EY-CD017-RK-002

TOPS-20 Monitor Internals

digital

DIGITAL

The material in this document is for informational purposes and is subject to change without notice; it should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such license. Digital Equipment Corporation assumes no responsibility for the use or reliability of its software on equipment that is not supplied by Digital or its affiliated companies.

The following are trademarks of Digital Equipment Corporation, Maynard, Massachusetts:

COMPUTER LABS	COMTEX	DBMS-10
DBMS-11	DBMS-20	DDT
DEC	DECCOMM	DECsystem-10
DECSYSTEM-20	DECtape	DECUS
DIBOL	DIGITAL	EDUSYSTEM
FLIPCHIP	FOCAL	INDAC
LAB-8	MASSBUS	OMNIBUS
OS/8	PDP	PHA
RSTS	RSX	TYPESET-8
TYPESET-10	TYPESET-11	TYPESET-20
UNIBUS	DECSYSTEM-2020	

TABLE OF CONTENTS

Monitor Program Logic Manual

This reflects version 3 of TOPS-2Ø.

DECSYSTEM-20 MONITOR FLOWCHARTS

I. Scheduler

II. Page Fault Handling

III. JSYS Calls - Device Independent Level

IV. JSYS Calls - Disk Dependent Level

V. JSYS Calls - Magtape Dependent Level

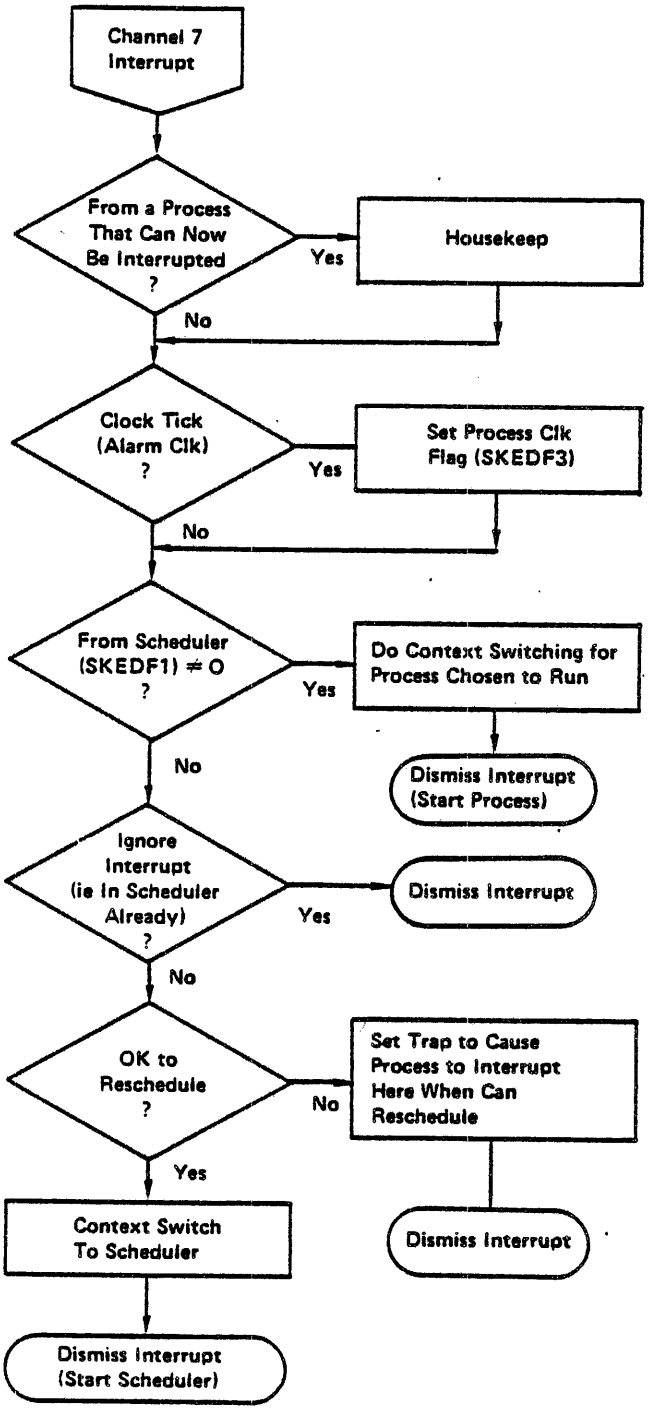
VI. Requesting DSK/MTA I/O & Interrupt Handling

VII. JSYS Calls - TTY Dependent Level

VIII. Requesting TTY I/O & Interrupt Handling

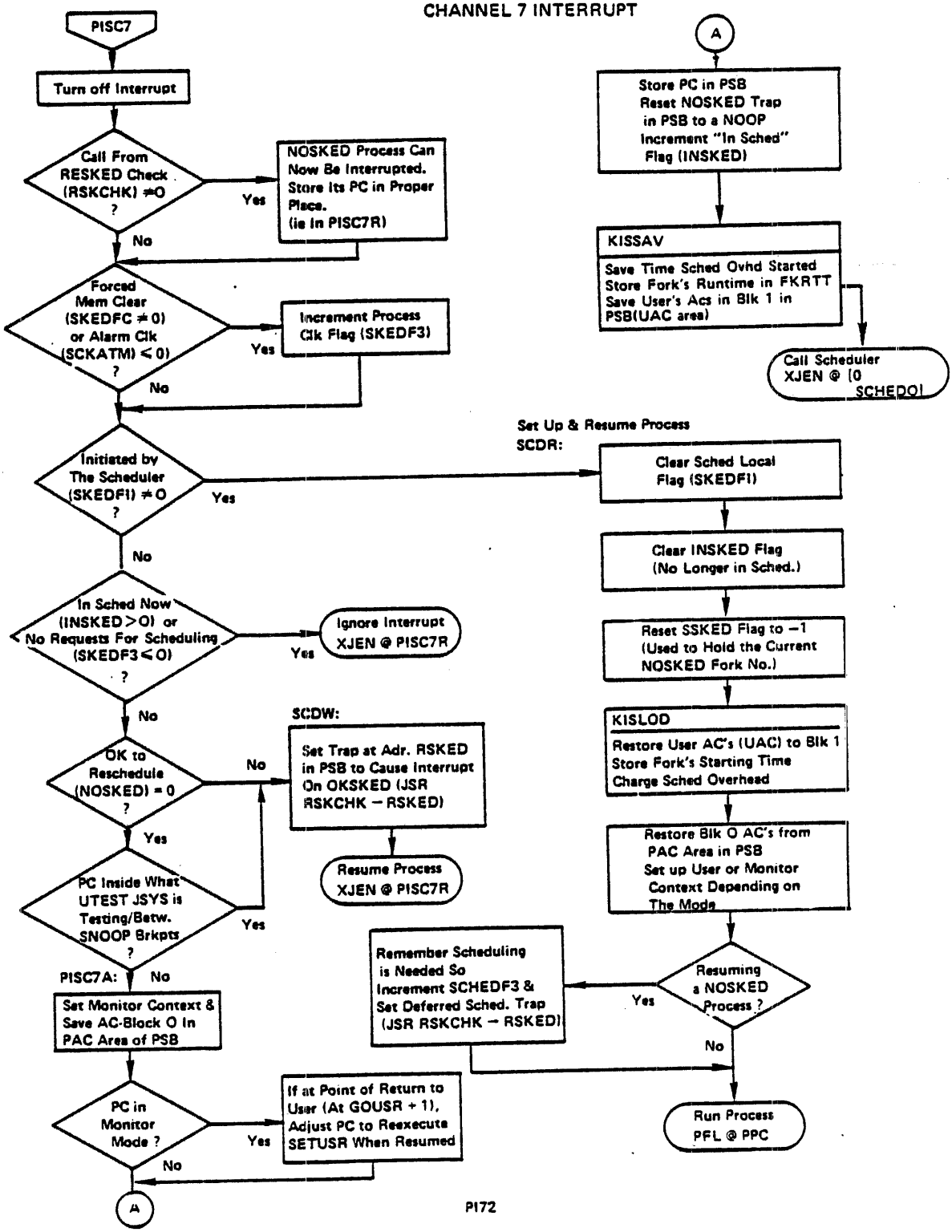
SCHEDULER FLOWCHARTS

CHANNEL 7 INTERRUPT
AN OVERVIEW

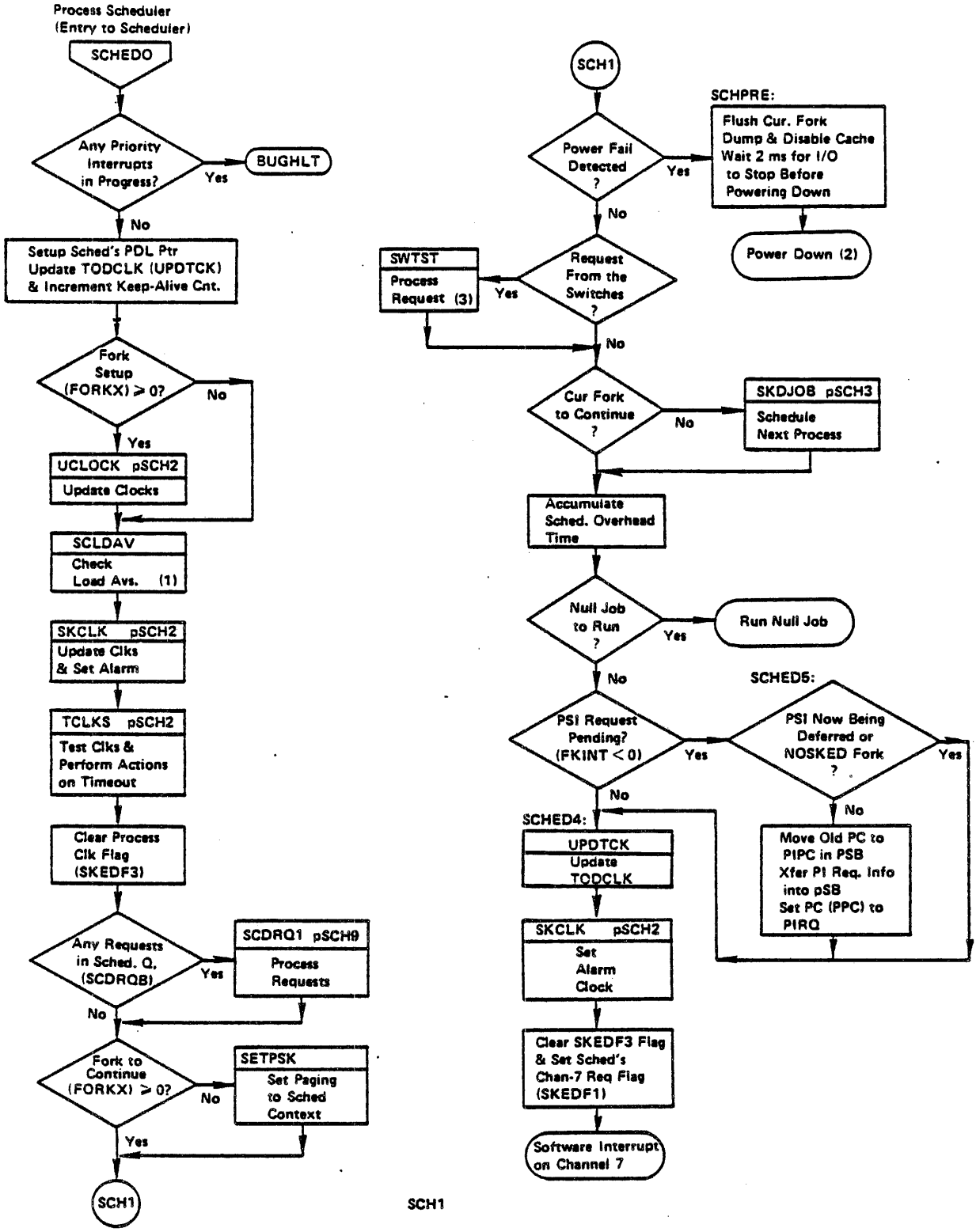


PI71

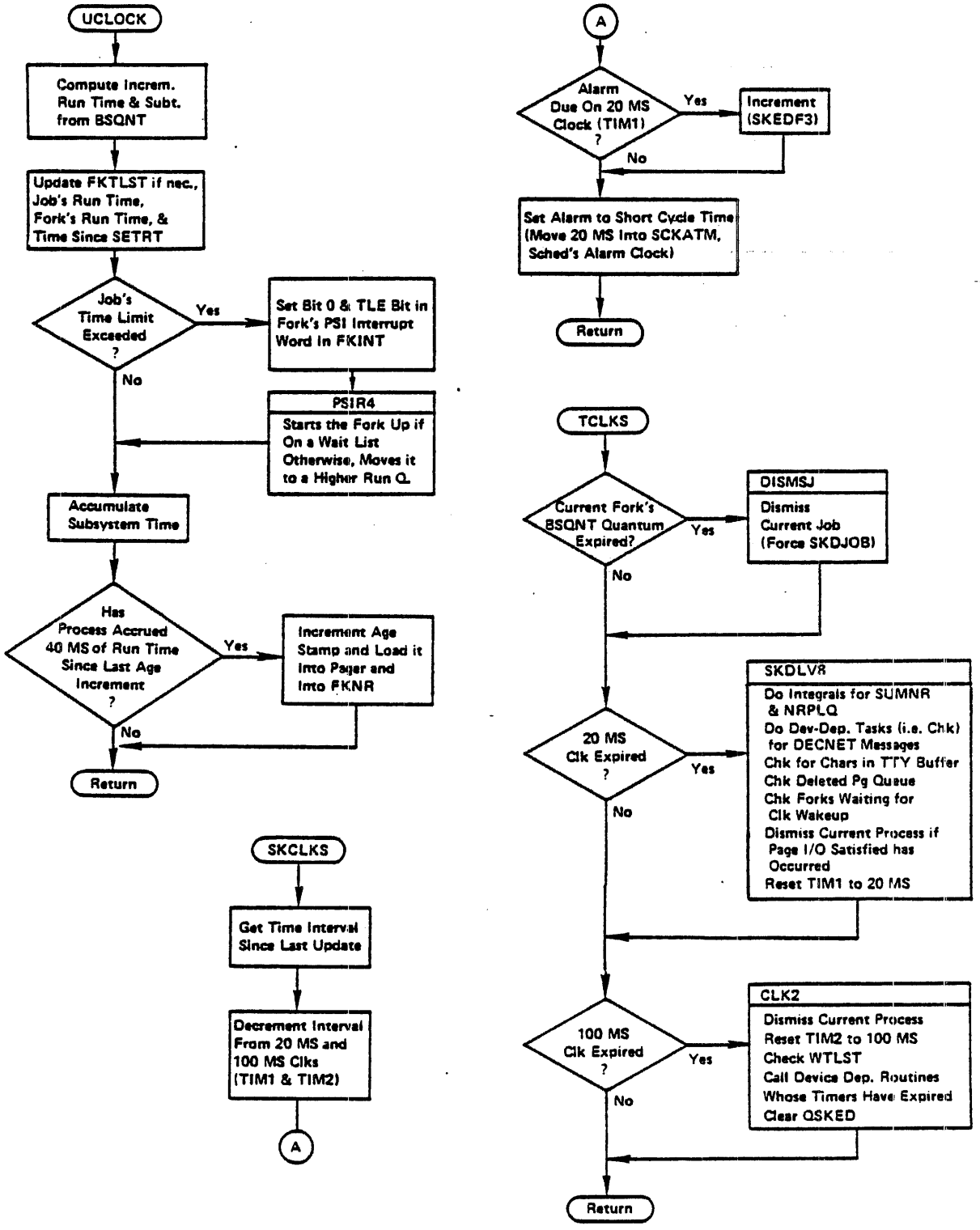
CHANNEL 7 INTERRUPT



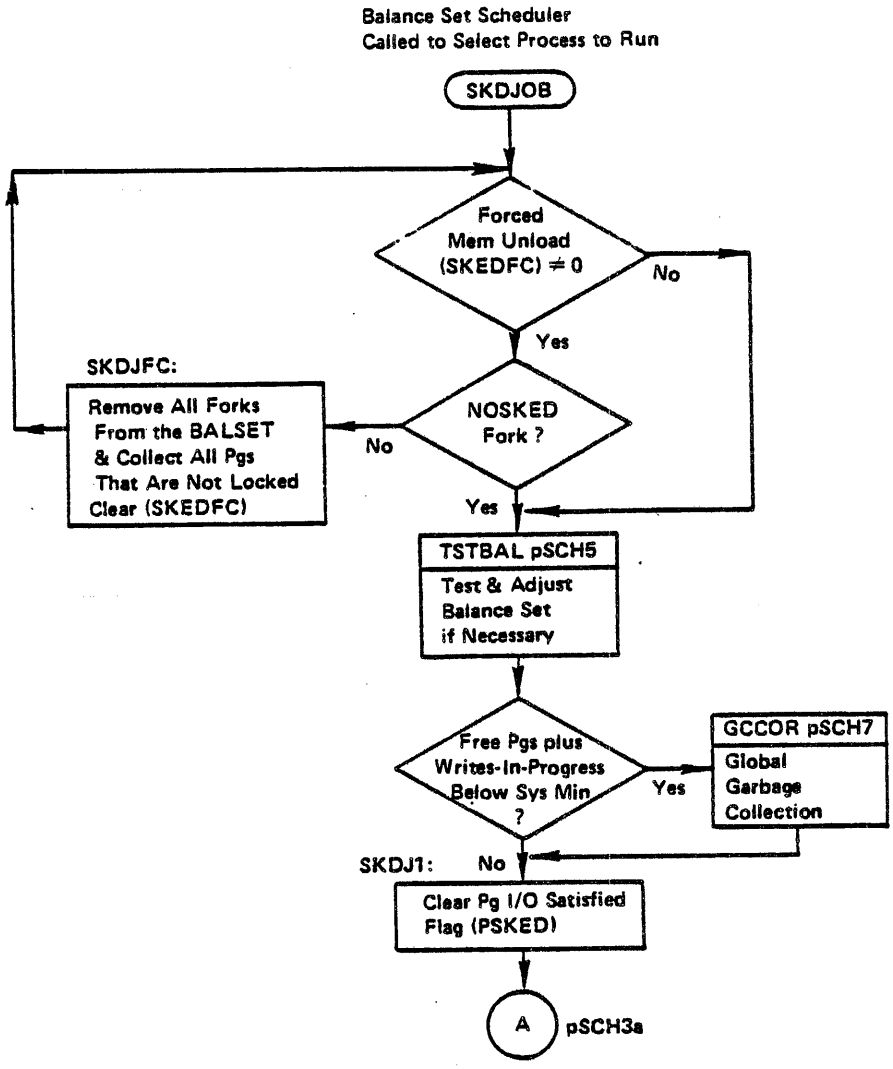
PI72


Process Scheduler
(Entry to Scheduler)
SCHEDO
Any Priority Interrupts in Progress?
Yes
BUGHLT
No
Setup Sched's PDL Ptr
Update TODCLK (UPDTCK)
& Increment Keep-Alive Cnt.
Fork Setup (FORKX) ≥ 0?
No
Yes
UCLOCK pSCH2
Update Clocks
SCLDAV
Check Load Avs. (1)
SKCLK pSCH2
Update Clks & Set Alarm
TCLKS pSCH2
Test Clks & Perform Actions on Timeout
Clear Process Clk Flag (SKEDF3)
Any Requests in Sched. Q. (SCDRQB)
Yes
SCDRQ1 pSCH9
Process Requests
No
Fork to Continue (FORKX) ≥ 0?
No
SETPSK
Set Paging to Sched Context
Yes
SCH1
SCH1
SCHPRE:
Power Fail Detected ?
Yes
Flush Cur. Fork
Dump & Disable Cache
Wait 2 ms for I/O
to Stop Before
Powering Down
Power Down (2)
No
SWTST
Process Request (3)
Yes
Request From the Switches ?
No
Cur Fork to Continue ?
No
SKDJOB pSCH3
Schedule Next Process
Accumulate Sched. Overhead Time
Null Job to Run ?
Yes
Run Null Job
No
SCHED5:
PSI Request Pending? (FKINT < 0)
Yes
PSI Now Being Deferred or NOSKED Fork ?
Yes
No
No
SCHED4:
Move Old PC to PIPC in PSB
Xfer PI Req. Info into pSB
Set PC (PPC) to PIRQ
UPDTCK
Update TODCLK
SKCLK pSCH2
Set Alarm Clock
Clear SKEDF3 Flag & Set Sched's Chan-7 Req Flag (SKEDF1)
Software Interrupt on Channel 7

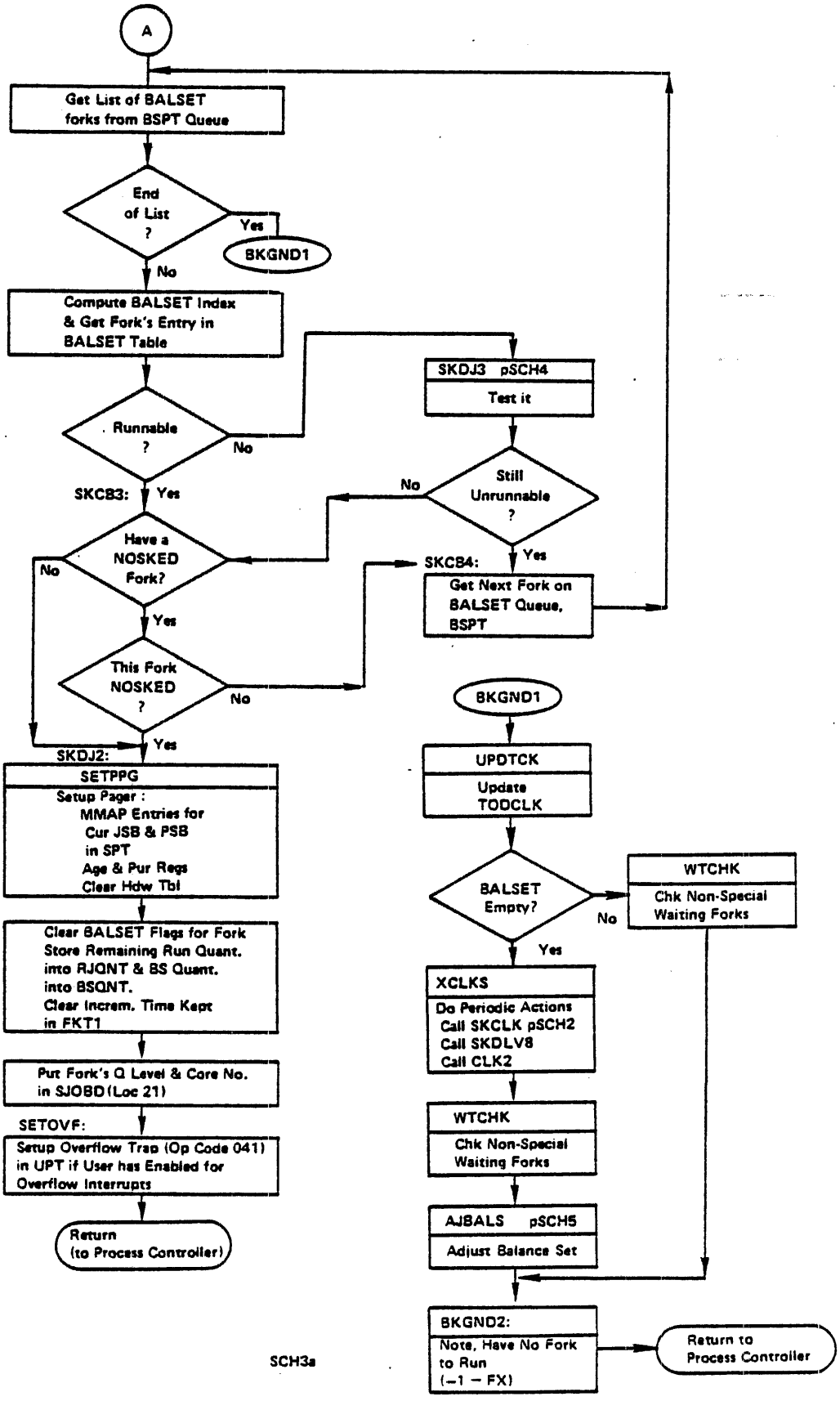
SCH1


UCLOCK
Compute Increm. Run Time & Subt. from BSQNT
Update FKTLST if nec., Job's Run Time, Fork's Run Time, & Time Since SETRT
Job's Time Limit Exceeded ?
Yes
Set Bit 0 & TLE Bit in Fork's PSI Interrupt Word In FKINT
No
PSIR4
Starts the Fork Up if On a Wait List Otherwise, Moves it to a Higher Run Q.
Accumulate Subsystem Time
Has Process Accrued 40 MS of Run Time Since Last Age Increment ?
Yes
Increment Age Stamp and Load it Into Pager and Into FKNR
No
Return
SKCLKS
Get Time Interval Since Last Update
Decrement Interval From 20 MS and 100 MS Clks (TIM1 & TIM2)
A
A
Alarm Due On 20 MS Clock (TIM1) ?
Yes
Increment (SKEDF3)
No
Set Alarm to Short Cycle Time (Move 20 MS Into SCKATM, Sched's Alarm Clock)
Return
TCLKS
Current Fork's BSQNT Quantum Expired?
Yes
DISMSJ
Dismiss Current Job (Force SKDJOB)
No
20 MS Clk Expired ?
Yes
SKDLVR
Do Integrals for SUMNR & NRPLQ
Do Dev-Dep. Tasks (i.e. Chk) for DECNET Messages
Chk for Chars in TTY Buffer
Chk Deleted Pg Queue
Chk Forks Waiting for Clk Wakeup
Dismiss Current Process if Page I/O Satisfied has Occurred
Reset TIM1 to 20 MS
No
100 MS Clk Expired ?
Yes
CLK2
Dismiss Current Process
Reset TIM2 to 100 MS
Check WTLST
Call Device Dep. Routines Whose Timers Have Expired
Clear QSKED
No
Return

