$_A$ ON CP QUEUE.

8. WHEN ACTIVITY$_B$ IS POPPED OFF THE CP QUEUE, IT CHECKS THE POPCELL DAL QUEUE. IF THIS IS EMPTY, IT PUSHES ITSELF ONTO THE POPCELL QUEUE (PARCELS 2 & 3).

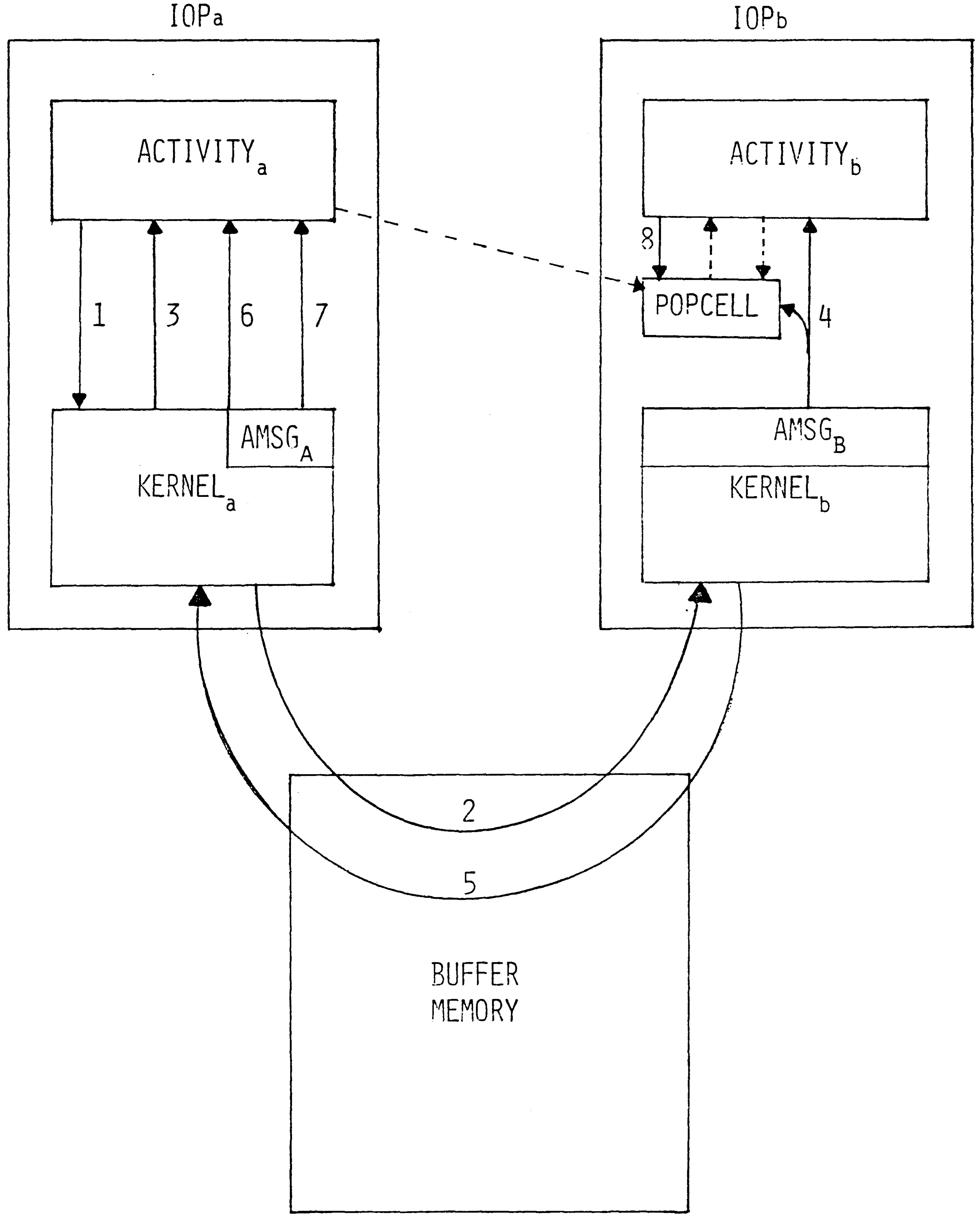


FIGURE 12-4. ALERT MECHANISM FLOW

AWAKE MECHANISM FLOW:

1. ACTIVITY$_A$ BUILDS A DAL FOR ACTIVITY$_B$ AND DOES AN <u>AWAKE</u> SPECIFYING WHICH IOP, POPCELL, AND DAL, AND WHETHER A RESPONSE IS DESIRED.

2. KERNEL$_A$ THEN PASSES DAL TO KERNEL$_B$ THROUGH BUFFER MEMORY.

3. KERNEL$_A$ THEN IDLES ACTIVITY$_A$ OR RETURNS CONTROL TO IT DEPENDING ON WAIT/NO WAIT PARAMETER.

4. AMSG$_B$ THEN PLACES DAL ON POPCELL DAL QUEUE AND ACTIVATES ACTIVITY$_B$ IF NOT ACTIVE.

5. ACTIVITY$_B$ THEN PROCESSES THE NEXT DAL ON THE QUEUE.

6. ACTIVITY$_B$ DOES A <u>RESPOND</u> AND KERNEL$_B$ PLACES RETURNED PARAMETER IN DAL.

7. KERNEL$_B$ SENDS DAL TO KERNEL$_A$ THROUGH BUFFER MEMORY.

8. IF WAIT WAS SPECIFIED, AMSG$_A$ RETURNS PARAMETER TO ACTIVITY$_A$ AND PLACES IT ON CPU QUEUE.
   8A. IF MORE DALS ON POPCELL QUEUE, GO TO 5.

9. ACTIVITY$_B$ PUSHES ITSELF ON THE POPCELL QUEUE, AWAITING ANOTHER AWAKE.

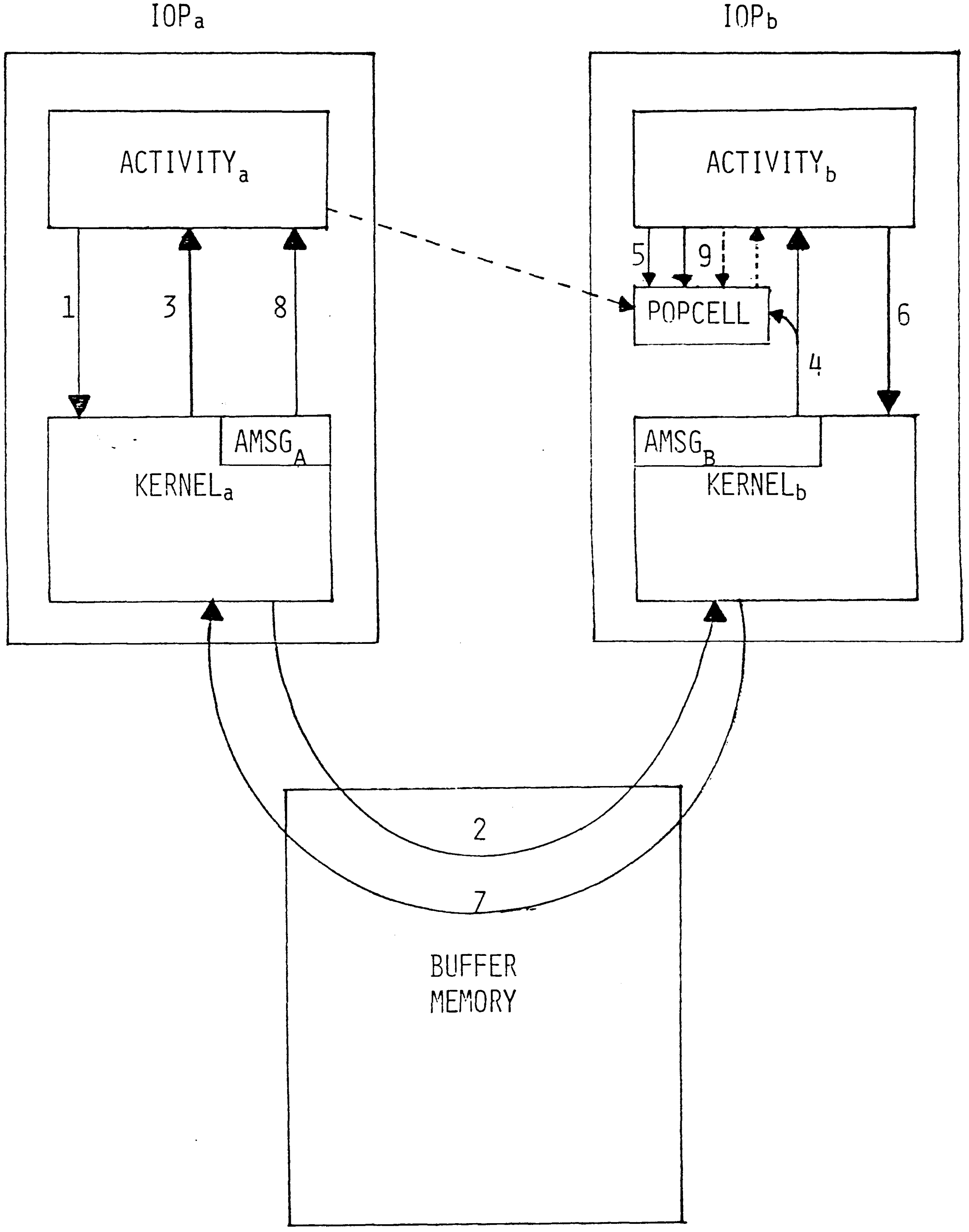


FIGURE 12-5. - AWAKE MECHANISM FLOW

Disk Driving (DAL Processing) routines

INTERNAL SUBROUTINES

PERFORM COMMON TASKS REQUIRED BY DIFFERENT PARTS OF THE SYSTEM

CALLED BY KERNEL ROUTINES, DEMON ACTIVITIES, AND OTHER OVERLAYS ASSEMBLED WITH THE KERNEL.

CALL LOCKS OUT INTERRUPTS

EXECUTE IN NON-INTERRUPTIBLE MODE USE OF KERNEL REGS

ROUTINES AVAILABLE FOR:

- SETTING UP ACTIVITY DESCRIPTOR AND BUFFER MEMORY SOFTWARE STACK AREAS.
- MOVING REGISTERS IN AND OUT OF SMODS.
- MAINTAINING QUEUES.
- MAINTAINING OVERLAY MEMORY CHAIN.

EMSG 13015 - in middle of Internal subs:

? EMSGTOP 3222 : Subroutine in Service Routine code

LOCAL MEMORY CONTROL

MAINTAINS LOCAL FREE MEMORY CHAIN

ALLOCATED IN MULTIPLES OF 4 PARCELS

MAINTAINS LOCAL DISK BUFFER CHAIN

ALLOCATED IN 4000_8 PARCEL BLOCKS

MAINTAINS DAL CHAIN

ALLOCATED IN 40_8 PARCEL BLOCKS

KRECORD MACRO to CALL

MAINTAINS A 1000_8 PARCEL TRACE BUFFER FOR RECORDING THE OCCURRENCE OF SPECIFIC EVENTS.

EACH TRACE ENTRY IS 10_8 PARCELS

Δ

WHEN FULL, THE TRACE BUFFER IS WRITTEN TO A 4000_8 WORD CIRCULAR BUFFER IN BUFFER MEMORY.

→ REG OVLAYS CANT CALL

(Common)

BUFFER MEMORY CONTROL

CONTROLS ALLOCATION OF 512 WORD BLOCKS IN BUFFER MEMORY KERNEL STORAGE AREAS.

CONTROLS ALLOCATION OF BUFFER MEMORY SOFTWARE STACK AREAS.

CONTROLS ALLOCATION OF DALs IN MESSAGE AREAS

PERFORMS READS AND WRITES TO BUFFER MEMORY AND OVER BIOP'S HIGH SPEED MEMORY CHANNEL.

CHAPTER 13

DISK SUBSYSTEM

FUNCTIONS

MOVES DATA BETWEEN CENTRAL MEMORY AND DISK

PERFORMS DISK ERROR RECOVERY

CPU MUST INITIATE I/O BY MAKING A DISK REQUEST

CPU IS RESPONSIBLE FOR DEVICE ASSIGNMENTS AND DATASET ALLOCATIONS.

OVERVIEW OF DISK I/O VIA DIOP

1. CPU PASSES DISK REQUEST TO MIOP.

2. MIOP TRANSFERS REQUEST TO DIOP THROUGH BUFFER MEMORY.

WRITE:

3. BIOP TRANSFERS DATA FROM CENTRAL MEMORY TO BUFFER MEMORY.

4. DIOP TRANSFERS DATA FROM BUFFER MEMORY TO DISK.

READ:

3. DIOP TRANSFERS DATA FROM DISK TO BUFFER MEMORY.

   3A. DIOP TRANSFERS ADDITIONAL SECTORS TO BUFFER MEMORY (READ AHEAD).

4. BIOP TRANSFERS DATA FROM BUFFER MEMORY TO CENTRAL MEMORY.

5. DIOP RETURNS REQUEST TO MIOP INDICATING I/O IS FINISHED.

6. MIOP PASSES RESPONSE TO CPU.

WRITE BEHIND
ON DIOP NO HSP

1140 .8 μsecs Hardware
~~fora~~ write on DIOP

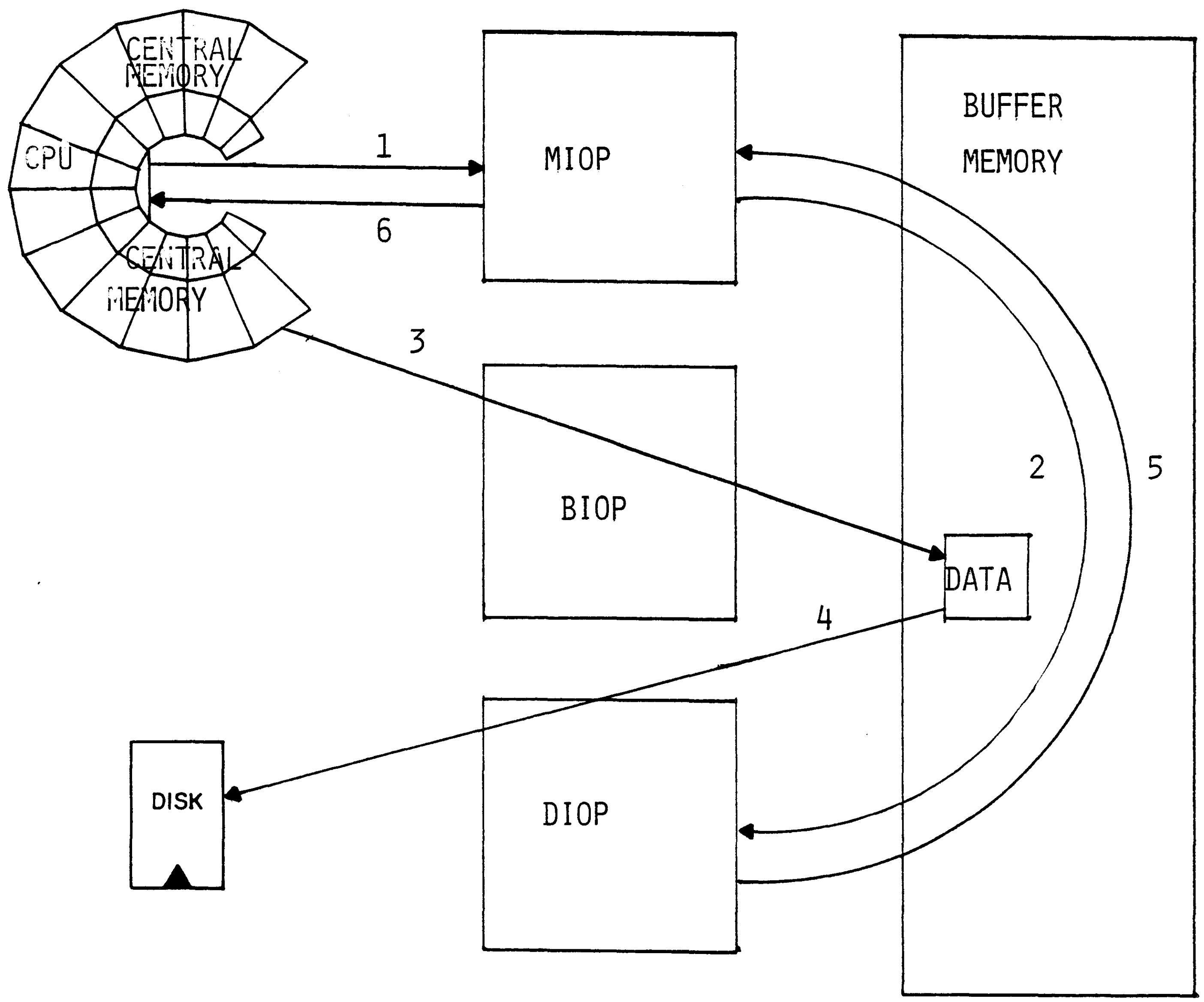


FIGURE 13-1 . DISK WRITE VIA DIOP

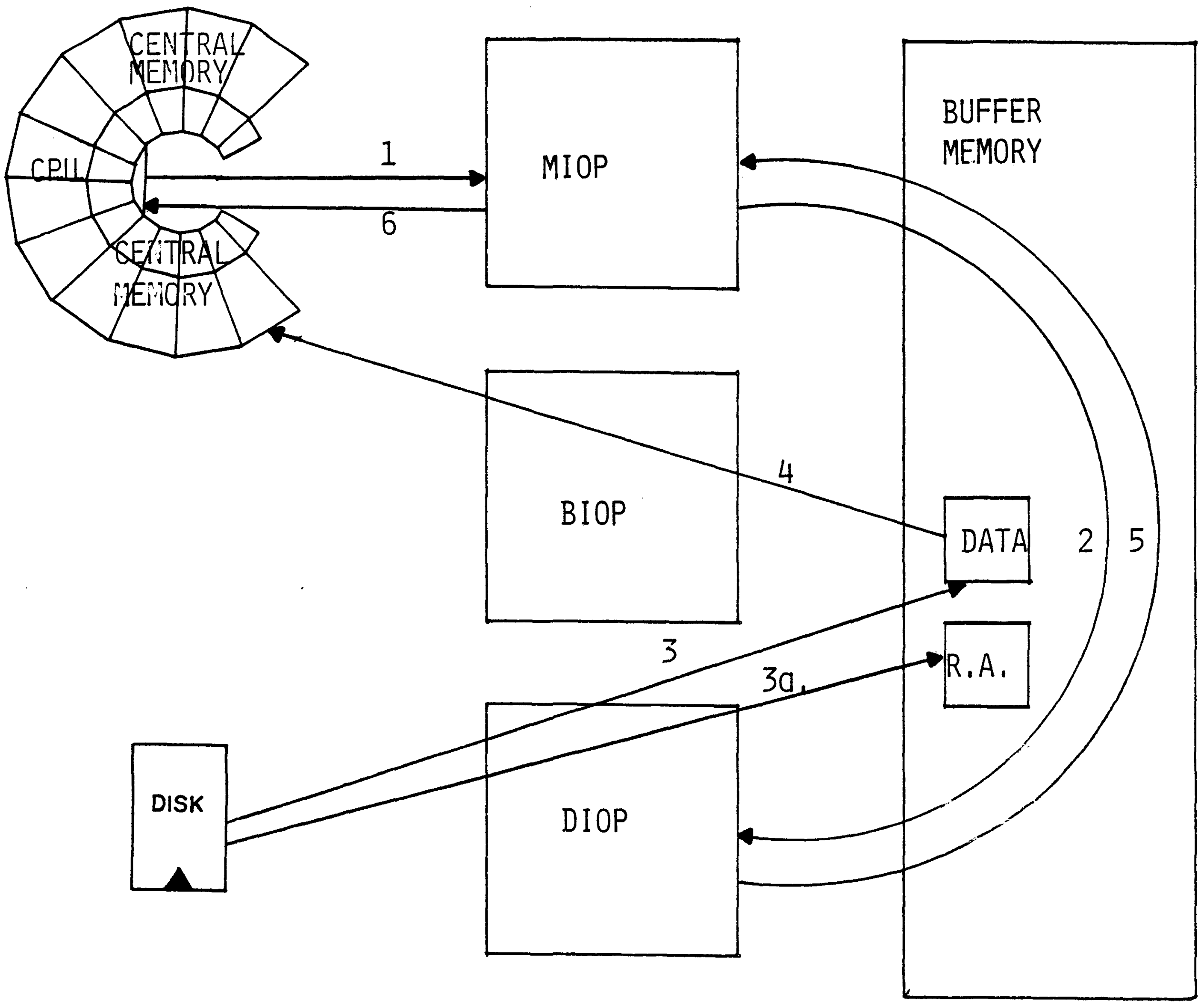


FIGURE 13-2 DISK READ VIA DIOP

OVERVIEW OF DISK I/O VIA BIOP

1. CPU PASSES DISK REQUEST TO MIOP

2. MIOP TRANSFERS REQUEST TO BIOP THROUGH BUFFER MEMORY.

WRITE:

3A. BIOP TRANSFERS DATA FROM CENTRAL MEMORY TO LOCAL MEMORY.
3B. BIOP TRANSFERS DATA FROM LOCAL MEMORY TO DISK.

READ:

3A. BIOP TRANSFERS DATA FROM DISK TO LOCAL MEMORY.

3B. BIOP TRANSFERS DATA FROM LOCAL MEMORY TO CENTRAL MEMORY.

3C. BIOP TRANSFERS ADDITIONAL SECTORS FROM DISK TO BUFFER MEMORY (READ AHEAD).