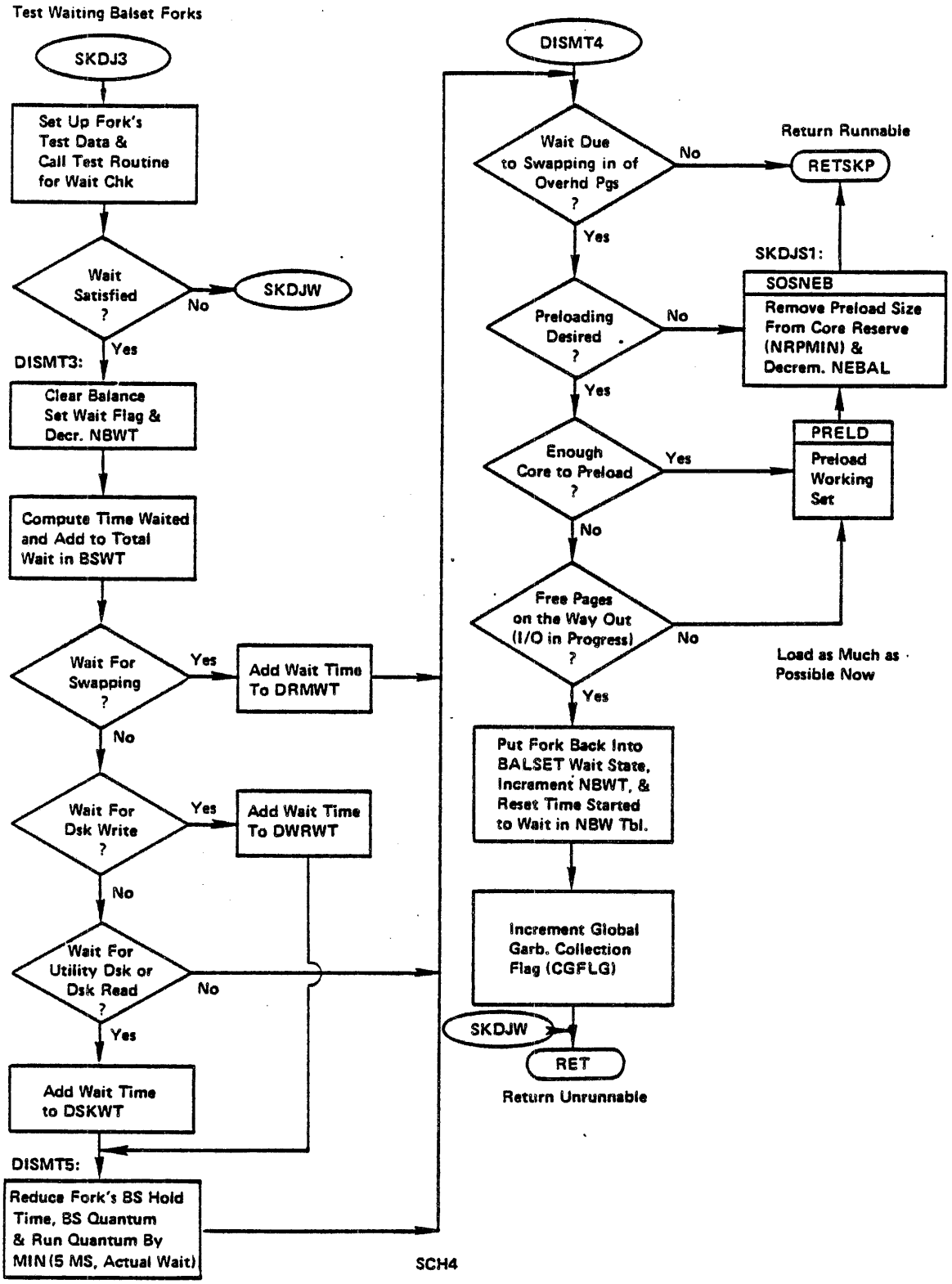
SCH2

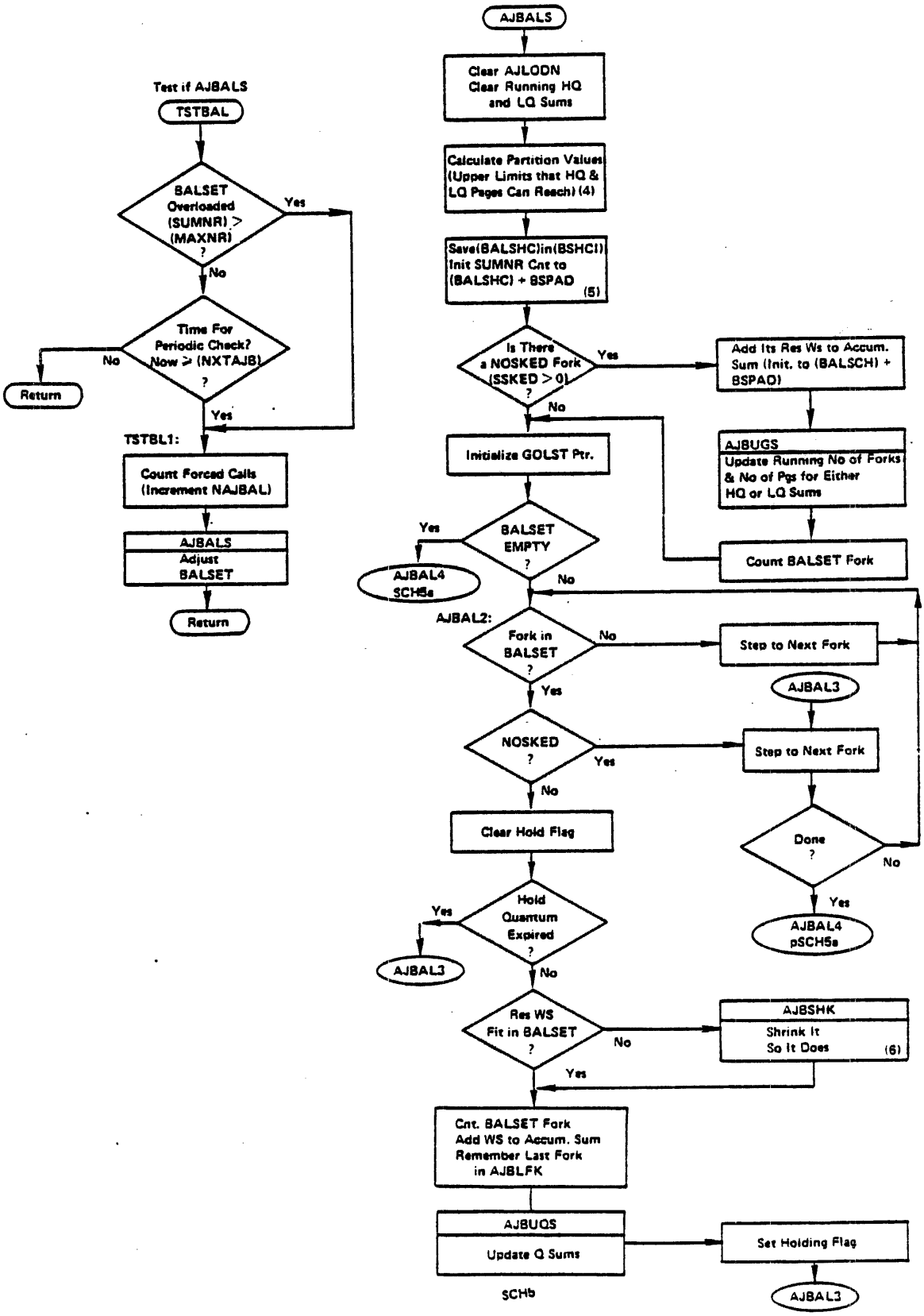

Balance Set Scheduler
Called to Select Process to Run
SKDJOB
Forced
Mem Unload
(SKEDFC) ≠ 0
No
Yes
SKDJFC:
Remove All Forks
From the BALSET
& Collect All Pgs
That Are Not Locked
Clear (SKEDFC)
NOSKED
Fork ?
No
Yes
TSTBAL pSCH5
Test & Adjust
Balance Set
if Necessary
Free Pgs plus
Writes-In-Progress
Below Sys Min
?
Yes
GCCOR pSCH7
Global
Garbage
Collection
SKDJ1:
No
Clear Pg I/O Satisfied
Flag (PSKED)
A
pSCH3a

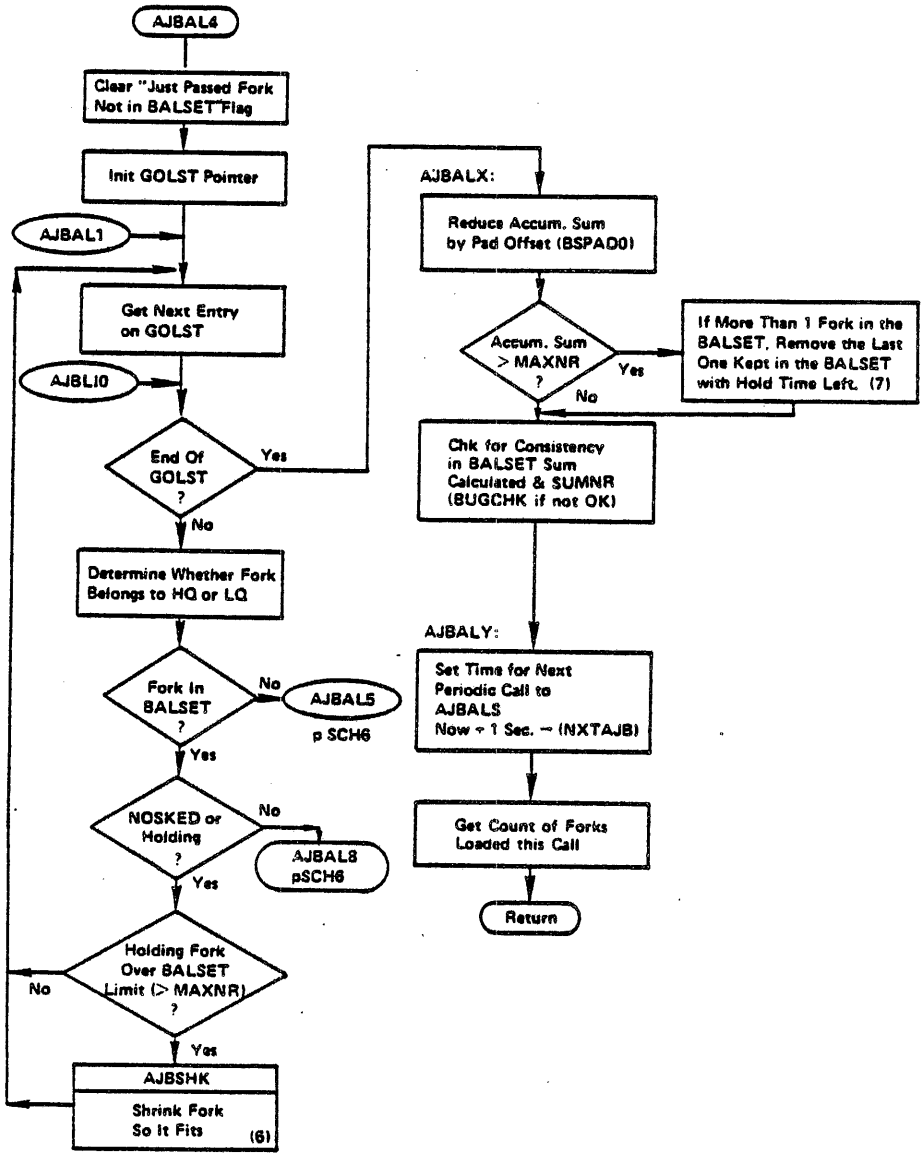
SCH3


A
Get List of BALSET forks from BSPT Queue
End of List ?
Yes
BKGND1
No
Compute BALSET Index & Get Fork's Entry in BALSET Table
Runnable ?
No
SKDJ3 pSCH4
Test it
Still Unrunnable ?
No
Yes
SKCB3:
Yes
Have a NOSKED Fork?
No
Yes
SKCB4:
Get Next Fork on BALSET Queue, BSPT
This Fork NOSKED ?
No
Yes
SKDJ2:
SETPPG
Setup Pager :
MMAP Entries for
Cur JSB & PSB
in SPT
Age & Pur Regs
Clear Hdw Tbl
Clear BALSET Flags for Fork
Store Remaining Run Quant.
into RJQNT & BS Quant.
into BSQNT.
Clear Increm. Time Kept
in FKT1
Put Fork's Q Level & Core No.
in SJOBD (Loc 21)
SETOVF:
Setup Overflow Trap (Op Code 041)
in UPT if User has Enabled for
Overflow Interrupts
Return
(to Process Controller)
BKGND1
UPDTCK
Update
TODCLK
BALSET Empty?
No
WTCHK
Chk Non-Special
Waiting Forks
Yes
XCLKS
Do Periodic Actions
Call SKCLK pSCH2
Call SKDLV8
Call CLK2
WTCHK
Chk Non-Special
Waiting Forks
AJBALS pSCH5
Adjust Balance Set
BKGND2:
Note, Have No Fork
to Run
(−1 − FX)
Return to
Process Controller

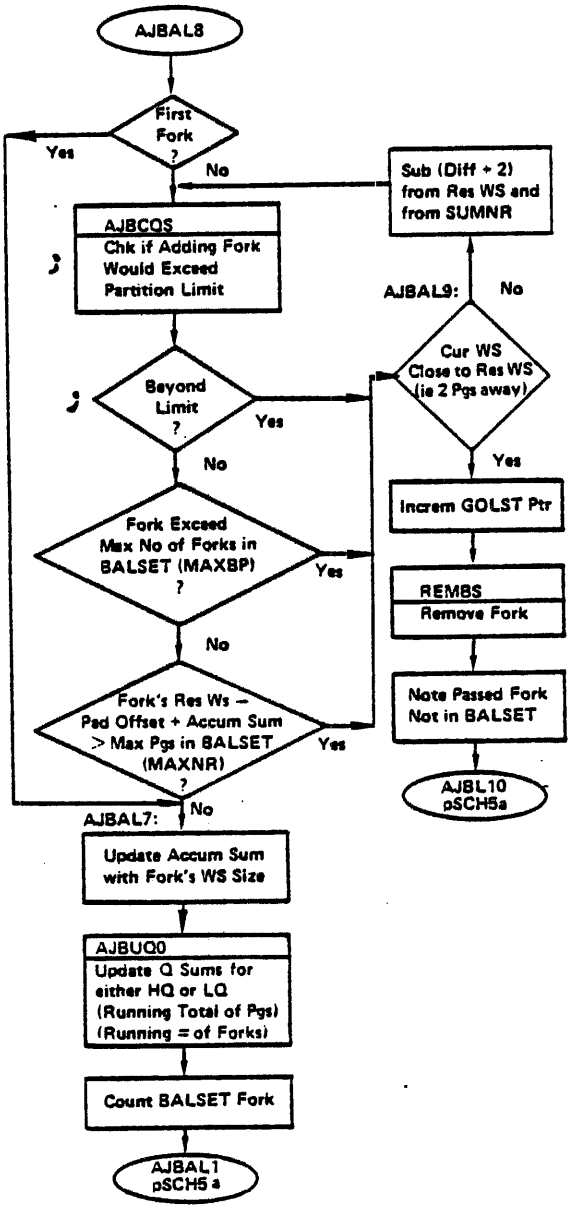
SCH3a

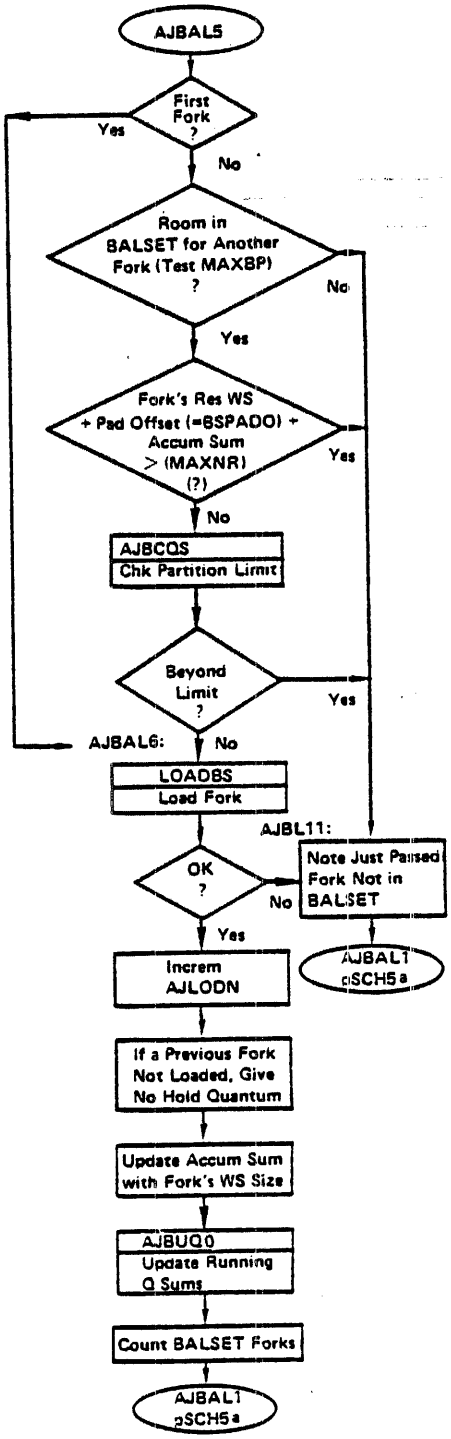

Test Waiting Balset Forks
SKDJ3
Set Up Fork's Test Data & Call Test Routine for Wait Chk
Wait Satisfied ?
No
SKDJW
Yes
DISMT3:
Clear Balance Set Wait Flag & Decr. NBWT
Compute Time Waited and Add to Total Wait in BSWT
Wait For Swapping ?
Yes
Add Wait Time To DRMWT
No
Wait For Dsk Write ?
Yes
Add Wait Time To DWRWT
No
Wait For Utility Dsk or Dsk Read ?
No
Yes
Add Wait Time to DSKWT
DISMT5:
Reduce Fork's BS Hold Time, BS Quantum & Run Quantum By MIN(5 MS, Actual Wait)
SCH4
DISMT4
Wait Due to Swapping in of Overhd Pgs ?
No
Return Runnable
RETSKP
Yes
SKDJS1:
SOSNEB
Remove Preload Size From Core Reserve (NRPMIN) & Decrem. NEBAL
Preloading Desired ?
No
Yes
PRELD
Preload Working Set
Enough Core to Preload ?
Yes
No
Free Pages on the Way Out (I/O in Progress) ?
No
Load as Much as Possible Now
Yes
Put Fork Back Into BALSET Wait State, Increment NBWT, & Reset Time Started to Wait in NBW Tbl.
Increment Global Garb. Collection Flag (CGFLG)
SKDJW
RET
Return Unrunnable


Test if AJBALS
TSTBAL
BALSET Overloaded (SUMNR) > (MAXNR) ?
Yes
No
Time For Periodic Check? Now ≥ (NXTAJB) ?
No
Return
Yes
TSTBL1:
Count Forced Calls (Increment NAJBAL)
AJBALS
Adjust BALSET
Return
AJBALS
Clear AJLODN
Clear Running HQ and LQ Sums
Calculate Partition Values (Upper Limits that HQ & LQ Pages Can Reach) (4)
Save(BALSHC)in(BSHCI) Init SUMNR Cnt to (BALSHC) + BSPAD (5)
Is There a NOSKED Fork (SSKED > 0) ?
Yes
Add Its Res Ws to Accum. Sum (Init. to (BALSCH) + BSPAD)
No
Initialize GOLST Ptr.
AJBUGS
Update Running No of Forks & No of Pgs for Either HQ or LQ Sums
BALSET EMPTY ?
Yes
AJBAL4 SCH5a
Count BALSET Fork
No
AJBAL2:
Fork in BALSET ?
No
Step to Next Fork
AJBAL3
Yes
NOSKED ?
Yes
Step to Next Fork
No
Clear Hold Flag
Done ?
No
Yes
AJBAL4 pSCH5a
Hold Quantum Expired ?
Yes
AJBAL3
No
Res WS Fit in BALSET ?
No
AJBSHK
Shrink It So It Does (6)
Yes
Cnt. BALSET Fork
Add WS to Accum. Sum
Remember Last Fork in AJBLFK
AJBUQS
Update Q Sums
Set Holding Flag
AJBAL3
SCH5b

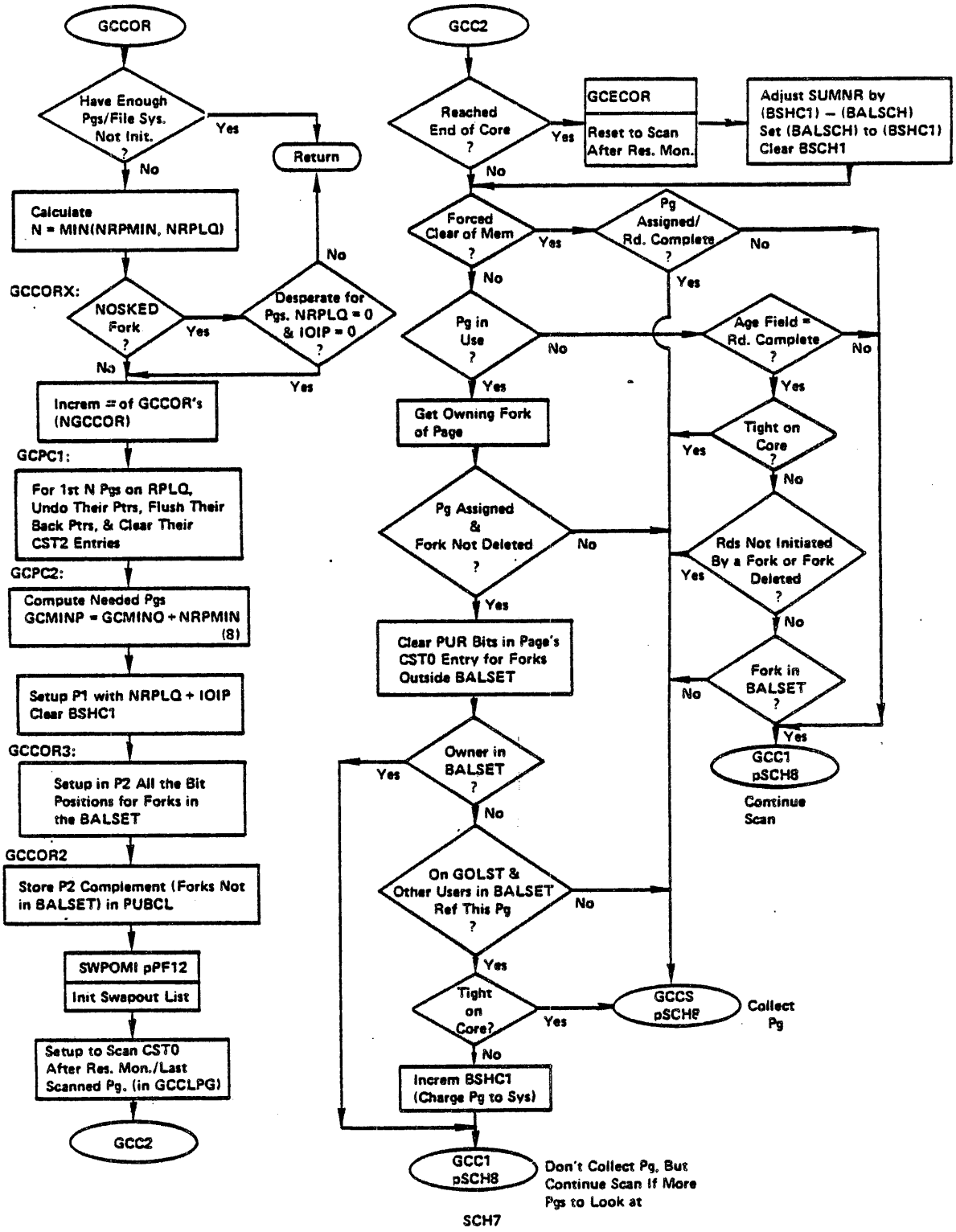

AJBAL4
Clear "Just Passed Fork Not in BALSET" Flag
Init GOLST Pointer
AJBAL1
Get Next Entry on GOLST
AJBLI0
End Of GOLST ?
Yes
No
Determine Whether Fork Belongs to HQ or LQ
Fork In BALSET ?
No
AJBAL5
p SCH6
Yes
NOSKED or Holding ?
No
AJBAL8 pSCH6
Yes
Holding Fork Over BALSET Limit (> MAXNR) ?
No
Yes
AJBSHK
Shrink Fork So It Fits (6)
AJBALX:
Reduce Accum. Sum by Pad Offset (BSPAD0)
Accum. Sum > MAXNR ?
Yes
If More Than 1 Fork in the BALSET, Remove the Last One Kept in the BALSET with Hold Time Left. (7)
No
Chk for Consistency in BALSET Sum Calculated & SUMNR (BUGCHK if not OK)
AJBALY:
Set Time for Next Periodic Call to AJBALS Now + 1 Sec. → (NXTAJB)
Get Count of Forks Loaded this Call
Return

SCH5a

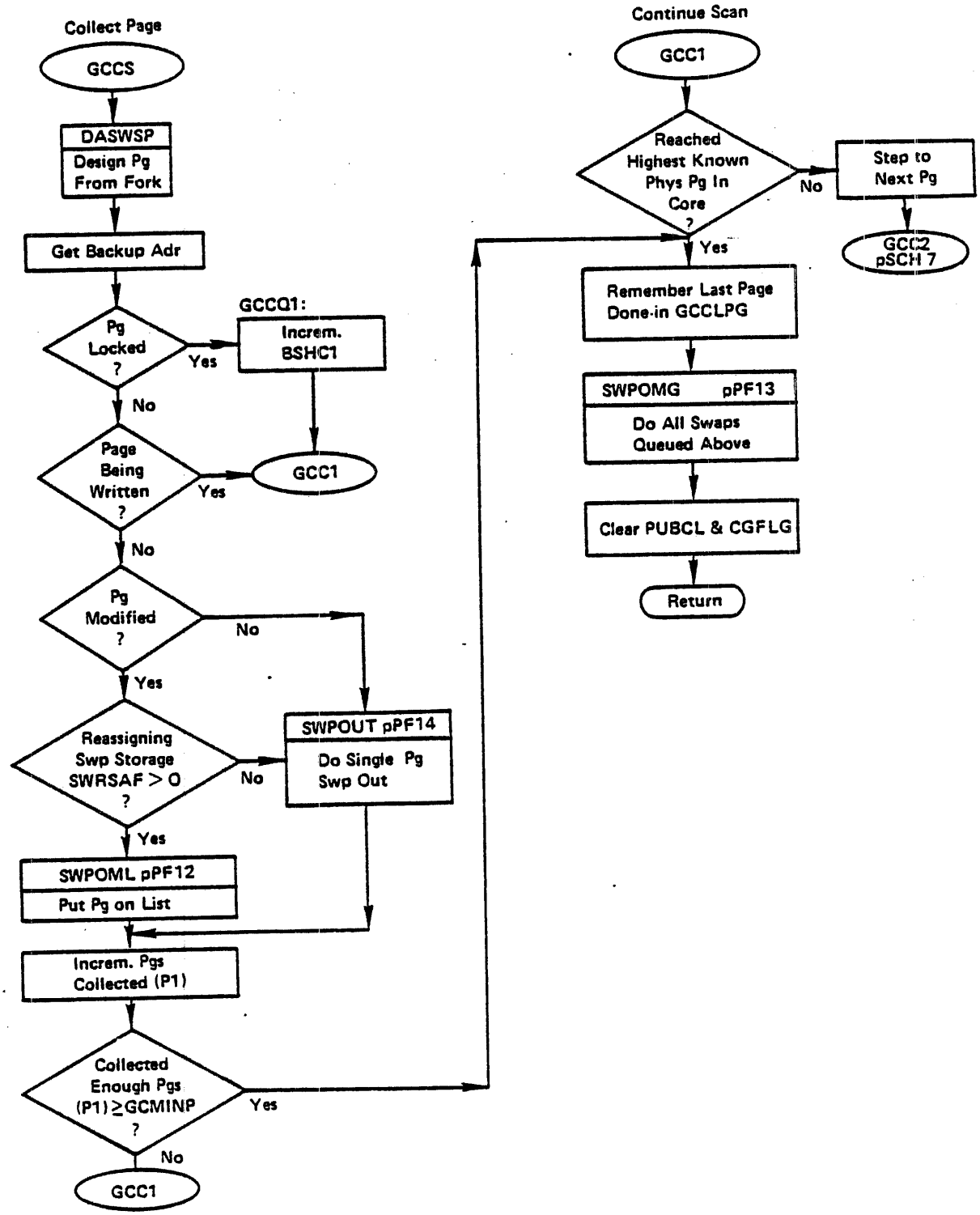

AJBAL8
First Fork ?
Yes
No
AJBCQS
Chk if Adding Fork Would Exceed Partition Limit
Beyond Limit ?
Yes
No
Fork Exceed Max No of Forks in BALSET (MAXBP) ?
Yes
No
Fork's Res Ws – Psd Offset + Accum Sum > Max Pgs in BALSET (MAXNR) ?
Yes
No
AJBAL7:
Update Accum Sum with Fork's WS Size
AJBUQ0
Update Q Sums for either HQ or LQ (Running Total of Pgs) (Running = of Forks)
Count BALSET Fork
AJBAL1 pSCH5a
Sub (Diff ÷ 2) from Res WS and from SUMNR
AJBAL9:
No
Cur WS Close to Res WS (ie 2 Pgs away)
Yes
Increm GOLST Ptr
REMBS
Remove Fork
Note Passed Fork Not in BALSET
AJBL10 pSCH5a


AJBAL5
First Fork ?
Yes
No
Room in BALSET for Another Fork (Test MAXBP) ?
No
Yes
Fork's Res WS + Psd Offset (=BSPADO) + Accum Sum > (MAXNR) (?)
Yes
No
AJBCQS
Chk Partition Limit
Beyond Limit ?
Yes
AJBAL6:
No
LOADBS
Load Fork
AJBL11:
OK ?
No
Note Just Paused Fork Not in BALSET
AJBAL1 pSCH5a
Yes
Increm AJLODN
If a Previous Fork Not Loaded, Give No Hold Quantum
Update Accum Sum with Fork's WS Size
AJBUQ0
Update Running Q Sums
Count BALSET Forks
AJBAL1 pSCH5a

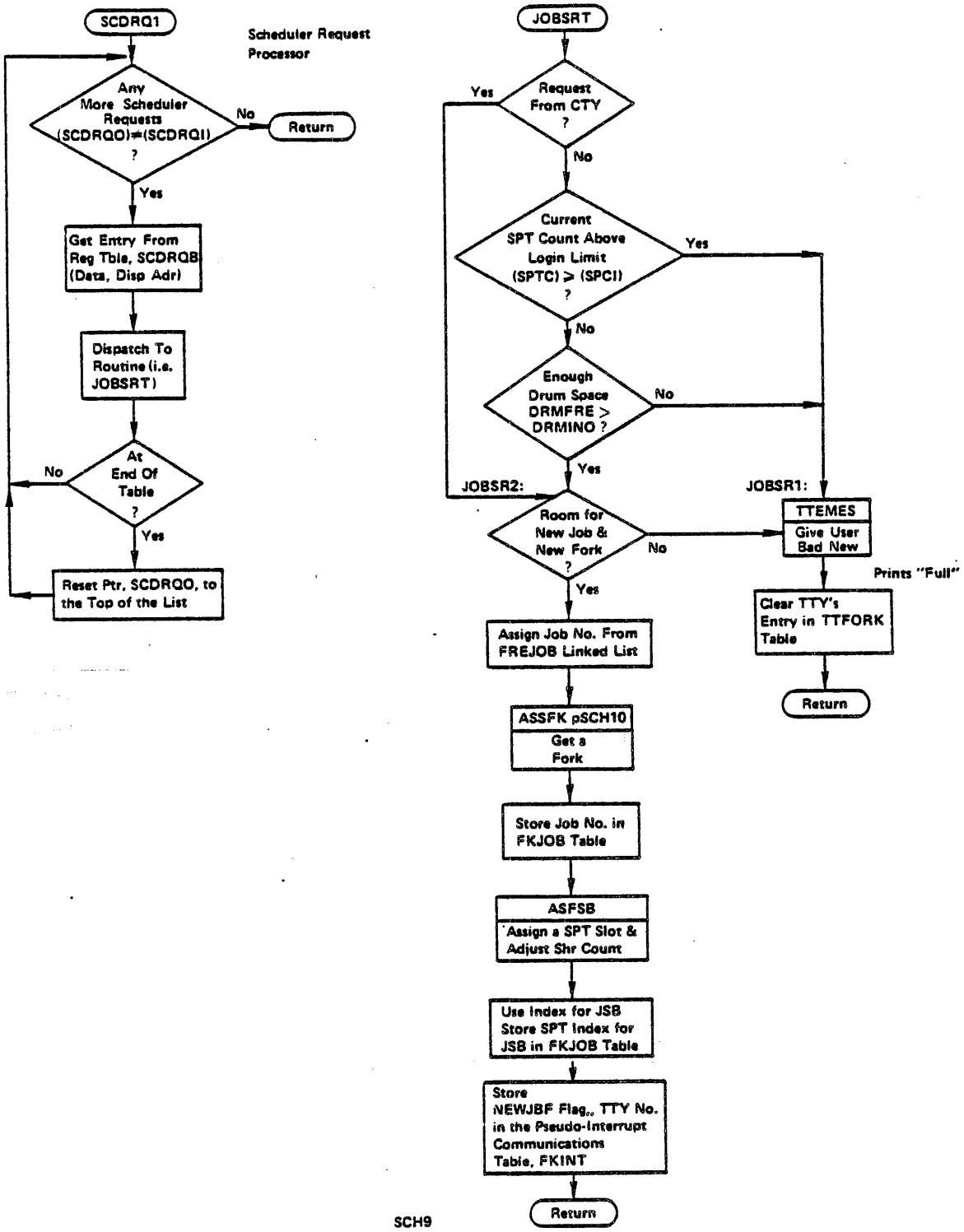
SCH6


GCCOR
Have Enough Pgs/File Sys. Not Init. ?
Yes
No
Return
Calculate N = MIN(NRPMIN, NRPLQ)
GCCORX:
NOSKED Fork ?
Yes
No
Desperate for Pgs. NRPLQ = 0 & IOIP = 0 ?
No
Yes
Increm # of GCCOR's (NGCCOR)
GCPC1:
For 1st N Pgs on RPLQ, Undo Their Ptrs, Flush Their Back Ptrs, & Clear Their CST2 Entries
GCPC2:
Compute Needed Pgs GCMINP = GCMINO + NRPMIN (8)
Setup P1 with NRPLQ + IOIP Clear BSHC1
GCCOR3:
Setup in P2 All the Bit Positions for Forks in the BALSET
GCCOR2
Store P2 Complement (Forks Not in BALSET) in PUBCL
SWPOMI pPF12
Init Swapout List
Setup to Scan CST0 After Res. Mon./Last Scanned Pg. (in GCCLPG)
GCC2
GCC2
Reached End of Core ?
Yes
GCECOR
Reset to Scan After Res. Mon.
Adjust SUMNR by (BSHC1) – (BALSCH) Set (BALSCH) to (BSHC1) Clear BSCH1
No
Forced Clear of Mem ?
Yes
No
Pg Assigned/ Rd. Complete ?
No
Yes
Pg in Use ?
No
Yes
Age Field = Rd. Complete ?
No
Yes
Get Owning Fork of Page
Tight on Core ?
Yes
No
Pg Assigned & Fork Not Deleted ?
No
Yes
Rds Not Initiated By a Fork or Fork Deleted ?
Yes
No
Clear PUR Bits in Page's CST0 Entry for Forks Outside BALSET
Fork in BALSET ?
No
Yes
GCC1 pSCH8
Continue Scan
Owner in BALSET ?
Yes
No
On GOLST & Other Users in BALSET Ref This Pg ?
No
Yes
Tight on Core?
Yes
GCCS pSCH8
Collect Pg
No
Increm BSHC1 (Charge Pg to Sys)
GCC1 pSCH8
Don't Collect Pg, But Continue Scan If More Pgs to Look at

SCH7

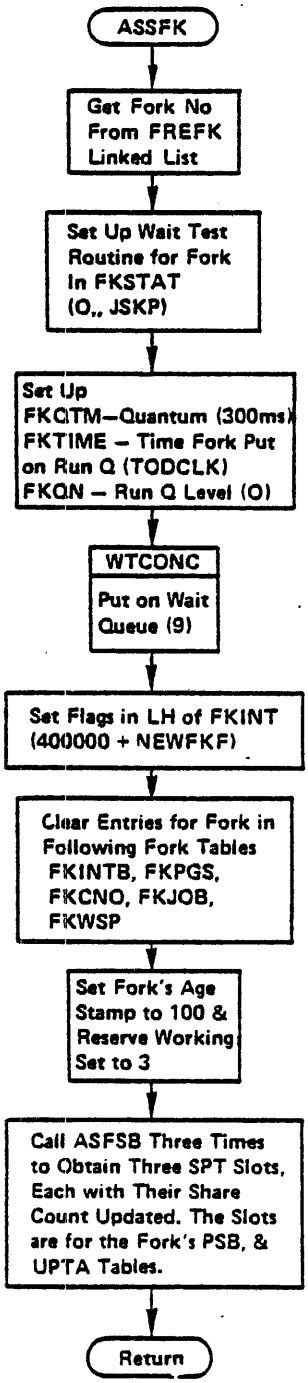

Collect Page
GCCS
DASWSP
Design Pg From Fork
Get Backup Adr
GCCQ1:
Pg Locked ?
Yes
Increm. BSHC1
No
Page Being Written ?
Yes
GCC1
No
Pg Modified ?
No
Yes
Reassigning Swp Storage SWRSAF > 0 ?
No
SWPOUT pPF14
Do Single Pg Swp Out
Yes
SWPOML pPF12
Put Pg on List
Increm. Pgs Collected (P1)
Collected Enough Pgs (P1) ≥ GCMINP ?
Yes
No
GCC1
Continue Scan
GCC1
Reached Highest Known Phys Pg In Core ?
No
Step to Next Pg
GCC2 pSCH 7
Yes
Remember Last Page Done-in GCCLPG
SWPOMG pPF13
Do All Swaps Queued Above
Clear PUBCL & CGFLG
Return

SCH8


SCDRQ1
Scheduler Request Processor
Any More Scheduler Requests (SCDRQO)≠(SCDRQI) ?
No
Return
Yes
Get Entry From Req Tble, SCDRQB (Data, Disp Adr)
Dispatch To Routine (i.e. JOBSRT)
At End Of Table ?
No
Yes
Reset Ptr, SCDRQO, to the Top of the List
JOBSRT
Request From CTY ?
Yes
No
Current SPT Count Above Login Limit (SPTC) ≥ (SPCI) ?
Yes
No
Enough Drum Space DRMFRE > DRMINO ?
No
Yes
JOBSR2:
JOBSR1:
Room for New Job & New Fork ?
No
Yes
TTEMES
Give User Bad New
Prints "Full"
Clear TTY's Entry in TTFORK Table
Return
Assign Job No. From FREJOB Linked List
ASSFK pSCH10
Get a Fork
Store Job No. in FKJOB Table
ASFSB
Assign a SPT Slot & Adjust Shr Count
Use Index for JSB Store SPT Index for JSB in FKJOB Table
Store NEWJBF Flag, TTY No. in the Pseudo-Interrupt Communications Table, FKINT
Return

SCH9

Assign Fork Slot



SCHIO

Scheduler Comments

SCHEDO:

(1) Running averages, exponentially weighed over intervals of 1, 5, and 15 minutes,are maintained for the number of runnable processes overall, as well as for those in High Run Queues and those in the Low Run Queues.

(2) Final phase of powerdown seq. clears the priority interrupt system and causes the system to loop in the ACs until power actually vanishes. If the power fail interrupt was spurious, the loop will time out after a few seconds and the system will be continued at address SYSRST.

(3) A very limited set of central functions for debugging purposes has been built into the Scheduler. To invoke a function, the appropriate bit or bits are set into loc 20 (SCTLW) via MDDT. The word is scanned from left to right (JFFO); the first bit found set on the scan selects the function.

Bit 0 Causes the scheduler to dismiss the current process and to stop timesharing. Useful to effect a clean manual transfer to Exec-mode DDT. System may be resumed at SCHEDO if no IOB reset is done.

Bit 1 Causes job specified by $(20)_{RH}$ to be run exclusively.

Bit 2 Forces running of Job 0 back-up function before halting the system.

If loc 30 (SHLTW) is set not equal to Ø, the system will crash. (Same as setting bit 2 of SCTLW word.)

AJBALS

(4) Upper Limit for LQ=MAXNR-MIN [Max HQ Reserve, HQ Load Avg.* (16)]

Upper Limit for HQ=MAXNR-MIN [Max LQ Reserve, No. of LQ forks * (32

(5) SUMNR reflects the number of timesharing pages in use. Its value after AJBALS equals the number of pages reserved for balance set members plus BALSHC (the number of pages shared, but not owned, by balance set members plus the number of locked pages).

BSPAD reflects the number of pages set aside for balance set members as their working set reserves grow. The real value of BSPAD is offset by a factor of BSPADO. When forks are trying to stay in the balance set, the adjustment algorithm allows the pad offset to be subtracted from the accumulated sum before it checks if the fork can fit.

i.e.,

$$(BSPAD + \sum_{i=1}^{n} Res.\ WS_i) - BSPADO + Res.\ WS_{n+1} \geq MAXNR$$

The adjustment algorithm does the opposite (i.e., adds the BSPADO factor) for forks trying to get into the balance set. The overall affect of this is to ensure (as much as possible) a certain number of pages be available for balance set forks.

(6) The shrink algorithm shrinks the fork's reserve working set by:

MIN [Reserve WS - Current WS, Accum. Sum + Fork's Res WS-MAXNR]

Notice that the fork's reserve working set will not be reduced below its current working set.

(7) This is the rare case of forks, with hold-time left, expanding. The lowest priority one is removed. If there is only one fork in the balance set, it is not removed. (Note: it is possible for one fork to be greater than MAXNR due to the BALSHC count changing).

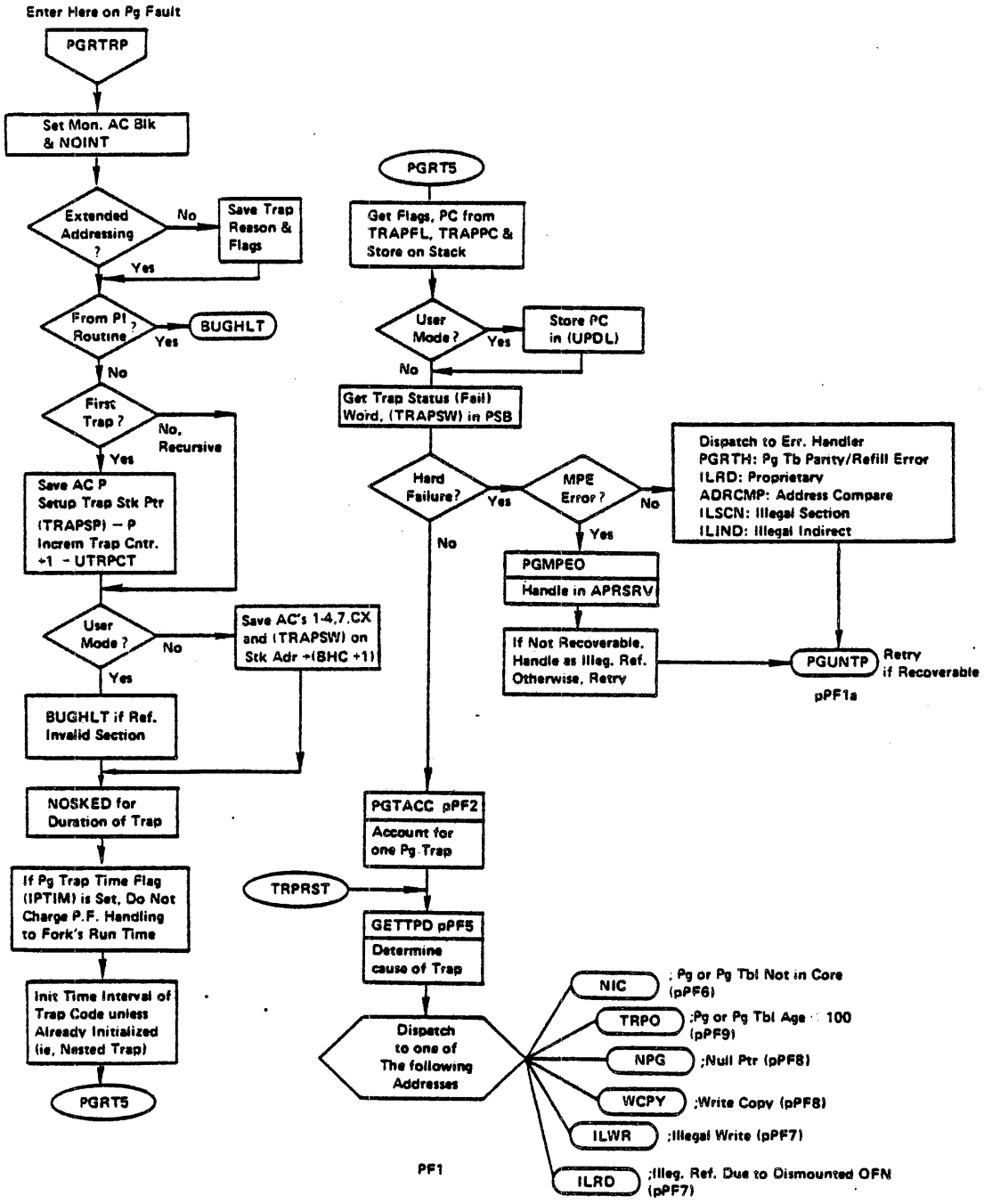
GCCOR

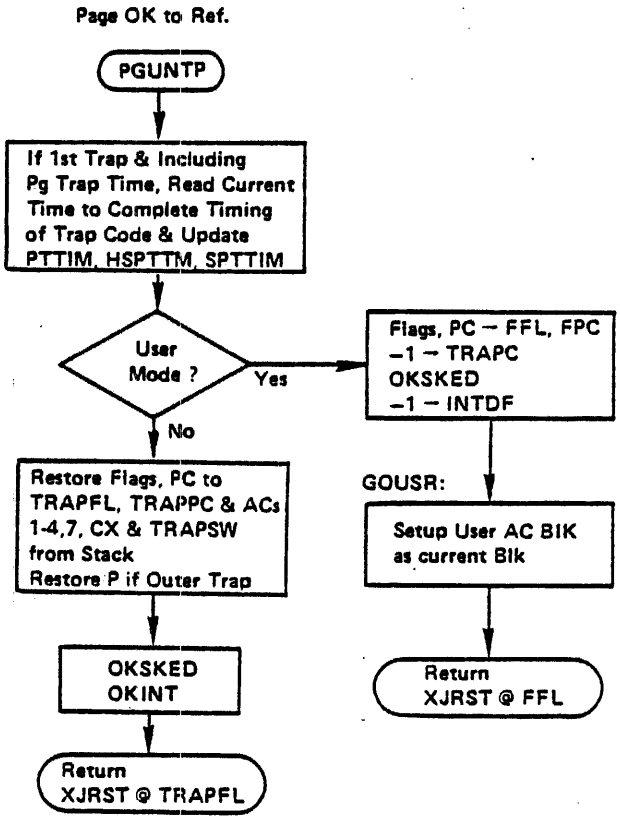
(8) If it is a forced clear, then GCMINO is made very large so all of core will be collected. However, its usual value is much lower. (Currently 64 decimal).

(9) The fork is actually placed on the GOLST at this time. WTCONC, after putting a fork on WTLST, checks if the wait condition is satisfied. The test routine, JSKP, gives a skip return indicating that the wait is satisfied. This causes UNBLK1 to be called which in turn calls SCHEDJ to unblock the fork and to requeue it from the WTLST to the GOLST.

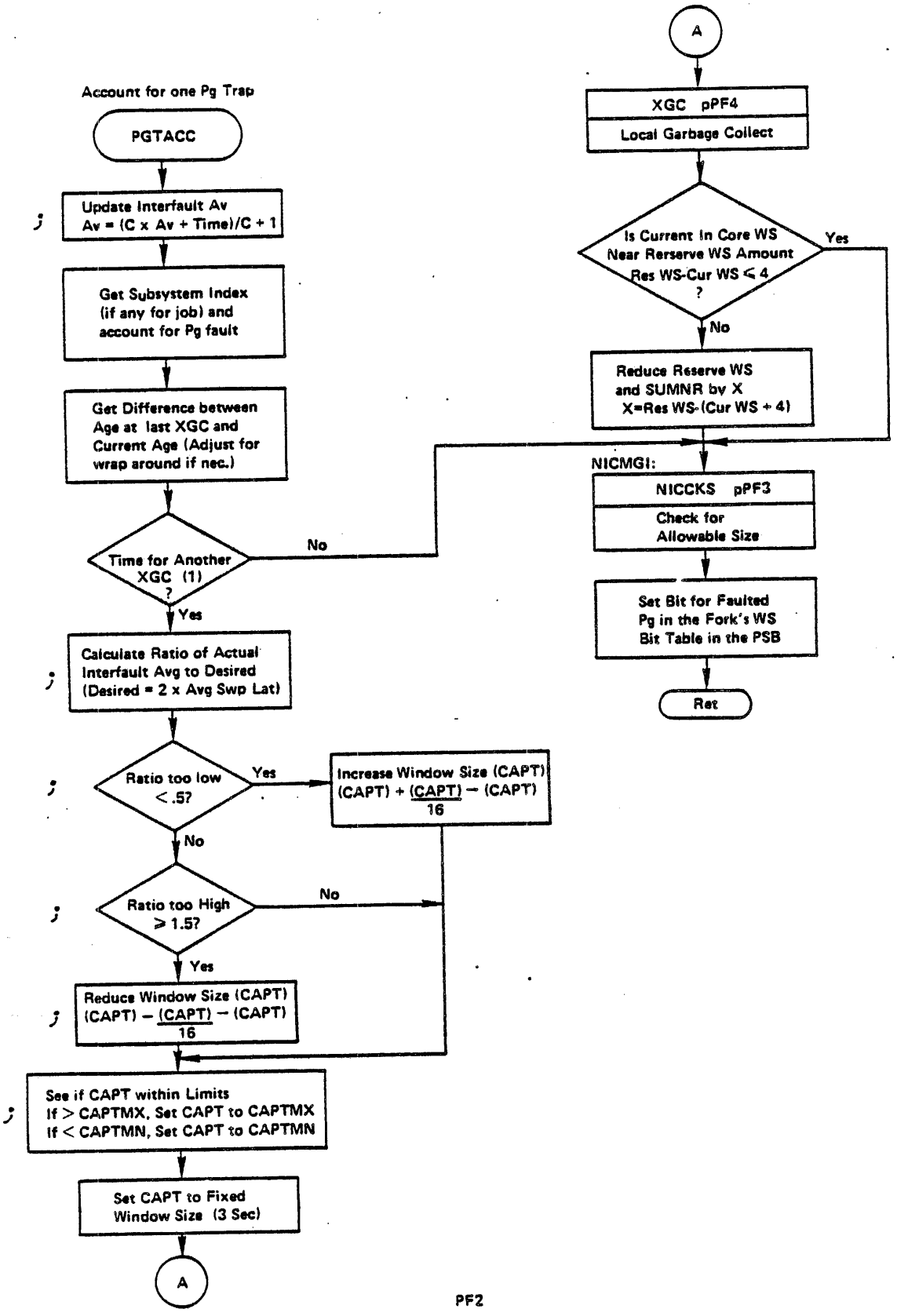
PAGE FAULT HANDLING FLOWCHARTS

PGRTRP -	Performs the Principal Accounting, Analysis, and Resolution of Page Faults		PF1
	PGTACC -	Accounts for Page Traps	PF2
		XGC - Local Garbage Collection	PF4
		SWPOUT - Swapping Out a Page	PF14
		NICCKS - Check In-Core Size Limits	PF3
	GETTPD -	Determine Cause of Trap	PF5
	NIC -	Not in Core Trap	PF6
		SWPINW - Swap In and Wait	PF10
		SWPIN - Swap In a Page	PF11
	WCPY -	Write Copy Trap	PF8
	ILRD -	Illegal Read Trap	PF7
	ILWR -	Illegal Write Trap	PF7
	TRPO -	Age < 100 Trap	PF9

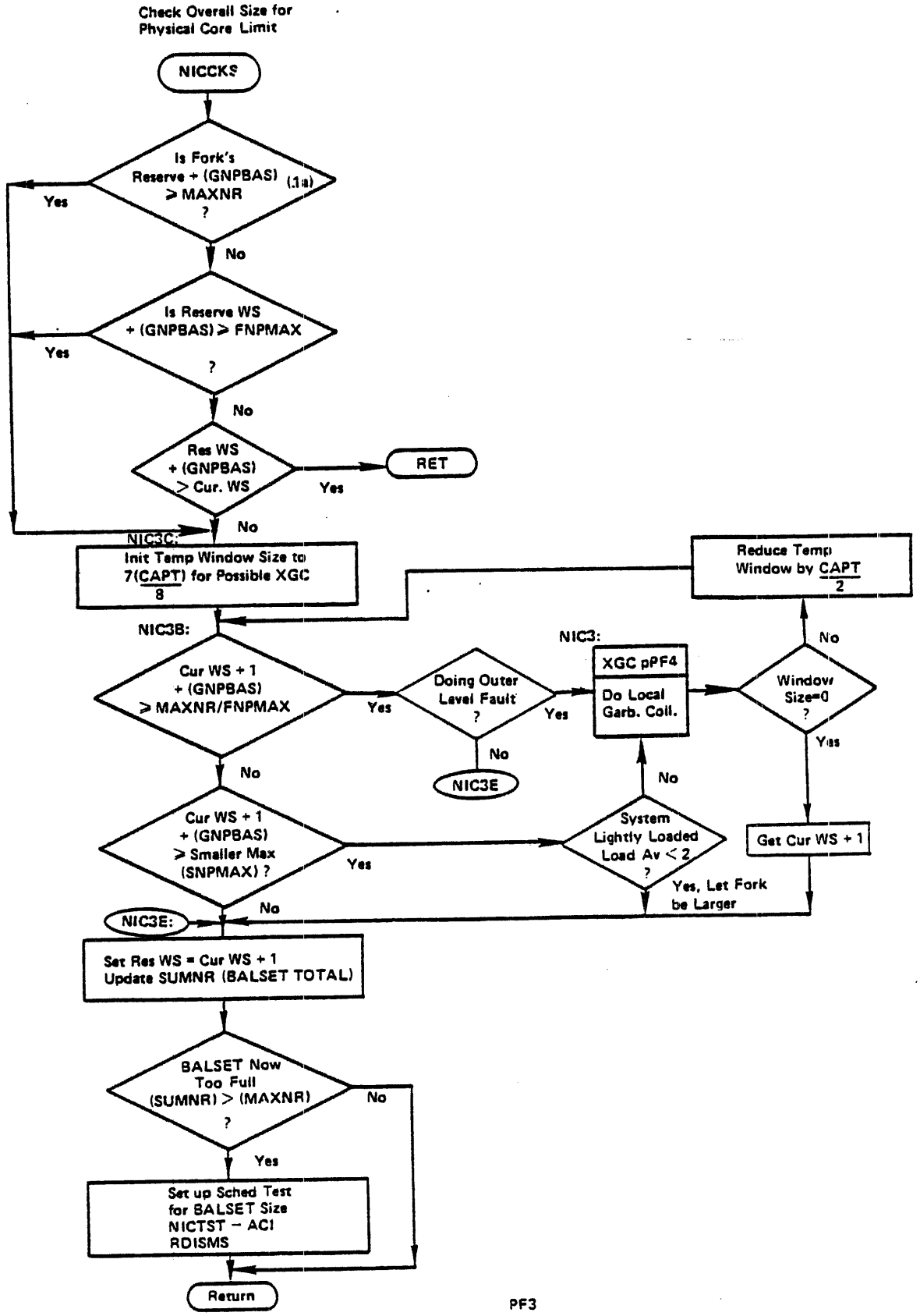

Enter Here on Pg Fault
PGRTRP
Set Mon. AC Blk & NOINT
Extended Addressing ?
No
Save Trap Reason & Flags
Yes
From PI Routine ?
Yes
BUGHLT
No
First Trap ?
No, Recursive
Yes
Save AC P
Setup Trap Stk Ptr (TRAPSP) – P
Increm Trap Cntr. +1 - UTRPCT
User Mode ?
No
Save AC's 1-4,7,CX and (TRAPSW) on Stk Adr +(BHC +1)
Yes
BUGHLT if Ref. Invalid Section
NOSKED for Duration of Trap
If Pg Trap Time Flag (IPTIM) is Set, Do Not Charge P.F. Handling to Fork's Run Time
Init Time Interval of Trap Code unless Already Initialized (ie, Nested Trap)
PGRT5
PGRT5
Get Flags, PC from TRAPFL, TRAPPC & Store on Stack
User Mode ?
Yes
Store PC in (UPDL)
No
Get Trap Status (Fail) Word, (TRAPSW) in PSB
Hard Failure?
Yes
MPE Error ?
No
Yes
Dispatch to Err. Handler
PGRTH: Pg Tb Parity/Refill Error
ILRD: Proprietary
ADRCMP: Address Compare
ILSCN: Illegal Section
ILIND: Illegal Indirect
PGMPEO
Handle in APRSRV
If Not Recoverable, Handle as Illeg. Ref. Otherwise, Retry
PGUNTP
Retry if Recoverable
pPF1a
No
PGTACC pPF2
Account for one Pg Trap
TRPRST
GETTPD pPF5
Determine cause of Trap
Dispatch to one of The following Addresses
NIC
; Pg or Pg Tbl Not in Core (pPF6)
TRPO
;Pg or Pg Tbl Age = 100 (pPF9)
NPG
;Null Ptr (pPF8)
WCPY
;Write Copy (pPF8)
ILWR
;Illegal Write (pPF7)
ILRD
;Illeg. Ref. Due to Dismounted OFN (pPF7)
PF1