4. BIOP RETURNS REQUEST TO MIOP INDICATING I/O IS FINISHED.

5. MIOP PASSES RESPONSE TO CPU.

Hardware Timing

47 μs CPU-LM 1044.4 μs
980 μs To disk
1027 μs to disk

2.4 μs 13.6 Packet traffic
15 μs CPU ↔ MIOP

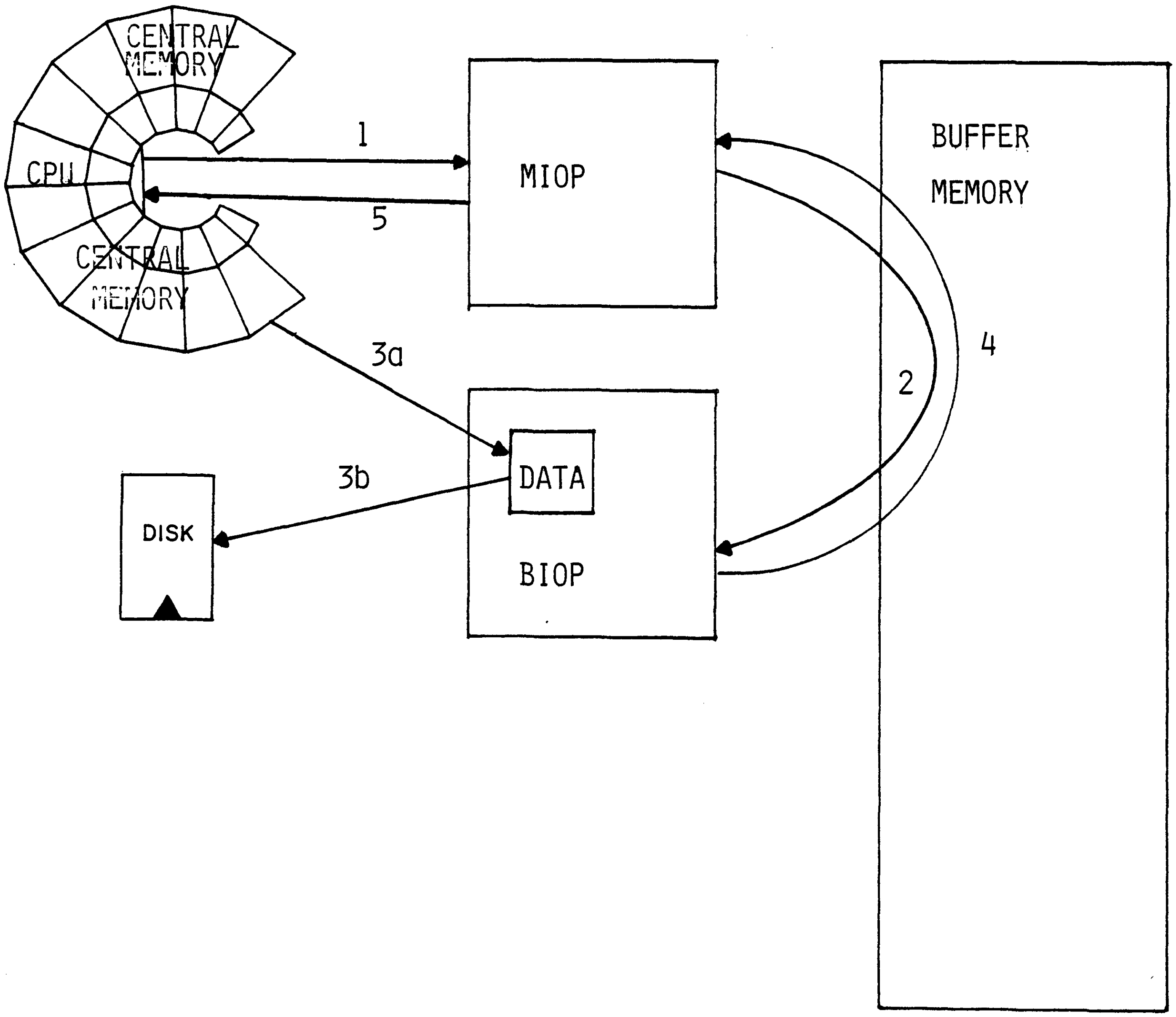


FIGURE 13-3. DISK WRITE VIA BIOP

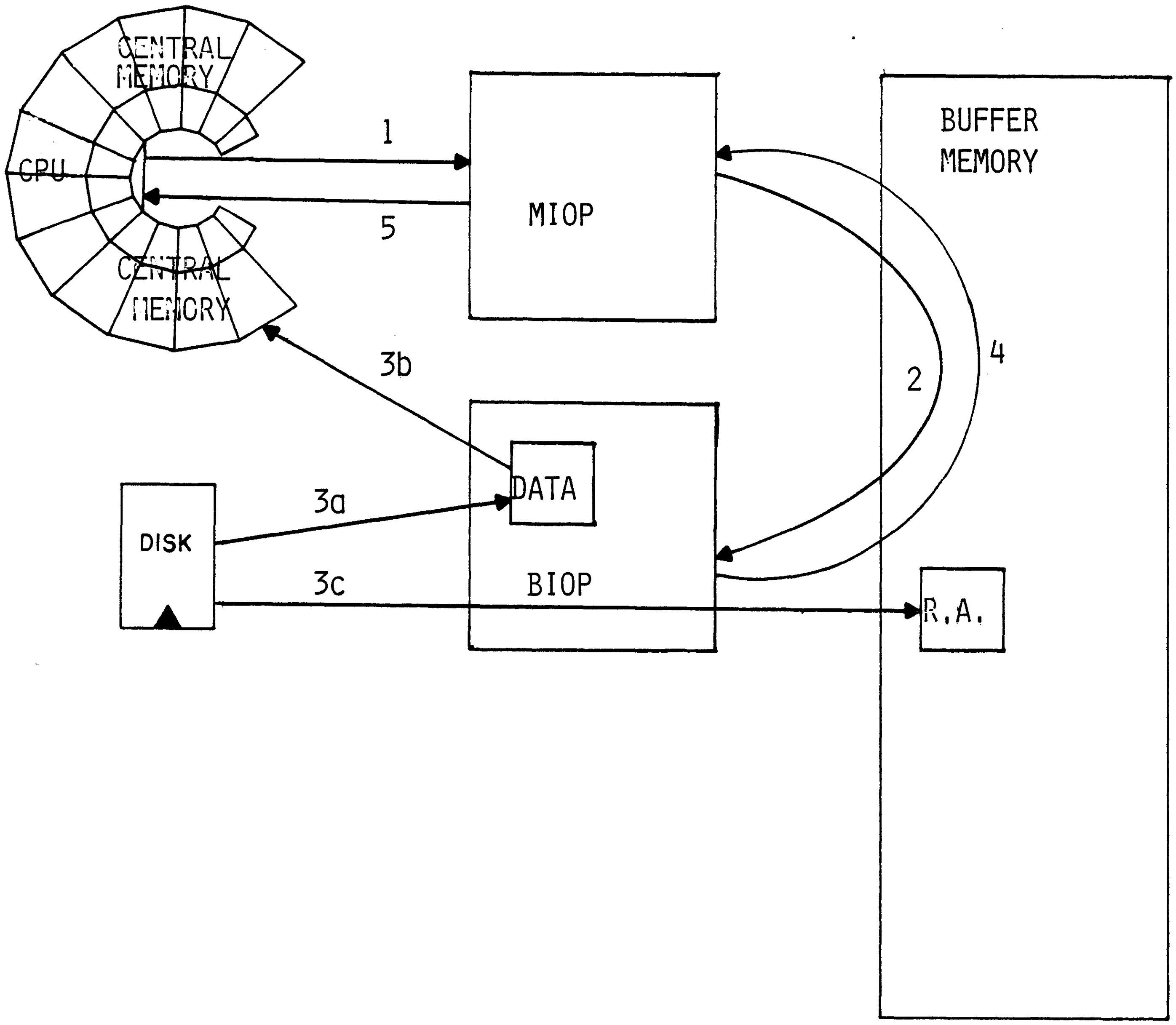


FIGURE 13-4. DISK READ VIA BIOP

MAJOR COMPONENTS

ACOM DEMON ACTIVITY

CDEM DEMON ACTIVITY

DISK DEMON ACTIVITY

DISK INTERRUPT ANSWERING

DISK ERROR RECOVERY ACTIVITY (ERRCK)

DISK DRIVING ROUTINES

ACOM DEMON ACTIVITY

PHYSICAL SECTOR

NORMALLY BUILDS INITIAL EDALS AND PLACES THEM ON EXECUTABLE QUEUE IN DCB.

NORMALLY INITIATES DISK I/O.

ACTIVATES DISK DEMON. (DISCO)

DIOP:

- STARTS I/O ON READS.
- SENDS INITIAL EDALS TO BIOP ON WRITES.
- SENDS FIRST STATUS TO MIOP ON WRITES.

BIOP:

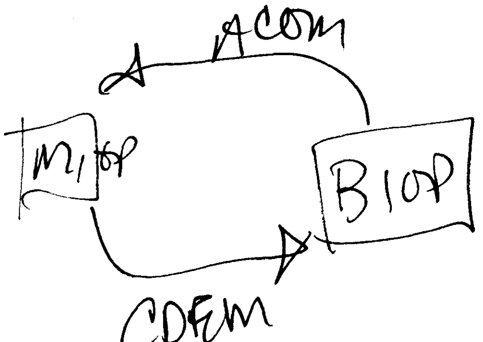
- STARTS I/O ON READS.
- MOVES DATA BETWEEN CENTRAL AND BUFFER MEMORY FOR DIOP.

MIOP: ENTIRE REQ

- SENDS RESPONSE MDAL TO CPU.

EXECUTES OFTEN IN NON-INTERRUPTIBLE MODE.

RESIDES IN BUFFER MEMORY AS ACOM OVERLAY.



DISK DEMON ACTIVITY

NUCLEUS OF THE DISK SUBSYSTEM

EVALUATES REQUESTS PENDING ON A DISK CHANNEL'S DONE QUEUE

BUILDS EDALS AS NEEDED (SO 3 ON EDAL QUEUE)

PRIMARY RESPONSIBILITY IN DIOP IS MOVING DATA BETWEEN LOCAL AND BUFFER MEMORY.

SENDS EDALS TO BIOP

IN BIOP TRANSFERS DISK DATA OVER HIGH SPEED CHANNEL

CREATES DISK ERROR RECOVERY ACTIVITY IF NECESSARY

USUALLY ACTIVATED BY DISK INTERRUPT ANSWERING

INITIALLY ACTIVATED BY ACOM DEMON

EXECUTES OFTEN IN NON-INTERRUPTIBLE MODE

RESIDES IN BUFFER MEMORY AS DISK OVERLAY

SENDS DONE STATUS TO MIOP

17624 @ IDID @ KERNEL

DISK INTERRUPT ANSWERING

ENTERED WHEN A DISK CHANNEL INTERRUPTS

INITIATES NEXT I/O AND SCHEDULES DISK DEMON OR DISK ERROR RECOVERY

EXECUTES IN NON-INTERRUPTIBLE MODE

MOVES FINISHED EDALS TO DONE QUEUE.

ALLOCATES LOCAL BUFFERS ON READS.

DEALLOCATES LOCAL BUFFERS ON WRITE

DISK DRIVING ROUTINES

PERFORM MOST OF THE PHYSICAL I/O

USED BY DISK INTERRUPT ANSWERING, DISK DEMON, ACOM DEMON
AND DISK ERROR RECOVERY.

EXECUTE IN NON-INTERRUPTIBLE MODE

ROUTINES AVAILABLE FOR:

- SELECTING HEAD AND CYLINDER
- SETTING UP DISK BUFFERS
- BUILDING AND SENDING A DAL
- INITIATING DISK I/O

ERRECK DISK ERROR RECOVERY ACTIVITY

PERFORMS A SET, TABLE DRIVEN, ERROR RECOVERY ALGORITHM

CPU IS NOTIFIED OF ANY UNRECOVERABLE ERRORS

SCHEDULED BY DISK INTERRUPT ANSWERING

TERMINATES WHEN RECOVERY ALGORITHM COMPLETED

ONCE IN A WHILE

EXECUTES ~~MOSTLY~~ IN NON-INTERRUPTIBLE MODE

RESIDES IN BUFFER MEMORY AS ERRECK OVERLAY

DISK CHANNEL CONTROL TABLES

DISK CONTROL BLOCK

40_8 PARCELS IN LENGTH + R.A. MODS (64 octal)

ONE FOR EACH DISK CHANNEL

CONTAINS STATUS AND NECESSARY INFORMATION TO DETERMINE A CHANNEL'S STATE.

USED BY DISK DEMON, DISK INTERRUPT ANSWERING, AND DISK ERROR RECOVERY.

KERNEL MAINTAINS A DCB POINTER TABLE.

DISK READ AHEAD MODULE

6 PARCELS IN LENGTH

ONE FOR EACH SECTOR READ AHEAD ON A CHANNEL

LOCATED AFTER DCB

POINTED TO BY DCB

PARCEL	
0	FLAG:0=DATA ON DISK; 1=DATA IN LOCAL; 2=DATA IN MOS
1	CYLINDER (11); HEAD (5)
2	SECTOR (7); UNUSED (9)
3	MOS ADDRESS OF BUFFER (UPPER)
4	MOS ADDRESS OF BUFFER (LOWER)
5	LOCAL BUFFER ADDRESS

FIGURE 13-5. DISK READ AHEAD MODULE

PARCEL	
0	FLAG:0=NOT BUSY NO BUFF, NO DATA: 1=READ; 2=WRITE; 4=R.A.; 10=SEEK; 2N=ERROR REC.; 100=ERRECK; 4N=CHANNEL WAIT; 1000=CREATE ERRECK; 200=ACTIVATE ERRECK
1	CURRENT CYLINDER & HEAD
2	EXECUTABLE DAL QUEUE HEAD
3	EXECUTABLE DAL QUEUE TAIL
4	EXECUTABLE DAL QUEUE POPULATION
5	MASTER DAL QUEUE HEAD
6	MASTER DAL QUEUE TAIL
7	DISK SELECTED BITS (2) DISK TYPE (5), DISK CHANNEL (9)
10	READ AHEAD (R.A.) COUNT CONSTANT (RA$NUM)
11	READ AHEADS DONE FOR CURRENT REQUEST
12	POINTER TO READ AHEAD MODULE TABLE
13	# OF R.A. SECTORS ACTUALLY USED
14	DISK DAL DONE QUEUE HEAD
15	DISK DAL DONE QUEUE TAIL
16	SECTORS OF READ - UPPER
17	SECTORS OF READ - LOWER
20	ERROR COUNT
21	UNRECOVERABLE ERRORS
22	SECTORS OF WRITE - UPPER
23	SECTORS OF WRITE - LOWER
24	# OF TIMES RETRIED
25	FLAG: TYPE OF ERROR
26	SEQUENCE # OF ERROR RECOVERY PROCEEDINGS
27	CYLINDER, HEAD OF ERROR
30	SECTOR, OFFSET AT ERROR
31	ORIGINAL ERROR STATUS FOR THIS RECOVERY ATTEMPT
32	INTERLOCK STATUS SAVE CELL
33	DKA: ~~1/7001 - HEAD GROUP~~ UNUSED
34	PUSH/POP CELL, FIRST (FOR ERRECK)
35	PUSH/POP CELL, LAST (FOR ERRECK)
36	TEMPORARY STATUS CELL
37	LINK FOR DISK QUEUE
40	FIRST ENTRY IN READ AHEAD MODULE

FIGURE 13-6. DISK CONTROL BLOCK

DISK REQUEST CONTROL PACKETS

CPU I/O REQUEST PACKET

30_8 PARCELS IN LENGTH

CONTAINS INFORMATION NECESSARY FOR I/O SUBSYSTEM TO ACCOMPLISH I/O

SENT OVER CPU LOW SPEED CHANNEL TO MIOP
RETURNED WHEN I/O FINISHED

MIOP USES PACKET TO BUILD A DISK ACTIVITY LINK FOR THIS REQUEST.
NORMALLY ONLY ONE PACKET PER DISK CHANNEL IN IOS (EXCEPT IF WRITE BEHIND).

PARCEL	
0	DESTINATION I.D. ("C1")
1	SOURCE I.D. ("A" = DISK; "B" = STATION)
2	UNUSED
3	UNUSED
4	UNUSED
5	UNUSED
6	UNUSED
7	UNUSED
10	COS REQUEST I.D. (PSEUDO CHANNEL)
11	" " " (TASK XP)
12	" " " (BIPOLAR ADDRESS OF DCT)
13	" " " (BIPOLAR ADDRESS OF EQT)
14	CENTRAL MEMORY ADDRESS OF DATA UPPER
15	CENTRAL MEMORY ADDRESS OF DATA LOWER
16	FUNCTION (8 BITS); STATUS (8 BITS)
17	UNUSED (5 BITS); IOP (2 BITS); CHANNEL (9 BITS)
20	CYLINDER (11 BITS); HEAD (5 BITS)
21	SECTOR (7 BITS); ~~OFFSET FOR PARTIAL SECTORS (9 BITS)~~
22	LENGTH IN WORDS UPPER
23	LENGTH IN WORDS LOWER
24	FOR IOS USE
25	FOR IOS USE
26	UNUSED
27	FOR IOS USE

FIGURE 13-7. COS I/O REQUEST PACKET

DISK I/O DISK ACTIVITY LINKS

MASTER DAL:

40_8 PARCELS IN LENGTH

BUILT BY MIOP FROM THE CPU I/O REQUEST PACKET

ONE FOR EACH I/O REQUEST

PASSED TO DISK SUBSYSTEM IN APPROPRIATE IOP

DISK SUBSYSTEM RETURNS MASTER DAL TO MIOP WHEN I/O REQUEST IS COMPLETED.

DISK SUBSYSTEM USES MASTER DAL AS A TEMPLATE FOR BUILDING EXECUTABLE DALS.

EXECUTABLE DAL:

40_8 PARCELS IN LENGTH

BUILT BY DISK SUBSYSTEM FROM A MASTER DAL

ONE FOR EACH SECTOR OF DISK REQUESTED

USED BY DISK SUBSYSTEM TO KEEP TRACK OF WHERE EACH SECTOR OF DATA IS.

USUALLY PASSED BY DIOP TO BIOP FOR HIGH SPEED TRANSFER REQUESTS.
RETURNED WHEN TRANSFER COMPLETE

ALSO REFERRED TO AS SLAVE DAL OR EDAL

PARCEL	
0	LINK TO NEXT MASTER DAL IN CHAIN (0 IF LAST)
1	FUNCTION OF MSG: 1=R/W DISK; 2=RELEASE MOS DAL; 5=STATUS TO COS
2	MOS UPPER OF THIS DAL
3	MOS LOWER OF THIS DAL
4	COUNT OF STATUS SENT TO MIOP
5	ACCUMULATOR SENT VIA EMSGIOP
6	CHANNEL MESSAGE RECEIVED ON
7	UNUSED
10	DESTINATION ID
11	SOURCE ID
12	SEQUENCE # OF LAST E-DAL BUILT
13	READ CONTROL, # SECTORS MOVED TO CPU BY BIOP
14	# OF FULL SECTORS TO MOVE (COMPUTED FROM 32 & 33)
15	UNUSED
16	UNUSED
17	UNUSED
20	CPU REQUEST IDENTITY
21	" " "
22	" " "
23	" " "
24	CENTRAL MEMORY ADDRESS (UPPER)
25	CENTRAL MEMORY ADDRESS (LOWER)
26	FUNCTION (8); STATUS (8)
27	UNUSED (5); IOP (2); CHANNEL (9)
30	CYLINDER (11); HEAD (5)
31	SECTOR (7); OFFSET (9)
32	LENGTH IN WORDS (UPPER)
33	LENGTH IN WORDS (LOWER)
34	IF ERROR, ORIGINAL ERROR STATUS
35	IF ERROR, INTERLOCK STATUS
36	IF ERROR, CYLINDER FROM DK*:1
37	UNUSED

FIGURE 13-8. MASTER DISK ACTIVITY LINK.

PARCEL	
0	LINK TO NEXT EDAL IN CHAIN
1	FUNCTION OF MSG: 3=CENTRAL TO MOS; 4=MOS TO CENTRAL; 6=BIPOLAR TO MOS DN; 7=MOS TO BIPOLAR DN
2	MOS UPPER OF THIS DAL
3	MOS LOWER OF THIS DAL
4	UNUSED
5	ACCUMULATOR SENT VIA EMSGIOP
6	CHANNEL MESSAGE RECEIVED ON
7	UNUSED
10	ADDRESS OF MASTER DAL
11	FLAG: 1 IF EDAL FOR LAST SECTOR
12	SEQUENCE # OF THIS EDAL
13	DATA CONTROL: 0=DATA IN CENTRAL ON DISK; 1=DATA IN LOCAL; 2=DATA IN MOS
14	LOCAL MEMORY ADDRESS OF THIS EDAL FOR BIOP RESPONSE
15	UNUSED
16	UNUSED
17	UNUSED
20	UNUSED
21	UNUSED
22	UNUSED
23	UNUSED
24	CENTRAL MEMORY ADDRESS (UPPER)
25	CENTRAL MEMORY ADDRESS (LOWER)
26	FUNCTION (8); STATUS (8)
27	UNUSED (5); IOP (2); CHANNEL (9)
30	CYLINDER (11); HEAD (5)
31	SECTOR (7); OFFSET (9)
32	UNUSED
33	SIZE OF TRANSFER (DEFAULT 1000_8)
34	MOS BUFFER ADDRESS (UPPER)
35	MOS BUFFER ADDRESS (LOWER)
36	UNUSED
37	LOCAL DISK BUFFER ADDRESS

FIGURE 13-9. EXECUTABLE DISK ACTIVITY LINK.

DISK READ SEQUENCE VIA DIOP

1. CPU TO MIOP: CPU I/O REQUEST PACKET (CDEM)

2. MIOP TO MOS: MASTER DAL TO MIOP MESSAGE AREA (CDEM)

3. MIOP TO DIOP: MOS ADDRESS OF MASTER DAL IN ACCUMULATOR (A=10xxxx)

4. MOS TO DIOP: READ MASTER DAL INTO LOCAL MEMORY (ACOM)
   4A. DIOP BUILDS EDALS AS NEEDED (DISK)
   4B. DIOP SETS UP LOCAL AND MOS DISK BUFFERS (DISK)

5. DISK TO DIOP: DATA INTO LOCAL MEMORY (DISK)

6. DIOP TO MOS: DATA TO MOS (DISK)

7. DIOP TO MOS: EDAL (PARCEL 1=4) TO DIOP MESSAGE AREA (DISK)

8. DIOP TO BIOP: MOS ADDRESS OF EDAL IN ACCUMULATOR (A=12xxxx)
   8A. BIOP SETS UP LOCAL BUFFER (ACOM)

9. MOS TO BIOP: DATA INTO LOCAL MEMORY (ACOM)

10. BIOP TO CPU: DATA INTO CENTRAL MEMORY (ACOM)

11. BIOP TO DIOP: REQUEST COMPLETED;PARCEL 1 OF EDAL IS 7 (ACOM)

12. DIOP TO MIOP: REQUEST COMPLETED;PARCEL 1 OF MDAL IS 5 (DISK)

13. MIOP TO CPU: CPU I/O REQUEST PACKET; PARCEL 16, BYTE 1 IS 0 (ACOM)

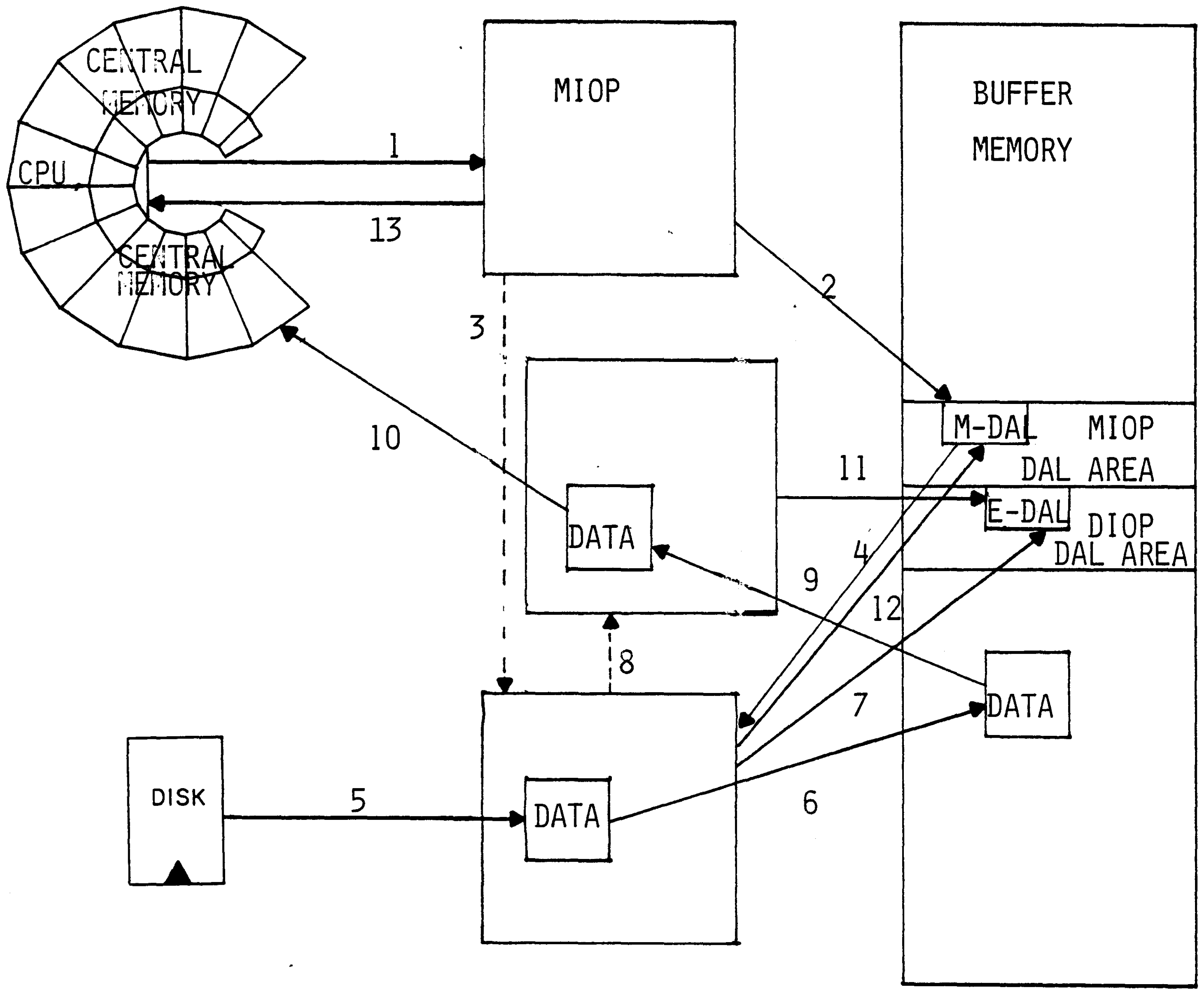


FIGURE 13-10 . DISK READ VIA DIOP

CHAPTER 14

CONCENTRATOR SUBSYSTEM

FUNCTIONS

ALLOWS APPARENT DIRECT COMMUNICATION BETWEEN THE CPU AND A FRONT END.