Page OK to Ref.
PGUNTP
If 1st Trap & Including Pg Trap Time, Read Current Time to Complete Timing of Trap Code & Update PTTIM, HSPTTM, SPTTIM
User Mode ?
Yes
No
Flags, PC ← FFL, FPC
-1 → TRAPC
OKSKED
-1 → INTDF
GOUSR:
Setup User AC BIK as current Blk
Return XJRST @ FFL
Restore Flags, PC to TRAPFL, TRAPPC & ACs 1-4,7, CX & TRAPSW from Stack
Restore P if Outer Trap
OKSKED
OKINT
Return XJRST @ TRAPFL

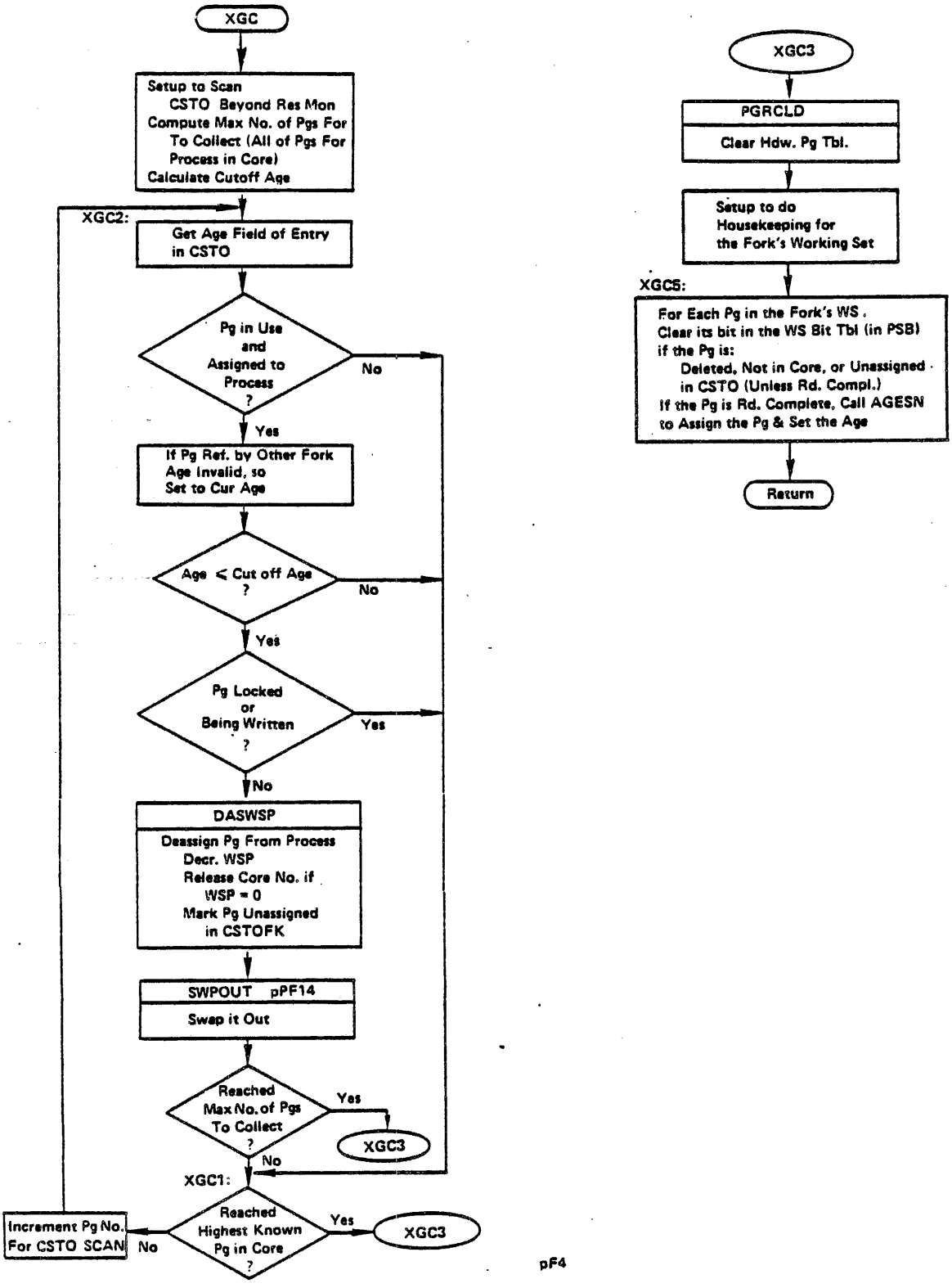
PF1a


Account for one Pg Trap
PGTACC
Update Interfault Av
Av = (C x Av + Time)/C + 1
Get Subsystem Index
(if any for job) and
account for Pg fault
Get Difference between
Age at last XGC and
Current Age (Adjust for
wrap around if nec.)
Time for Another
XGC (1)
?
No
Yes
Calculate Ratio of Actual
Interfault Avg to Desired
(Desired = 2 x Avg Swp Lat)
Ratio too low
< .5?
Yes
Increase Window Size (CAPT)
(CAPT) + (CAPT) − (CAPT)
16
No
Ratio too High
> 1.5?
No
Yes
Reduce Window Size (CAPT)
(CAPT) − (CAPT) − (CAPT)
16
See if CAPT within Limits
If > CAPTMX, Set CAPT to CAPTMX
If < CAPTMN, Set CAPT to CAPTMN
Set CAPT to Fixed
Window Size (3 Sec)
A
A
XGC pPF4
Local Garbage Collect
Is Current In Core WS
Near Rerserve WS Amount
Res WS-Cur WS ≤ 4
?
Yes
No
Reduce Reserve WS
and SUMNR by X
X=Res WS-(Cur WS + 4)
NICMGI:
NICCKS pPF3
Check for
Allowable Size
Set Bit for Faulted
Pg in the Fork's WS
Bit Table in the PSB
Ret

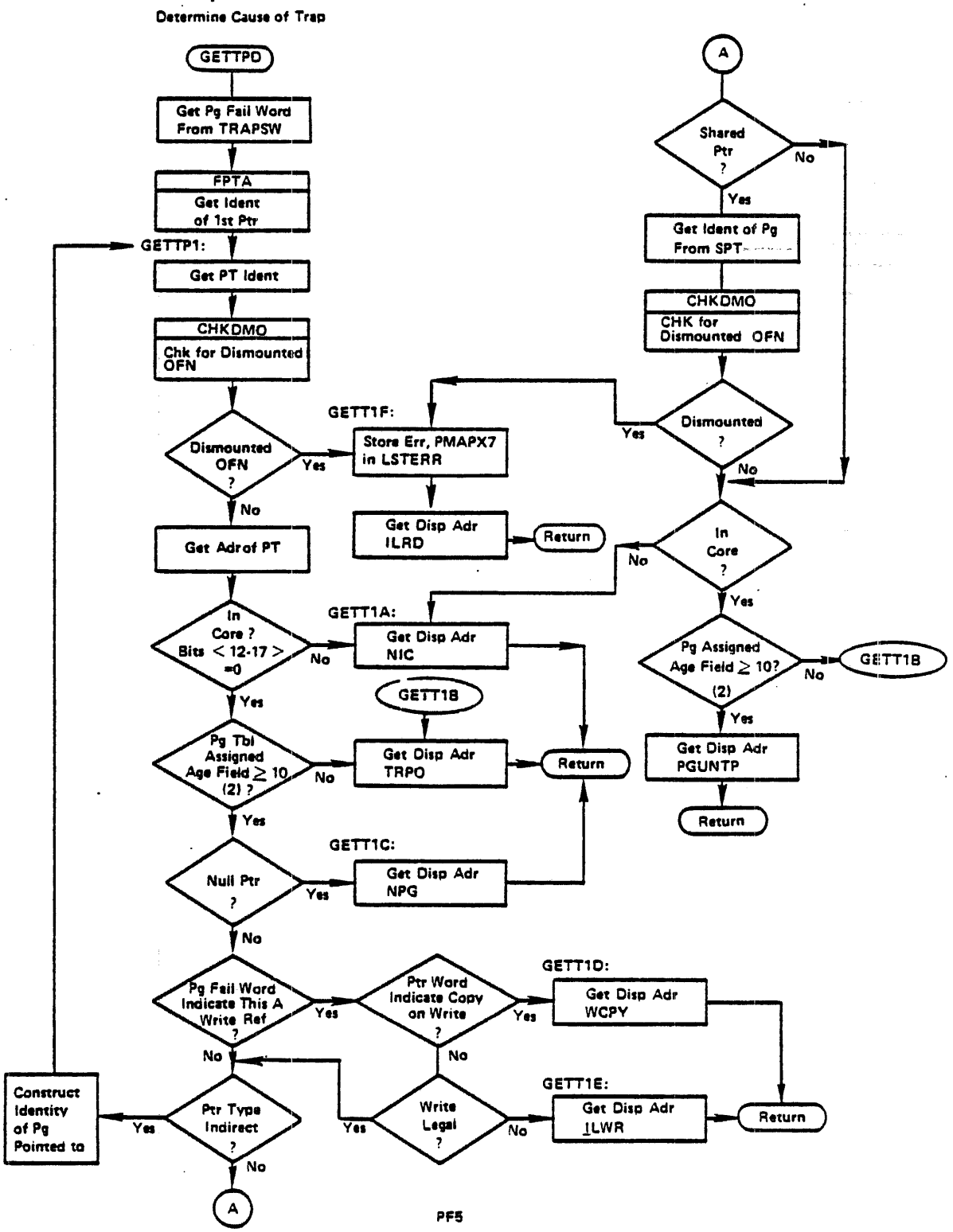
PF2

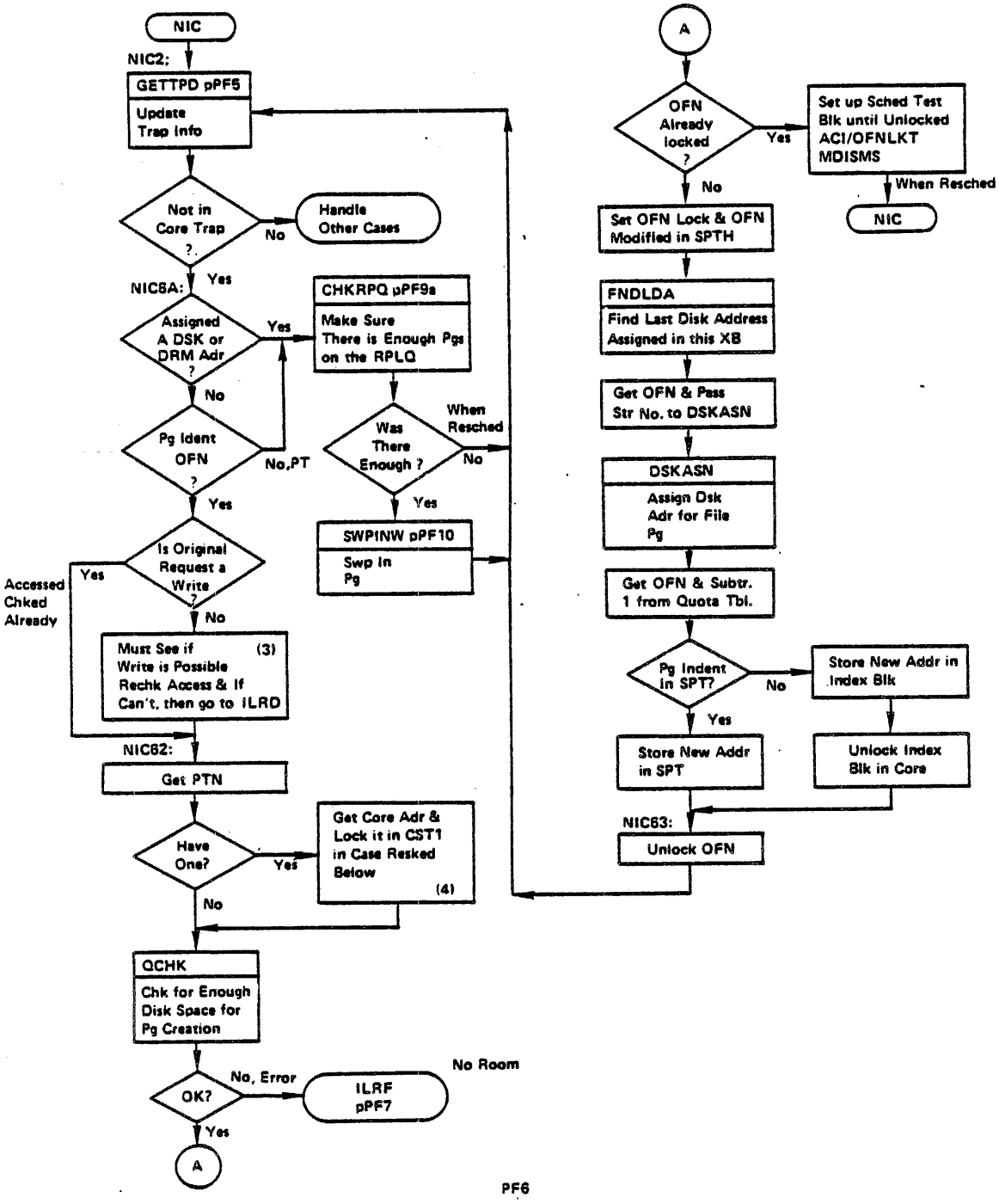

Check Overall Size for Physical Core Limit
NICCKS
Is Fork's Reserve + (GNPBAS) ≥ MAXNR ?
(1a)
Yes
No
Is Reserve WS + (GNPBAS) ≥ FNPMAX ?
Yes
No
Res WS + (GNPBAS) > Cur. WS
RET
Yes
No
NIC3C:
Init Temp Window Size to 7(CAPT/8) for Possible XGC
NIC3B:
Cur WS + 1 + (GNPBAS) ≥ MAXNR/FNPMAX
Yes
Doing Outer Level Fault ?
Yes
No
NIC3E
NIC3:
XGC pPF4
Do Local Garb. Coll.
Window Size=0 ?
No
Yes
Reduce Temp Window by CAPT/2
No
Cur WS + 1 + (GNPBAS) ≥ Smaller Max (SNPMAX) ?
Yes
No
System Lightly Loaded Load Av < 2 ?
No
Yes, Let Fork be Larger
Get Cur WS + 1
NIC3E:
Set Res WS = Cur WS + 1
Update SUMNR (BALSET TOTAL)
BALSET Now Too Full (SUMNR) > (MAXNR) ?
No
Yes
Set up Sched Test for BALSET Size
NICTST – ACI
RDISMS
Return
PF3

Local Garbage Collection

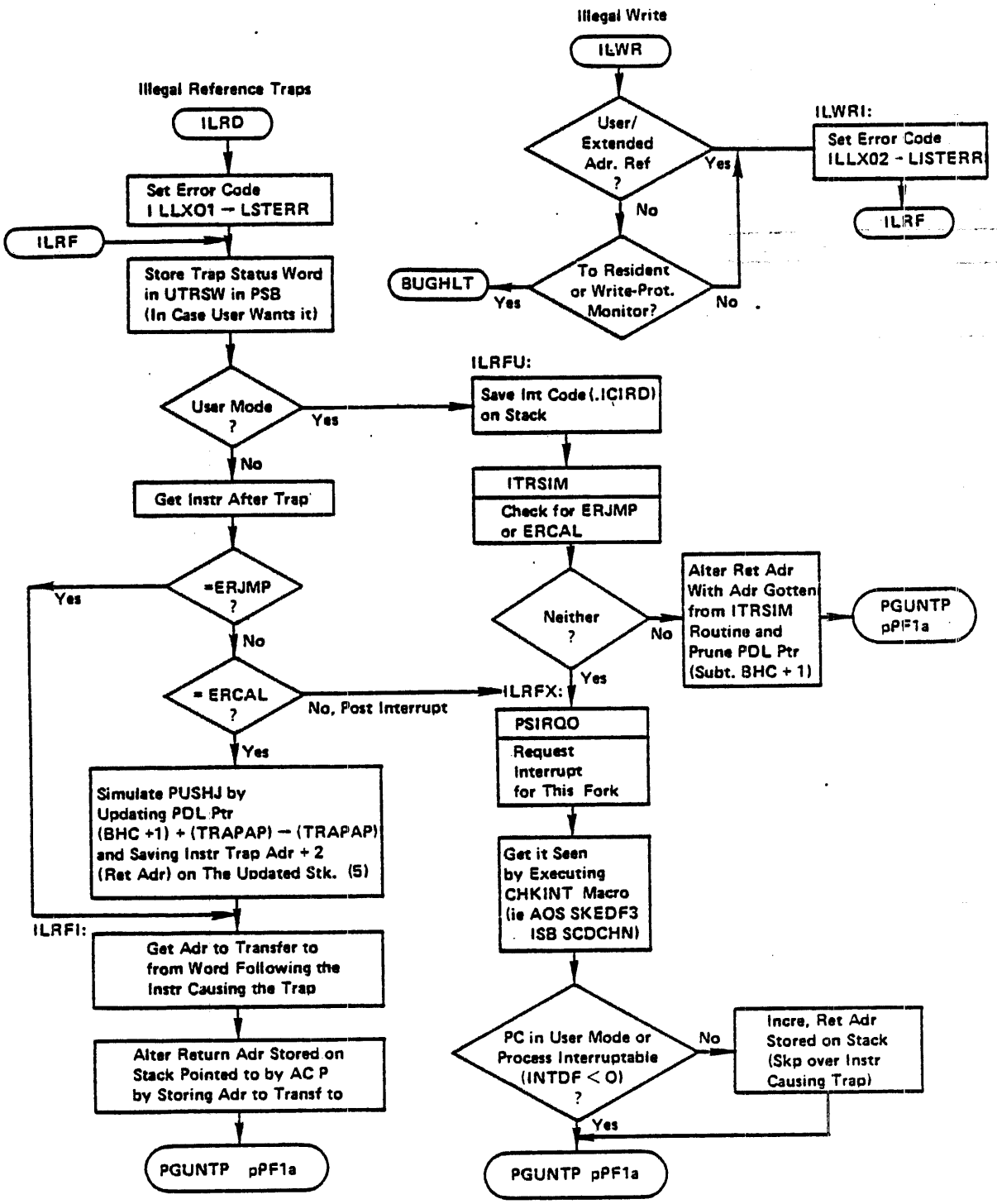


pF4

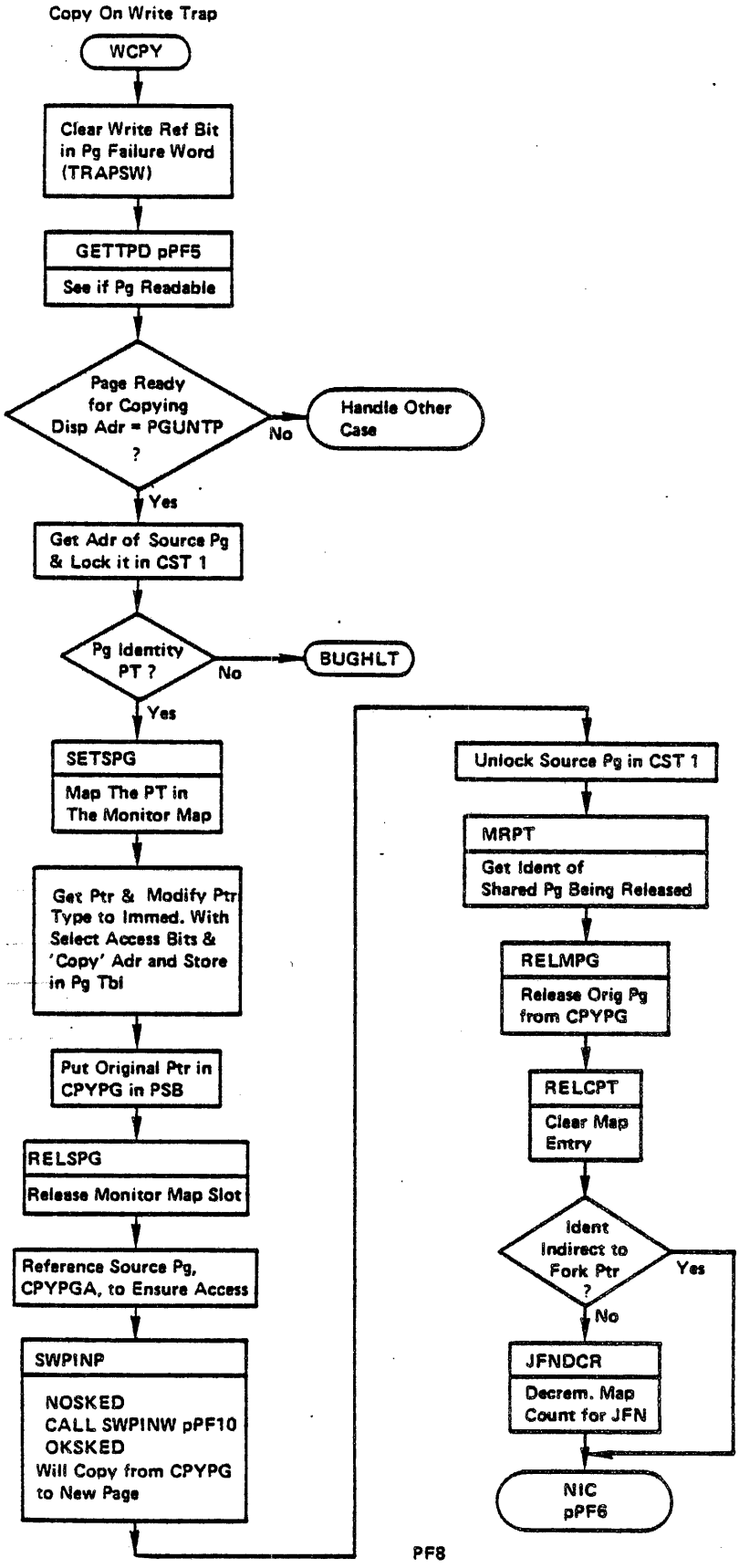

Determine Cause of Trap
GETTPD
Get Pg Fail Word From TRAPSW
FPTA
Get Ident of 1st Ptr
GETTP1:
Get PT Ident
CHKDMO
Chk for Dismounted OFN
Dismounted OFN ?
Yes
No
GETT1F:
Store Err, PMAPX7 in LSTERR
Get Disp Adr ILRD
Return
Get Adr of PT
In Core ? Bits < 12-17 > =0
No
Yes
GETT1A:
Get Disp Adr NIC
GETT1B
Pg Tbl Assigned Age Field ≥ 10 (2) ?
Get Disp Adr TRPO
Return
Null Ptr ?
GETT1C:
Get Disp Adr NPG
Pg Fail Word Indicate This A Write Ref ?
Ptr Word Indicate Copy on Write ?
GETT1D:
Get Disp Adr WCPY
Construct Identity of Pg Pointed to
Ptr Type Indirect ?
Write Legal ?
GETT1E:
Get Disp Adr ILWR
A
Shared Ptr ?
Get Ident of Pg From SPT
CHKDMO
CHK for Dismounted OFN
Dismounted ?
In Core ?
Pg Assigned Age Field ≥ 10? (2)
Get Disp Adr PGUNTP
PF5

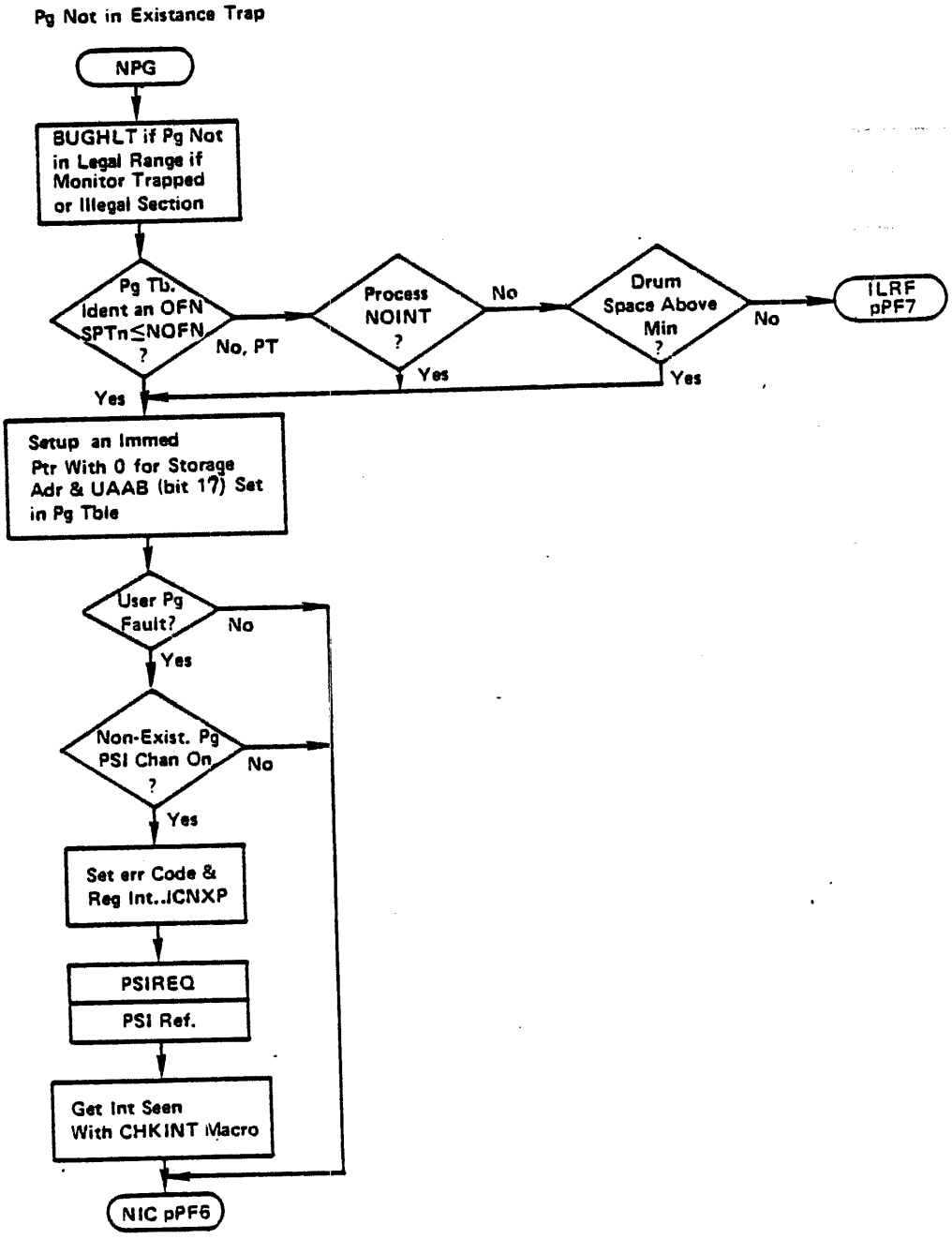

NIC
NIC2:
GETTPD pPF5
Update Trap Info
Not in Core Trap ?
No
Handle Other Cases
Yes
NIC8A:
Assigned A DSK or DRM Adr ?
Yes
CHKRPQ pPF9a
Make Sure There is Enough Pgs on the RPLQ
No
Pg Ident OFN ?
No,PT
Was There Enough ?
When Resched
No
Yes
SWPINW pPF10
Swp In Pg
Yes
Is Original Request a Write ?
Yes
Accessed Chked Already
No
Must See if Write is Possible Rechk Access & If Can't, then go to ILRD
(3)
NIC62:
Get PTN
Have One?
Yes
Get Core Adr & Lock it in CST1 in Case Resked Below
(4)
No
QCHK
Chk for Enough Disk Space for Pg Creation
OK?
No, Error
ILRF pPF7
No Room
Yes
A
A
OFN Already locked ?
Yes
Set up Sched Test Blk until Unlocked ACI/OFNLKT MDISMS
When Resched
NIC
No
Set OFN Lock & OFN Modified in SPTH
FNDLDA
Find Last Disk Address Assigned in this XB
Get OFN & Pass Str No. to DSKASN
DSKASN
Assign Dsk Adr for File Pg
Get OFN & Subtr. 1 from Quota Tbl.
Pg Indent In SPT?
No
Store New Addr in Index Blk
Yes
Store New Addr in SPT
Unlock Index Blk in Core
NIC63:
Unlock OFN