LOOKS LIKE A CRAY-1 S CHANNEL PAIR TO FRONT END.

THUS NO CHANGES NECESSARY TO EXISTING FRONT-END STATIONS.

MAY REDUCE THE NUMBER OF INTERRUPTS TO THE CPU PER FRONT-END MESSAGE.

<u>CHARACTERISTICS</u>

RESIDES IN BUFFER MEMORY AS OVERLAYS.

EXECUTES MOSTLY IN MIOP WITH HIGH SPEED TRANSFERS TO CPU THROUGH BIOP.

ONE ACTIVE CONCENTRATOR FOR EACH FRONT-END CHANNEL PAIR.

- MAY HAVE SEVERAL LOGICAL ID'S LOGGED ON TO ONE CONCENTRATOR.
  - EACH ID MAY HAVE A DIFFERENT SEGMENT SIZE.

CONTROLLED VIA <u>CONC</u> AND <u>ENDCONC</u> KERNEL CONSOLE COMMANDS.

MAIN COMPONENTS

CONC ACTIVITY:

INITIALIZES CONCENTRATOR RESOURCES

CREATES FEREAD, FEWRIT, CONCO AND CONCI ACTIVITIES

CONCI ACTIVITY:

ACCEPTS MESSAGE FROM A FRONT END VIA FEREAD.

PUTS MESSAGE IN BUFFER MEMORY.

CONCO ACTIVITY:

SENDS A MESSAGE TO A FRONT END VIA FEWRIT.

MESSAGE IS IN BUFFER MEMORY.

FEREAD ACTIVITY:

READS A MESSAGE FROM A FRONT END INTO LOCAL MEMORY.

FEWRIT ACTIVITY:

WRITES A MESSAGE TO A FRONT END FROM LOCAL MEMORY.

CRAYMSG OVERLAY:

GETS CENTRAL MEMORY ADDRESSES FOR MESSAGES VIA CHANNEL EXTENSION TABLE (CXT).

AWAKENS MSGIO ACTIVITY IN BIOP TO MOVE MESSAGES INTO OR OUT OF CENTRAL MEMORY.

CONCENTRATOR ACTIVITY INTERACTION

INTERACTION IS VIA SYNC SERVICE REQUEST CALLS.

A SYNC SERVICES A 2 PARCEL PUSH QUEUE IN LOCAL MEMORY.

IF QUEUE IS EMPTY, A SYNC CALL RESULTS IN A PUSH ON TO THE QUEUE.

IF FULL, QUEUED ACTIVITY IS POPPED OFF THE QUEUE AND PLACED ON THE CP QUEUE. THE SYNCING ACTIVITY REGAINS CONTROL.

"SYNC"ING ACTIVITIES MAY PASS ONE PARCEL MESSAGES THROUGH PARCELS 17 AND 13 OF THEIR ACTIVITY DESCRIPTORS.

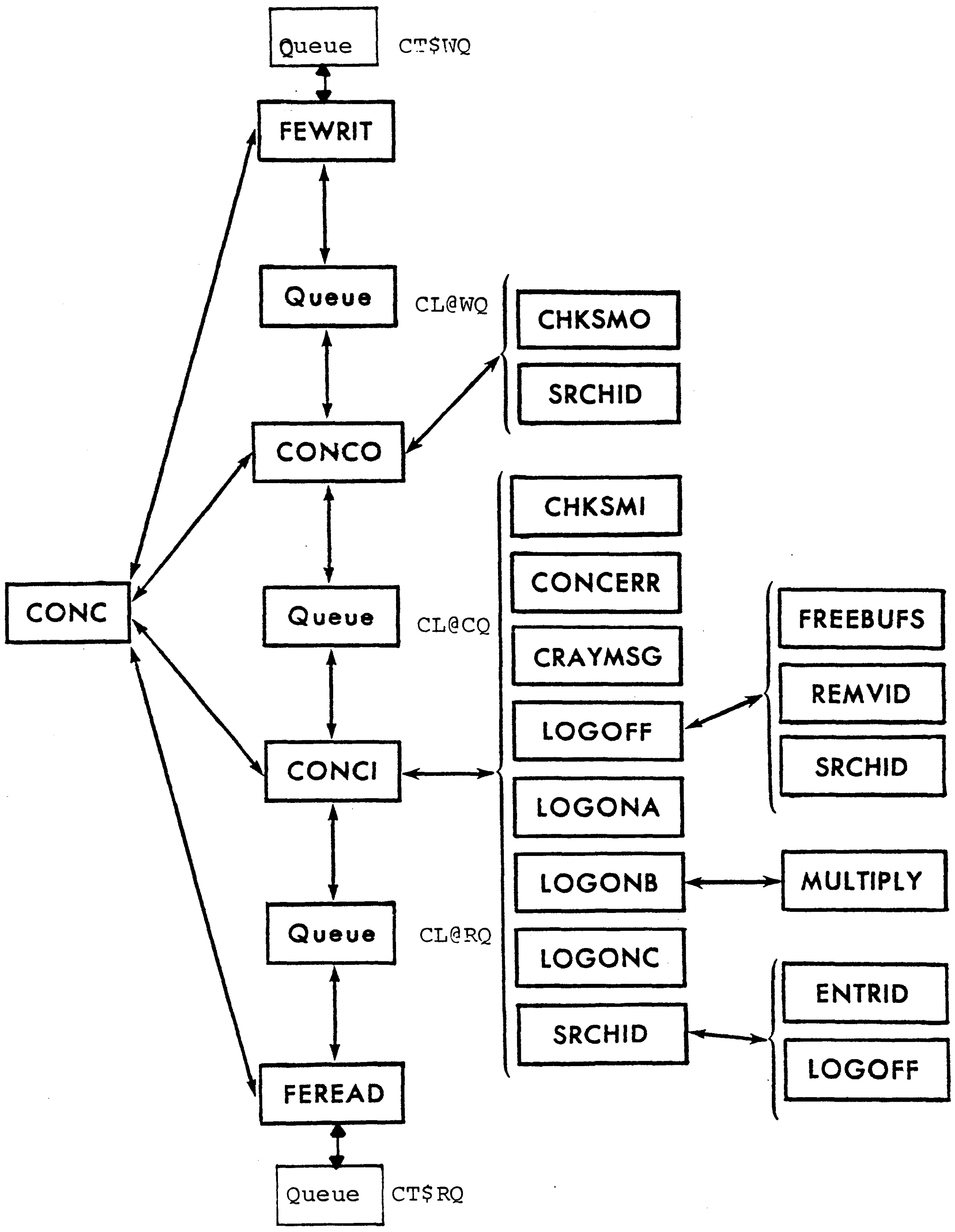


Figure 14-1. Tree structure of Concentrator software

OVERVIEW OF FRONT-END MESSAGE FLOW

1. FRONT END SENDS MESSAGE CONSISTING OF LCP AND POSSIBLY SUB-SEGMENTS AND LTP TO MIOP. (FEREAD)

2. MIOP WRITES THE MESSAGE TO BUFFER MEMORY. (CONCI)

3. MIOP GETS CENTRAL MEMORY ADDRESSES FROM CPU FOR INPUT MESSAGE. (CRAYMSG)

4. MIOP SENDS ADDRESS INFORMATION TO BIOP VIA BUFFER MEMORY. (CRAYMSG)

5. BIOP READS THE MESSAGE INTO LOCAL MEMORY. (MSGIO $\rightarrow$ MSGIN)

6. BIOP WRITES THE MESSAGE TO CENTRAL MEMORY. (MSGIN)

7. BIOP TELLS MIOP IT IS DONE VIA BUFFER MEMORY. (MSGIO)

8. MIOP TELLS CPU MESSAGE IS IN CENTRAL MEMORY. (CRAYMSG)
   8A. CPU PROCESSES MESSAGE AND BUILDS A RESPONSE. (SCP)

9. MIOP RECEIVES RESPONSE MESSAGE CENTRAL MEMORY ADDRESSES FROM CPU. (CRAYMSG)

10. MIOP SENDS ADDRESS INFORMATION TO BIOP VIA BUFFER MEMORY. (CRAYMSG)

11. BIOP READS CPU RESPONSE MESSAGE INTO LOCAL MEMORY. (MSGIO $\rightarrow$ MSGOUT)

12. BIOP WRITES CPU RESPONSE MESSAGE TO BUFFER MEMORY. (MSGOUT)

13. MIOP READS RESPONSE MESSAGE INTO LOCAL MEMORY. (CONCO)

14. MIOP SENDS RESPONSE MESSAGE TO FRONT END. (FEWRIT)

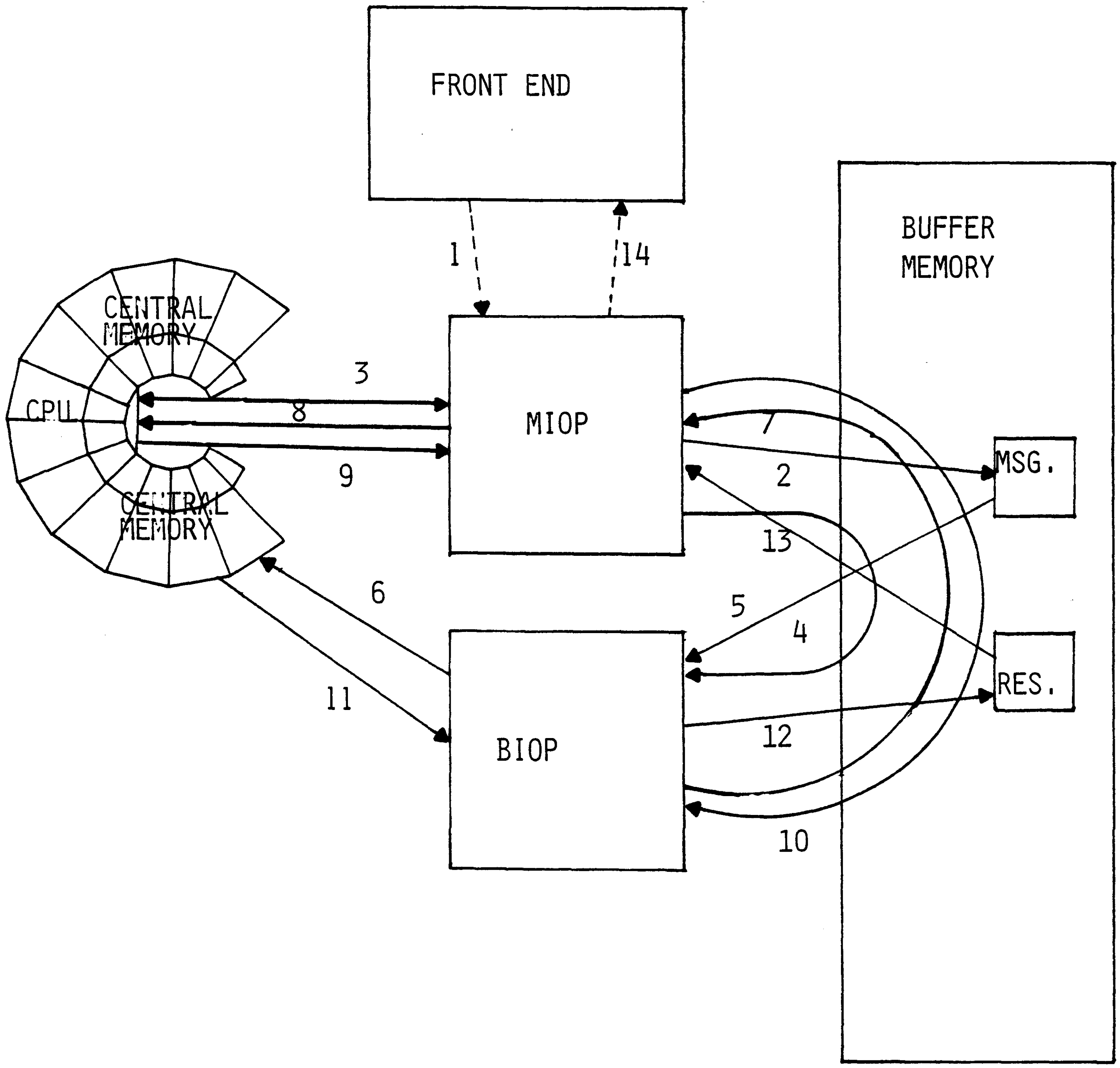


FIGURE 14-2. FRONT-END MESSAGE FLOW

CHAPTER 15

STATION SUBSYSTEM

FUNCTIONS

PROVIDES A MEANS FOR OPERATOR-CPU COMMUNICATION.
CONTROLS OPERATOR CONSOLES

MAY BE USED AS A BATCH JOB ENTRY STATION.
JOBS OR DATASETS STAGED FROM TAPE

MAY ACCEPT CPU OUTPUT AND DISTRIBUTE IT TO MAG TAPE OR PRINTER

ALLOWS ON-LINE DEBUGGING OF CPU

OVERLAYS
COMMØ - COMMØ3

CHARACTERISTICS

RESIDES IN BUFFER MEMORY AS OVERLAYS.

EXECUTES MOSTLY IN MIOP WITH SOME HIGH SPEED TRANSFERS THROUGH BIOP.

MAY HAVE MORE THAN ONE STATION ACTIVE AT A TIME.

EACH STATION MUST HAVE A DEDICATED CONSOLE.
THEY MUST SHARE THE EXPANDER PERIPHERALS.
TWO OR MORE CONSOLES MAY BE SUPPORTED BY
ONE STATION.

COMMUNICATES WITH CPU IN STANDARD CRAY MESSAGE FORMAT.

APPEARS TO BE JUST ANOTHER FRONT-END STATION TO CPU.

COMMUNICATION PROTOCOL

A MESSAGE IS A VARIABLE SIZE SET OF TRANSMISSIONS BETWEEN A STATION AND THE CPU.

- IT IS ALWAYS HEADED BY A LINK CONTROL PACKAGE (LCP)
- MAY CONTAIN ONE OR MORE ADDITIONAL TRANSMISSIONS TERMED SUB-SEGMENTS.
  - A GROUP OF SUB-SEGMENTS ASSOCIATED WITH ONE LCP IS TERMED A SEGMENT.
    - MAXIMUM SEGMENT SIZE IS DETERMINED BY CPU START UP PARAMETER.
  - SUB-SEGMENT SIZE IS DETERMINED BY STATION.
- MAY CONTAIN A LINK TRAILER PACKAGE (LTP) FOR VALIDATING THE MESSAGE.

DATASETS ARE TRANSFERRED IN ONE OR MORE MESSAGES.

- ALL OF THE MESSAGES RELATED TO A SINGLE DATASET IS TERMED A STREAM.
- STREAMS ARE MAINTAINED THROUGH STREAM CONTROL BYTES PRESENT IN THE LCP.
- THE MAXIMUM NUMBER OF STREAMS (UP TO 8 INPUT AND 8 OUTPUT) IS DETERMINED BY THE STATION.

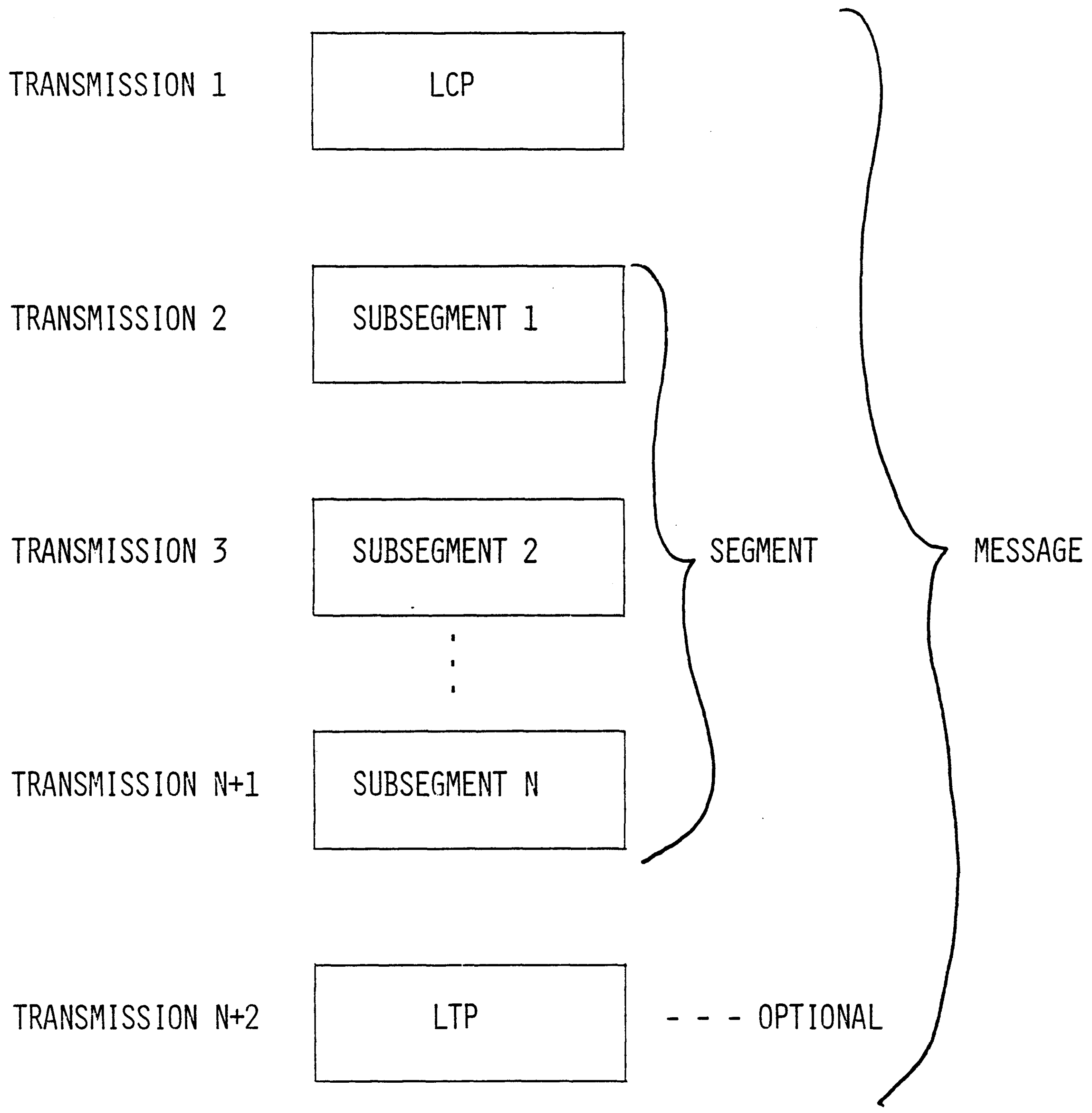


FIGURE 15-1. CRAY MESSAGE FORMAT

LINK CONTROL PACKAGE

ALWAYS CONSISTS OF SIX 64 BIT WORDS (24 PARCELS)

CONTAINS INFORMATION NECESSARY TO PROCESS ITS' ASSOCIATED SEGMENT.

ALSO PROVIDES INFORMATION CONCERNING ALL STREAMS.

PARCEL		
0	DESTINATION ID ('C1' OR STATION LOGON ID)	
1	SOURCE ID ('C1' OR STATION LOGON ID)	
2	NUMBER OF SUBSEGMENTS (NSSG); MESSAGE NUMBER (MN)	
3	MESSAGE CODE (MC); MESSAGE SUB-CODE (MSC)	
4	STREAM NUMBER (STN); SEGMENT NUMBER (SGN) (UPPER)	
5	SEGMENT NUMBER (LOWER)	
6	SEGMENT BIT COUNT (SGBC) (UPPER)	
7	SEGMENT BIT COUNT (SGBC) (LOWER)	
10	UNUSED	
11	UNUSED	
12	UNUSED	
13	UNUSED	
14	INPUT STREAM CONTROL BYTE 1 ($ISCB_1$)	; $ISCB_2$
15	$ISCB_3$	; $ISCB_4$
16	$ISCB_5$	; $ISCB_6$
17	$ISCB_7$	; $ISCB_8$
20	$OSCB_1$	; $OSCB_2$
21	$OSCB_3$	; $OSCB_4$
22	$OSCB_5$	; $OSCB_6$
23	$OSCB_7$	; $OSCB_8$
24	UNUSED	
25	UNUSED	
26	UNUSED	
27	UNUSED	

FIGURE 15-2. LINK CONTROL PACKAGE

STREAM CONTROL BYTES

PROVIDE A MEANS OF PASSING STREAM STATUS INFORMATION.

USED BY BOTH THE STATION AND THE CPU.

OCTAL CODE	ACRONYM	MEANING	SENDER	RECEIVER
00	IDL	IDLE	X	X
01	RTS	REQUEST TO SEND	X	
02	PTR	PREPARING TO RECEIVE		X
03	SND	SENDING	X	
04	RCV	RECEIVING		X
05	SUS	SUSPEND		X
06	END	END OF DATASET	X	
07	SVG	SAVING DATASET		X
10	SVD	DATASET SAVED		X
11	PPN	POSTPONE		X
12	CAN	CANCEL	X	X
13	MCL	MASTER CLEAR	X	X

TABLE 15-1. STREAM CONTROL BYTES.

MAIN COMPONENTS

STATION OVERLAY:

INITIALIZES A STATION WHEN 'STATION' IS TYPED IN AT THE MIOP KERNEL CONSOLE.

ALL OTHER STATION COMMANDS TYPED IN A A STATION CONSOLE.

INITIATES ONE SET OF STATION CONSOLE HANDLING ACTIVITIES:
KEYBD, CLI, AND DISPLAY

KEYBD ACTIVITY:

RECEIVES CHARACTERS ENTERED AT THE STATION CONSOLE KEYBOARD.

ONE KEYBD ACTIVITY FOR EACH ACTIVE STATION.

CALLS THE CONSL OVERLAY TO ECHO THE CHARACTERS.

ACTIVATES THE CLI ACTIVITY TO PROCESS COMMANDS.

CLI ACTIVITY:

INTERPRETS AND EXECUTES THE OPERATOR COMMANDS.

ONE CLI ACTIVITY FOR EACH ACTIVE STATION.

GETS COMMANDS FROM A CIRCULAR BUFFER FILLED BY THE KEYBD ACTIVITY, VALIDATES THEM, AND CALLS APPROPRIATE OVERLAY (COMMO-13) TO PROCESS THEM.

DISPLAY ACTIVITY:

FORMATS THE OPERATOR DISPLAY.

ONE DISPLAY ACTIVITY FOR EACH ACTIVE STATION.

RESPONDS TO REQUESTS FROM CLI AND CALLS APPROPRIATE DISPLAY OVERLAY (DISP01, DISP02).

PROTOCOL ACTIVITY:

MANAGES STATION-CPU COMMUNICATIONS FOR ALL ACTIVE STATIONS.

INITIATED BY LOGON COMMAND.

TERMINATED BY LOGOFF OR COMMUNICATION BREAKDOWN.

RESPONSIBLE FOR:
- GENERATING MESSAGES SENT TO CPU.
- VALIDATING CPU RESPONSES.
- MAINTAINING STREAM STATES.
- CREATING ACTIVITIES TO MANAGE DATASET TRANSFERS.
- SCHEDULING MESSAGES TO CPU.
- DISTRIBUTING CPU RESPONSES.

STAGEIN ACTIVITY:

STAGES A DATASET FROM THE I/O SUBSYSTEM TO CPU.

CREATED BY PROTOCOL ACTIVITY.

REQUEST ORIGINATES FROM A SAVE OR SUBMIT COMMAND FROM OPERATOR, OR AN ACQUIRE MESSAGE FROM A JOB IN CPU.

ONE FOR EACH ACTIVE INPUT STAGING OPERATION..