PF6

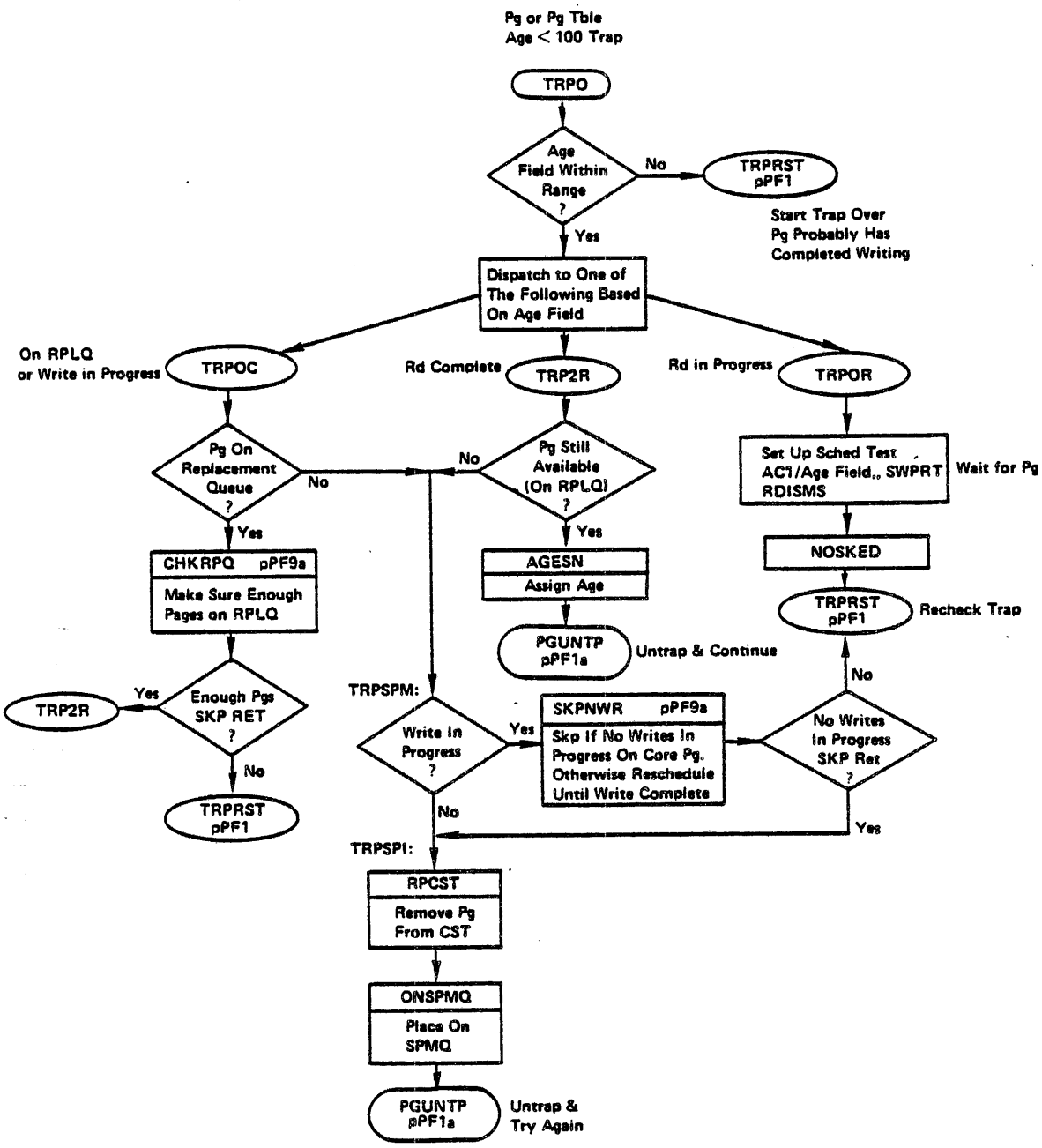

Illegal Write
ILWR
User/ Extended Adr. Ref ?
Yes
ILWRI:
Set Error Code ILLX02 - LISTERR
ILRF
No
To Resident or Write-Prot. Monitor?
Yes
BUGHLT
No
Illegal Reference Traps
ILRD
Set Error Code I LLX01 - LSTERR
ILRF
Store Trap Status Word in UTRSW in PSB (In Case User Wants it)
ILRFU:
User Mode ?
Yes
Save Int Code (.ICIRD) on Stack
No
Get Instr After Trap
ITRSIM
Check for ERJMP or ERCAL
=ERJMP ?
Yes
Neither ?
No
Alter Ret Adr With Adr Gotten from ITRSIM Routine and Prune PDL Ptr (Subt. BHC + 1)
PGUNTP pPF1a
No
Yes
= ERCAL ?
No, Post Interrupt
ILRFX:
PSIRQO
Request Interrupt for This Fork
Yes
Simulate PUSHJ by Updating PDL Ptr (BHC +1) + (TRAPAP) - (TRAPAP) and Saving Instr Trap Adr + 2 (Ret Adr) on The Updated Stk. (5)
Get it Seen by Executing CHKINT Macro (ie AOS SKEDF3 ISB SCDCHN)
ILRFI:
Get Adr to Transfer to from Word Following the Instr Causing the Trap
PC in User Mode or Process Interruptable (INTDF < 0) ?
No
Incre, Ret Adr Stored on Stack (Skp over Instr Causing Trap)
Alter Return Adr Stored on Stack Pointed to by AC P by Storing Adr to Transf to
Yes
PGUNTP pPF1a
PGUNTP pPF1a

PF7

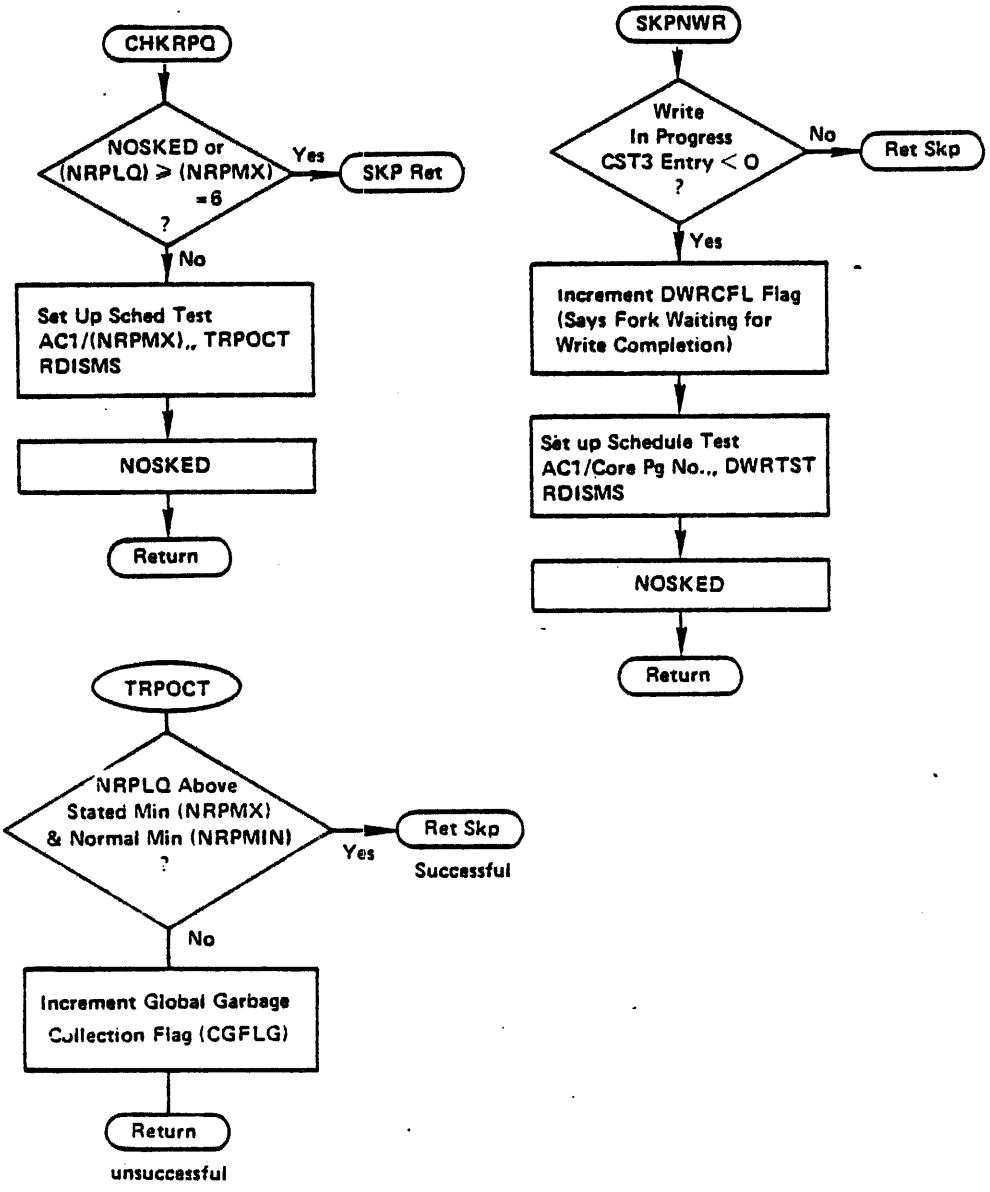

Copy On Write Trap
WCPY
Clear Write Ref Bit in Pg Failure Word (TRAPSW)
GETTPD pPF5
See if Pg Readable
Page Ready for Copying Disp Adr = PGUNTP ?
No
Handle Other Case
Yes
Get Adr of Source Pg & Lock it in CST 1
Pg Identity PT ?
No
BUGHLT
Yes
SETSPG
Map The PT in The Monitor Map
Get Ptr & Modify Ptr Type to Immed. With Select Access Bits & 'Copy' Adr and Store in Pg Tbl
Put Original Ptr in CPYPG in PSB
RELSPG
Release Monitor Map Slot
Reference Source Pg, CPYPGA, to Ensure Access
SWPINP
NOSKED
CALL SWPINW pPF10
OKSKED
Will Copy from CPYPG to New Page
PF8
Unlock Source Pg in CST 1
MRPT
Get Ident of Shared Pg Being Released
RELMPG
Release Orig Pg from CPYPG
RELCPT
Clear Map Entry
Ident Indirect to Fork Ptr ?
Yes
No
JFNDCR
Decrem. Map Count for JFN
NIC pPF6


Pg Not in Existance Trap
NPG
BUGHLT if Pg Not in Legal Range if Monitor Trapped or Illegal Section
Pg Tb. Ident an OFN SPTn≤NOFN ?
No, PT
Process NOINT ?
No
Drum Space Above Min ?
No
ILRF pPF7
Yes
Yes
Yes
Setup an Immed Ptr With 0 for Storage Adr & UAAB (bit 17) Set in Pg Tble
User Pg Fault?
No
Yes
Non-Exist. Pg PSI Chan On ?
No
Yes
Set err Code & Reg Int..ICNXP
PSIREQ
PSI Ref.
Get Int Seen With CHKINT Macro
NIC pPF6

PF 8a


Pg or Pg Tble
Age < 100 Trap
TRPO
Age
Field Within
Range
?
No
TRPRST
pPF1
Start Trap Over
Pg Probably Has
Completed Writing
Yes
Dispatch to One of
The Following Based
On Age Field
On RPLQ
or Write in Progress
TRPOC
Rd Complete
TRP2R
Rd in Progress
TRPOR
Pg On
Replacement
Queue
?
No
Pg Still
Available
(On RPLQ)
?
No
Set Up Sched Test
ACT/Age Field,, SWPRT
RDISMS
Wait for Pg
Yes
Yes
CHKRPQ pPF9a
Make Sure Enough
Pages on RPLQ
AGESN
Assign Age
NOSKED
PGUNTP
pPF1a
Untrap & Continue
TRPRST
pPF1
Recheck Trap
Yes
Enough Pgs
SKP RET
?
TRP2R
TRPSPM:
Write In
Progress
?
Yes
SKPNWR pPF9a
Skp If No Writes In
Progress On Core Pg.
Otherwise Reschedule
Until Write Complete
No Writes
In Progress
SKP Ret
?
No
Yes
No
TRPRST
pPF1
No
Yes
TRPSPI:
RPCST
Remove Pg
From CST
ONSPMQ
Place On
SPMQ
PGUNTP
pPF1a
Untrap &
Try Again

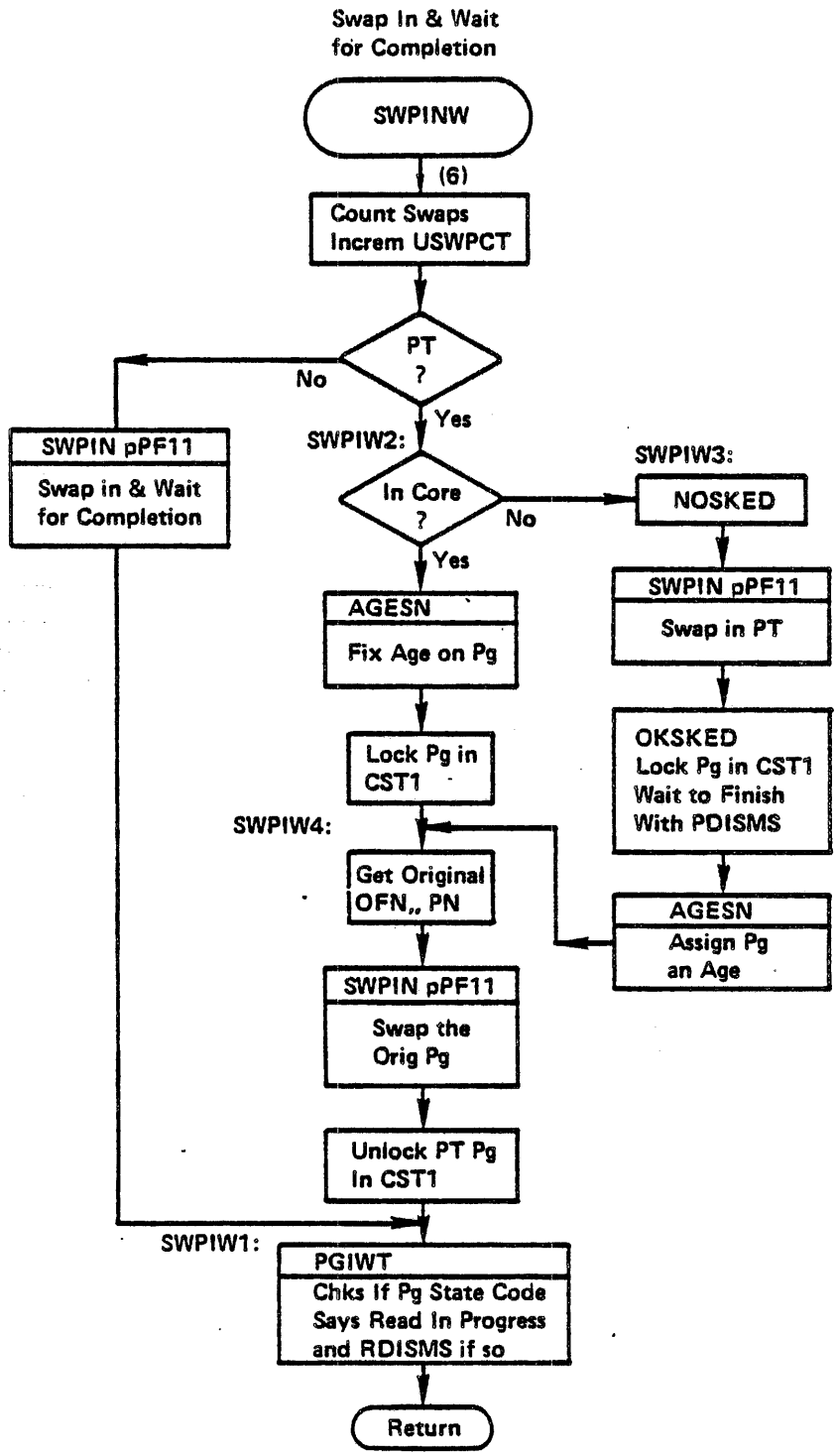
PF9


CHKRPQ
NOSKED or (NRPLQ) ≥ (NRPMX) =6 ?
Yes
SKP Ret
No
Set Up Sched Test AC1/(NRPMX),, TRPOCT RDISMS
NOSKED
Return
SKPNWR
Write In Progress CST3 Entry < 0 ?
No
Ret Skp
Yes
Increment DWRCFL Flag (Says Fork Waiting for Write Completion)
Set up Schedule Test AC1/Core Pg No.., DWRTST RDISMS
NOSKED
Return
TRPOCT
NRPLQ Above Stated Min (NRPMX) & Normal Min (NRPMIN) ?
Yes
Ret Skp
Successful
No
Increment Global Garbage Collection Flag (CGFLG)
Return
unsuccessful

PF9a

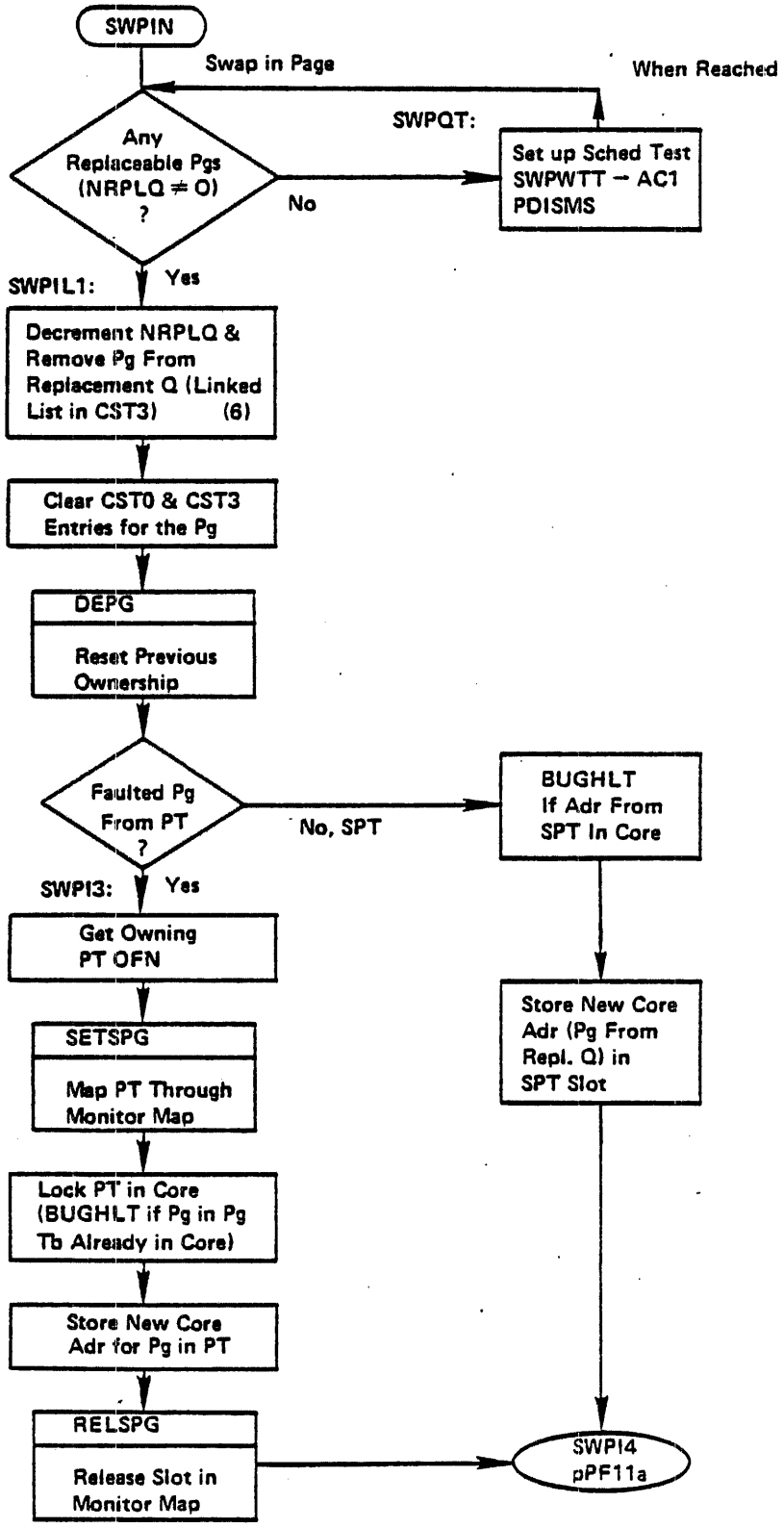
REQUESTING DRUM OR DISK READ

(PAGEM LEVEL)

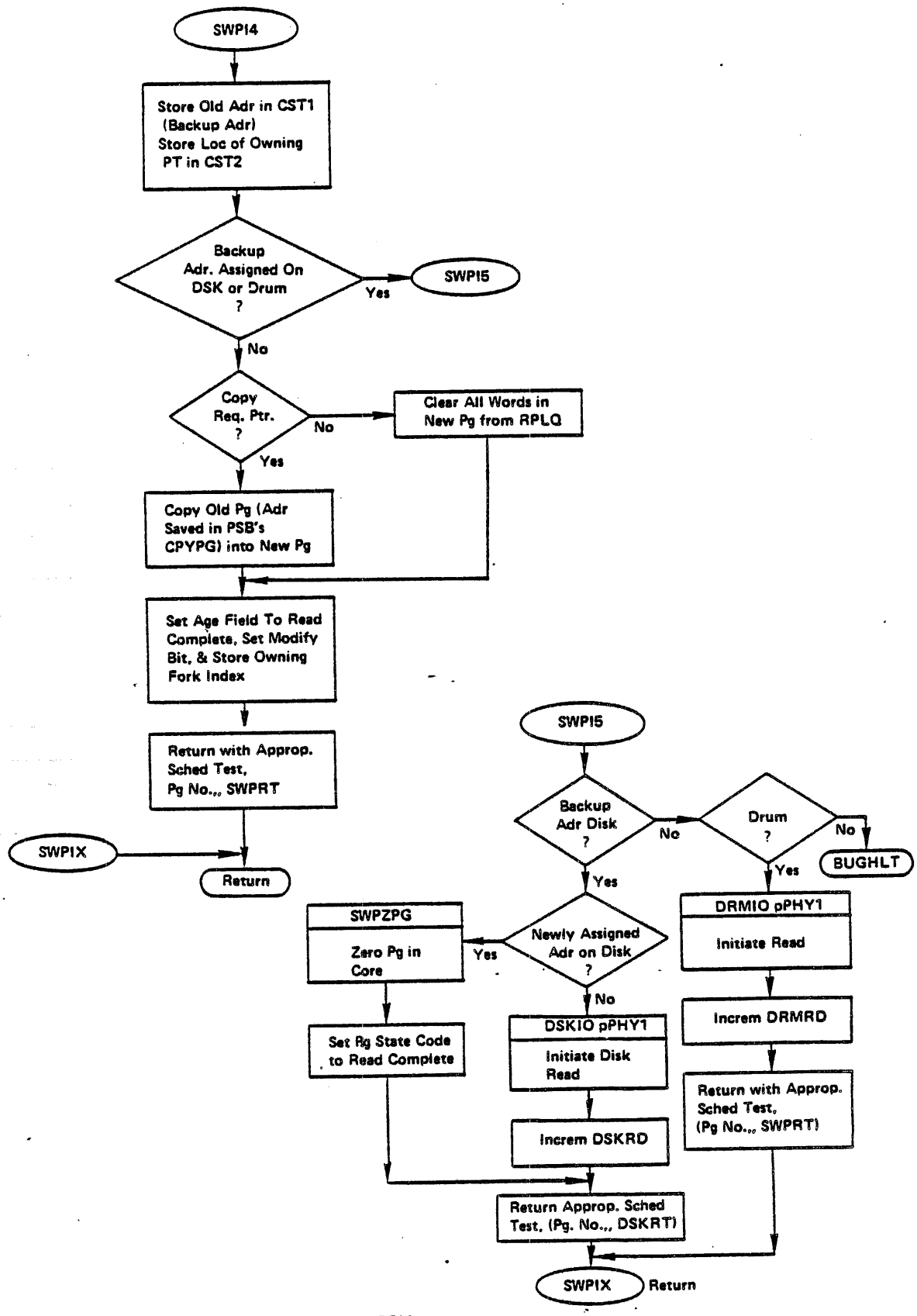


PF10

REQUESTING DRUM OR DISK READ (Continued)
(PAGEM LEVEL)