STAGEOUT ACTIVITY:

STAGES A DATASET FROM CPU TO THE I/O SUBSYSTEM.

CREATED BY THE PROTOCOL ACTIVITY WHEN CPU INTIATES STAGING ON AN OUTPUT STREAM.

ONE FOR EACH ACTIVE STAGING OPERATION.

STATION ACTIVITY INTERACTION

ACTIVITIES PASS PARAMETERS VIA SHARED LOCAL MEMORY AREAS.

POINTERS TO THESE AREAS (TABLES) ARE MAINTAINED IN GLOBAL REGISTERS.

THUS AN ACTIVITY MAY MODIFY AN ENTRY IN ONE OF THESE TABLES AND PASS CONTROL TO ANOTHER ACTIVITY WITHOUT PASSING THE TABLE ADDRESS AS A PARAMETER.

ACTIVITIES INTERACT VIA THE PUSH, POP, AND TPUSH SERVICE REQUESTS.

ACTIVITIES MAY PASS PARAMETERS AND INTERACT VIA THESE SERVICE REQUESTS BY USING THE SIGNAL AND WATCH MACROS.

THESE SERVICE A 3 PARCEL AREA, 2 OF WHICH ARE USED AS A QUEUE, AND THE THIRD FOR PASSING CODED MESSAGES TO ANOTHER ACTIVITY.

OVERVIEW OF STATION-CPU MESSAGE FLOW

1. OPERATOR INITIATES STAGING OR AN OPERATOR COMMAND.

   1A. MIOP BUILDS A MESSAGE CONSISTING OF AN LCP AND POSSIBLY SUBSEGMENTS.

2. MIOP WRITES THE MESSAGE TO BUFFER MEMORY.

3. MIOP GETS CENTRAL MEMORY ADDRESSES FROM CPU FOR INPUT MESSAGE.

4. MIOP SENDS ADDRESS INFORMATION TO BIOP VIA BUFFER MEMORY.

5. BIOP READS THE MESSAGE INTO LOCAL MEMORY.

6. BIOP WRITES THE MESSAGE TO CENTRAL MEMORY.

7. BIOP TELLS MIOP IT IS DONE VIA BUFFER MEMORY.

8. MIOP TELLS CPU MESSAGE IS IN CENTRAL MEMORY.

   8A. CPU PROCESSES MESSAGE AND BUILDS A RESPONSE.

9. MIOP RECEIVES RESPONSE MESSAGE CENTRAL MEMORY ADDRESSES FROM CPU.

10. MIOP SENDS ADDRESS INFORMATION TO BIOP VIA BUFFER MEMORY.

11. BIOP READS CPU RESPONSE MESSAGE INTO LOCAL MEMORY.

12. BIOP WRITES CPU RESPONSE MESSAGE TO BUFFER MEMORY.

13. MIOP READS RESPONSE MESSAGE INTO LOCAL MEMORY.

14. IF APPROPRIATE, MIOP SENDS RESPONSE TO DISPLAY.

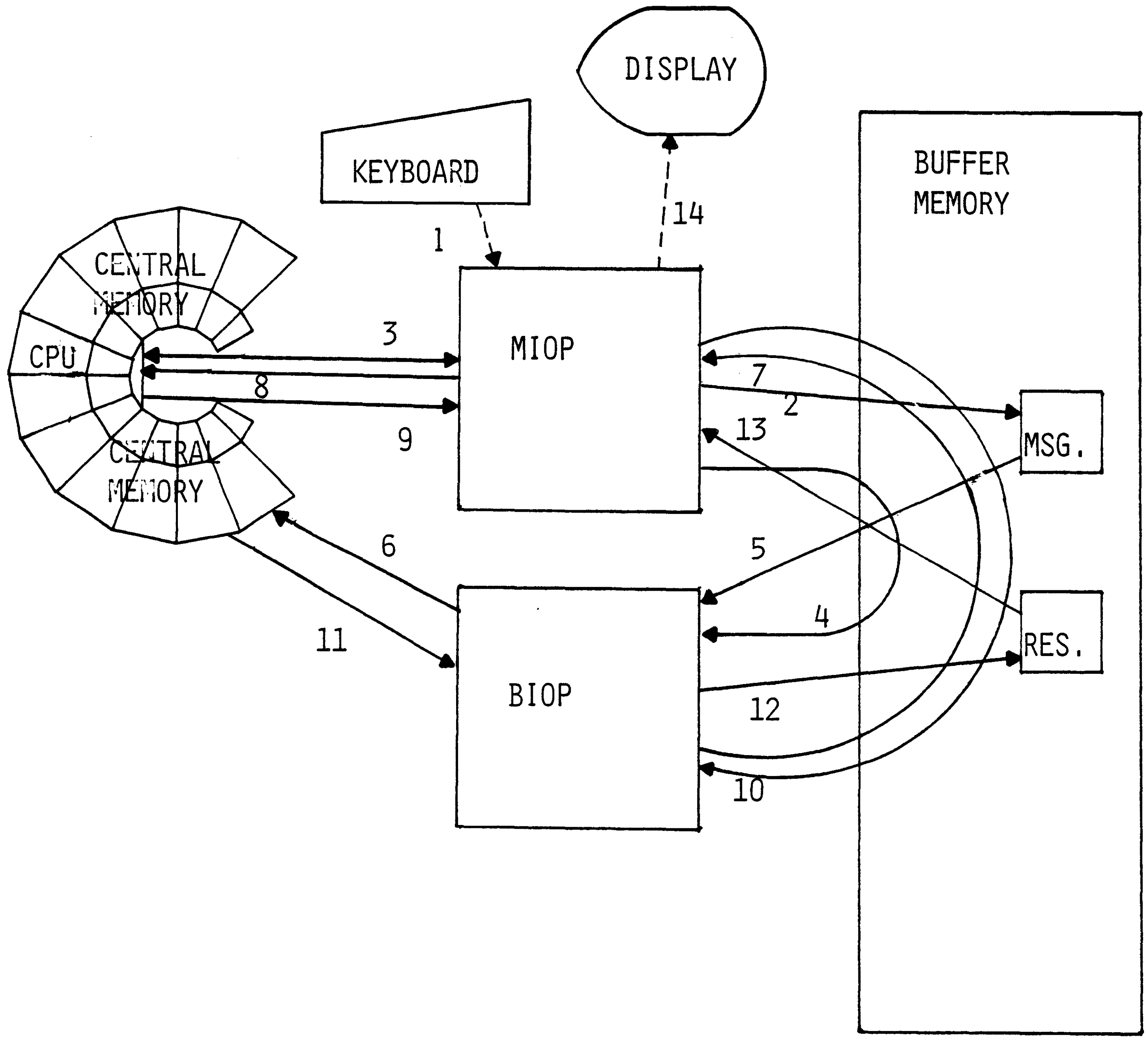


FIGURE 15-3 . STATION-CPU MESSAGE FLOW

CHAPTER 16

INTERACTIVE STATION SUBSYSTEM

FUNCTIONS

ALLOWS OPERATOR TO RUN JOBS IN THE CPU IN AN INTERACTIVE FASHION.

CHARACTERISTICS

RESIDES IN BUFFER MEMORY AS OVERLAYS

EXECUTES MOSTLY IN MIOP WITH HIGH SPEED TRANSFERS TO CPU THROUGH BIOP.

MAY SUPPORT SEVERAL CONSOLES

CONSISTS OF TWO PARTS:

INTERACTIVE CONCENTRATOR

INTERACTIVE CONSOLE

INTERACTIVE CONCENTRATOR COMPONENTS

IAIOP ACTIVITY:

INITIALIZES THE INTERACTIVE CONCENTRATOR AND ACCEPTS COMMANDS FOR IT.

CURRENT COMMANDS ARE LOG, LOGOFF, POLL AND END.

CREATES THE IAIOP1 ACTIVITY

IAIOP1 ACTIVITY:

MAIN CONTROL OF INTERACTIVE CONCENTRATOR

IAIOP1 OVERLAY CALLS IAFUNC, IAMSG AND CRAYMSG OVERLAYS.

IAFUNC OVERLAY:

PROCESSES INTERACTIVE CONCENTRATOR COMMANDS.

IAMSG OVERLAY:

DISTRIBUTES RESPONSES TO INTERACTIVE CONSOLES.

CRAYMSG OVERLAY:

SENDS MESSAGES TO THE CPU

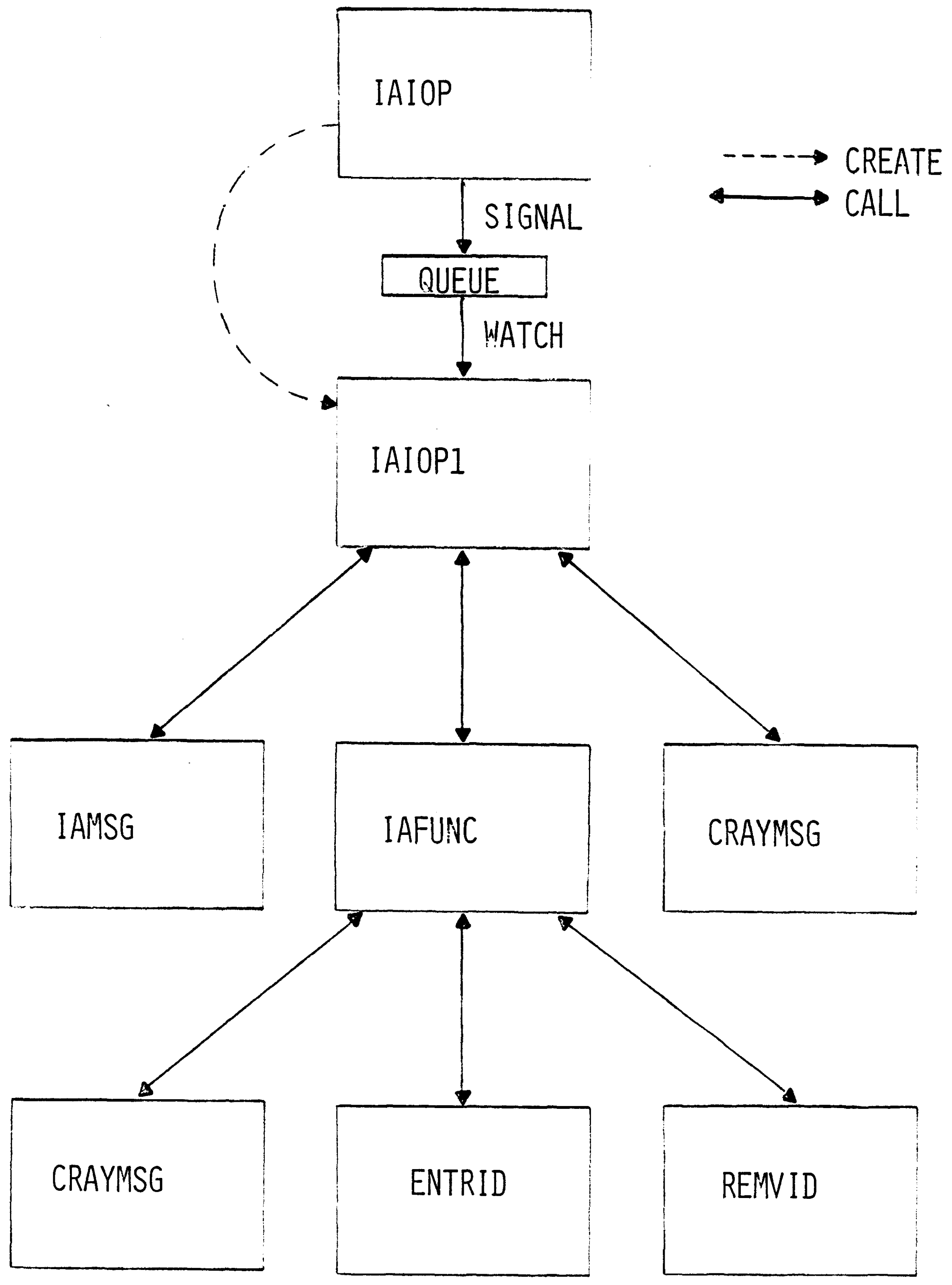


FIGURE 16-1. TREE STRUCTURE OF INTERATIVE CONCENTRATOR SOFTWARE

INTERACTIVE CONSOLE COMPONENTS

IACON ACTIVITY:

INITIALIZES THE INTERACTIVE CONSOLE

CREATES THE KEYBD ACTIVITY FOR INPUT AND THE IAOUT ACTIVITY TO UPDATE THE SCREEN.

PASSES CONTROL TO THE IACON1 OVERLAY.

IACON1 ACTIVITY:

MAIN CONTROL ACTIVITY FOR THE INTERACTIVE CONSOLE.

ONE PER INTERACTIVE CONSOLE

PROCESSES INPUT FROM THE KEYBOARD BUFFER AND NOTIFIES THE INTERACTIVE CONCENTRATOR.

IACMD OVERLAY:

PROCESSES COMMANDS TO THE INTERACTIVE CONSOLE.

COMMANDS ARE PRECEDED BY A COMMAND CONTROL CHARACTER (/)

CURRENTLY SUPPORTED COMMANDS ARE:

ABORT
ATTENTION
BYE
CHANGE
COMMENT
EOF
LOGOFF
LOGON

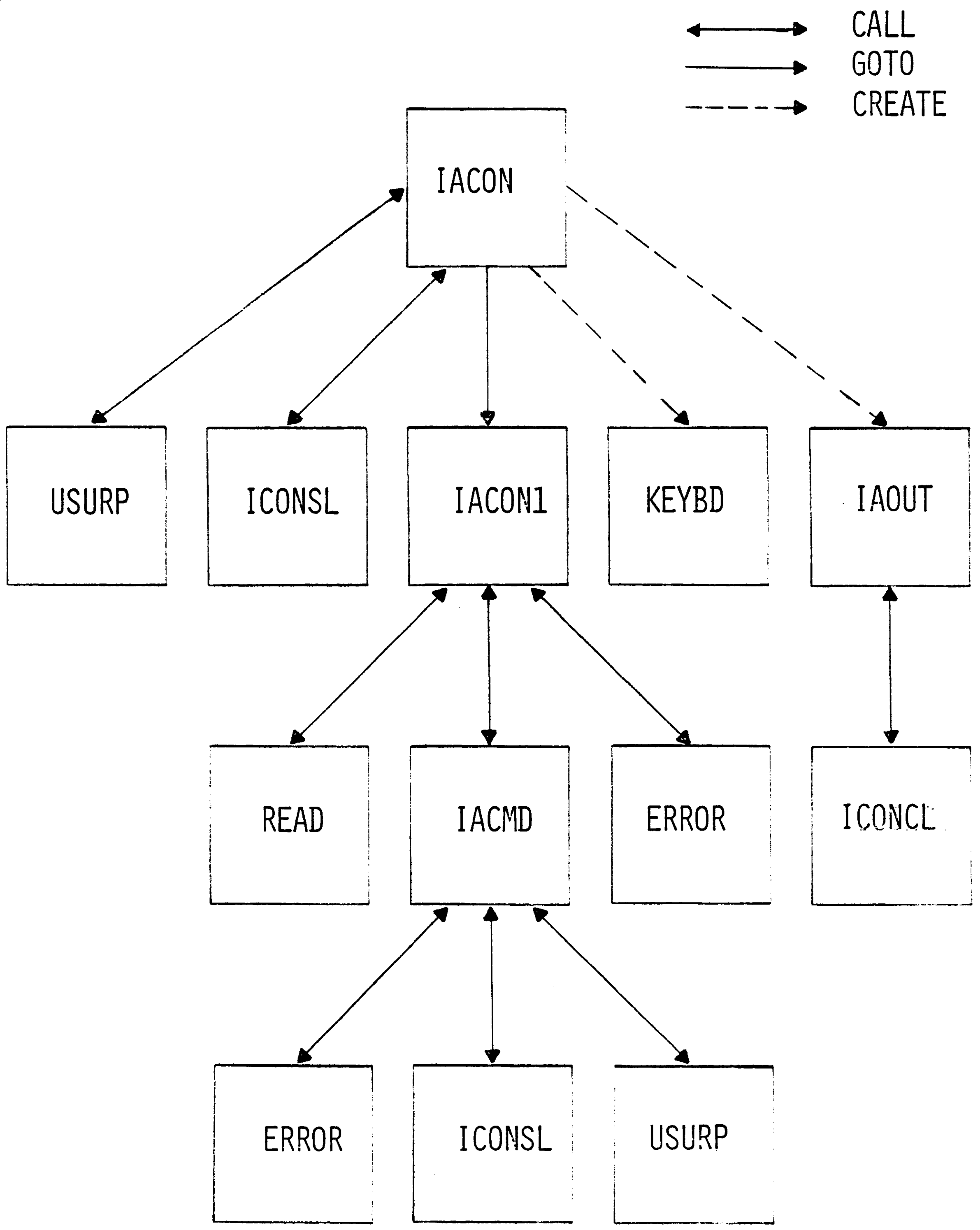


FIGURE 16-2. TREE STRUCTURE OF INTERACTIVE CONSOLE SOFTWARE

CHAPTER 17

DEADSTART

OVERVIEW

IOS IS INITIALLY DEADSTARTED FROM TAPE.

SUBSEQUENT RESTARTS MAY BE FROM DISK.

THE CPU MAY BE DEADSTARTED FROM TAPE OR DISK.

SYS.IN in MIOP }
SYSS in others } for init

CONFIG DATA in AMAP only

I/O SUBSYSTEM DEADSTART

MIOP IS INITIALLY DEADSTARTED FROM TAPE THROUGH THE EXPANDER CHANNEL.

MIOP INITIALIZES THE BUFFER MEMORY CONFIGURATION AND WRITES A COPY OF THE KERNEL TO BUFFER MEMORY.

MIOP THEN DEADSTARTS THE OTHER IOPS IN THE CONFIGURATION WHICH CAUSES THE KERNEL TO BE READ IN FROM BUFFER MEMORY.

THESE IOPS ARE THEN INITIALIZED BY SYSS OVERLAY.

THE AMAP OVERLAY IS REFERENCED AT DEADSTART BY ALL IOPS FOR CONFIGURATION INFORMATION.

AMAP OVERLAY

THE AMAP OVERLAY IS USED TO PROVIDE CONFIGURATION INFORMATION FOR IOS INITIALIZATION.

CHANGES TO AMAP ARE MADE USING THE UPDATE UTILITY.

THERE ARE THREE TYPES OF TABLES IN AMAP:

- INITIAL AMAP TABLE
- IOP INFORMATION TABLE
- CHANNEL CONFIGURATION TABLE

THE INITIAL AMAP TABLE IS 12_8 PARCELS IN LENGTH AND IS USED TO PROVIDE SOME BUFFER MEMORY INFORMATION AND TO POINT TO THE IOP INFORMATION TABLES.

PARCEL (IN AMAP)	DESCRIPTION
* 5	BUFFER MEMORY SIZE IN 131K WORD UNITS
6	NUMBER OF I/O PROCESSORS IN SUBSYSTEM
7	MIOP BUFFER MEMORY MESSAGE AREA SIZE IN WORDS
10	BIOP BUFFER MEMORY MESSAGE AREA SIZE IN WORDS
11	IOP-2 BUFFER MEMORY MESSAGE AREA SIZE IN WORDS
12	IOP-3 BUFFER MEMORY MESSAGE AREA SIZE IN WORDS
13	POINTER TO MIOP INFORMATION TABLE
14	POINTER TO BIOP INFORMATION TABLE
15	POINTER TO IOP-2 INFORMATION TABLE (0 IF NOT CONFIGURED)
16	POINTER TO IOP-3 INFORMATION TABLE (0 IF NOT CONFIGURED)

TABLE 17-1. INITIAL AMAP TABLE

* SINCE THE OVERLAY MACRO GENERATES A 6-PARCEL HEADER, THIS PARAMETER MUST BE SPECIFIED AS THE FP PARAMETER ON THE OVERLAY MACRO, AND NP MUST BE 0. FOR EXAMPLE:

LOCATION	RESULT	OPERAND	COMMENT
	OVERLAY	AMAP,NP=0,FP=4	.HALF MILLION WORDS

THE IOP INFORMATION TABLE IS 7 PARCELS PER I/O PROCESSOR AND IS USED TO PROVIDE LOCAL AND BUFFER MEMORY ALLOCATION INFORMATION AND TO POINT TO THE CHANNEL CONFIGURATION TABLE.

OFFSET	DESCRIPTION
0	BUFFER MEMORY ALLOCATED TO THIS IOP IN 1K WORD UNITS
1	NUMBER OF 512-WORD BUFFERS TO RESERVE IN LOCAL MEMORY
2	NUMBER OF SOFTWARE STACK AREAS TO ALLOCATE IN BUFFER MEMORY.
3	SIZE OF OVERLAY MEMORY IN LOCAL MEMORY.
4	NUMBER OF MESSAGE PACKETS (DALS) TO RESERVE IN LOCAL MEMORY.
5	LOCAL MEMORY SIZE IN 65K PARCEL UNITS; THIS IS ALWAYS 1.
6	POINTER TO CHANNEL CONFIGURATION TABLE FOR THIS IOP.

TABLE 17-2. IOP INFORMATION TABLE

THE CHANNEL CONFIGURATION TABLE IDENTIFIES DEVICES ATTACHED TO CHANNELS 6 TO 47_8 OF AN I/O PROCESSOR. THE STATUS OF THE CHANNEL IS ALSO INDICATED.

ENTRIES IN THIS TABLE ARE DEFINED VIA THE CHANNEL MACRO.
THERE ARE TWO TYPES OF TABLE ENTRIES:

```
        CHANNEL       NUM
```

NUM - HIGHEST CHANNEL NUMBER DESCRIBED BY TABLE.

```
    CHANNEL    (CHANNEL), ST=STATUS,TY=TYPE
```

CHANNEL - CHANNEL NUMBER(S)

STATUS - CHANNEL STATUS:

UP - DEFAULT
DOWN - CHANNEL NOT TO BE USED

TYPE - TYPE OF CHANNEL OR DEVICE ON CHANNEL:

A0	MIOP ACCUMULATOR CHANNEL
A1	BIOP ACCUMULATOR CHANNEL
A2	IOP-2 ACCUMULATOR CHANNEL
A3	IOP-3 ACCUMULATOR CHANNEL
BM	BLOCK MULTIPLEXER CHANNEL
CH	CPU HIGH-SPEED CHANNEL
CL	CPU LOW-SPEED CHANNEL
C0	TEC 455 DISPLAY
C1	TEC 1440 DISPLAY
CS	AMPEX DISPLAY
D1	DD-19 DISK DRIVE
D2	DD-29 DISK DRIVE
EX	EXPANDER CHANNEL
ST	FRONT-END CHANNEL
EM	UNUSED
(EM,N)	N UNUSED CHANNEL ENTRIES

IOS TAPE DEADSTART

THE CONVENTIONAL IOS TAPE LAYOUT IS:

FILE 0 - TAPELOAD must BE
FILE 1 - DMP
FILE 2 - KERNEL
FILE 3 - OVERLAYS must Follow Kernel

PROCEDURE:

1. MOUNT THE IOS DEADSTART TAPE ON THE IOS TAPE UNIT
2. PUSH MASTER CLEAR AND DEADSTART BUTTONS AT THE POWER UNIT.
3. TYPE "2" IN RESPONSE TO THE TAPELOAD "FROM MTO:" MESSAGE AT THE MIOP KERNEL CONSOLE.
4. IF THE KERNEL WAS ASSEMBLED WITH THE ON-LINE DEBUGGER, TYPE "X" WHEN THE ! PROMPT CHARACTER APPEARS. CTRL D
5. WHEN DEADSTART IS COMPLETE, A SYSTEM MESSAGE WILL BE POSTED AT EACH KERNEL CONSOLE.
6. DEADSTART THE CPU, IF APPROPRIATE.

MIOP INITIAL DEADSTART SEQUENCE

1) OPERATOR PUSHES MASTER CLEAR BUTTON

   THIS CAUSES EXIT STACK LOCATION ZERO TO BE SET TO ZERO.

   CLEARS CHANNELS' DN AND BZ FLAGS

2) OPERATOR PUSHES DEADSTART BUTTON

   THIS CAUSES FIRST BLOCK OF TAPE TO BE LOADED INTO LOW MEMORY.

   INTERRUPT OCCURS WHEN DONE.

   HARDWARE BEGINS EXECUTION AT ADDRESS IN EXIT STACK LOCATION ZERO, WHICH IS ZERO.

3) TAPELOAD ROUTINE (LOCATED IN FIRST BLOCK) LOADS REST OF KERNEL FROM TAPE.

4) DISABLE INTERRUPTS ON CHANNELS 3 TO 47

5) PERFORM LOCAL MEMORY DIAGNOSTICS

6) JUMP TO SYSTEM INITIALIZATION ROUTINE

BIOP, DIOP, & XIOP DEADSTART SEQUENCE

1) MIOP ISSUES A0*:1 COMMAND WITH MASTER CLEAR AND DEADSTART BITS SET IN A.

2) MIOP ISSUES A0*:1 COMMAND WITH MASTER CLEAR AND DEADSTART BITS CLEARED.

   THIS INITIATES TRANSFER OF LOWER 65K PARCELS OF BUFFER MEMORY

   WHEN TRANSFER COMPLETES, AN INTERRUPT IS GENERATED AND EXECUTION BEGINS AT LOCATION 0.

3) DISABLE INTERRUPTS ON CHANNELS 3 TO 47.

4) PERFORM LOCAL MEMORY DIAGNOSTICS.

@ 8777

5) LOAD SYSS OVERLAY AND BEGIN RECONFIGURATION OF KERNEL FOR THIS IOP.

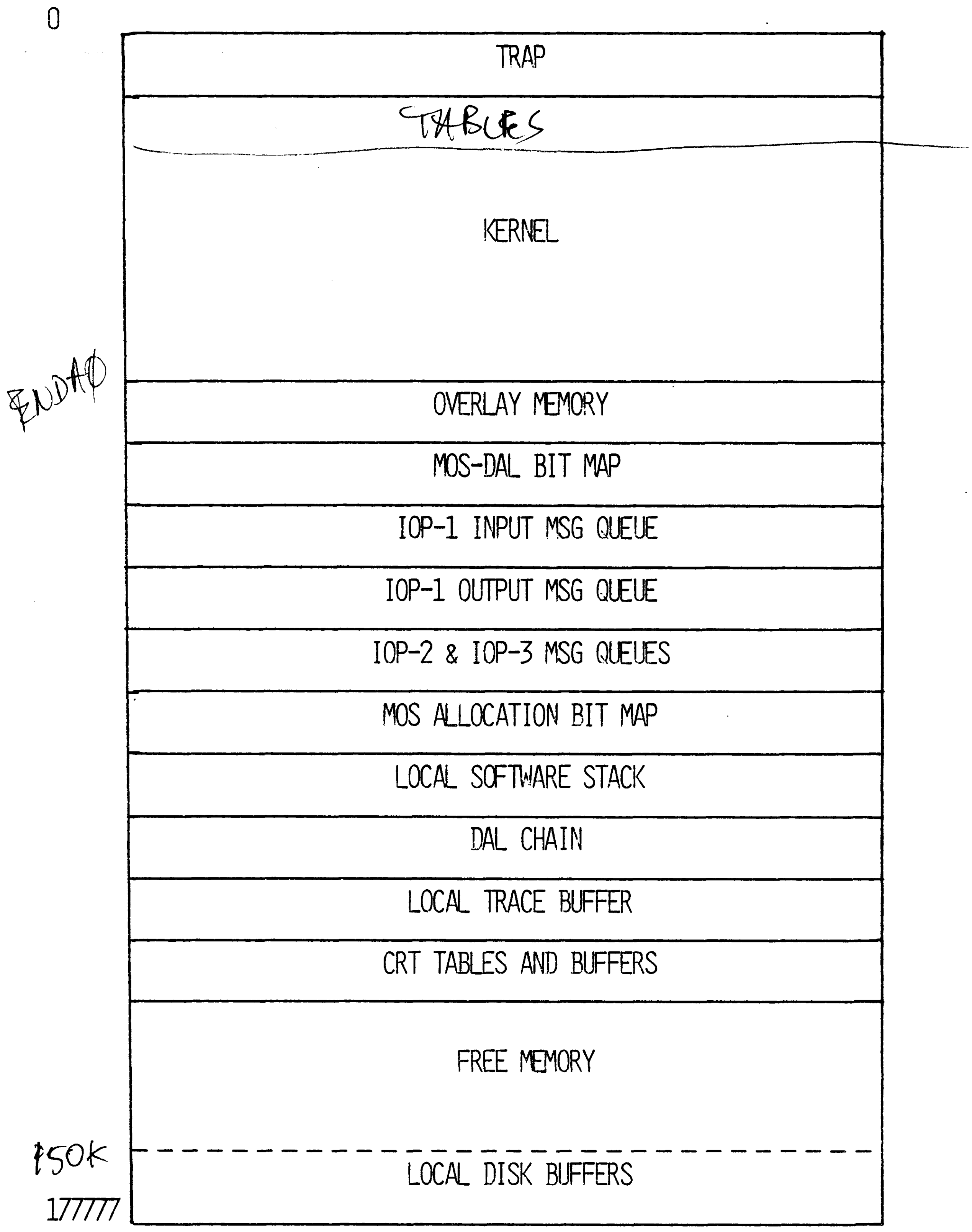


FIGURE 17-1. MIOP LOCAL MEMORY

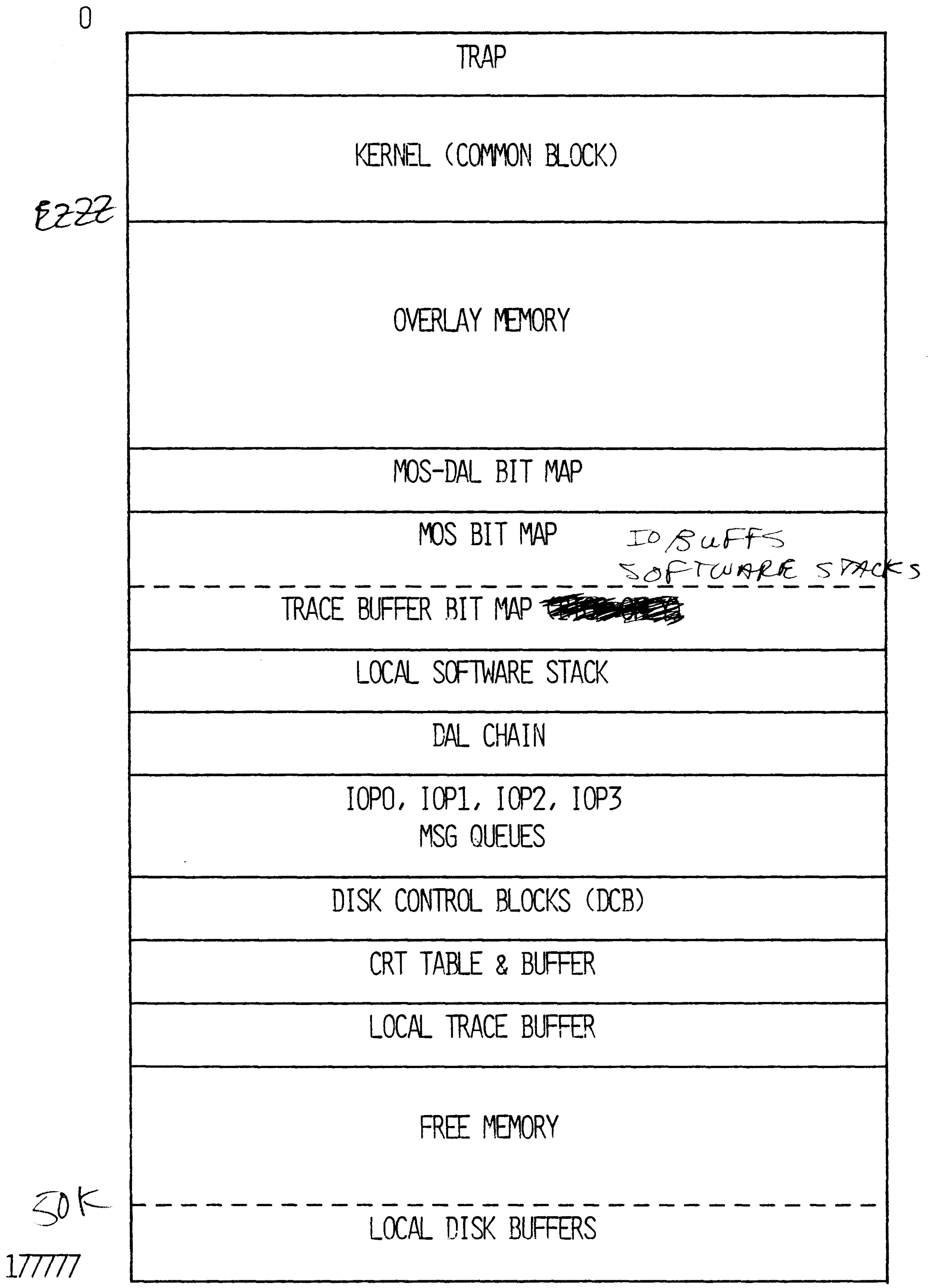


FIGURE 17-2. BIOP, DIOP, OR XIOP LOCAL MEMORY

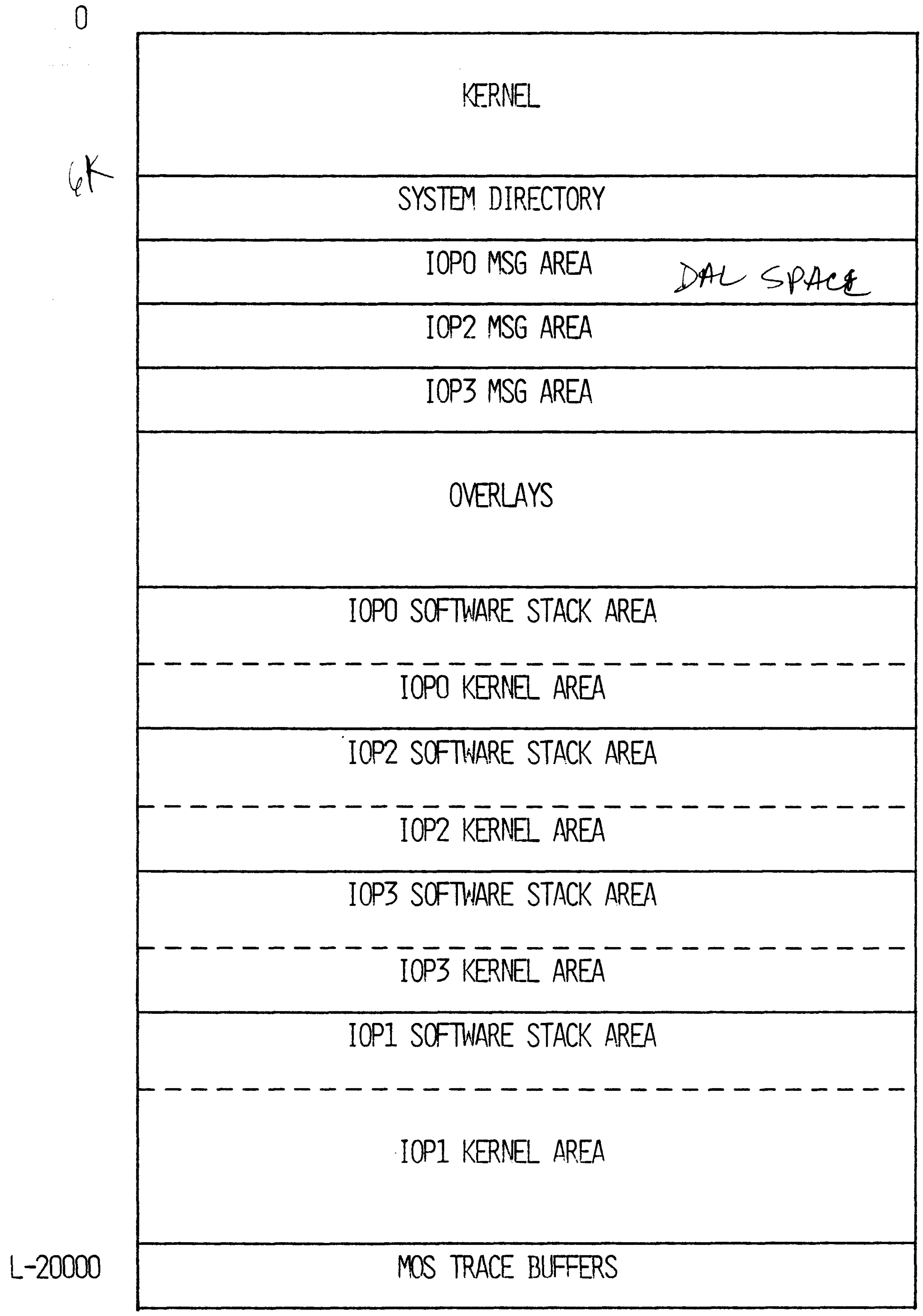


FIGURE 17-3. BUFFER MEMORY

DEADSTART DISK FILES

THREE DISK DIRECTORIES ARE SET ASIDE BY COS AT INSTALL TIME FOR DEADSTART FILES.

THE DIRECTORIES ARE:

COS	USED TO STORE COS BINARY FILES. THE FILES ARE CREATED, NAMED AND SAVED USING THE SV OPTION ON THE START COMMAND OR THE COPY UTILITY.
PAR	USED TO STORE PARAMETER TEXT FILES. THESE ARE CREATED USING THE SV OPTION ON THE START COMMAND; THE COPY UTILITY; AND THE PARAMETER FILE EDITOR.
IOS	USED TO STORE IOS BINARY FILES. THESE ARE CREATED USING THE COPY UTILITY.

THE NAMES OF FILES RESIDING IN THESE DIRECTORIES MUST BE 15 OR LESS ASCII CHARACTERS.

THEY CANNOT BEGIN WITH MT OR TT.

IOS DISK DEADSTART

UNDER CERTAIN CONDITIONS, THE IOS MAY BE RESTARTED FROM A FILE IN THE IOS DIRECTORY ON DISK.

PREREQUISITE:

A FILE, ios, HAS PREVIOUSLY BEEN SAVED WITH THE COPY FILE UTILITY.

PROCEDURE:

1. TYPE CNTRL-D AT THE MIOP KERNEL CONSOLE. IF "SYSDUMP?" APPEARS, GO TO 5.

2. IF NO RESPONSE, MAKE SURE THERE IS NO TAPE LOADED ON THE TAPE DRIVE AND PUSH MASTER CLEAR AND DEADSTART AT THE POWER UNIT.

3. IF 2 RESULTS IN ENTERING THE DEBUGGER. TYPE CNTRL-D TO EXIT.

4. TYPE CNTRL-D AGAIN. IF "SYSDUMP?" DOES NOT APPEAR, A TAPE DEADSTART MUST BE PERFORMED.

5. TYPE "Y" OR "N" IN RESPONSE TO "SYSDUMP?."

6. WHEN DUMP COMPLETE (OR IMMEDIATELY), "RESTART?" WILL BE POSTED. TYPE "Y".

7. ENTER ios IN RESPONSE TO "ENTER RESTART FILE NAME:" MESSAGE.

8. IF AN ERROR OCCURS, IT MAY BE NECESSARY TO DEADSTART FROM TAPE.

MESSAGE	MEANING
DISK ERROR	AN UNRECOVERABLE DISK ERROR OCCURED.
LABEL NOT FOUND	MASTER DEVICE LABEL COULD NOT BE FOUND.
DIRECTORY NOT FOUND	THE IOS DIRECTORY COULD NOT BE FOUND.
FILE NOT FOUND	THE NAMED FILE COULD NOT BE FOUND IN IOS DIRECTORY.
MOS ERROR	AN UNRECOVERABLE ERROR OCCURRED WHILE READING BUFFER MEMORY.
RETRY?	DISPLAYED AFTER ERROR MESSAGES. ENTER "Y" IF ANOTHER TRY AT RESTART IS DESIRED. A NEW PROMPT FOR FILE NAME WILL ALSO BE DISPLAYED.

TABLE 17-3. IOS DISK RESTART ERROR MESSAGES

CPU DEADSTART

CPU DEADSTART REQUIRES A COS BINARY FILE AND A PARAMETER FILE.

EITHER OF THESE CAN RESIDE ON TAPE OR DISK.

THE PARAMETER FILE MAY ALSO BE INPUT FROM THE CONSOLE; OR AN EXISTING ONE MAY BE EDITED THROUGH THE CONSOLE.

THE FORMAT OF THE START COMMAND, INPUT AT THE MIOP KERNEL CONSOLE, IS:

```
START     COSFILE        PARFILE [,ED]
```

WHERE COSFILE IS:

```
MTO:N  [,SV/SYSDSN]
       N   IS TAPE FILE NUMBER.
       SYSDSN IS DESIRED NAME OF SAVED FILE.

SYSDSN - NAME OF FILE IN COS DIRECTORY ON DISK.
```

PARFILE IS:

```
MTO:N  [, SV/PARDSN]
       N   IS TAPE FILE NUMBER.
       PARDSN IS DESIRED NAME OF SAVED FILE

PARDSN - NAME OF FILE IN PAR DIRECTORY ON DISK.

TTI - PARAMETER FILE IS INPUT FROM CONSOLE
```

ED INDICATES PARAMETER FILE IS TO BE EDITED FIRST.

START EXAMPLES

```
START MTO:0    MTO:3
```

- COS BINARY ON TAPE FILE 0; PARAMETER FILE ON TAPE FILE 3.

```
START MTO:0, SV/COS1     MTO:2, SV/PAR1
```

- STARTUP FROM TAPE FILES 0 AND 2; SAVE COS BINARY FILE IN COS DIRECTORY AS COS1; SAVE PARAMETER FILE IN PAR DIRECTORY AS PAR1.

```
START COS1  PAR1, ED
```

- STARTUP FROM DISK FILE COS1 WITH PARAMETER FILE PAR1 BEING EDITED FIRST.

```
START MTO:2  TTI
```

- STARTUP FROM TAPE FILE 2 WITH PARAMETER FILE ENTERED AT CONSOLE.

FILE UTILITIES

THERE ARE SEVEN UTILITIES AVAILABLE FOR MANIPULATING FILES IN THE COS, PAR AND IOS DIRECTORIES.

1. EDIT FN

   INVOKES THE PARAMETER FILE EDITOR.

   FN MAY BE THE NAME OF A FILE ALREADY IN THE PAR DIRECTORY; OR TTI, IF A NEW FILE IS TO BE CREATED.

2. COPY FN_1 FN_2

   COPY FILE FN_1 TO FILE FN_2. THE COPY IS EITHER FROM TAPE TO DISK OR DISK TO TAPE. IF COPY IS FROM TAPE TO DISK, FN_2 CANNOT ALREADY BE IN USE IN THE SPECIFIED DIRECTORY.

   DISK FILES ARE DENOTED AS DIR/FN, WHERE DIR IS COS, PAR OR IOS.

   WHEN COPYING TO THE IOS DIRECTORY, THE OVERLAY FILE MUST IMMEDIATELY FOLLOW THE KERNEL FILE. WHEN COPYING THE OTHER WAY, ALLOW TWO CONSECUTIVE TAPE FILES.

3. FSTAT DIR $[/FN_1, \ldots]$

   DISPLAY FILE STATUS (CREATED, WORD LENGTH) OF ONE OR MORE FILES WITHIN THE SPECIFIED DIRECTORY.

   IF NO FILE NAMES SPECIFIED, THEN STATUS OF ALL FILES IN THE DIRECTORY WILL BE DISPLAYED.

4. DELETE DIR [/FN$_1$,...]

DELETE THE SPECIFIED FILES FROM THE SPECIFIED DIRECTORY.

5. CLEAR DIR

DELETE ALL FILES FROM THE NAMED DIRECTORY.

6. DUMP MTO:Y DIR [/FN$_1$,...]

EXECUTE A FORMATTED DUMP OF THE SPECIFIED FILES TO TAPE FILE Y.

IF NO FILE NAMES SPECIFIED, ALL FILES IN THE DIRECTORY WILL BE DUMPED.

7. LOAD MTO:Y [FN$_1$,FN$_2$,...]

LOAD PREVIOUSLY 'DUMPED' TAPE FILE INTO THE ORIGINAL DIRECTORY.

IF NO FILE NAMES SPECIFIED, ALL FILES ON THE TAPE WILL BE LOADED.

IF A FILE ALREADY EXISTS IN THE DIRECTORY, THE FILE ON TAPE WILL NOT BE LOADED.

DUMP AND LOAD ARE USEFUL WHEN A DIRECTORY GETS FRAGMENTED.

PARAMETER FILE EDITOR

PROVIDES FOR CREATION AND MODIFICATION OF PARAMETER TEXT FILES REQUIRED FOR CPU DEADSTART.

THE EDITOR IS RUN FROM THE MIOP KERNEL CONSOLE.

EACH OF THE FOLLOWING WILL INVOKE THE EDITOR:

1. ED OPTION ON THE START COMMAND.
2. SPECIFYING TTI FOR PARFILE ON THE START COMMAND.
3. EDIT FN.

THE EDITOR OPERATES IN TWO MODES:

1. COMMAND INPUT MODE.

   THIS MODE IS RECOGNIZED BY A '>' IN COLUMN 1.

2. TEXT INPUT MODE.

   INDICATED BY A LINE NUMBER IN COLUMN 1

   INPUT IS ACCEPTED ON A LINE-BY-LINE BASIS.

   TERMINATE LINES BY CARRIAGE RETURNS OR LINE FEEDS.

   THE ESC KEY RETURNS CONTROL TO COMMAND INPUT MODE.

EDITOR COMMANDS

THERE ARE SEVEN COMMANDS AVAILABLE FOR EDITING PARAMETER TEXT FILES.

1. INSERT LN

   INSERT TEXT FOLLOWING THE SPECIFIED LINE NUMBER.

2. APPEND

   APPEND TEXT TO THE FILE.

   IF FILE IS EMPTY, TEXT WILL BE ACCEPTED STARTING AT LINE 1.

3. DELETE LN_1 $[LN_2]$

   DELETE LINES LN_1 TO LN_2 INCLUSIVE.

4. REPLACE LN_1 $[LN_2]$

   REPLACE LINES LN_1 TO LN_2, INCLUSIVE, WITH TEXT TO BE INPUT.

5. TYPE LN_1 $[LN_2]$

   TYPE LINES LN_1 TO LN_2,INCLUSIVE, TO THE CONSOLE.

6. PRINT LN_1 $[LN_2]$

PRINT LINES LN_1 TO LN_2, INCLUSIVE, ON THE PRINTER.

7. BYE

TERMINATE THE EDITOR.

THE FOLLOWING MESSAGE IS DISPLAYED:

"SAVE?"

NO - EDITED VERSION IS DISCARDED. IF EDITOR WAS CALLED FROM START, EDITED VERSION WILL BE SENT TO CPU BUT NOT MADE PERMANENT.

YES - "ENTER FILE NAME:" MESSAGE IS DISPLAYED. EDITED VERSION OF THE FILE WILL BE SAVED IN THE PAR DIRECTORY UNDER THE SPECIFIED NAME.

MESSAGE	MEANING
COMMAND SYNTAX ERROR	THE COMMAND ENTERED WAS NOT IN LEGAL FORMAT.
EXPANDER DEVICE ERROR	AN ERROR WAS ENCOUNTERED ON THE EXPANDER DEVICE BEING USED.
MOS NOT AVAILABLE	BUFFER MEMORY SPACE NOT AVAILABLE.
LOCAL MEMORY NOT AVAILABLE	LOCAL MEMORY NOT AVAILABLE.
DISK ERROR	AN UNRECOVERABLE DISK ERROR OCCURRED.
FILE NOT FOUND:NAME	THE SPECIFIED FILE COULD NOT BE FOUND IN THE CURRENT DIRECTORY.
LABEL NOT FOUND	THE LABEL ON THE MASTER DEVICE COULD NOT BE FOUND.
FILE DIRECTORY FULL	NO MORE ROOM IN THE CURRENT DIRECTORY FOR NEW FILES.
FILE BUFFERS DEPLETED	NOT ENOUGH DISK SPACE REMAINS IN THE CURRENT DIRECTORY TO LOAD THE FILE.
FILE DELETED:NAME	FILE NAMED WAS DELETED FROM THE CURRENT DIRECTORY.
FILE CREATED:NAME	FILE NAME WAS CREATED IN THE CURRENT DIRECTORY.
FILE ALREADY EXITS: NAME	THE NAMED FILE ALREADY EXISTS. IT MUST BE DELETED BEFORE IT CAN BE RE-CREATED.
FILE BEING UPDATED: NAME	NAMED FILE IS BEING WRITTEN OVER.
FILE DUMPED:NAME	NAMED FILE HAS BEEN DUMPED TO THE TAPE FILE.
FILE LOADED:NAME	NAMED FILE HAS BEEN LOADED FROM THE DUMP TAPE AND CREATED IN THE CURRENT DIRECTORY.