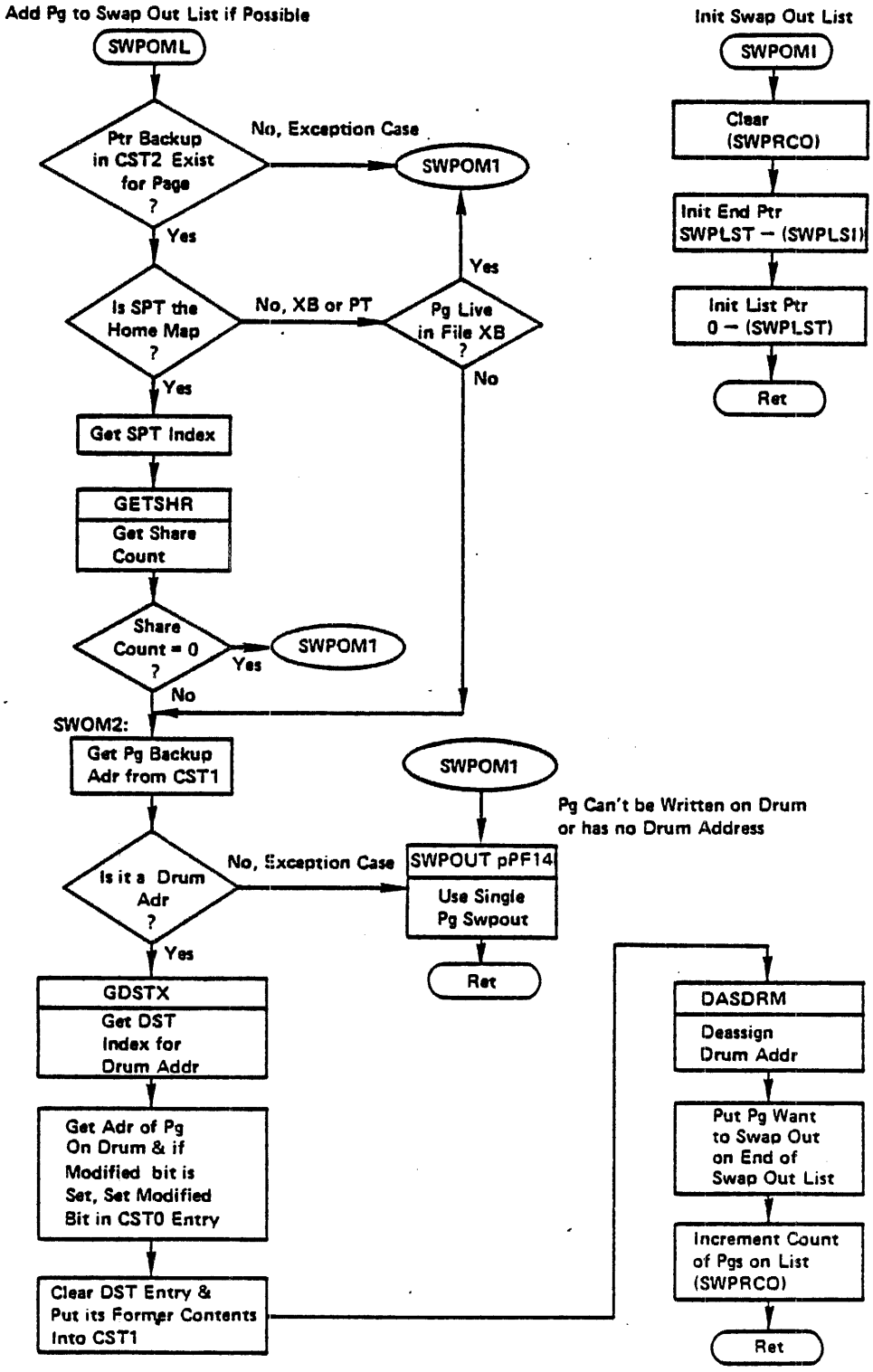


PF11

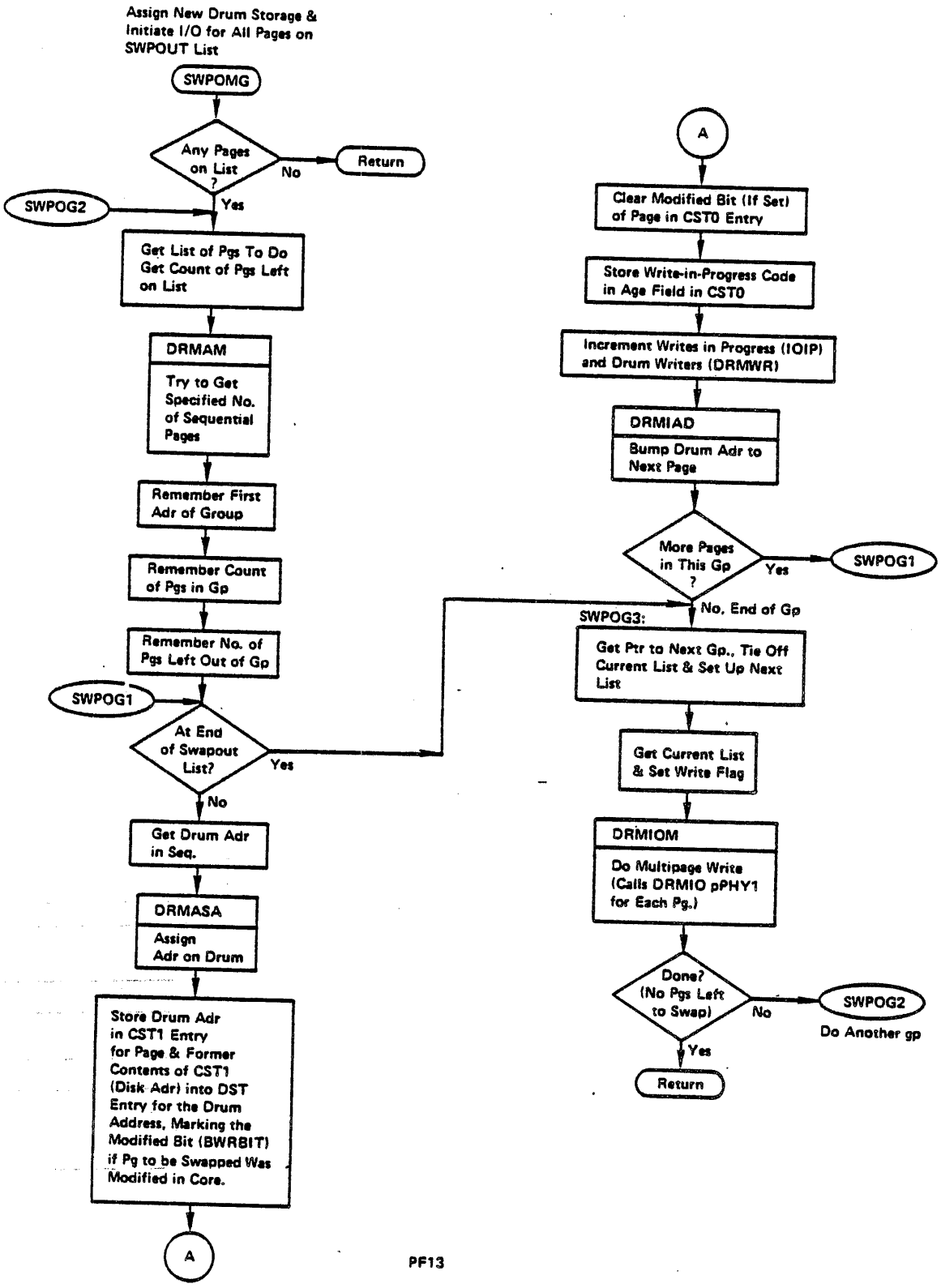

SWPI4
Store Old Adr in CST1 (Backup Adr) Store Loc of Owning PT in CST2
Backup Adr. Assigned On DSK or Drum ?
Yes
SWPI5
No
Copy Req. Ptr. ?
No
Clear All Words in New Pg from RPLQ
Yes
Copy Old Pg (Adr Saved in PSB's CPYPG) into New Pg
Set Age Field To Read Complete, Set Modify Bit, & Store Owning Fork Index
Return with Approp. Sched Test, Pg No.,, SWPRT
SWPIX
Return
SWPI5
Backup Adr Disk ?
No
Drum ?
No
BUGHLT
Yes
Yes
SWPZPG
Zero Pg in Core
Yes
Newly Assigned Adr on Disk ?
No
DRMIO pPHY1
Initiate Read
Increm DRMRD
Set Pg State Code to Read Complete
DSKIO pPHY1
Initiate Disk Read
Increm DSKRD
Return with Approp. Sched Test, (Pg No.,, SWPRT)
Return Approp. Sched Test, (Pg. No.,, DSKRT)
SWPIX
Return

PF11a

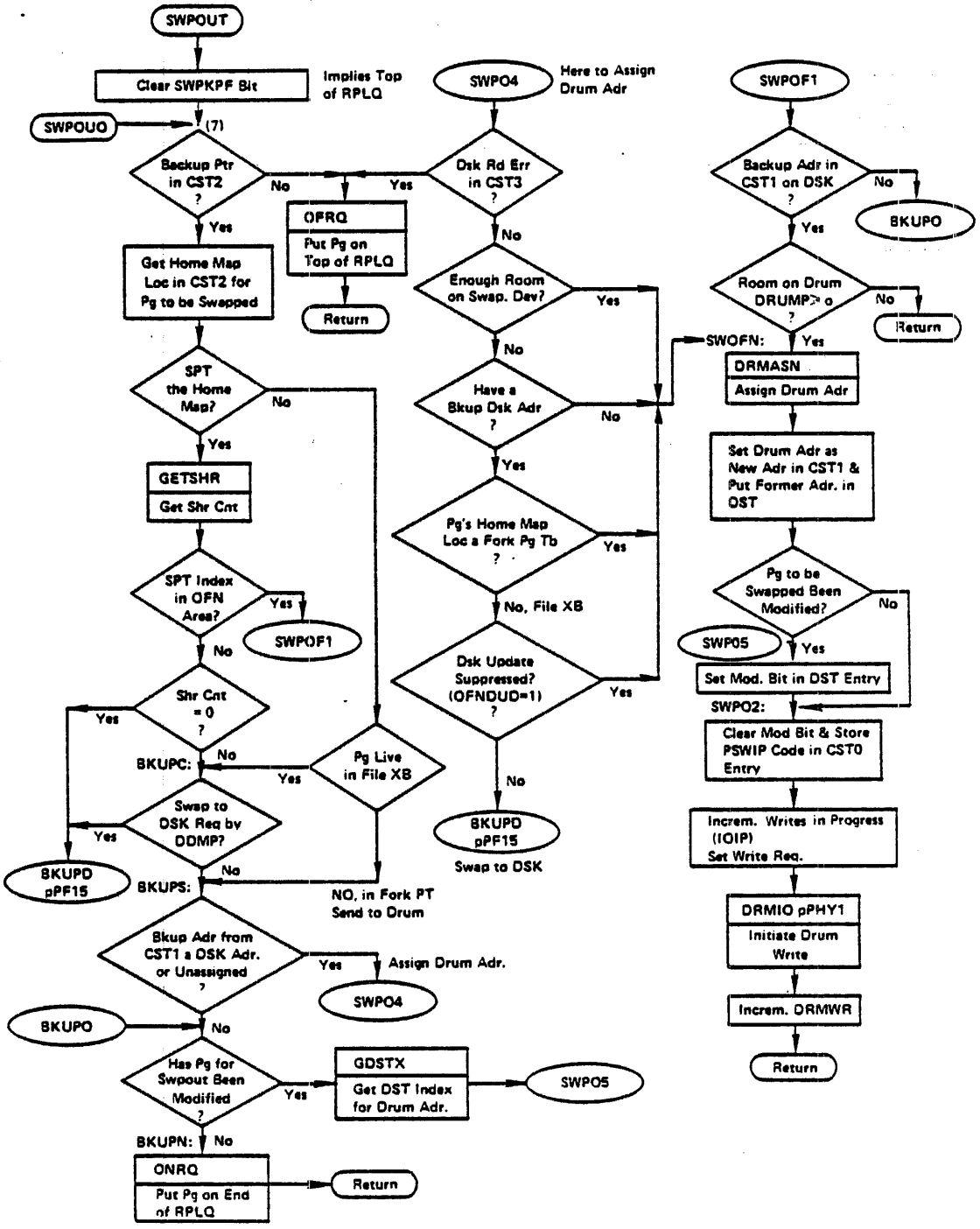
MULTIPLE PAGE SWAP OUT ROUTINE
SWPOMI - Init List
SWPOML - Called to Add Page to Swap Out List if Possible
SWPOMG - To Begin I/O for All Pages on Swap Out List
SWPOUT - Initiate Swap Out of Single Page



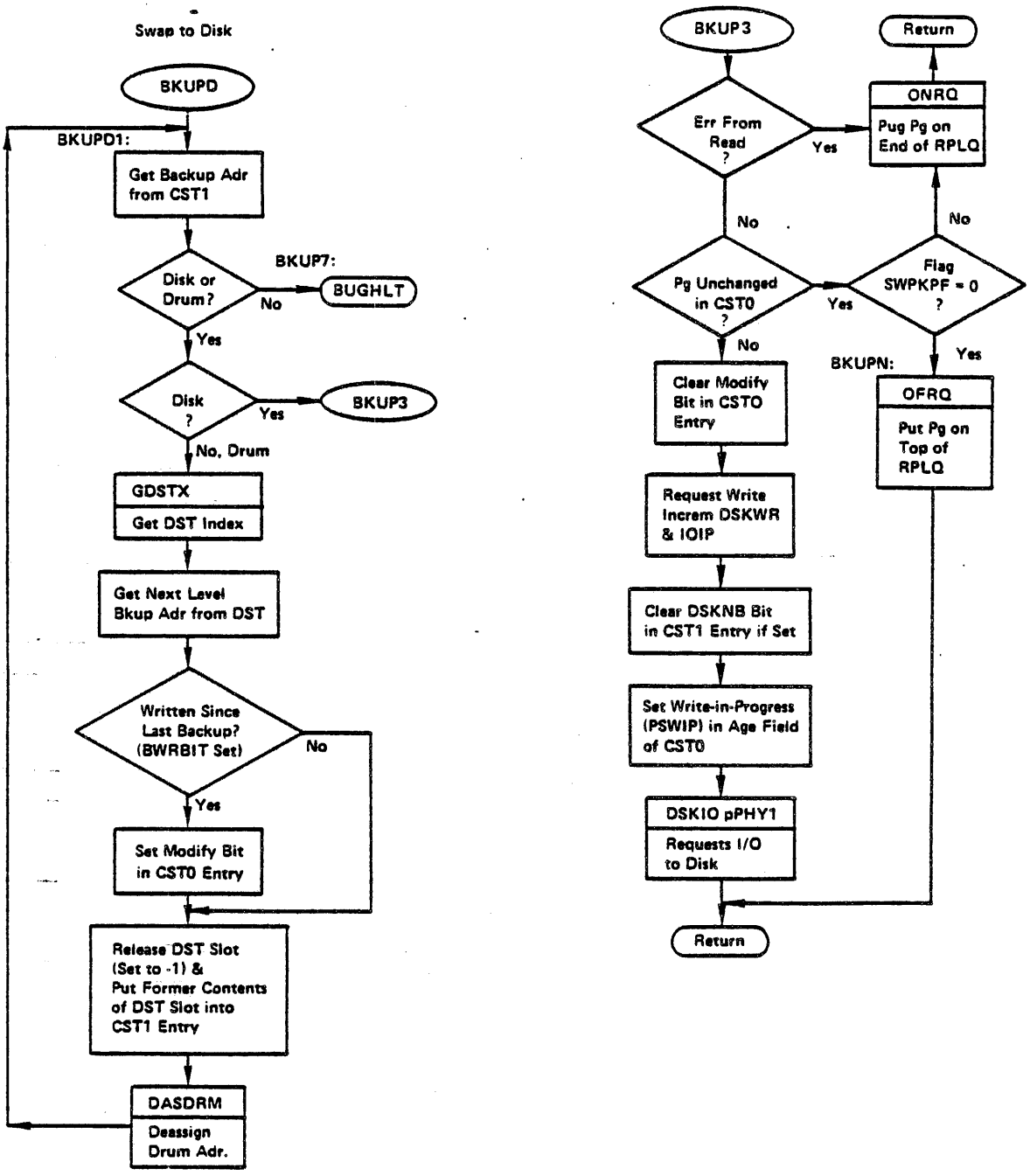
PF12


Assign New Drum Storage & Initiate I/O for All Pages on SWPOUT List
SWPOMG
Any Pages on List ?
No
Return
Yes
SWPOG2
Get List of Pgs To Do Get Count of Pgs Left on List
DRMAM
Try to Get Specified No. of Sequential Pages
Remember First Adr of Group
Remember Count of Pgs in Gp
Remember No. of Pgs Left Out of Gp
SWPOG1
At End of Swapout List?
Yes
No
Get Drum Adr in Seq.
DRMASA
Assign Adr on Drum
Store Drum Adr in CST1 Entry for Page & Former Contents of CST1 (Disk Adr) into DST Entry for the Drum Address, Marking the Modified Bit (BWRBIT) if Pg to be Swapped Was Modified in Core.
A
A
Clear Modified Bit (If Set) of Page in CST0 Entry
Store Write-in-Progress Code in Age Field in CST0
Increment Writes in Progress (IOIP) and Drum Writers (DRMWR)
DRMIAD
Bump Drum Adr to Next Page
More Pages in This Gp ?
Yes
SWPOG1
No, End of Gp
SWPOG3:
Get Ptr to Next Gp., Tie Off Current List & Set Up Next List
Get Current List & Set Write Flag
DRMIOM
Do Multipage Write (Calls DRMIO pPHY1 for Each Pg.)
Done? (No Pgs Left to Swap)
No
SWPOG2
Do Another gp
Yes
Return

PF13


SWPOUT
Clear SWPKPF Bit
Implies Top of RPLQ
SWPOUO
(7)
Backup Ptr in CST2 ?
No
Yes
OFRQ
Put Pg on Top of RPLQ
Return
Get Home Map Loc in CST2 for Pg to be Swapped
SPT the Home Map?
No
Yes
GETSHR
Get Shr Cnt
SPT Index in OFN Area?
Yes
SWPOF1
No
Shr Cnt = 0 ?
Yes
No
BKUPC:
Yes
Pg Live in File XB
Swap to DSK Req by DDMP?
Yes
BKUPD pPF15
No
BKUPS:
NO, in Fork PT Send to Drum
Bkup Adr from CST1 a DSK Adr. or Unassigned ?
Yes
Assign Drum Adr.
SWPO4
BKUPO
No
Has Pg for Swpout Been Modified ?
Yes
GDSTX
Get DST Index for Drum Adr.
SWPO5
BKUPN:
No
ONRQ
Put Pg on End of RPLQ
Return
SWPO4
Here to Assign Drum Adr
Dsk Rd Err in CST3 ?
No
Enough Room on Swap. Dev?
Yes
No
Have a Bkup Dsk Adr ?
No
Yes
Pg's Home Map Loc a Fork Pg Tb ?
Yes
No, File XB
Dsk Update Suppressed? (OFNDUD=1) ?
Yes
No
BKUPD pPF15
Swap to DSK
SWPOF1
Backup Adr in CST1 on DSK ?
No
BKUPO
Yes
Room on Drum DRUMP> 0 ?
No
Return
Yes
SWOFN:
DRMASN
Assign Drum Adr
Set Drum Adr as New Adr in CST1 & Put Former Adr. in OST
Pg to be Swapped Been Modified?
No
SWP05
Yes
Set Mod. Bit in DST Entry
SWPO2:
Clear Mod Bit & Store PSWIP Code in CST0 Entry
Increm. Writes in Progress (IOIP) Set Write Req.
DRMIO pPHY1
Initiate Drum Write
Increm. DRMWR
Return

PF14


Swap to Disk
BKUPD
BKUPD1:
Get Backup Adr from CST1
Disk or Drum?
No
BUGHLT
BKUP7:
Yes
Disk ?
Yes
BKUP3
No, Drum
GDSTX
Get DST Index
Get Next Level Bkup Adr from DST
Written Since Last Backup? (BWRBIT Set)
No
Yes
Set Modify Bit in CST0 Entry
Release DST Slot (Set to -1) & Put Former Contents of DST Slot into CST1 Entry
DASDRM
Deassign Drum Adr.
BKUP3
Err From Read ?
Yes
ONRQ
Pug Pg on End of RPLQ
Return
No
Pg Unchanged in CST0 ?
Yes
Flag SWPKPF = 0 ?
No
Yes
BKUPN:
OFRQ
Put Pg on Top of RPLQ
No
Clear Modify Bit in CSTO Entry
Request Write Increm DSKWR & IOIP
Clear DSKNB Bit in CST1 Entry if Set
Set Write-in-Progress (PSWIP) in Age Field of CST0
DSKIO pPHY1
Requests I/O to Disk
Return

PF15

Page Fault Handling Comments

PGTACC

(1) Checks if process has accrued more than or equal to the number of age ticks of GCRATE. Currently, this is set to 50, which implies 2 sec. of process virtual time (i.e., the age stamp is incremented every 40 ms of process run time).

NICCKS

(1) GNPBAS is currently initialized at system startup to zero and is incremented/decremented only when pages are locked/unlocked. It is currently only tested by NICCKS as well.

GETTPD

(2) The age field when used to hold the age stamp, will always have a value of 100 or greater. This checks if any of the lefthand 6 bits of the age field are set.

NIC

(3) Could take the ILRD path, for example, when OPENed file for write, but PMAPed for each of a nonexistent page. A page would have to be created which would then imply a write which was not enabled under PMAP.

(4) If file page faulted does not have its own SPT slot, but has to be mapped (using indirect pointer) via the index blk slot in the OFN area, then the index blk will be locked in core. (So can't be swapped in case of reschedule.)

(5) Note in the predispatch code that AC1 was stored in BHC + 1 and AC, P, which holds a push down list pointer, was saved in TRAPAP.

SWPINW

(6) SWPINW will invoke SWPIN to swap in a page into a page from the RPLQ. However, this same code can also be entered with different flag settings and be used to swap in a page into a page from the special memory queue (SPMQ), a queue used by the memory error handling code.

SWPOUT

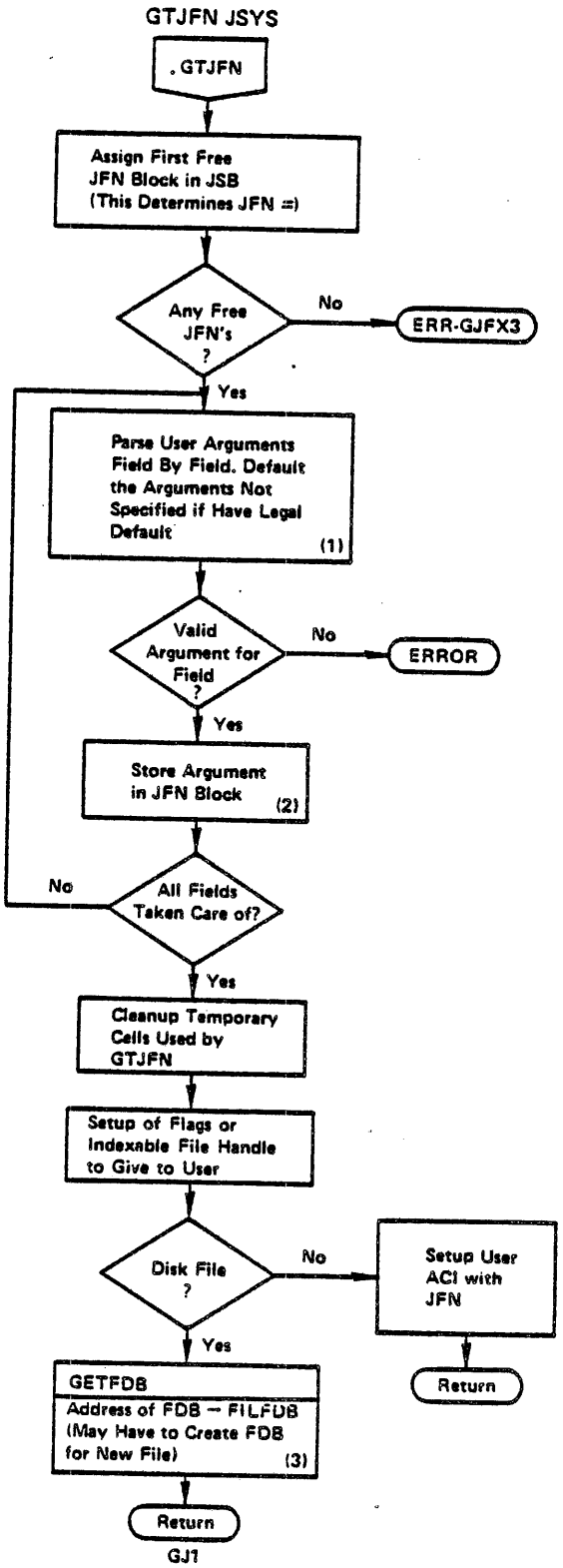
(7) SWPOUO is called from:

SWPOTO which clears the SWPKPF bit (for top of RPLQ) before calling SWPOUO and

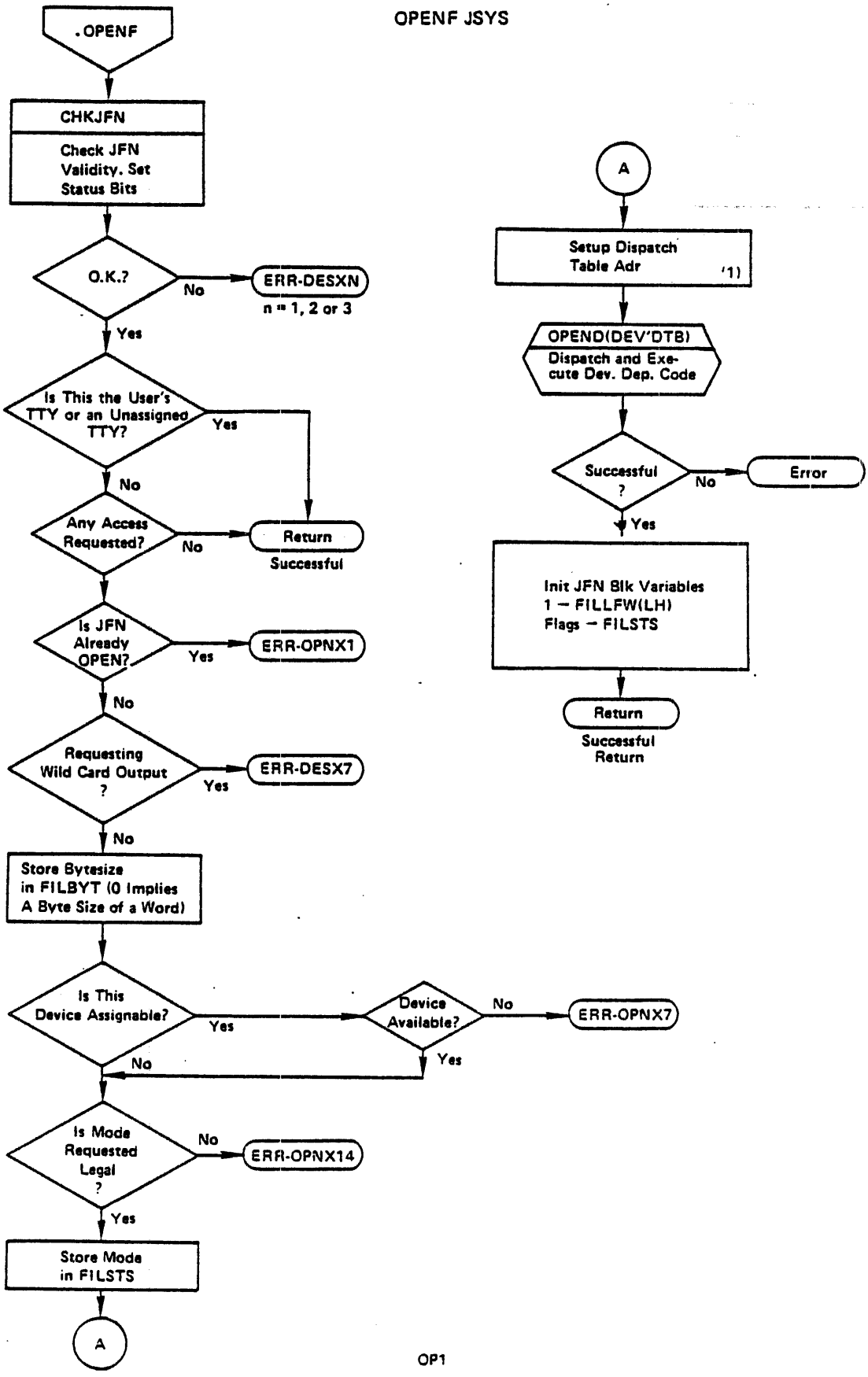
SWPOTK (called from the UPDPGS JSYS) which sets the SWPKPF bit (for end of RPLQ) before calling SWPOUO.