TABLE 17-4. START COMMAND AND FILE UTILITY MESSAGES

CHAPTER 18

UTILITIES

HISTORY TRACE

PROVIDES A MEANS FOR TRACING, TO A CERTAIN DEGREE, THE PATH OF EXECUTION THROUGH THE CODE.

STORES PERTINENT DATA RELATING TO SELECTED EVENTS IN A LOCAL MEMORY BUFFER.

LOCAL TRACE BUFFER IS DUMPED TO A CIRCULAR BUFFER IN BUFFER MEMORY.

MAY BE USED IN DEBUGGING AND FINE TUNING THE SYSTEM.

TRACE FORMAT

EACH TRACE ENTRY IS EIGHT PARCELS LONG.

$\boxed{\text{EVENT TIME OVERLAY PAR}_1\ \text{PAR}_2\ \text{PAR}_3\ \text{PAR}_4\ \text{PAR}_5}$

EVENT - OCTAL CODE OF TRACE EVENT

TIME - LOW ORDER 16 BITS OF RTC AT TIME OF RECORDING Millisecs

OVERLAY - NUMBER OF CONTROLLING OVERLAY AT TIME OF RECORDING Kernel = 177777

PAR_I - PARAMETER TO BE RECORDED

EVENT CODE	DESCRIPTION
TF$INT(1)	EXIT FROM COMMON INTERRUPT HANDLER (IPOI)
TF$CALL(2)	ENTRANCE TO KERNEL FUNCTION PROCESSOR (ENTR)
TF$TSK(3)	EXIT FROM ACTIVITY DISPATCHING (ELDPA)
TF$CHN(4)	INDIVIDUAL INTERRUPT HANDLERS
TF$FCT(5)	INDIVIDUAL KERNEL FUNCTION PROCESSOR
TF$SEK(6)	DISK SEEK ROUTINE (DIOH)
TF$DSK(7)	DISK READ/WRITE PROCESSOR (DIOSTRT)
TF$DSKER(10)	DISK ERROR HANDLER (IDERROR)
TF$HSP(11)	MEMORY CHANNEL I/O (CDEM)
TF$OLAY(12)	OVERLAY LOADING (OVLBA)
TF$ACOM(13)	RECEIVE MESSAGES FROM OTHER I/O PROCESSORS (ACOM)
TF$ATA(14)	SEND MESSAGES TO OTHER I/O PROCESSORS (EMSGIOP)

FIGURE 18-1. EVENT CODE DESCRIPTIONS.

TRACE ON-LINE COMMANDS

ENABLE/DISABLE SELECTED EVENTS

PROVIDE A FORMATTED LISTING OF TRACE BUFFERS

```
1)  TRACE { ON  }  EVENT [/SUBCODE]
          { OFF }
```

TURNS ON OR OFF A SELECTED EVENT AND ONE OR ALL OR ITS ASSOCIATED SUBCODES.

```
2)  TRACE { ON  }  ALL
          { OFF }
```

TURNS TRACE ON OR OFF FOR ALL EVENTS.

```
3)  TRACE { ON  }  MOS
          { OFF }
```

CONTROLS DUMPING OF LOCAL BUFFER TO BUFFER MEMORY.

```
4)  TRACE DUMP { LOCAL }
               { MOS   }
```

PRINTS A FORMATTED LISTING OF SPECIFIED TRACE BUFFER. EVENTS ARRANGED MOST RECENT TO LEAST RECENT.

DMP

GIVES UNFORMATTED DUMP OF DIFFERENT PARTS OF THE SYSTEM AS AN AID IN DEBUGGING.

IS A STAND-ALONE PROGRAM DEADSTARTED INTO MIOP.

PRINTS OUT THE FOLLOWING REGISTERS AND MEMORIES:

- CENTRAL MEMORY
- BUFFER MEMORY
- IOP LOCAL MEMORIES
- IOP A, B, C, OPERAND REGISTERS AND EXIT STACK
- LOCAL AND BUFFER MEMORY TRACE BUFFERS

Cal C.

CRAY-1 I/O SUBSYSTEM

EXERCISE 11

1. When deadstarting an I/O Processor, what causes execution to begin at Location 0?

Master clear sets P = ∅

2. When deadstarting an I/O Subsystem from tape, how does the kernel get into BIOP?

Read from MOS after interrupt from MIOP during startup

3. What is the history trace used for?

Selective trace of kernel and overlay entrys and parameters

4. When would the DMP utility be used instead of SYSDUMP?

During startup

5. List two ways to enter the Debugger.

i) Debug command

ii) In MIOP at startup

SYSDUMP

DUMPS SELECTED RESOURCES TO AN AREA OF DISK PRE-SELECTED AT INSTALL TIME, OR SPECIFIED DURING SYSDUMP.

THIS DUMP MAY THEN BE FORMATTED VIA FDUMP AND DISPOSED APPROPRIATELY.

RESTART MAY OCCUR WHEN THE DUMP IS COMPLETE.

THE FOLLOWING MEMORIES AND REGISTERS MAY BE DUMPED:

- CENTRAL MEMORY
- BUFFER MEMORY
- IOP LOCAL MEMORIES
- IOP OPERAND REGISTERS
- IOP A, B, C, E REGISTERS AND EXIT STACK
- IOP CHANNELS' BZ AND DN FLAGS
- CPU B, T, V AND VM REGISTERS

SYSDUMP IS ENTERED BY TYPING CNTRL-D AT THE MIOP KERNEL CONSOLE.

DEBUGGER

ALLOWS ON-LINE DEBUGGING OF IOS.

ASSEMBLED WITH THE KERNEL AND IS MIOP RESIDENT AT INITIALIZATION.

> SUBSEQUENT REFERENCES TO THE DEBUGGER LOAD IT FROM BUFFER MEMORY INTO AN I/O BUFFER.

ALLOWS SETTING OF BREAKPOINTS AND EXAMINATION AND MODIFICATION OF BUFFER MEMORY AND THE I/O PROCESSOR'S REGISTERS AND LOCAL MEMORY.

DEBUGGING COMMANDS ENTERED AT THE KERNEL CONSOLE.

> MUST HAVE A KERNEL CONSOLE ON AN IOP IN ORDER TO DEBUG IT WITH THE DEBUGGER.

THE DEBUGGER MAY BE ENTERED SEVERAL WAYS:

- DURING SYSTEM INITIALIZATION
- WHEN A R=XFAR INSTRUCTION IS ENCOUNTERED IN NON-INTERRUPTIBLE CODE.
- WHEN AN I/O PROCESSOR HALT OCCURS
- WHEN THE DEBUG COMMAND IS ENTERED AT THE KERNEL CONSOLE.

DEBUGGER COMMANDS ALLOW OPERATOR TO DISPLAY AND MODIFY THE FOLLOWING:

A REGISTER

B REGISTER

C REGISTER

P REGISTER

E REGISTER

EXIT STACK

OPERAND REGISTERS

LOCAL MEMORY

BUFFER MEMORY

CHANNEL STATES MAY ALSO BE EXAMINED AND CHANNEL FUNCTIONS ISSUED WITH THE DEBUGGER.

UP TO 4 ACTIVE BREAKPOINTS MAY BE SET IN THE CODE.

DOUBLE BREAKPOINTS MAY BE SPECIFIED.

APPENDICES

APPENDIX I

I/O PROCESSOR INSTRUCTION SUMMARY

I/O PROCESSOR INSTRUCTION SUMMARY

IOP	APML	Description
000	PASS	No operation
001	EXIT	Exit from subroutine
002	I = 0	Disable system interrupts
003	I = 1	Enable system interrupts
004	A = A > d	Right shift C and A by d places, end off
005	A = A < d	Left shift C and A by d places, end off
006	A = A >> d	Right shift C and A by d places, circular
007	A = A << d	Left shift C and A by d places, circular
010	A = d	Transmit d to A
011	A = A & d	Logical product of A and d to A
012	A = A + d	Add d to A
013	A = A - d	Subtract d from A
014	A = k	Transmit k to A
015	A = A & k	Logical product of A and k to A
016	A = A + k	Add k to A
017	A = A - k	Subtract k from A
020	A = dd	Transmit operand register d to A
021	A = A & dd	Logical product of A and operand register d to A
022	A = A + dd	Add operand register d to A
023	A = A - dd	Subtract operand register d from A
024	dd = A	Transmit A to register d
025	dd = A + dd	Add operand register d to A, result to operand register d
026	dd = dd + 1	Transmit register d to A, add 1, result to operand register d
027	dd = dd - 1	Transmit register d to A, subtract 1, result to operand register d
030	A = (dd)	Transmit contents of memory addressed by register d to A
031	A = A & (dd)	Logical product of A and contents of memory addressed by register d, result to A
032	A = A + (dd)	Add contents of memory addressed by register d to A, result to A
033	A = A - (dd)	Subtract contents of memory addressed by register d from A, result to A

IOP	APML	Description
034	(dd) = A	Transmit A to memory addressed by register d
035	(dd) = A + (dd)	Add memory addressed by register d to A, result to same memory location
036	(dd) = (dd) + 1	Transmit memory addressed by register d to A, add 1, result to same memory location
037	(dd) = (dd) - 1	Transmit memory addressed by register d to A, subtract 1, result to same memory location
040	C = 1, iod = DN	Set carry equal to channel d done
041	C = 1, iod = BZ	Set carry equal to channel d busy
042	C = 1, IOB = DN	Set carry equal to channel B done
043	C = 1, IOB = BZ	Set carry equal to channel B busy
044	A = A > B	Right shift C and A by B places, end off
045	A = A < B	Left shift C and A by B places, end off
046	A = A >> B	Right shift C and A by B places, circular
047	A = A << B	Left shift C and A by B places, circular
050	A = B	Transmit B to A
051	A = A & B	Logical product of A and B to A
052	A = A + B	Add B to A, result to A
053	A = A - B	Subtract B from A, result to A
054	B = A	Transmit A to B
055	B = A + B	Add B to A, result to B
056	B = B + 1	Transmit B to A, add 1, result to B
057	B = B - 1	Transmit B to A, subtract 1, result to B
060	A = (B)	Transmit operand register B to A
061	A = A & (B)	Logical product of A and operand register B to A
062	A = A + (B)	Add operand register B to A, result to A
063	A = A - (B)	Subtract operand register B from A, result to A
064	(B) = A	Transmit A to operand register B
065	(B) = A + (B)	Add operand register B to A, result to operand register B
066	(B) = (B) + 1	Transmit operand register B to A, add 1, result to operand register B
067	(B) = (B) - 1	Transmit operand register B to A, subtract 1, result to operand register B
070	P = P + d	Jump to P + d
071	P = P - d	Jump to P - d
072	P = P + d	Return jump to P + d
073	P = P - d	Return jump to P - d
074	P = dd	Jump to address in operand register d
075	P = dd + k	Jump to sum of k and operand register d
076	R = dd	Return jump to address in operand register d
077	R = dd + k	Return jump to sum of k and operand register d

IOP	APML	Description
100	P = P + d, C = 0	Jump to P + d if carry = 0
101	P = P + d, C # 0	Jump to P + d if carry ≠ 0
102	P = P + d, A = 0	Jump to P + d if A = 0
103	P = P + d, A # 0	Jump to P + d if A ≠ 0
104	P = P - d, C = 0	Jump to P - d if carry = 0
105	P = P - d, C # 0	Jump to P - d if carry ≠ 0
106	P = P - d, A = 0	Jump to P - d if A = 0
107	P = P - d, A # 0	Jump to P - d if A ≠ 0
110	R = P + d, C = 0	Return jump to P + d if carry = 0
111	R = P + d, C # 0	Return jump to P + d if carry ≠ 0
112	R = P + d, A = 0	Return jump to P + d if A = 0
113	R = P + d, A # 0	Return jump to P + d if A ≠ 0
114	R = P - d, C = 0	Return jump to P - d if carry = 0
115	R = P - d, C # 0	Return jump to P - d if carry ≠ 0
116	R = P - d, A = 0	Return jump to P - d if A = 0
117	R = P - d, A # 0	Return jump to P - d if A ≠ 0
120	P = dd, C = 0	Jump to address in operand register d if carry = 0
121	P = dd, C # 0	Jump to address in operand register d if carry ≠ 0
122	P = dd, A = 0	Jump to address in operand register d if A = 0
123	P = dd, A # 0	Jump to address in operand register d if A ≠ 0
124	P = dd + k, C = 0	Jump to address in operand register d + k if carry = 0
125	P = dd + k, C # 0	Jump to address in operand register d + k if carry ≠ 0
126	P = dd + k, A = 0	Jump to address in operand register d + k if A = 0
127	P = dd + k, A # 0	Jump to address in operand register d + k if A ≠ 0
130	R = dd, C = 0	Return jump to address in operand register d if carry = 0
131	R = dd, C # 0	Return jump to address in operand register d if carry ≠ 0
132	R = dd, A = 0	Return jump to address in operand register d if A = 0
133	R = dd, A # 0	Return jump to address in operand register d if A ≠ 0

IOP	APML	Description
134	R = dd + k, C = 0	Return jump to address in operand register d + k if carry = 0
135	R = dd + k, C # 0	Return jump to address in operand register d + k if carry ≠ 0
136	R = dd + k, A = 0	Return jump to address in operand register d + k if A = 0
137	R = dd + k, A # 0	Return jump to address in operand register d + k if A ≠ 0
140	iod : 0	Channel d function 0
141	iod : 1	Channel d function 1
142	iod : 2	Channel d function 2
143	iod : 3	Channel d function 3
144	iod : 4	Channel d function 4
145	iod : 5	Channel d function 5
146	iod : 6	Channel d function 6
147	iod : 7	Channel d function 7
150	iod : 10	Channel d function 10
151	iod : 11	Channel d function 11
152	iod : 12	Channel d function 12
153	iod : 13	Channel d function 13
154	iod : 14	Channel d function 14
155	iod : 15	Channel d function 15
156	iod : 16	Channel d function 16
157	iod : 17	Channel d function 17
160	IOB : 0	Channel B function 0
161	IOB : 1	Channel B function 1
162	IOB : 2	Channel B function 2
163	IOB : 3	Channel B function 3
164	IOB : 4	Channel B function 4
165	IOB : 5	Channel B function 5
166	IOB : 6	Channel B function 6
167	IOB : 7	Channel B function 7
170	IOB : 10	Channel B function 10
171	IOB : 11	Channel B function 11
172	IOB : 12	Channel B function 12
173	IOB : 13	Channel B function 13
174	IOB : 14	Channel B function 14
175	IOB : 15	Channel B function 15
176	IOB : 16	Channel B function 16
177	IOB : 17	Channel B function 17

APPENDIX II

SYSTEM CHANNEL ASSIGNMENTS

SYSTEM CHANNEL ASSIGNMENTS

Typical Model 4400 system channel assignments

PROCESSOR	CHANNEL	MNEMONIC	FUNCTION
Master I/O Processor	0	IOR	Interrupt request
	1	PFR	Program fetch request
	2	PXS	Program exit stack
	3	LME	I/O Memory error
	4	RTC	Real-time clock
	5	MOS	Buffer Memory Interface (DMA 3)
	6	AIA	Input from Buffer I/O Processor
	7	AOA	Output to Buffer I/O Processor
	10	AIB	Input from Disk I/O Processor
	11	AOB	Output to Disk I/O Processor
	12	AIC	Input from Auxiliary I/O Processor
	13	AOC	Output to Auxiliary I/O Processor
	14		
	15		
	16	ERA	Error log
	17	EXB	Peripheral Expander (DMA 0)
	20	CIA	Input from CRAY-1 channel (DMA 1)
	21	COA	Output to CRAY-1 channel (DMA 1)
	22		
	23		
	24	CIB	Input from F.-E. Interface (DMA 2)
	25	COB	Output to F.-E. Interface (DMA 2)
	26		
	27		
	30	CIC	Input from F.-E. Interface (DMA 4)
	31	COC	Output to F.-E. Interface (DMA 4)
	32		
	33		
	34	CID	Input from F.-E. Interface (DMA 5)
	35	COD	Output to F.-E. Interface (DMA 5)
	36		
	37		
	40	TIA	Console 0 keyboard
	41	TOA	Console 0 display
	42	TIB	Console 1 keyboard
	43	TOB	Console 1 display
	44		
	45		
	46		
	47		

Typical Model 4400 system channel assignments (continued)

PROCESSOR	CHANNEL	MNEMONIC	FUNCTION
Buffer I/O Processor	0	IOR	Interrupt request
	1	PFR	Program fetch request
	2	PXS	Program exit stack
	3	LME	I/O Memory error
	4	RTC	Real-time clock
	5	MOS	Buffer Memory Interface (DMA 3)
	6	AIA	Input from Master I/O Processor
	7	AOA	Output from Master I/O Processor
	10	AIB	Input from Disk I/O Processor
	11	AOB	Output to Disk I/O Processor
	12	AIC	Input from Auxiliary I/O Processor
	13	AOC	Output to Auxiliary I/O Processor
	14	HIA	Input from Memory Channel (DMA 4)
	15	HOA	Output to Memory Channel (DMA 4)
	16		
	17		
	20	DKA	Disk Storage Unit 0 (DMA 0)
	21	DKB	Disk Storage Unit 1 (DMA 0)
	22	DKC	Disk Storage Unit 2 (DMA 1)
	23	DKD	Disk Storage Unit 3 (DMA 1)
	24	DKE	Disk Storage Unit 4 (DMA 2)
	25	DKF	Disk Storage Unit 5 (DMA 2)
	26	DKG	Disk Storage Unit 6 (DMA 5)
	27	DKH	Disk Storage Unit 7 (DMA 5)
	30		
	31		
	32		
	33		
	34		
	35		
	36		
	37		
	40		
	41		
	42		
	43		
	44		
	45		
	46		
	47		

Typical Model 4400 system channel assignments (continued)

PROCESSOR	CHANNEL	MNEMONIC	FUNCTION
Disk	0	IOR	Interrupt request
I/O	1	PFR	Program fetch request
Processor	2	PXR	Program exit stack
	3	LME	I/O Memory error
	4	RTC	Real-time clock
	5	MOS	Buffer Memory Interface (DMA 3)
	6	AIA	Input from Master I/O Processor
	7	AOA	Output to Master I/O Processor
	10	AIB	Input from Buffer I/O Processor
	11	AOB	Output to Buffer I/O Processor
	12	AIC	Input from Auxiliary I/O Processor
	13	AOC	Output to Auxiliary I/O Processor
	14		
	15		
	16		
	17		
	20	DKA	Disk Storage Unit 0 (DMA 1)
	21	DKB	Disk Storage Unit 1 (DMA 1)
	22	DKC	Disk Storage Unit 2 (DMA 1)
	23	DKD	Disk Storage Unit 3 (DMA 1)
	24	DKE	Disk Storage Unit 4 (DMA 2)
	25	DKF	Disk Storage Unit 5 (DMA 2)
	26	DKG	Disk Storage Unit 6 (DMA 2)
	27	DKH	Disk Storage Unit 7 (DMA 2)
	30	DKI	Disk Storage Unit 8 (DMA 4)
	31	DKJ	Disk Storage Unit 9 (DMA 4)
	32	DKK	Disk Storage Unit 10 (DMA 4)
	33	DKL	Disk Storage Unit 11 (DMA 4)
	34	DKM	Disk Storage Unit 12 (DMA 5)
	35	DKN	Disk Storage Unit 13 (DMA 5)
	36	DKO	Disk Storage Unit 14 (DMA 5)
	37	DKP	Disk Storage Unit 15 (DMA 5)
	40		
	41		
	42		
	43		
	44		
	45		
	46		
	47		

Typical Model 4400 system channel assignments (continued)

PROCESSOR	CHANNEL	MNEMONIC	FUNCTION
Auxiliary I/O Processor	0	IOR	Interrupt request
	1	PFR	Program fetch request
	2	PXS	Program exit stack
	3	LME	I/O Memory error
	4	RTC	Real time clock
	5	MOS	Buffer Memory Interface (DMA 3)
	6	AIA	Input from Master I/O Processor
	7	AOA	Output to Master I/O Processor
	10	AIB	Input from Buffer I/O Processor
	11	AOB	Output to Buffer I/O Processor
	12	AIC	Input from Disk I/O Processor
	13	AOC	Output to Disk I/O Processor
	14		
	15		
	16		
	17		
	20	BMA	Block Multiplexer Channel 0 (DMA 0)
	21	BMB	Block Multiplexer Channel 1 (DMA 0)
	22	BMC	Block Multiplexer Channel 2 (DMA 0)
	23	BMD	Block Multiplexer Channel 3 (DMA 0)
	24	BME	Block Multiplexer Channel 4 (DMA 1)
	25	BMF	Block Multiplexer Channel 5 (DMA 1)
	26	BMG	Block Multiplexer Channel 6 (DMA 1)
	27	BMH	Block Multiplexer Channel 7 (DMA 1)
	30	BMI	Block Multiplexer Channel 10 (DMA 2)
	31	BMJ	Block Multiplexer Channel 11 (DMA 2)
	32	BMK	Block Multiplexer Channel 12 (DMA 2)
	33	BML	Block Multiplexer Channel 13 (DMA 2)
	34	BMM	Block Multiplexer Channel 14 (DMA 5)
	35	BMN	Block Multiplexer Channel 15 (DMA 5)
	36	BMO	Block Multiplexer Channel 16 (DMA 5)
	37	BMP	Block Multiplexer Channel 17 (DMA 5)
	38		
	39		
	40		
	41		
	42		
	43		
	44		
	45		
	46		
	47		

APPENDIX III

IOP BLOCK DIAGRAM IN DETAIL

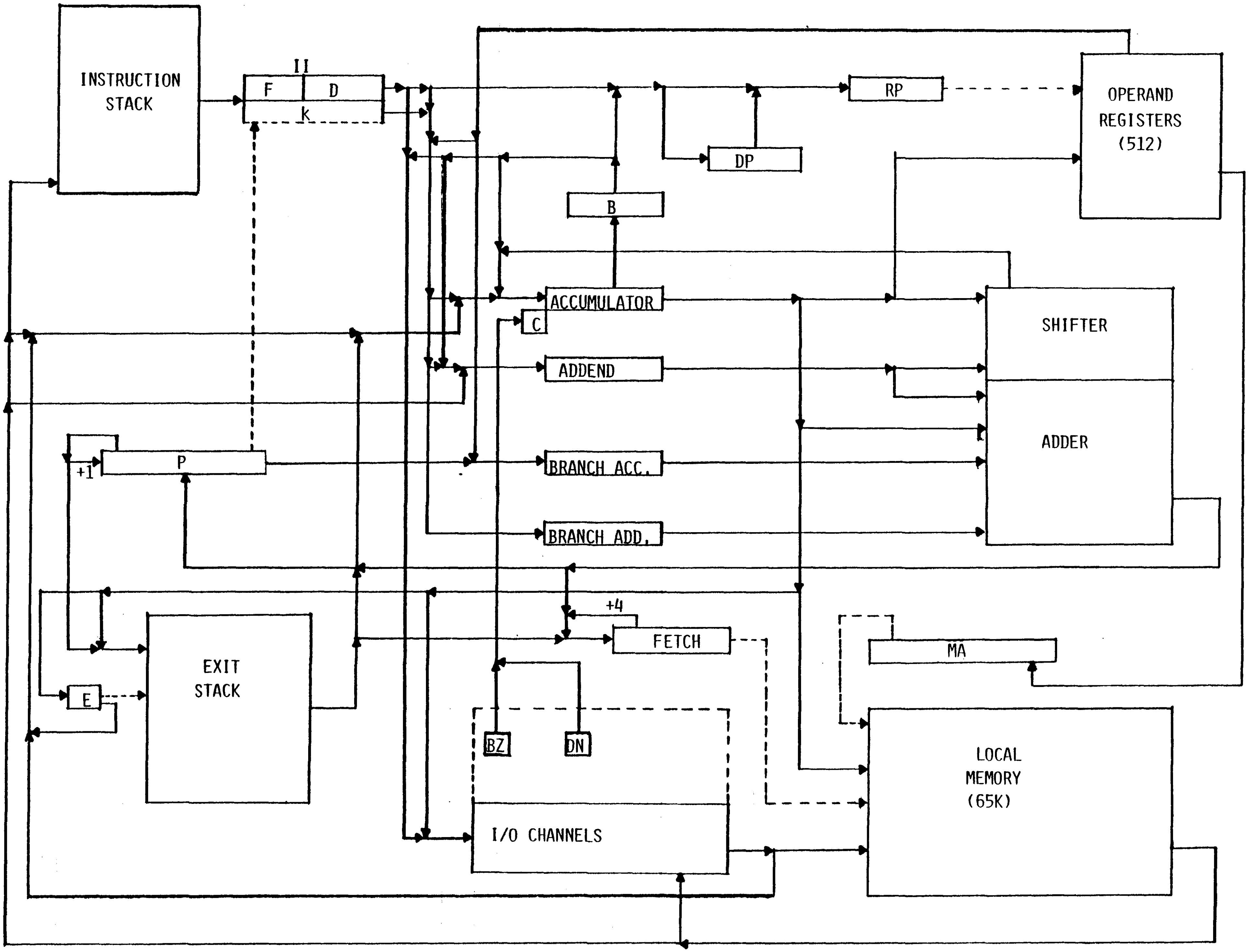


FIGURE III-1. [illegible] BLOCK DIAGRAM

THE FOLLOWING ARE PART OF THE INSTRUCTION CONTROL NETWORK:

RP (REGISTER POINTER) REGISTER

DP (DESTINATION POINTER) REGISTER

FETCH REGISTER

MA (MEMORY ADDRESS) REGISTER

RP REGISTER

9 BITS WIDE

POINTS TO AN OPERAND REGISTER

LOADED FROM II REGISTER D FIELD OR B REGISTER

DP REGISTER

9 BITS WIDE

STORES ADDRESS OF OPERAND REGISTER TO BE WRITTEN

LOADED FROM II REGISTER D FIELD OR B REGISTER

PROTECTS READING OF OPERAND REGISTER BEFORE NEW DATA AVAILABLE

CONTENTS GO TO RP WHEN ACCUMULATOR READY TO WRITE OPERAND REGISTER.