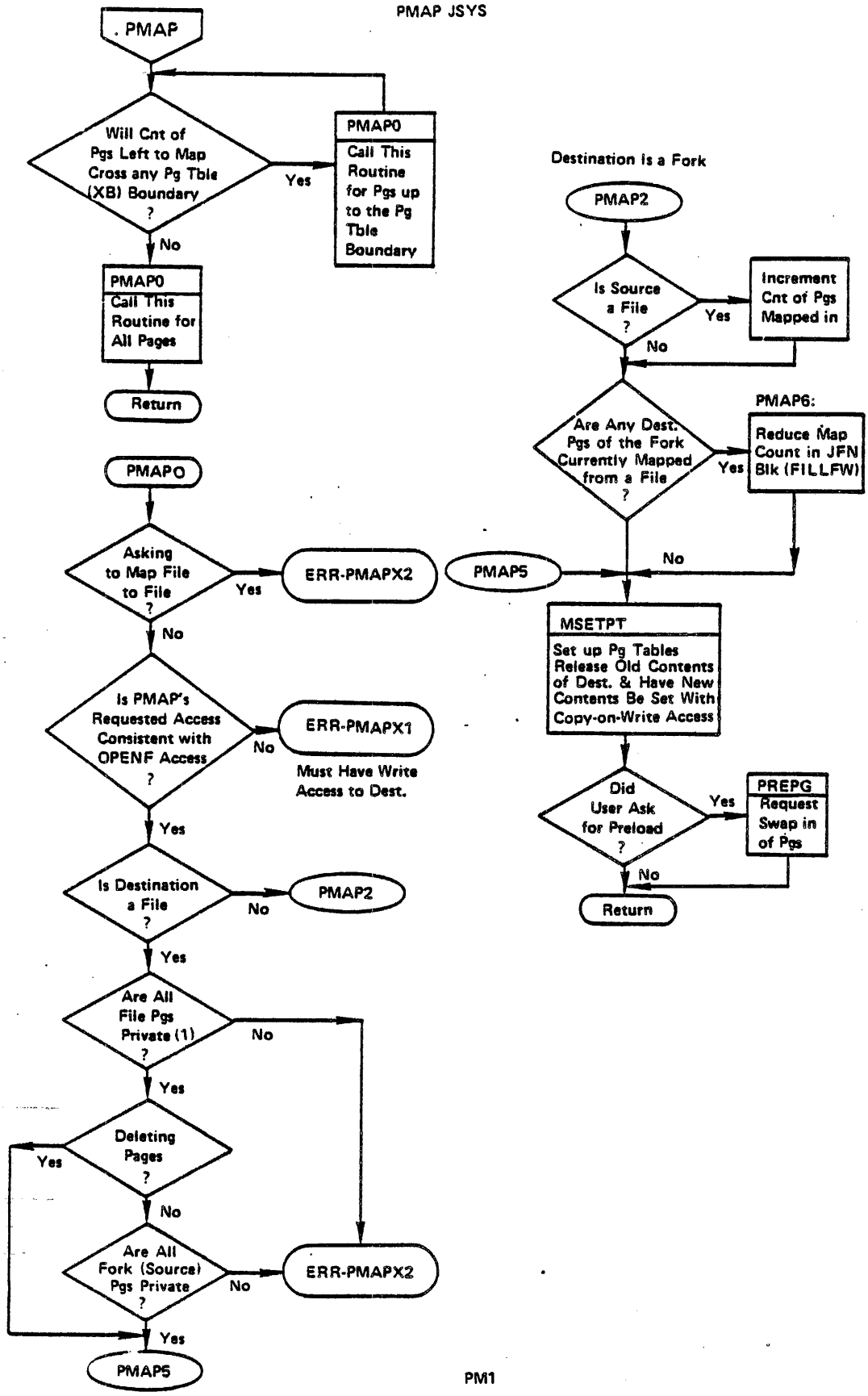
JSYS CALL FLOWCHARTS

DEVICE INDEPENDENT LEVEL

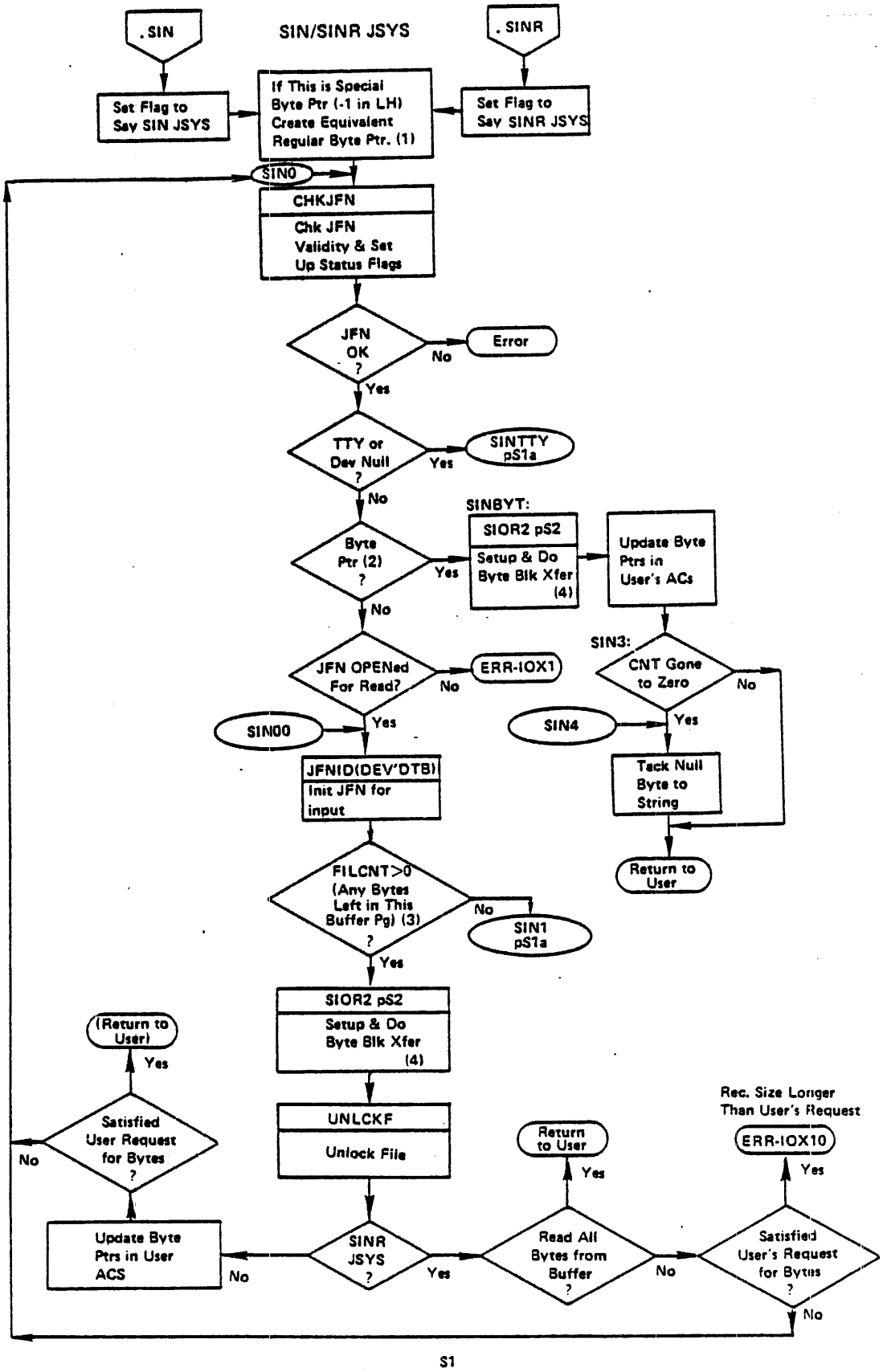

GTJFN JSYS
.GTJFN
Assign First Free JFN Block in JSB (This Determines JFN =)
Any Free JFN's ?
No
ERR-GJFX3
Yes
Parse User Arguments Field By Field. Default the Arguments Not Specified if Have Legal Default
(1)
Valid Argument for Field ?
No
ERROR
Yes
Store Argument in JFN Block
(2)
All Fields Taken Care of?
No
Yes
Cleanup Temporary Cells Used by GTJFN
Setup of Flags or Indexable File Handle to Give to User
Disk File ?
No
Setup User ACI with JFN
Return
Yes
GETFDB
Address of FDB — FILFDB (May Have to Create FDB for New File)
(3)
Return
GJ1

OPENF JSYS


.OPENF
CHKJFN
Check JFN Validity. Set Status Bits
O.K.?
No
ERR-DESXN
n = 1, 2 or 3
Yes
Is This the User's TTY or an Unassigned TTY?
Yes
No
Any Access Requested?
No
Return
Successful
Is JFN Already OPEN?
Yes
ERR-OPNX1
No
Requesting Wild Card Output ?
Yes
ERR-DESX7
No
Store Bytesize in FILBYT (0 Implies A Byte Size of a Word)
Is This Device Assignable?
Yes
Device Available?
No
ERR-OPNX7
Yes
No
Is Mode Requested Legal ?
No
ERR-OPNX14
Yes
Store Mode in FILSTS
A
A
Setup Dispatch Table Adr
(1)
OPEND(DEV'DTB)
Dispatch and Execute Dev. Dep. Code
Successful ?
No
Error
Yes
Init JFN Blk Variables
1 – FILLFW(LH)
Flags – FILSTS
Return
Successful Return

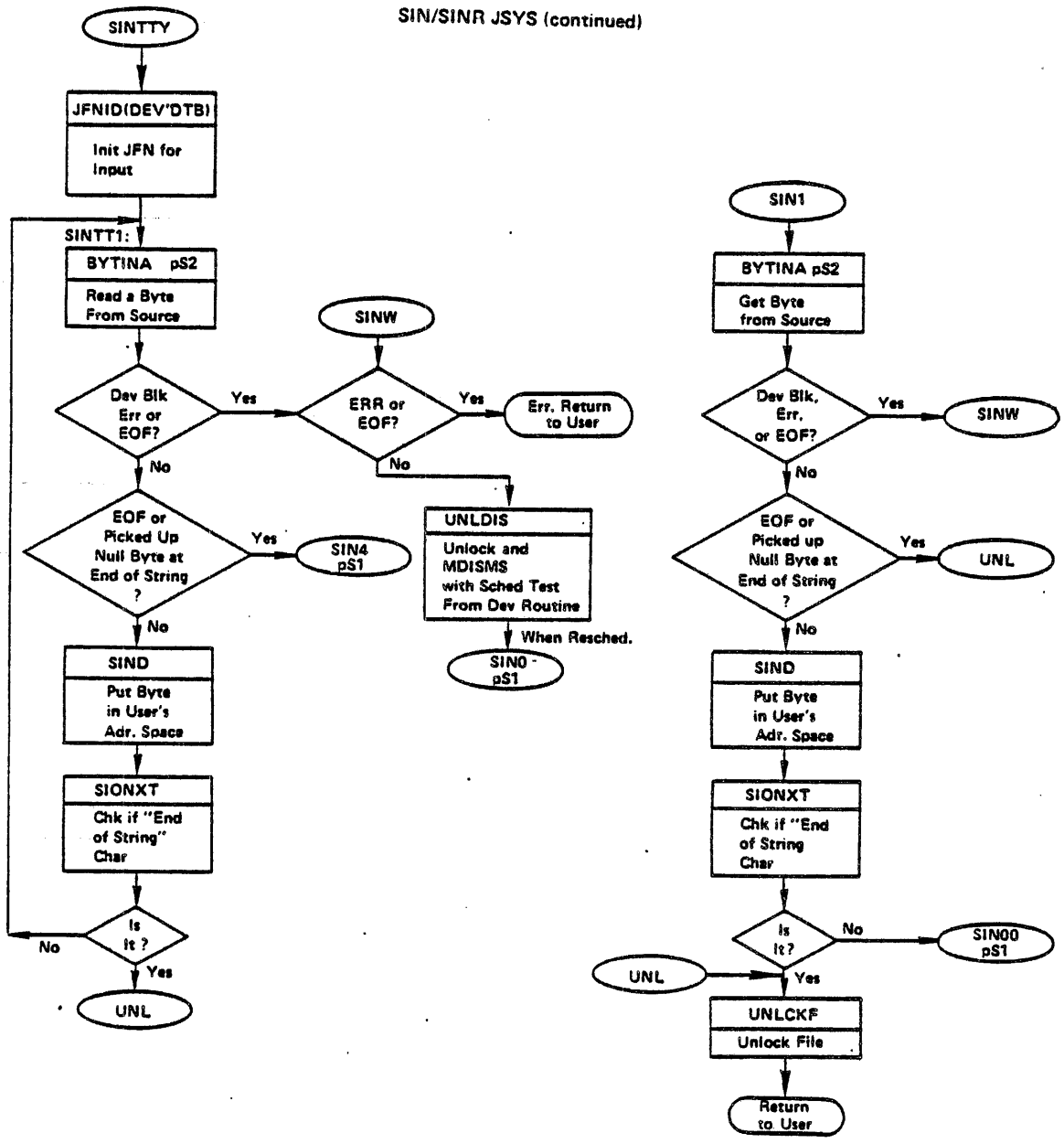

PMAP JSYS
. PMAP
Will Cnt of Pgs Left to Map Cross any Pg Tble (XB) Boundary ?
Yes
PMAP0
Call This Routine for Pgs up to the Pg Tble Boundary
No
PMAP0
Call This Routine for All Pages
Return
PMAP0
Asking to Map File to File ?
Yes
ERR-PMAPX2
No
Is PMAP's Requested Access Consistent with OPENF Access ?
No
ERR-PMAPX1
Must Have Write Access to Dest.
Yes
Is Destination a File ?
No
PMAP2
Yes
Are All File Pgs Private (1) ?
No
Yes
Deleting Pages ?
Yes
No
Are All Fork (Source) Pgs Private ?
No
ERR-PMAPX2
Yes
PMAP5
Destination is a Fork
PMAP2
Is Source a File ?
Yes
Increment Cnt of Pgs Mapped in
No
PMAP6:
Are Any Dest. Pgs of the Fork Currently Mapped from a File ?
Yes
Reduce Map Count in JFN Blk (FILLFW)
PMAP5
No
MSETPT
Set up Pg Tables Release Old Contents of Dest. & Have New Contents Be Set With Copy-on-Write Access
Did User Ask for Preload ?
Yes
PREPG
Request Swap in of Pgs
No
Return

PM1

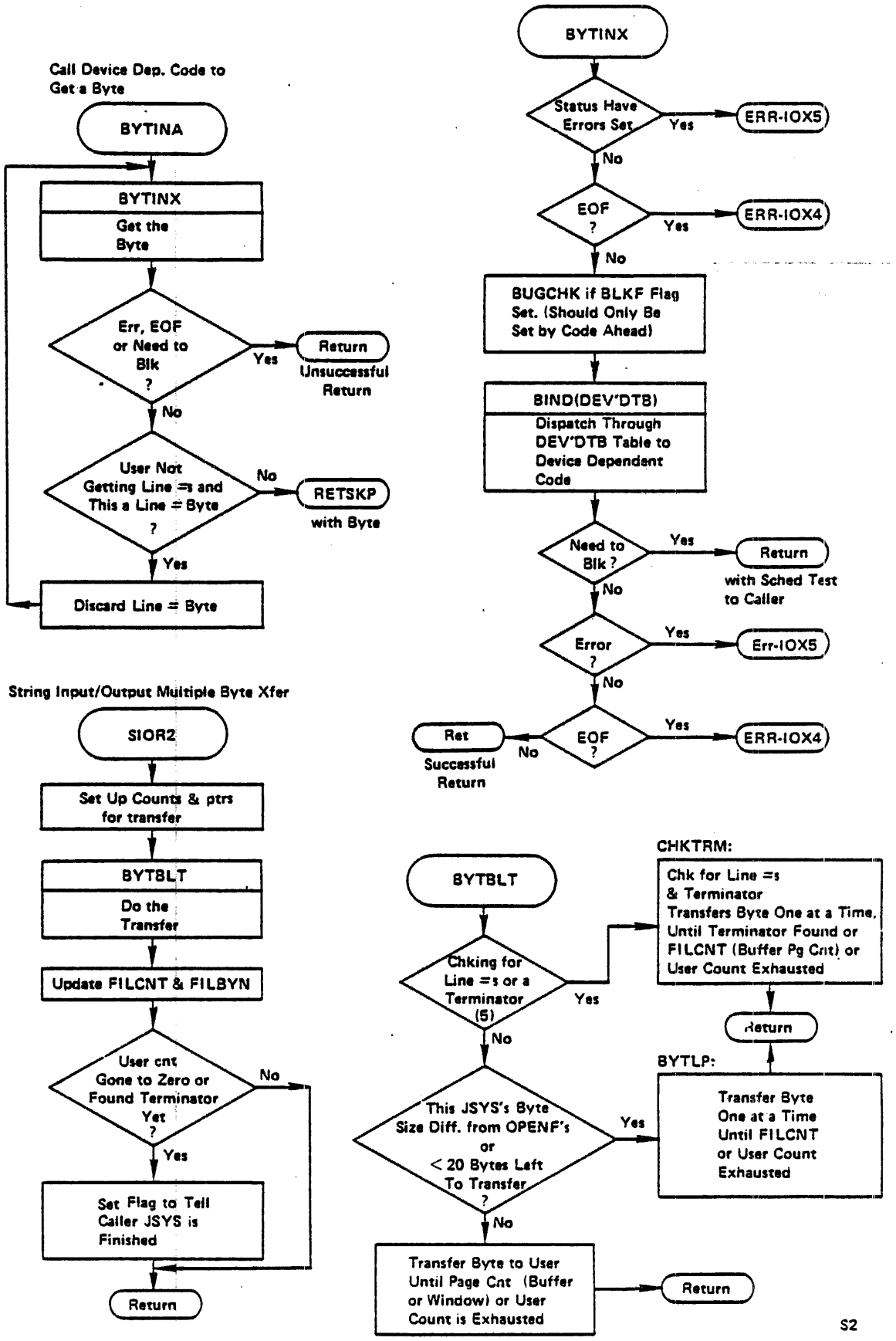

.SIN
SIN/SINR JSYS
.SINR
Set Flag to Say SIN JSYS
If This is Special Byte Ptr (-1 in LH) Create Equivalent Regular Byte Ptr. (1)
Set Flag to Say SINR JSYS
SIN0
CHKJFN
Chk JFN Validity & Set Up Status Flags
JFN OK ?
No
Error
Yes
TTY or Dev Null ?
Yes
SINTTY pS1a
No
SINBYT:
Byte Ptr (2) ?
Yes
SIOR2 pS2
Setup & Do Byte Blk Xfer (4)
Update Byte Ptrs in User's ACs
No
JFN OPENed For Read?
No
ERR-IOX1
Yes
SIN3:
CNT Gone to Zero
No
Yes
SIN00
SIN4
Tack Null Byte to String
JFNID(DEV'DTB)
Init JFN for input
Return to User
FILCNT>0 (Any Bytes Left in This Buffer Pg) (3) ?
No
SIN1 pS1a
Yes
SIOR2 pS2
Setup & Do Byte Blk Xfer (4)
(Return to User)
Yes
Satisfied User Request for Bytes ?
No
UNLCKF
Unlock File
Return to User
Yes
Rec. Size Longer Than User's Request
ERR-IOX10
Yes
Update Byte Ptrs in User ACS
No
SINR JSYS ?
Yes
Read All Bytes from Buffer ?
No
Satisfied User's Request for Bytes ?
No

S1

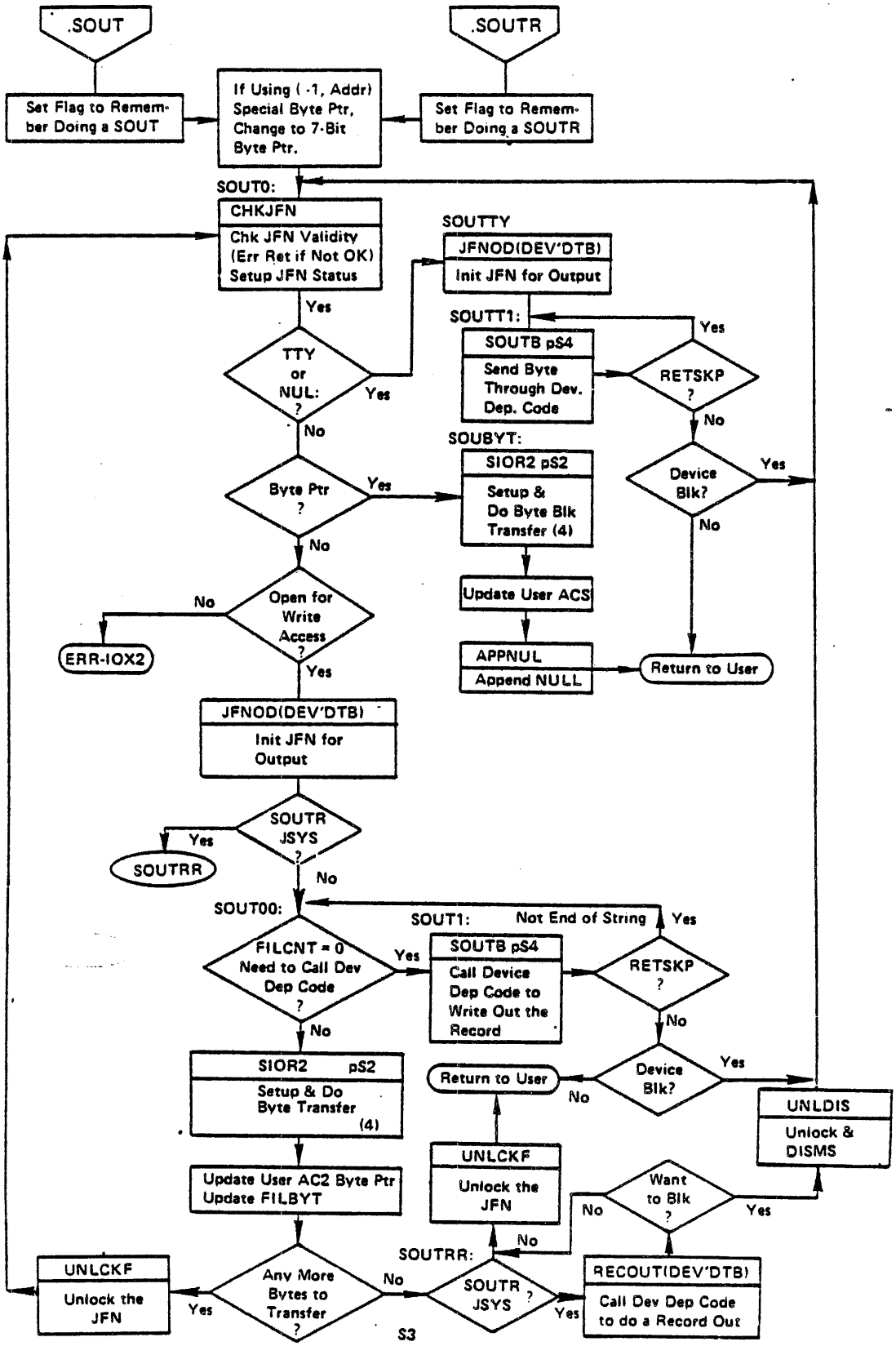
SIN/SINR JSYS (continued)

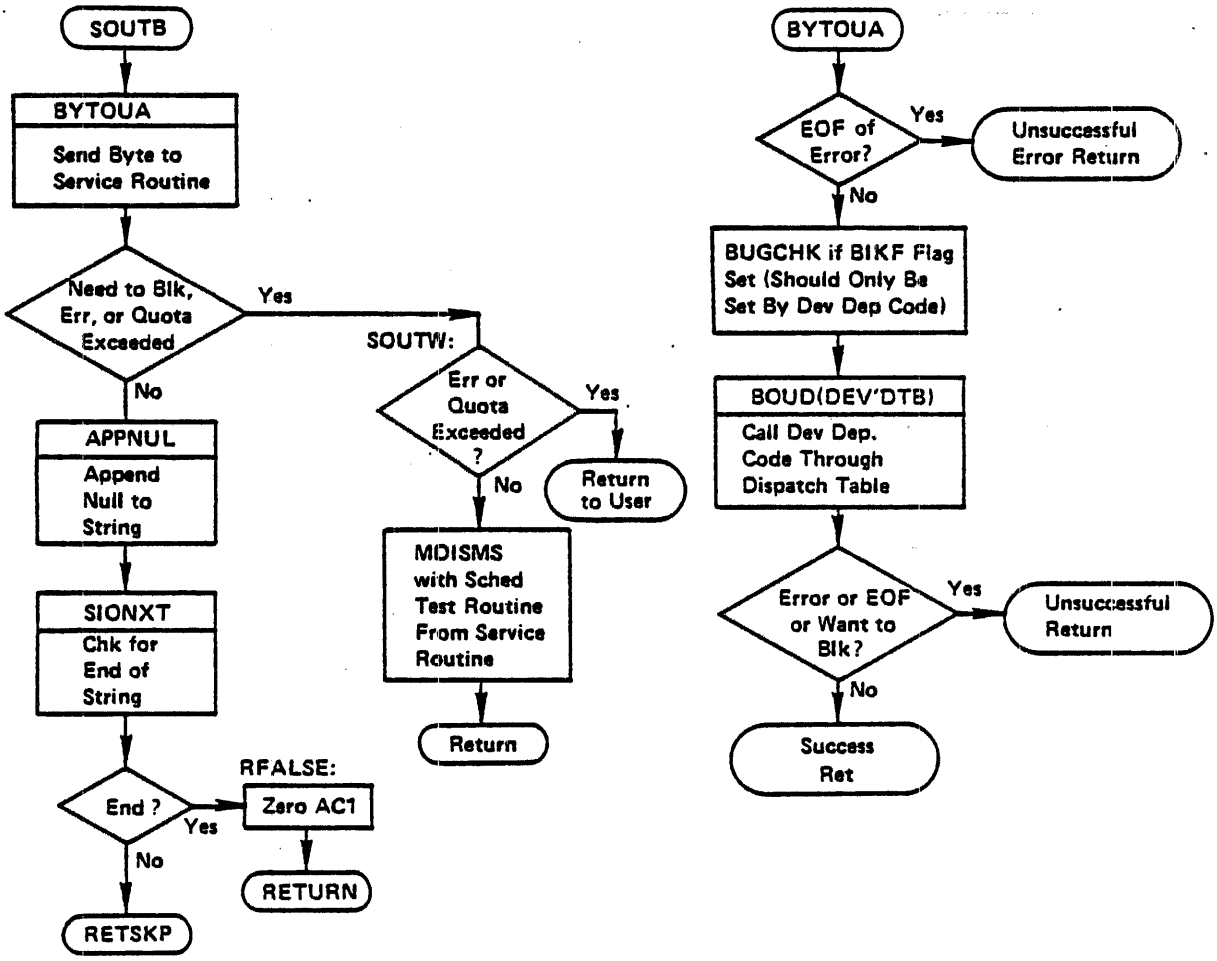


S1a

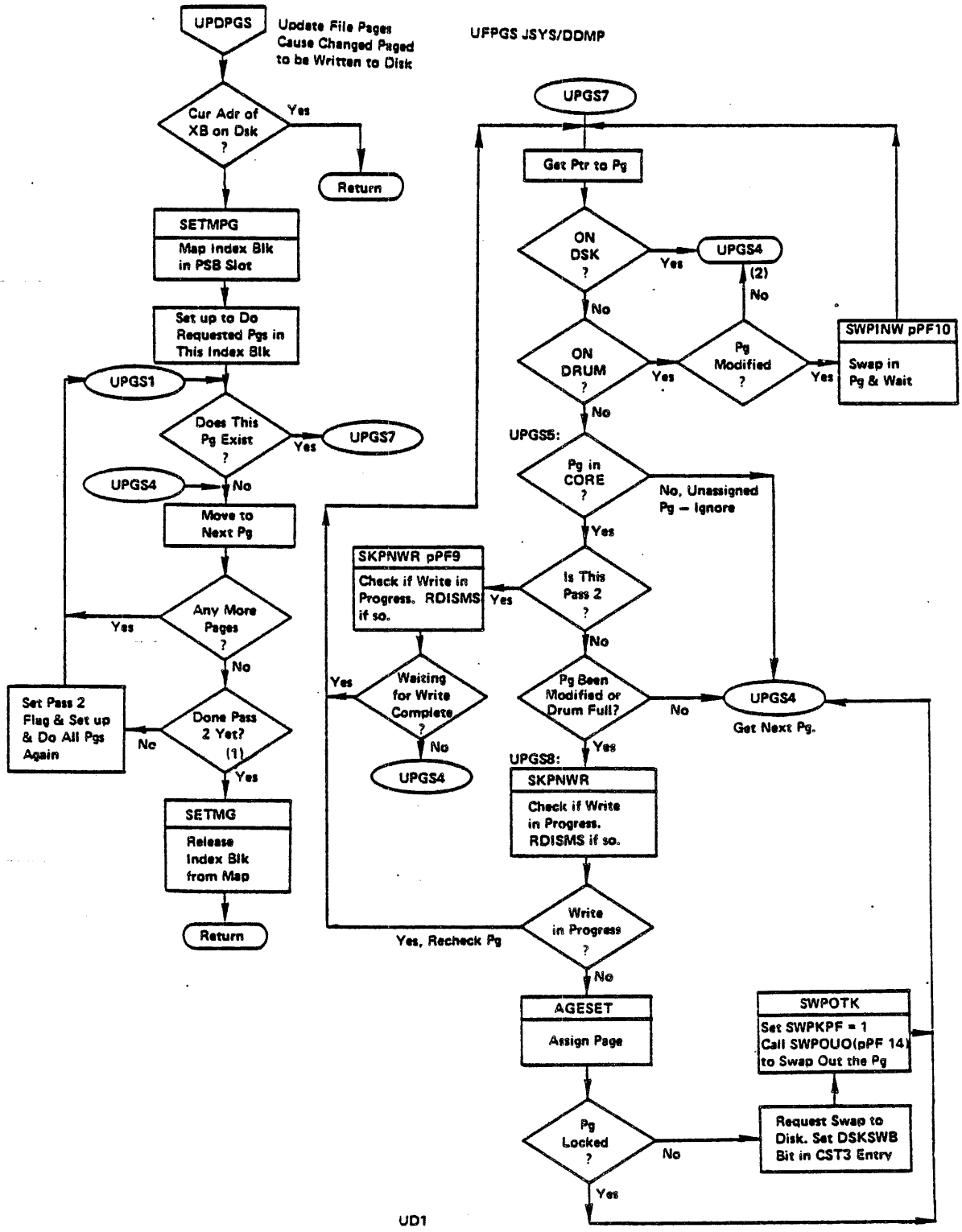

Call Device Dep. Code to Get a Byte
BYTINA
BYTINX
Get the Byte
Err, EOF or Need to Blk ?
Yes
Return Unsuccessful Return
No
User Not Getting Line =s and This a Line = Byte ?
No
RETSKP with Byte
Yes
Discard Line = Byte
String Input/Output Multiple Byte Xfer
SIOR2
Set Up Counts & ptrs for transfer
BYTBLT
Do the Transfer
Update FILCNT & FILBYN
User cnt Gone to Zero or Found Terminator Yet ?
No
Yes
Set Flag to Tell Caller JSYS is Finished
Return
BYTINX
Status Have Errors Set
Yes
ERR-IOX5
No
EOF ?
Yes
ERR-IOX4
No
BUGCHK if BLKF Flag Set. (Should Only Be Set by Code Ahead)
BIND(DEV'DTB)
Dispatch Through DEV'DTB Table to Device Dependent Code
Need to Blk ?
Yes
Return with Sched Test to Caller
No
Error ?
Yes
Err-IOX5
No
EOF ?
Yes
ERR-IOX4
No
Ret Successful Return
CHKTRM:
Chk for Line =s & Terminator Transfers Byte One at a Time, Until Terminator Found or FILCNT (Buffer Pg Cnt) or User Count Exhausted
Return
BYTBLT
Chking for Line =s or a Terminator (5)
Yes
No
BYTLP:
Transfer Byte One at a Time Until FILCNT or User Count Exhausted
This JSYS's Byte Size Diff. from OPENF's or < 20 Bytes Left To Transfer ?
Yes
No
Transfer Byte to User Until Page Cnt (Buffer or Window) or User Count is Exhausted
Return
S2