FETCH REGISTER

16 BITS WIDE

HOLDS ADDRESS OF FIRST INSTRUCTION PARCEL OF FOUR PARCEL GROUP TO BE FETCHED FROM MEMORY.

INCREMENTED BY 4 EVERY CP

MAY BE LOADED FROM EXIT STACK OR ADDER

MA REGISTER

16 BITS WIDE

HOLDS ADDRESS FOR A LOCAL MEMORY REFERENCE

LOADED FROM AN OPERAND REGISTER

BRANCH ACCUMULATOR

16 BITS WIDE

LOADED BY P OR AN OPERAND REGISTER ON A BRANCH INSTRUCTION

SUPPLIES OPERAND TO ADDER

BRANCH ADDEND REGISTER

16 BITS WIDE

LOADED BY D OR K FIELD ON A BRANCH INSTRUCTION

SUPPLIES OPERAND TO ADDER

APPENDIX IV

IOS ACTIVITY SUMMARY

IOS ACTIVITIES

NAME	SUBSYSTEM	I/O PROCESSOR
ACOM	DISK,STATION,CONCENTRATOR,INTERACTIVE	ALL
AMSG	STATION,CONCENTRATOR,INTERACTIVE	MIOP, BIOP
CDEM	ALL	MIOP, BIOP
DISK	DISK	BIOP, DIOP
ERRECK	DISK	BIOP, DIOP
CONC	CONCENTRATOR	MIOP
ENDCONC	CONCENTRATOR	MIOP
CONCI	CONCENTRATOR	MIOP
CONCO	CONCENTRATOR	MIOP
FEREAD	CONCENTRATOR	MIOP
FEWRIT	CONCENTRATOR	MIOP
MSGIO	STATION,CONCENTRATOR,INTERACTIVE	BIOP
STATION	STATION	MIOP
KEYBD	STATION	MIOP
CLI	STATION	MIOP
DISPLAY	STATION	MIOP

NAME	SUBSYSTEM	I/O PROCESSOR
PROTOCOL	STATION	MIOP
STAGEIN	STATION	MIOP
STAGEOUT	STATION	MIOP
CONFIG	- -	ALL
LISTO	- -	MIOP
CRAY	STATION,CONCENTRATOR,INTERACTIVE	MIOP
HPLOAD	STATION,CONCENTRATOR,INTERACTIVE	MIOP
IACON	INTERACTIVE	MIOP
IAIOP	INTERACTIVE	MIOP
PATCH	- -	ALL
START	- -	MIOP
TRACE	- -	ALL

APPENDIX V

KERNEL SERVICE REQUEST FUNCTIONS

CODE	NAME	DESCRIPTION	RETURN TO
1	PUSH	PUT ACTIVITY ON A QUEUE AT PRIORITY	KERNEL (ESwP)
2	POP	REMOVE ACTIVITY FROM A QUEUE AND PLACE IT ON CP QUEUE AT PRIORITY.	REQUESTER
3	TERMINATE	TERMINATE AN ACTIVITY BY RELEASING ITS' AD AND SMOD AREAS.	KERNEL
4	GIVEUP	RESCHEDULE AN ACTIVE TASK BY PRIORITY	KERNEL
7	PAUSE	SUSPEND AN ACTIVITY FOR TENTHS OF A SECOND.	KERNEL
10	DELAY	SUSPEND AN ACTIVITY FOR MILLISECONDS	KERNEL
11	TPUSH	PUT ACTIVITY ON A QUEUE AND ON A TIMER QUEUE FOR TENTHS OF A SECOND.	KERNEL
12	SYNC	SYNCHRONIZE TWO ACTIVITIES	REQUESTER
15	ALERT	REQUEST ANOTHER IOP TO CREATE AN ACTIVITY.	KERNEL
16	AWAKE	REQUEST ANOTHER IOP TO ACTIVATE AN ACTIVITY.	KERNEL/ REQUESTER
17	RESPOND	SEND RESPONSE TO ANOTHER IOP	REQUESTER
20	MSG	SEND A MESSAGE TO A CRT	KERNEL/ REQUESTER
21	MSGR	SEND A MESSAGE TO A CRT AND WAIT FOR RESPONSE.	KERNEL/ REQUESTER

CODE	NAME	DESCRIPTION	RETURN TO
22	OUTPUT	OUTPUT A MESSAGE TO A CRT (STATION)	KERNEL
23	FRNTNDIO	INITIATE I/O BETWEEN A CONCENTRATOR AND A FRONT END.	KERNEL
25	RECEIVE	INPUT ONE CHARACTER FROM A CONSOLE	REQUESTER
26	SBMXIO	INITIATE I/O ON A BLOCK MUX CHANNEL	KERNEL
30	GETMEM	ALLOCATE LOCAL MEMORY	REQUESTER
31	RELMEM	RELEASE LOCAL MEMORY	REQUESTER
32	BGET	ALLOCATE A 512 WORD (4000 PARCEL) LOCAL BUFFER.	REQUESTER
33	BRET	RELEASE A 512 WORD LOCAL BUFFER	REQUESTER
35	MGET	ALLOCATE A 512 WORD MOS BUFFER	KERNEL/ REQUESTER
36	MPUT	RELEASE A 512 WORD MOS BUFFER	REQUESTER
44	POLL	SEND A MESSAGE TO THE CPU	KERNEL
45	TRANSFER	MOVE DATA BETWEEN MOS AND CENTRAL MEMORY.	KERNEL
46	MOSR	READ DATA FROM MOS TO LOCAL MEMORY	REQUESTER
47	MOSW	WRITE DATA FROM LOCAL TO MOS MEMORY	REQUESTER

CODE	NAME	DESCRIPTION	RETURN TO
50	CALL	PASS CONTROL TO AN OVERLAY WITH RETURN.	OVERLAY
51	GOTO	~~RETURN~~ PASS CONTROL TO AN OVERLAY	OVERLAY
52	RETURN	RETURN CONTROL TO AN OVERLAY	~~OVERLAY~~ KERNEL
53	FIND	FIND MOS ADDRESS AND WORD LENGTH OF AN OVERLAY.	REQUESTER
54	FLUSH	RE-INITIALIZE OVERLAY MEMORY	KERNEL
55	CREATE	SET UP AN INDEPENDENT ACTIVITY AND PLACE IT ON A CPU QUEUE.	REQUESTER

APPENDIX VI

INTERNAL SUBROUTINES

NAME	DESCRIPTION
ERGC	BUILDS A NEW ACTIVITY DESCRIPTOR AND SMOD. PLACES AD ON CP QUEUE.
EREB	FINDS SPACE FOR AD AND CLEARS IT. PLACES AD ON CHAIN OF AD.
EREC	GETS MOS FOR SMOD AND SETS UP PARAMETERS IN AD
ERED	INITIALIZES SMOD IN SCRATCH AREA
STOREGS	STORES SPECIFIED OPERAND REGISTERS IN SMOD
LODREGS	LOADS SPECIFIED OPERAND REGISTERS FROM SMOD
EDQU	REMOVES ACTIVITY FROM SPECIFIED QUEUE
ENQU	PUTS ACTIVITY ON SPECIFIED QUEUE
EQCP	PUTS ACTIVITY ON CP QUEUE AT PRIORITY
EQUE	PUTS ACTIVITY ON SPECIFIED QUEUE AT PRIORITY
EPOQ	PUTS ENTRY ON MESSAGE QUEUE
ETOQ	REMOVE ENTRY FROM MESSAGE QUEUE
QTIME	PUT AN ACTIVITY ON TIMER QUEUE. ACTIVITY MUST ALSO BE ON ANOTHER QUEUE.
DQTIME	REMOVE AN ACTIVITY FROM TIMER QUEUE
DQFIND	LOCATE AND REMOVE AN ENTRY FROM A QUEUE
EMSGIOP	SEND MESSAGE TO ANOTHER IOP VIA ACCUMULATOR CHANNEL

APPENDIX VII

DISK SUBSYSTEM DETAILED INTERACTION

DISK READ REQUEST VIA DIOP

1. MIOP - INTERRUPT ON LOW SPEED INPUT CHANNEL DUE TO TRANSFER OF 6 WORD I/O REQUEST PACKET.
   - CDEM IS ACTIVATED BY INTERRUPT ANSWERING (IA).
   - CDEM BUILDS A MASTER DISK ACTIVITY LINK (MDAL) BY PREFIXING 10_8 CONTROL PARCELS TO THE 6 WORD PACKET.
   - CDEM ALLOCATES A MOS DAL AND WRITES LOCAL MDAL TO MOS.
   - CDEM QUEUES ACCUMULATOR MESSAGE TO BE SENT TO DIOP (SPECIFYING ADDRESS OF DAL IN MOS).
   - CDEM DEALLOCATES LOCAL DAL.

2. DIOP - INTERRUPT ON IOP TO IOP INPUT CHANNEL.
   - ACOM IS ACTIVATED BY IA.
   - ACOM DECODES ACCUMULATOR MSG, ALLOCATES A LOCAL DAL, AND READS IN MDAL FROM MOS.
   - ACOM BUILDS FIRST, UP TO 3, EXECUTABLE DALS (EDALS) AND ALLOCATES LOCAL AND MOS BUFFERS.
   - ACOM STARTS FIRST READ (OR SEEK) IF DISK IS NOT BUSY, AND PUTS EDALS ON DISK CONTROL BLOCK (DCB) EDAL QUEUE.

3. DIOP - INTERRUPT ON DISK CHANNEL.
   - IA ALLOCATES A LOCAL BUFFER FOR NEXT TRANSFER, IF NECESSARY, AND STARTS NEXT SECTOR.
   - IA MOVES FINISHED EDAL TO DCB DONE QUEUE AND ACTIVATES THE DISK DEMON (DISK).

4. DIOP - DISK TAKES TOP ENTRY OFF THE DONE QUEUE.
   - DISK ALLOCATES A MOS BUFFER IF NECESSARY, AND MOVES THE DATA FROM LOCAL MEMORY TO MOS, AND DEALLOCATES THE LOCAL BUFFER.
   - DISK ALLOCATES A MOS DAL AND WRITES LOCAL EDAL TO MOS AND DEALLOCATES THE LOCAL EDAL.
   - DISK QUEUES AN ACCUMULATOR MSG FOR BIOP.
   - DISK CHECKS THE EDAL QUEUE, IF IT IS LESS THAN 3, BUILDS MORE EDALS UNTIL THERE ARE 3.

5. BIOP - INTERRUPT ON IOP TO IOP INPUT CHANNEL.
   - <u>ACOM</u> IS ACTIVATED BY IA.
   - <u>ACOM</u> ALLOCATES A LOCAL DAL AND READS IN EDAL FROM MOS.
   - <u>ACOM</u> ALLOCATES A LOCAL BUFFER AND MOVES DATA FROM MOS TO LOCAL MEMORY TO CENTRAL MEMORY.
   - <u>ACOM</u> DEALLOCATES LOCAL BUFFER, CHANGES PARCEL 1 IN EDAL, AND WRITES FIRST WORD OF EDAL OVER EDAL IN MOS.
   - <u>ACOM</u> DEALLOCATES LOCAL EDAL AND QUEUES AN ACCUMULATOR MSG FOR DIOP.

6. DIOP - INTERRUPT ON IOP TO IOP INPUT CHANNEL.
   - <u>ACOM</u> IS ACTIVATED BY IA.
   - <u>ACOM</u> ALLOCATES A LOCAL DAL AND READS IN MOS EDAL.
   - <u>ACOM</u> DEALLOCATES MOS BUFFER AND LOCAL AND MOS EDALS.

REPEAT STEPS 3 THROUGH 6 UNTIL ALL BUT LAST SECTOR TRANSFERRED.

7. DIOP (READ AHEAD)
   - INTERRUPT ON DISK CHANNEL.
   - <u>IA</u> DETECTS LAST SECTOR THIS REQUEST AND BEGINS READ AHEAD (IF NO OTHER EDALS) BY ALLOCATING A LOCAL BUFFER AND STARTING I/O.
   - <u>IA</u> MOVES LAST EDAL TO DONE QUEUE.
   - <u>IA</u> ACTIVATES THE DISK DEMON.

8. DIOP - <u>DISK</u> PROCESSES LAST EDAL AS PER STEP 4.

9. DIOP - INTERRUPT ON DISK CHANNEL.
   - <u>IA</u> DETECTS I/O AS READ AHEAD, ALLOCATES A LOCAL BUFFER AND STARTS NEXT READ AHEAD.
   - <u>IA</u> ACTIVATES THE DISK DEMON.

10. DIOP - <u>DISK</u> ALLOCATES A MOS BUFFER AND WRITES READ AHEAD DATA TO MOS.
   - <u>DISK</u> DEALLOCATES THE LOCAL BUFFER.

REPEAT STEPS 9 & 10 FOR 3 SECTORS OR UNTIL NEW REQUEST THIS CHANNEL.

11. DIOP - INTERRUPT ON IOP TO IOP INPUT CHANNEL FOR LAST EDAL (FROM BIOP).
- ACOM IS ACTIVATED BY IA.
- ACOM ALLOCATES A LOCAL DAL AND READS IN MOS EDAL.
- ACOM DEALLOCATES MOS BUFFER AND LOCAL AND MOS DAL.
- ACOM UPDATES LOCAL MDAL AND WRITES IT OVER MOS MDAL.
- ACOM DEALLOCATES LOCAL MDAL.
- ACOM QUEUES AN ACCUMULATOR MSG FOR MIOP.

12. MIOP - INTERRUPT ON IOP TO IOP INPUT CHANNEL
- IA ACTIVATES ACOM.
- ACOM ALLOCATES A LOCAL DAL AND READS IN MOS MDAL
- ACOM DEALLOCATES MOS MDAL.
- ACOM 'STRIPS' OFF FIRST 10_8 PARCELS FROM MDAL AND QUEUES 6 WORDS TO BE SENT TO CPU.
- WHEN CPU RECEIVES THE 6 WORDS THE LOCAL MDAL WILL BE DEALLOCATED.

DISK WRITE REQUEST VIA DIOP

1. MIOP - INTERRUPT ON LOW SPEED INPUT CHANNEL DUE TO TRANSFER OF 6 WORD I/O REQUEST PACKET.
   - CDEM IS ACTIVATED BY INTERRUPT ANSWERING (IA).
   - CDEM BUILDS A MASTER DISK ACTIVITY LINK (MDAL) BY PREFIXING 10_8 CONTROL PARCELS TO THE 6 WORD PACKET.
   - CDEM ALLOCATES A MOS DAL AND WRITES LOCAL MDAL TO MOS.
   - CDEM QUEUES ACCUMULATOR MESSAGE TO BE SENT TO DIOP (SPECIFYING ADDRESS OF DAL IN MOS).
   - CDEM DEALLOCATES LOCAL DAL.

2. DIOP - INTERRUPT ON IOP TO IOP INPUT CHANNEL.
   - ACOM IS ACTIVATED BY IA.
   - ACOM DECODES ACCUMULATOR MSG, ALLOCATES A LOCAL DAL, AND READS IN MDAL FROM MOS.
   - ACOM BUILDS FIRST, UP TO 3, EDALS, ALLOCATES MOS BUFFERS, AND QUEUES EDALS TO BE SENT TO BIOP.
   - ~~ACOM ACTIVATES DISK DEMON.~~

3. BIOP - INTERRUPT ON IOP TO IOP INPUT CHANNEL
   - ACOM IS ACTIVATED BY IA.
   - ACOM ALLOCATES A LOCAL DAL AND READS IN EDAL FROM MOS.
   - ACOM ALLOCATES A LOCAL BUFFER AND MOVES DATA FROM CENTRAL TO LOCAL MEMORY AND THEN TO MOS.
   - ACOM UPDATES LOCAL EDAL AND WRITES 1 WORD OVER MOS EDAL.
   - ACOM DEALLOCATES LOCAL BUFFER AND DAL AND QUEUES AN ACCUMULATOR MSG FOR DIOP.

4. DIOP - INTERRUPT ON IOP TO IOP INPUT CHANNEL.
   - ACOM IS ACTIVATED BY IA
   - ACOM ALLOCATES A LOCAL DAL AND READS IN EDAL FROM MOS.

- ACOM MATCHES 'NEW' EDAL WITH ORIGINAL EDAL AND DEALLOCATES 'NEW' EDAL.
- IF FIRST EDAL AND DISK NOT BUSY, ALLOCATE A LOCAL BUFFER; MOVE DATA TO LOCAL MEMORY; DEALLOCATE MOS BUFFER; AND START WRITE.
- ACOM PUTS EDAL ON DCB EDAL QUEUE.
- ACOM ACTIVATES DISK DEMON.

wtY2 0

5. DIOP
- INTERRUPT ON DISK CHANNEL
- IA STARTS NEXT WRITE IF DATA IS AVAILABLE.
- IA RETURNS LOCAL BUFFER.
- IA PUTS EDAL ON DONE QUEUE IN DCB.
- IA ACTIVATES DISK DEMON.

6. DIOP
- DISK TAKES TOP ENTRY OFF DONE QUEUE.
- DISK RETURNS LOCAL AND MOS EDAL AND MOS BUFFER, IF NECESSARY.
- DISK BUILDS MORE EDALS (SO THERE ARE 3), AND ALLOCATES MOS BUFFERS.
- DISK ALLOCATES MOS DAL(S) AND WRITES LOCAL EDAL(S) TO MOS.
- DISK QUEUES (AN) ACCUMULATOR MESSAGE(S) FOR BIOP.
- DISK ALLOCATES A LOCAL BUFFER FOR ANY EDALS ON EDAL QUEUE WHOSE DATA IS IN MOS AND READS THIS DATA INTO LOCAL.
- DISK DEALLOCATES THESE MOS BUFFERS.
- IF CHANNEL IS INACTIVE, AWAITING THIS DATA, DISK STARTS A WRITE TRANSFER.

REPEAT STEPS 3 TO 6 UNTIL ALL DATA TRANSFERRED.

7. DIOP
- ACOM DETECTS THE RETURN OF THE LAST EDAL, THIS REQUEST, FROM BIOP.
- ACOM ALLOCATES AN MOS DAL AND WRITES LOCAL MDAL ONTO IT.
- ACOM QUEUES AN ACCUMULATOR MSG FOR MIOP.
- ACOM DEALLOCATES 'NEW' LOCAL EDAL BUT NOT LOCAL MDAL.
- ACOM ACTIVATES DISK DEMON.

8. MIOP - INTERRUPT ON IOP TO IOP INPUT CHANNEL
- ACOM ACTIVATED BY IA.
- ACOM ALLOCATES A LOCAL DAL AND READS IN 'NEW' MDAL FROM MOS.
- ACOM CHANGES FUNCTION CODE IN 'NEW' MDAL TO 2 AND WRITES 1 WORD OVER 'NEW' MOS MDAL.
- ACOM QUEUES AN ACCUMULATOR MSG FOR DIOP. (ACOM IN DIOP WILL EVENTUALLY RELEASE 'NEW' MOS DAL).
- ACOM 'STRIPS' OFF FIRST 10_8 PARCELS FROM 'NEW' MDAL AND QUEUES 6 WORDS TO BE SENT TO CPU.
- WHEN CPU RECEIVES THE 6 WORDS THE 'NEW' LOCAL MDAL WILL BE DEALLOCATED.

9. DIOP - DISK DETECTS THE LAST EDAL, THIS REQUEST, ON DONE QUEUE.
- DISK DEALLOCATES LOCAL AND MOS EDAL AND MOS BUFFER, IF NECESSARY.
- DISK UPDATES LOCAL MDAL AND WRITES IT OVER ORIGINAL MOS MDAL.
- DISK DEALLOCATES LOCAL MDAL.
- DISK QUEUES AN ACCUMULATOR MSG FOR MIOP.

10. MIOP - INTERRUPT ON IOP TO IOP INPUT CHANNEL
- ACOM ACTIVATED BY IA.
- ACOM ALLOCATES A LOCAL DAL AND READS IN MDAL FROM MOS.
- ACOM DEALLOCATES MOS MDAL.
- ACOM 'STRIPS' OFF FIRST 10_8 PARCELS FROM MDAL AND QUEUES 6 WORDS TO BE SENT TO CPU.
- WHEN CPU RECEIVES THE 6 WORDS THE LOCAL MDAL WILL BE DEALLOCATED.

DISK READ REQUEST VIA BIOP

1. MIOP - INTERRUPT ON LOW SPEED INPUT CHANNEL DUE TO TRANSFER OF 6 WORD I/O REQUEST PACKET.
   - CDEM IS ACTIVATED BY INTERRUPT ANSWERING (IA).
   - CDEM BUILDS A MASTER DISK ACTIVITY LINK (MDAL) BY PREFIXING 10_8 CONTROL PARCELS TO THE 6 WORD PACKET.
   - CDEM ALLOCATES A MOS DAL AND WRITES LOCAL MDAL TO MOS.
   - CDEM QUEUES ACCUMULATOR MESSAGE TO BE SENT TO DIOP (SPECIFYING ADDRESS OF DAL IN MOS).
   - CDEM DEALLOCATES LOCAL DAL.

2. BIOP - INTERRUPT ON IOP TO IOP INPUT CHANNEL.
   - ACOM IS ACTIVATED BY IA.
   - ACOM DECODES ACCUMULATOR MSG, ALLOCATES A LOCAL DAL, AND READS IN MDAL FROM MOS.
   - ACOM BUILDS FIRST, UP TO 3, EXECUTABLE DALS (EDALS) AND ALLOCATES LOCAL BUFFERS.
   - ACOM STARTS FIRST READ (OR SEEK) IF DISK IS NOT ACTIVE, AND PUTS EDALS ON DISK CONTROL BLOCK (DCB) EDAL QUEUE.

3. BIOP - INTERRUPT ON DISK CHANNEL.
   - IA ALLOCATES A LOCAL BUFFER FOR NEXT TRANSFER, IF NECESSARY, AND STARTS NEXT SECTOR.
   - IA MOVES FINISHED EDAL TO DCB DONE QUEUE AND ACTIVATES THE DISK DEMON (DISK).

4. BIOP - DISK TAKES FIRST ENTRY OFF THE DONE QUEUE.
   - DISK MOVES DATA FROM LOCAL TO CENTRAL MEMORY.
   - DISK DEALLOCATES LOCAL BUFFER AND EDAL.
   - DISK ALLOCATES A LOCAL BUFFER FOR SECOND EDAL ON EDAL QUEUE.
   - DISK BUILDS MORE EDALS, IF NECESSARY.

REPEAT STEPS 3 & 4 UNTIL ALL BUT LAST SECTOR TRANSFERRED.

5. BIOP (READ AHEAD)
   - INTERRUPT ON DISK CHANNEL.
   - IA DETECTS LAST SECTOR THIS REQUEST AND BEGINS READ AHEAD (IF NO OTHER EDALS) BY ALLOCATING A LOCAL BUFFER AND STARTING I/O.
   - IA MOVES LAST EDAL TO DONE QUEUE.
   - IA ACTIVATES THE DISK DEMON.

6. BIOP
   - DISK PROCESSES LAST EDAL AS PER STEP 4.
   - DISK UPDATES LOCAL MDAL AND WRITES IT OVER ORIGINAL MOS MDAL.
   - DISK DEALLOCATES LOCAL MDAL AND QUEUES AN ACCUMULATOR MSG FOR MIOP.

7. MIOP
   - INTERRUPT ON IOP TO IOP INPUT CHANNEL.
   - ACOM IS ACTIVATED BY IA.
   - ACOM ALLOCATES A LOCAL DAL AND READS IN MOS MDAL.
   - ACOM DEALLOCATES MOS MDAL.
   - ACOM 'STRIPS' OFF FIRST 10_8 PARCELS FROM MDAL AND QUEUES 6 WORDS TO BE SENT TO CPU.
   - WHEN CPU RECEIVES THE 6 WORDS THE LOCAL MDAL WILL BE DEALLOCATED.

8. BIOP
   - INTERRUPT ON DISK CHANNEL
   - IA DETECTS I/O AS READ AHEAD, ALLOCATES A LOCAL BUFFER AND STARTS I/O.
   - IA ACTIVATES THE DISK DEMON.

9. BIOP
   - DISK ALLOCATES A MOS BUFFER AND WRITES READ AHEAD DATA TO MOS.
   - DISK DEALLOCATES THE LOCAL BUFFER.

REPEAT STEPS 8 & 9 FOR 3 SECTORS OR UNTIL NEW REQUEST THIS CHANNEL.

DISK WRITE REQUEST VIA BIOP

1. MIOP
   - INTERRUPT ON LOW SPEED INPUT CHANNEL DUE TO TRANSFER OF 6 WORD I/O REQUEST PACKET.
   - CDEM IS ACTIVATED BY INTERRUPT ANSWERING (IA).
   - CDEM BUILDS A MASTER DISK ACTIVITY LINK (MDAL) BY PREFIXING 10_8 CONTROL PARCELS TO THE 6 WORD PACKET.
   - CDEM ALLOCATES A MOS DAL AND WRITES LOCAL MDAL TO MOS.
   - CDEM QUEUES ACCUMULATOR MESSAGE TO BE SENT TO BIOP (SPECIFYING ADDRESS OF DAL IN MOS).
   - CDEM DEALLOCATES LOCAL DAL.

2. BIOP
   - INTERRUPT ON IOP TO IOP INPUT CHANNEL.
   - ACOM IS ACTIVATED BY IA.
   - ACOM DECODES ACCUMULATOR MSG, ALLOCATES A LOCAL DAL, AND READS IN MDAL FROM MOS.
   - ACOM BUILDS FIRST, UP TO 3, EDALS, AND PUTS THEM ON THE DISK CONTROL BLOCK (DCB) EDAL QUEUE.
   - ACOM ACTIVATES DISK DEMON.

3. BIOP
   - DISK LOOKS AT FIRST EDAL ON EDAL QUEUE AND ALLOCATES A LOCAL BUFFER.
   - DISK MOVES DATA FROM CENTRAL TO LOCAL MEMORY.
   - DISK STARTS WRITE (OR SEEK).
   - DISK LOOKS AT SECOND EDAL ON EDAL QUEUE.
   - DISK ALLOCATES A LOCAL BUFFER AND MOVES DATA FROM CENTRAL TO LOCAL MEMORY.

4. BIOP
   - INTERRUPT ON DISK CHANNEL.
   - IA STARTS WRITE FOR NEXT SECTOR.
   - IA DEALLOCATES LOCAL BUFFER.
   - IA MOVES FINISHED EDAL TO DCB DONE QUEUE.
   - IA ACTIVATES DISK DEMON.

5. BIOP - DISK GETS EDAL OFF DONE QUEUE.
  - DISK DEALLOCATES LOCAL EDAL.
  - IF LAST EDAL THIS REQUEST, UPDATE LOCAL MDAL; WRITE IT OVER ORIGINAL MDAL IN MOS; DEALLOCATE LOCAL MDAL; QUEUE AN ACCUMULATOR MSG FOR MIOP.
  - DISK BUILDS MORE EDALS, IF NECESSARY: ALLOCATE LOCAL BUFFERS; TRANSFER DATA FROM CENTRAL TO LOCAL MEMORY.

REPEAT STEPS 4 & 5 UNTIL ALL DATA TRANSFERRED.

6. MIOP - INTERRUPT ON IOP TO IOP INPUT CHANNEL.
  - ACOM IS ACTIVATED BY IA.
  - ACOM ALLOCATES A LOCAL DAL AND READS IN MOS MDAL.
  - ACOM DEALLOCATES MOS MDAL.
  - ACOM 'STRIPS' OFF FIRST 10_8 PARCELS FROM MDAL AND QUEUES 6 WORDS TO BE SENT TO CPU.
  - WHEN CPU RECEIVES THE 6 WORDS THE LOCAL MDAL WILL BE DEALLOCATED.

APPENDIX VIII

CONCENTRATOR TABLES AND DETAILED FLOW

CONCENTRATOR TABLES

KERNEL RESIDENT CONCENTRATOR TABLE (CT$CT)

LOCAL MEMORY CONCENTRATOR TABLE (CL)

CHANNEL EXTENSION TABLE (CXT)

STREAM DESCRIPTOR TABLE

KERNEL RESIDENT CONCENTRATOR TABLE

CONTAINS QUEUES, QUEUE ADDRESSES, TABLE POINTERS AND OTHER INFORMATION USED BY THE KERNEL TO MANAGE A CONCENTRATOR.

SIZE DEPENDS ON MAXIMUM NUMBER OF CONCENTRATORS ASSEMBLED.

LOCATED IN MIOP-ONLY SOFTWARE TABLE AREA OF KERNEL.

CONTAINS A 4 PARCEL ID ENTRY FOR EACH LOGICAL ID.

ADDRESS	PARCEL	
CT$CT	0	'ID'
CT$ID	1	MESSAGE CHANNEL ORDINAL FOR THIS ID (DEFERRED)
	2	LOGICAL ID
	3	DESCRIPTOR TABLE BUFFER MEMORY ADDRESS (UPPER)
	4	DESCRIPTOR TABLE BUFFER MEMORY ADDRESS (LOWER)
		⋮
	41	'MQ'
CT$MQ	42	POPCELL QUEUE ADDRESS FOR MSGIO IN BIOP
	43	'CQ'
CT$CQ	44	CONCENTRATOR 0 CONCI & CONCO SYNC QUEUE ADDRESS
	45	CONCENTRATOR 1 CONCI & CONCO SYNC QUEUE ADDRESS
	46	'MC'
CT$MC	47	CONCENTRATOR 0 MESSAGE COUNT
	50	CONCENTRATOR 1 MESSAGE COUNT
	51	'RQ'
CT$RQ	52	CONCENTRATOR 0 FEREAD QUEUE (FIRST)
	53	CONCENTRATOR 0 FEREAD QUEUE (LAST)
	54	CONCENTRATOR 1 FEREAD QUEUE (FIRST)
	55	CONCENTRATOR 1 FEREAD QUEUE (LAST)
	56	'WQ'
CT$WQ	57	CONCENTRATOR 0 FEWRIT QUEUE (FIRST)
	60	CONCENTRATOR 0 FEWRIT QUEUE (LAST)
	61	CONCENTRATOR 1 FEWRIT QUEUE (FIRST)
	62	CONCENTRATOR 1 FEWRIT QUEUE (LAST)
	63	'IC'
CT$ICH	64	CONCENTRATOR 0 PHYSICAL INPUT CHANNEL
	65	CONCENTRATOR 1 PHYSICAL INPUT CHANNEL
	66	'LM'
CT$LCL	67	CONCENTRATOR 0 LOCAL MEMORY TABLE
	70	CONCENTRATOR 1 LOCAL MEMORY TABLE
	71	'MO'
CT$MO	72	CONCENTRATOR 0 MESSAGE CHANNEL ORDINAL
	73	CONCENTRATOR 1 MESSAGE CHANNEL ORDINAL

FIGURE VIII-1. KERNEL RESIDENT CONCENTRATOR TABLE
WITH CONC$MX=2 AND ID$MXP=40.

LOCAL MEMORY CONCENTRATOR TABLE

CONTAINS LCP BUFFER, CXT, DSTB SCRATCH SPACE, CONCENTRATOR SYNC QUEUES, AND LCP BUFFER MEMORY ADDRESS.

ONE PER ACTIVE CONCENTRATOR

ALLOCATED FROM FREE MEMORY AT INITIALIZATION

EXISTS UNTIL CONCENTRATOR TERMINATES

254_8 PARCELS LONG

OFFSET	PARCEL	
CL@OLC	0	OUTPUT LCP
	27	
CL@ILC	30	INPUT LCP
	57	
CL@CXT	60	CHANNEL EXTENSION TABLE
	117	
CL@DSC	120	SYSTEM DESCRIPTOR TABLE MANIPULATION SPACE
	237	
CL@RQ	240	FEREAD-CONCI SYNC QUEUE (FIRST)
	241	FEREAD-CONCI SYNC QUEUE (LAST)
	242	# OF WORDS TO READ
	243	LOCAL ADDRESS TO STORE WORDS
CL@WQ	244	FEWRIT-CONCO SYNC QUEUE (FIRST)
	245	FEWRIT-CONCO SYNC QUEUE (LAST)
	246	# OF WORDS TO WRITE
	247	LOCAL ADDRESS TO WRITE FROM
CL@CQ	250	CONCI-CONCO SYNC QUEUE (FIRST)
	251	CONCI-CONCO SYNC QUEUE (LAST)
CL@LCO	252	BUFFER MEMORY ADDRESS OF LCP (UPPER)
	253	BUFFER MEMORY ADDRESS OF LCP (LOWER)

FIGURE VIII-2. LOCAL MEMORY CONCENTRATOR TABLE

CHANNEL EXTENSION TABLE

40_8 PARCELS LONG

USED TO OBTAIN CENTRAL MEMORY ADDRESSES FOR MESSAGES.

LAST 30_8 PARCELS PASSED TO CPU.

LOCATED IN LOCAL MEMORY CONCENTRATOR TABLE.

TREATED AS A DAL BY I/O SUBSYSTEM.

PARCEL	
0	LINK TO NEXT DAL IN CHAIN (0 IF LAST)
1	FUNCTION OF MESSAGE
2	BUFFER MEMORY ADDRESS OF DAL (UPPER)
3	BUFFER MEMORY ADDRESS OF DAL (LOWER)
4	ACTIVITY DESCRIPTOR OF SENDER (FOR RESPONSE)
5	ACCUMULATOR MESSAGE
6	POPCELL ADDRESS
7	UNUSED
10	SOURCE ID ('C1' OR 'B')
11	DESTINATION ID ('B' OR 'C1')
12	MESSAGE CHANNEL ORDINAL (8 BITS); MESSAGE COUNT (8 BITS)
13	UNUSED
14	BUFFER MEMORY OUTPUT LCP ADDRESS (UPPER)
15	BUFFER MEMORY OUTPUT LCP ADDRESS (LOWER)
16	BUFFER MEMORY INPUT LCP ADDRESS (UPPER)
17	BUFFER MEMORY INPUT LCP ADDRESS (LOWER)
20	RESEND FLAG (1 BIT); DSTB MOS ADDRESS (UPPER) (15 BITS)
21	DSTB MOS ADDRESS (LOWER)
22	STATUS (0 IF NO ERROR)
23	UNUSED
24	CENTRAL MEMORY OUTPUT LCP ADDRESS (UPPER)
25	CENTRAL MEMORY OUTPUT LCP ADDRESS (LOWER)
26	CENTRAL MEMORY INPUT LCP ADDRESS (UPPER)
27	CENTRAL MEMORY INPUT LCP ADDRESS (LOWER)
30	CENTRAL MEMORY OUTPUT SEGMENT ADDRESS (UPPER)
31	CENTRAL MEMORY OUTPUT SEGMENT ADDRESS (LOWER)
32	CENTRAL MEMORY INPUT SEGMENT ADDRESS (UPPER)
33	CENTRAL MEMORY INPUT SEGMENT ADDRESS (LOWER)
34	CENTRAL MEMORY OUTPUT LTP ADDRESS (UPPER)
35	CENTRAL MEMORY OUTPUT LTP ADDRESS (LOWER)
36	CENTRAL MEMORY INPUT LTP ADDRESS (UPPER)
37	CENTRAL MEMORY INPUT LTP ADDRESS (LOWER)

FIGURE VIII-3. CHANNEL EXTENSION TABLE

STREAM DESCRIPTOR TABLE

PROVIDES INFORMATION TO THE CONCENTRATOR CONCERNING STREAMS.

RESIDES IN BUFFER MEMORY

    ACCESSED BY MIOP AND BIOP

CONSISTS OF TWO PARTS:

- DESCRIPTOR TABLE (DSTB)
    - CONTAINS STREAM LIMITS AND POINTERS TO INDIVIDUAL STREAM DESCRIPTORS.
    - ONE PER LOGICAL ID
- INDIVIDUAL STREAM DESCRIPTOR
    - CONTAINS SUBSEGMENT AND MESSAGE BUFFER INFORMATION FOR A STREAM.

PARCEL	
0	UNUSED (8 BITS); CHECKSUM SIZE (CKZ) (8 BITS)
1	MAX. # INPUT STREAMS (MIS) (8 BITS); MAX. # OUTPUT STREAMS (MOS) (8 BITS)
2	MAX. # ACTIVE STREAMS (MAS) (8 BITS); MAX. # SUBSEGMENTS (MSS) (8 BITS)
3	SUBSEGMENT SIZE IN WORDS
4	LOGICAL ID
5	UNUSED
6	UNUSED
7	UNUSED
10	UNUSED
11	UNUSED
12	MOS ADDRESS OF INPUT STREAM 0 DESCRIPTOR (ISTO) (UPPER)
13	MOS ADDRESS OF INPUT STREAM 0 DESCRIPTOR (ISTO) (LOWER)
	.
	.
120	

FIGURE VIII-4. DESCRIPTOR TABLE

PARCEL	
0	# SUBSEGMENTS (NSS) (8 BITS); # BUFFERS THIS STREAM (ENT) (8 BITS)
1	OFFSET TO FIRST WORD USED IN BUFFER
2	BUFFER MEMORY DATA BIT COUNT (UPPER)
3	BUFFER MEMORY DATA BIT COUNT (LOWER)
4	NUMBER OF WORDS USED IN LAST BUFFER
5	UNUSED
6	UNUSED
7	UNUSED
10	UNUSED
11	UNUSED
12	MOS ADDRESS FOR MESSAGE BUFFER 0 (UPPER)
13	MOS ADDRESS FOR MESSAGE BUFFER 0 (LOWER)
	.
	.
N	

FIGURE VIII-5. STREAM DESCRIPTOR (1 PER STREAM)

<u>DETAILED MESSAGE FLOW</u>

0. <u>FEREAD</u> HAS PREVIOUSLY DONE A FRNTNDIO SERVICE REQUEST.

   THIS OPENS UP INPUT CHANNEL FROM FRONT-END.
   <u>FEREAD</u> IS ON KERNEL TABLE QUEUE (CT$RQ).
   <u>CONCI</u> IS ON CL@RQ.
   <u>CONCO</u> IS ON CL@CQ.
   <u>FEWRIT</u> IS ON CL@WQ.

1. FRONT-END SENDS AN LCP ACROSS THE INPUT CHANNEL. THIS GENERATES AN INTERRUPT.

2. INTERRUPT ANSWERING TAKES <u>FEREAD</u> OFF CT$RQ.

3. <u>FEREAD</u> DOUBLE SYNCS WITH <u>CONCI</u>. THIS ACTIVATES <u>CONCI</u> AND IDLES <u>FEREAD</u>.

4. <u>CONCI</u> VALIDATES LCP AND WRITES IT TO BUFFER MEMORY.

5. <u>CONCI</u> GETS A LOCAL MEMORY SUB-SEGMENT BUFFER AND ALL OF THE NECESSARY BUFFER MEMORY BUFFERS.

6. CONCI DOUBLES SYNCS WITH <u>FEREAD</u>.

7. <u>FEREAD</u> DOES A FRNTNDIO, INITIATING THE READ OF THE NEXT SUBSEGMENT, AND PLACING <u>FEREAD</u> ON CT$RQ.

8. WHEN THE SUB-SEGMENT (UP TO 512 WORDS) IS READ, INTERRUPT ANSWERING TAKES FEREAD OFF CT$RQ.

9. FEREAD DOUBLE SYNCS WITH CONCI.

10. CONCI WRITES THE SUB-SEGMENT TO BUFFER MEMORY.

11. GO TO 6 UNTIL ALL SUB-SEGMENTS ARE READ.

12. IF NO LTP, GO TO 18.

13. CONCI DOUBLE SYNCS WITH FEREAD.

14. FEREAD DOES A FRNTNDIO, INITIATING THE READ OF THE LTP, AND PLACING FEREAD ON CT$RQ.

15. WHEN THE LTP IS READ, INTERRUPT ANSWERING TAKES FEREAD OFF CT$RQ.

16. FEREAD DOUBLE SYNCS WITH CONCI.

17. CONCI CALLS CHKSMI WHICH COMPARES CHECKSUM AND LTP.

18. CONCI CALLS CRAYMSG.

19. CRAYMSG SENDS CXT TO CPU WITH RESEND FLAG SET.

20. CPU RETURNS CXT WITH CENTRAL MEMORY ADDRESS FOR INPUT MESSAGE.

21. CRAYMSG AWAKENS MSGIO IN BIOP.

22. MSGIO CALLS MSGIN WHICH MOVES MESSAGE TO CENTRAL MEMORY.

23. MSGIO RESPONDS TO CRAYMSG.

24. CRAYMSG SENDS CXT TO CPU WITH RESEND FLAG CLEAR.

25. CPU PROCESSES MESSAGE, BUILDS A RESPONSE, AND RETURNS CXT WITH CENTRAL MEMORY ADDRESS OF RESPONSE MESSAGE.

26. CRAYMSG AWAKENS MSGIO IN BIOP.

27. MSGIO CALLS MSGOUT WHICH MOVES RESPONSE MESSAGE TO BUFFER MEMORY.

28. MSGIO RESPONDS TO CRAYMSG.

29. CRAYMSG RETURNS TO CONCI.

30. CONCI DOUBLE SYNCS WITH CONCO VIA THE CL@CQ QUEUE.

31. CONCO READS RESPONSE LCP INTO MIOP LOCAL MEMORY FROM BUFFER MEMORY.

32. CONCO SYNCS WITH FEWRIT VIA CL@WQ.

33. CONCO SYNCS WITH CONCI. THIS IS TO ALLOW FRONT END TO SEND AN IMMEDIATE RESPONSE TO THE OUTPUT MESSAGE.
    A. CONCI DOUBLE SYNCS WITH FEREAD.
    B. FEREAD OPENS UP INPUT CHANNEL FOR NEXT LCP VIA FRNTNDIO. FEREAD GOES ON CT$RQ.

34. CONCO SYNCS WITH CL@WQ. THIS IDLES CONCO SINCE THIS QUEUE IS EMPTY.

35. FEWRIT INITIATES WRITE OF LCP TO FRONT END VIA FRNTNDIO. FEWRIT GOES ON CT$WQ.

36. WHEN LCP IS WRITTEN, INTERRUPT ANSWERING TAKES FEWRIT OFF CT$WQ.

37. FEWRIT DOUBLE SYNCS WITH CONCO.

38. CONCO READS A SUB-SEGMENT (UP TO 512 WORDS) INTO MIOP LOCAL MEMORY.

39. CONCO DOUBLE SYNCS WITH FEWRIT.

40. FEWRIT INITIATES WRITE OF SUB-SEGMENT TO FRONT END VIA FRNTNDIO. FEWRIT GOES ON CT$WQ.

41. WHEN SUB-SEGMENT IS WRITTEN, INTERRUPT ANSWERING TAKES FEWRIT OFF CT$WQ.

42. FEWRIT DOUBLE SYNCS WITH CONCO.

43. GO TO 38 UNTIL ALL SUB-SEGMENTS DONE.

44. IF NO LTP, GO TO 50.

45. CONCO CALLS CHKSMO WHICH BUILDS LTP.

46. CONCO DOUBLE SYNCS WITH FEWRIT.

47. FEWRIT INITIATES WRITE OF LTP TO FRONT END VIA FRNTNDIO. FEWRIT GOES ON CT$CW.

48. WHEN LTP IS WRITTEN, INTERRUPT ANSWERING TAKES FEWRIT OFF CT$WQ.

49. FEWRIT DOUBLE SYNCS WITH CONCO.

50. CONCO SYNCS WITH CONCI VIA CL@CQ, WHICH IS EMPTY, THUS, CONCO IS IDLED.

Cray Research, Inc.
Corporate Addresses

CORPORATE HEADQUARTERS

1440 Northland Drive
Mendota Heights, MN 55120
Tel: 612-452-6650
TLX 298444

≈

THE CHIPPEWA FACILITIES

Manufacturing:
Highway 178 North
Chippewa Falls, WI 54729
Tel: 715-723-2221
TWX 910285 1699

Engineering:
Highway 178 North
Chippewa Falls, WI 54729
Tel: 715-723-5501

CRAY LABORATORIES

Cray Labs Headquarters
5311 Western Avenue
Boulder, CO 80301
Tel: 303-449-3351

Hallie Lab
P.O. Box 169
Chippewa Falls, WI 54729
Tel: 715-723-0266

SALES OFFICES

Eastern Regional Sales Office
10750 Columbia Pike, Suite 602
Silver Spring, MD 20901
Tel: 301-681-9626

≈

Central Regional Sales Office
5330 Manhattan Circle, Suite F
Boulder, CO 80303
Tel: 303-499-3055

Houston Sales Office
3121 Buffalo Speedway, Suite 400
Houston, TX 77098
Tel: 713-877-8053

Austin Sales Office
3415 Greystone, Suite 201
Austin, TX 78731
Tel: 512-345-7034

≈

Western Regional Sales Office
Sunset Office Plaza
1874 Holmes Street
Livermore, CA 94550
Tel: 415-447-0201

Los Angeles Sales Office
101 Continental Boulevard, Suite 456
El Segundo, CA 90245
Tel: 213-640-2351

Seattle Sales Office
536A Medical Dental Building
2728 Colby Avenue
Everett, WA 98201
Tel: 206-259-5075

INTERNATIONAL (SUBSIDIARIES)

Cray Research (UK) Limited
James Glaisher House
Grenville Place
Bracknell, UK
Tel: 44-344-21515
TLX: 848841

Cray Research GmbH
Wartburgplatz 7
8000 Munich 40
West Germany
Tel: 49-89-3630-76
TLX: 05213211

Cray Research Japan, Limited
Shin Aoyama Building, West 1661
1-1 Minami-Aoyama 1-chome
Minato-ku, Tokyo 107 Japan
Tel: 81(03)403-3471

CRAY RESEARCH, INC.