SOUT/SOUTR JSYS

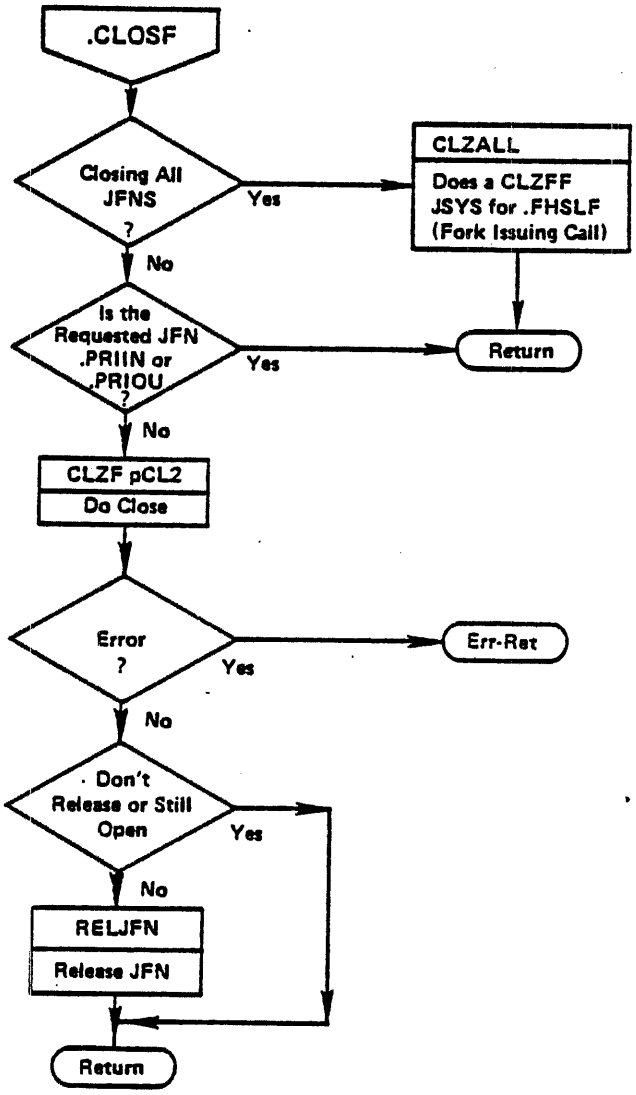



SOUTB
BYTOUA
Send Byte to Service Routine
Need to Blk, Err, or Quota Exceeded
Yes
No
APPNUL
Append Null to String
SIONXT
Chk for End of String
End ?
Yes
No
RETSKP
RFALSE:
Zero AC1
RETURN
SOUTW:
Err or Quota Exceeded ?
Yes
No
Return to User
MDISMS with Sched Test Routine From Service Routine
Return
BYTOUA
EOF of Error?
Yes
Unsuccessful Error Return
No
BUGCHK if BIKF Flag Set (Should Only Be Set By Dev Dep Code)
BOUD(DEV'DTB)
Call Dev Dep. Code Through Dispatch Table
Error or EOF or Want to Blk?
Yes
Unsuccessful Return
No
Success Ret

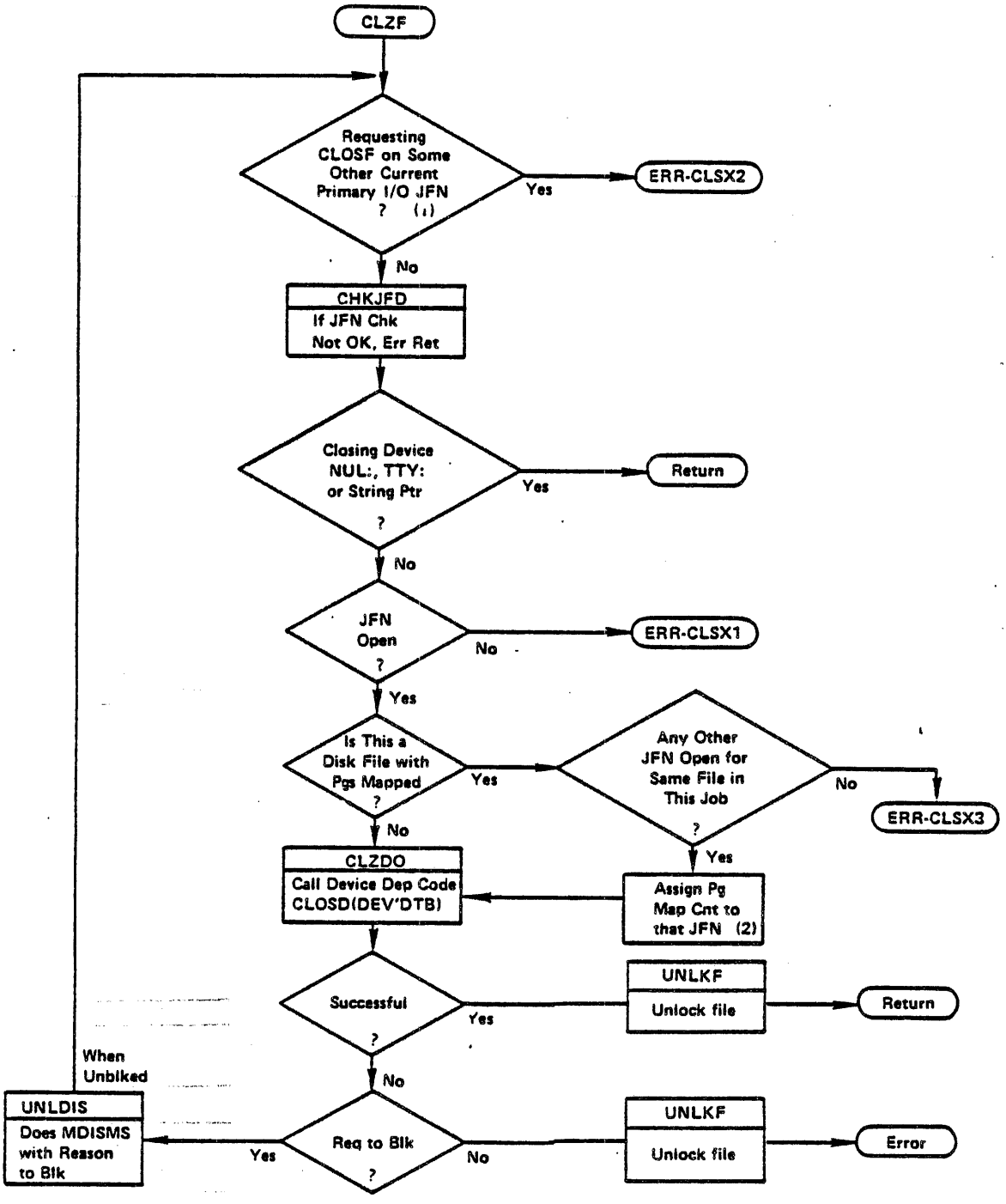
S4


UPDPGS
Update File Pages Cause Changed Paged to be Written to Disk
Cur Adr of XB on Dsk ?
Yes
Return
SETMPG
Map Index Blk in PSB Slot
Set up to Do Requested Pgs in This Index Blk
UPGS1
Does This Pg Exist ?
Yes
UPGS7
UPGS4
No
Move to Next Pg
Any More Pages ?
Yes
No
Set Pass 2 Flag & Set up & Do All Pgs Again
Done Pass 2 Yet? (1)
No
Yes
SETMG
Release Index Blk from Map
Return
UFPGS JSYS/DDMP
UPGS7
Get Ptr to Pg
ON DSK ?
Yes
UPGS4
(2)
No
No
ON DRUM ?
Yes
Pg Modified ?
Yes
SWPINW pPF10
Swap in Pg & Wait
No
UPGS5:
Pg in CORE ?
No, Unassigned Pg – Ignore
Yes
SKPNWR pPF9
Check if Write in Progress. RDISMS if so.
Is This Pass 2 ?
Yes
No
Yes
Waiting for Write Complete ?
No
UPGS4
Pg Been Modified or Drum Full?
No
UPGS4
Get Next Pg.
Yes
UPGS8:
SKPNWR
Check if Write in Progress. RDISMS if so.
Write in Progress ?
Yes, Recheck Pg
No
AGESET
Assign Page
SWPOTK
Set SWPKPF = 1
Call SWPOUO(pPF 14) to Swap Out the Pg
Pg Locked ?
No
Request Swap to Disk. Set DSKSWB Bit in CST3 Entry
Yes
UD1

CLOSF JSYS



CL1


CLZF
Requesting CLOSF on Some Other Current Primary I/O JFN ? (1)
Yes
ERR-CLSX2
No
CHKJFD
If JFN Chk Not OK, Err Ret
Closing Device NUL:, TTY: or String Ptr ?
Yes
Return
No
JFN Open ?
No
ERR-CLSX1
Yes
Is This a Disk File with Pgs Mapped ?
Yes
Any Other JFN Open for Same File in This Job ?
No
ERR-CLSX3
No
Yes
CLZDO
Call Device Dep Code CLOSD(DEV'DTB)
Assign Pg Map Cnt to that JFN (2)
Successful ?
Yes
UNLKF
Unlock file
Return
No
When Unblked
UNLDIS
Does MDISMS with Reason to Blk
Yes
Req to Blk ?
No
UNLKF
Unlock file
Error

CL2

GTJFN Comments

(1) This code is looking for a file specification of the form:

Dev:Directory Name,type,gen;T(temporary);P(protection);A(account

One or more fields can be defined by logical names. If any fields are omitted from the specification, the system will default the values as follows:

Device	DSK:
Directory	Connected directory
Name	No default for disk Null for other devices
Generation	Highest existing for input Next highest for output
Protection	As specified for directory or protection of next lower generation
Account	Current user account

(2) The internal GTJFN code uses several locations in the JFN block as temporary cells. These locations have two names in the JFN block table descriptions. The JFN block storage locations set up or used by GTJFN are:

FILLCK*	FILDDN	FILNEN
FILTMP*	FILPRT	FILVER
FILACT*	FILSTS1	FILCOD (LH)
FILOPT*	FILLNM*	
FILDEV	FILDNM	

*Used internally only by the GTJFN JSYS.

(3) The creation process of the FDB simply asks for space in the directory for the FDB.

.OPENF Comments

(1) Cell FILDEV in the JFN blk has the device dispatch table address. For example, for disk, GTJFN sets the dispatch table address to DSKDTB. If spooling to disk, GTJFN sets the dispatch table address to SPLDTB, but the OPENF code changes the dispatch table address to DSKDTB and sets up a file specification in the JFN block.

(1) TOPS-20 allows a user to specify a special byte pointer of -1,, Address which is interpreted as a 7-bit byte size beginning on the word boundary, Address.

(2) A user can do I/O from one place to another in core by specifying byte pointers for both source and destination. This differs from BLT in that the use can transfer on non-word boundaries.

(3) For disk files, FILCNT will be the number of bytes remaining in the window page. For magtape and other devices it will be the number of bytes remaining in the current page of the buffer.

(4) The routine BYTBLT only moves data up to the page boundary of the current buffer page.

(5) If the user has not specified OF%PLN in the OPENF, line numbers are stripped off the beginning of each line. (See SIN JSYS in Monitor Calls manual for definition of terminator.)

.PMAP Comments

(1) A page is private if it is not shared between a file and a fork.

UPDPGS Comments

(1) Routine scans page table twice: first time to request writes on all changed pages. Second time to wait for completion of writes. (This is faster than waiting for each write to complete as it is requested.)

(2) If page has not been modified, a check is made to see if the drum is full and if so, to release this page back to the drum. The map pointer to the page will be changed to its disk address.

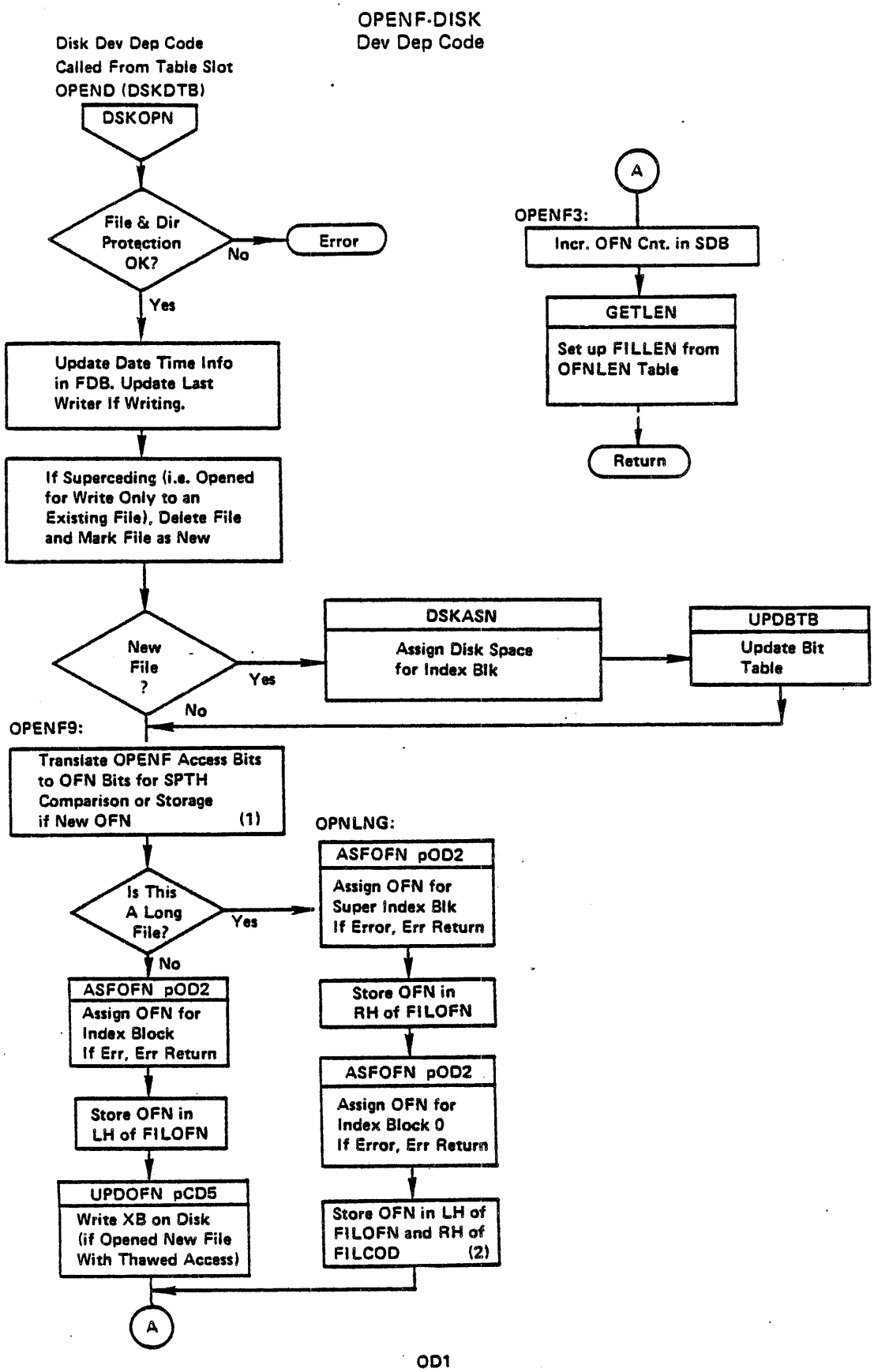
CLOSF Comments

(1) If user has switched primary I/O to some other JFN and attempts to close it, an error results.

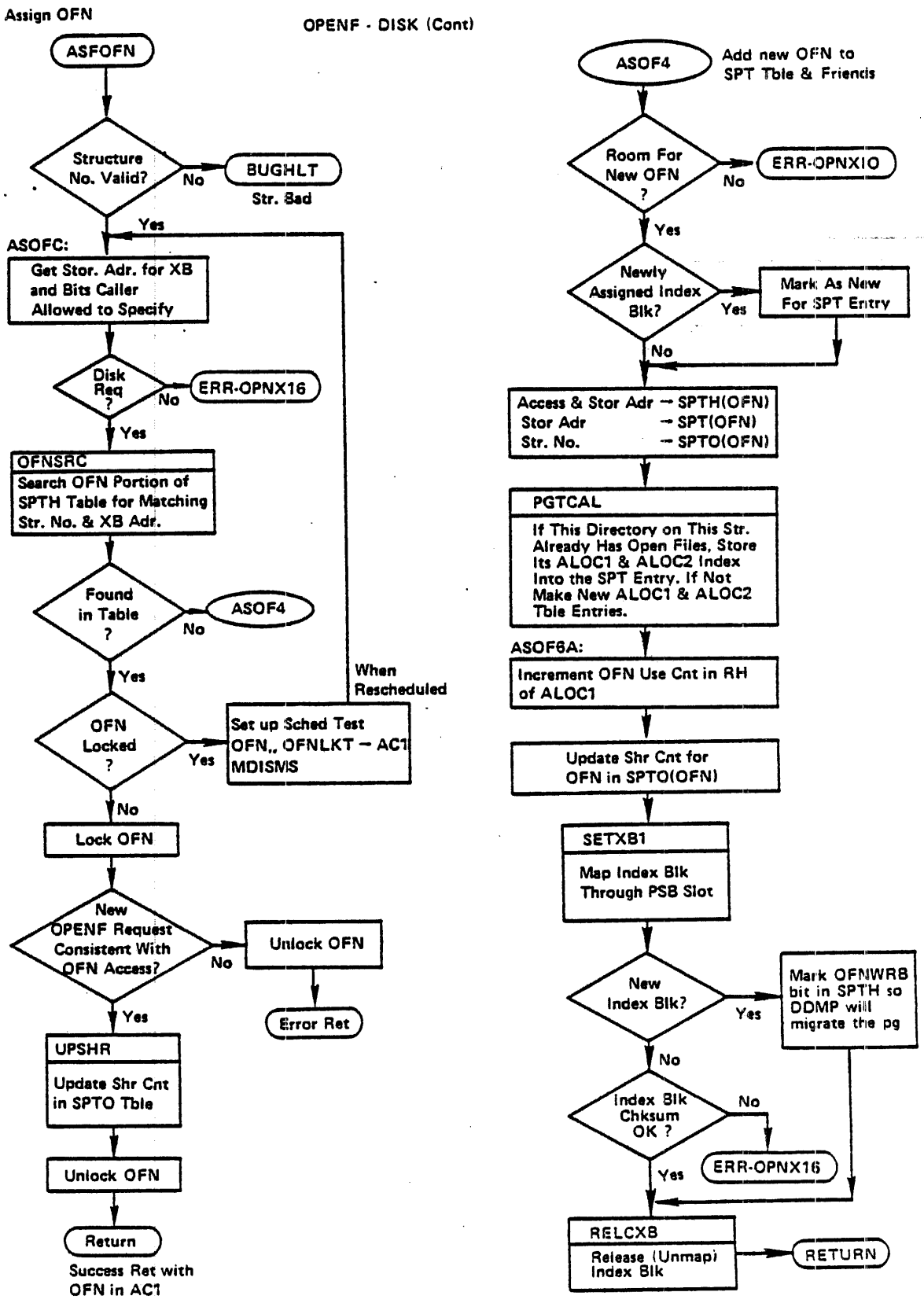
(2) The page map count in FILFW reflects the number of pages mapped and a CLOSF can't be done on a file if this count is greater than Ø .

JSYS CALL FLOWCHARTS

DSK DEPENDENT LEVEL

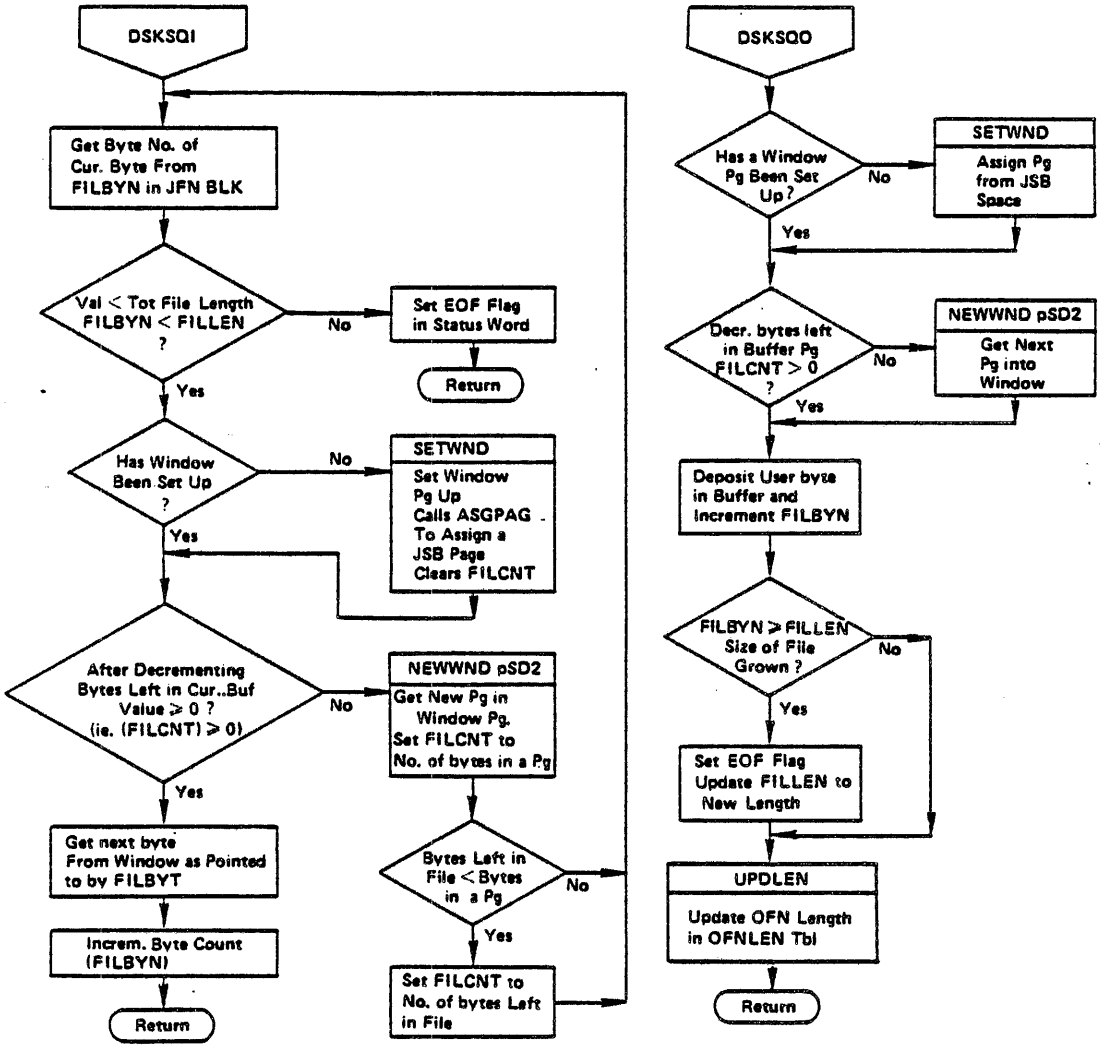

OPENF-DISK
Dev Dep Code
Disk Dev Dep Code
Called From Table Slot
OPEND (DSKDTB)
DSKOPN
File & Dir Protection OK?
No
Error
Yes
Update Date Time Info in FDB. Update Last Writer If Writing.
If Superceding (i.e. Opened for Write Only to an Existing File), Delete File and Mark File as New
New File ?
Yes
DSKASN
Assign Disk Space for Index Blk
UPDBTB
Update Bit Table
No
OPENF9:
Translate OPENF Access Bits to OFN Bits for SPTH Comparison or Storage if New OFN (1)
Is This A Long File?
Yes
No
ASFOFN pOD2
Assign OFN for Index Block If Err, Err Return
Store OFN in LH of FILOFN
UPDOFN pCD5
Write XB on Disk (if Opened New File With Thawed Access)
A
OPNLNG:
ASFOFN pOD2
Assign OFN for Super Index Blk If Error, Err Return
Store OFN in RH of FILOFN
ASFOFN pOD2
Assign OFN for Index Block 0 If Error, Err Return
Store OFN in LH of FILOFN and RH of FILCOD (2)
A
OPENF3:
Incr. OFN Cnt. in SDB
GETLEN
Set up FILLEN from OFNLEN Table
Return

OD1

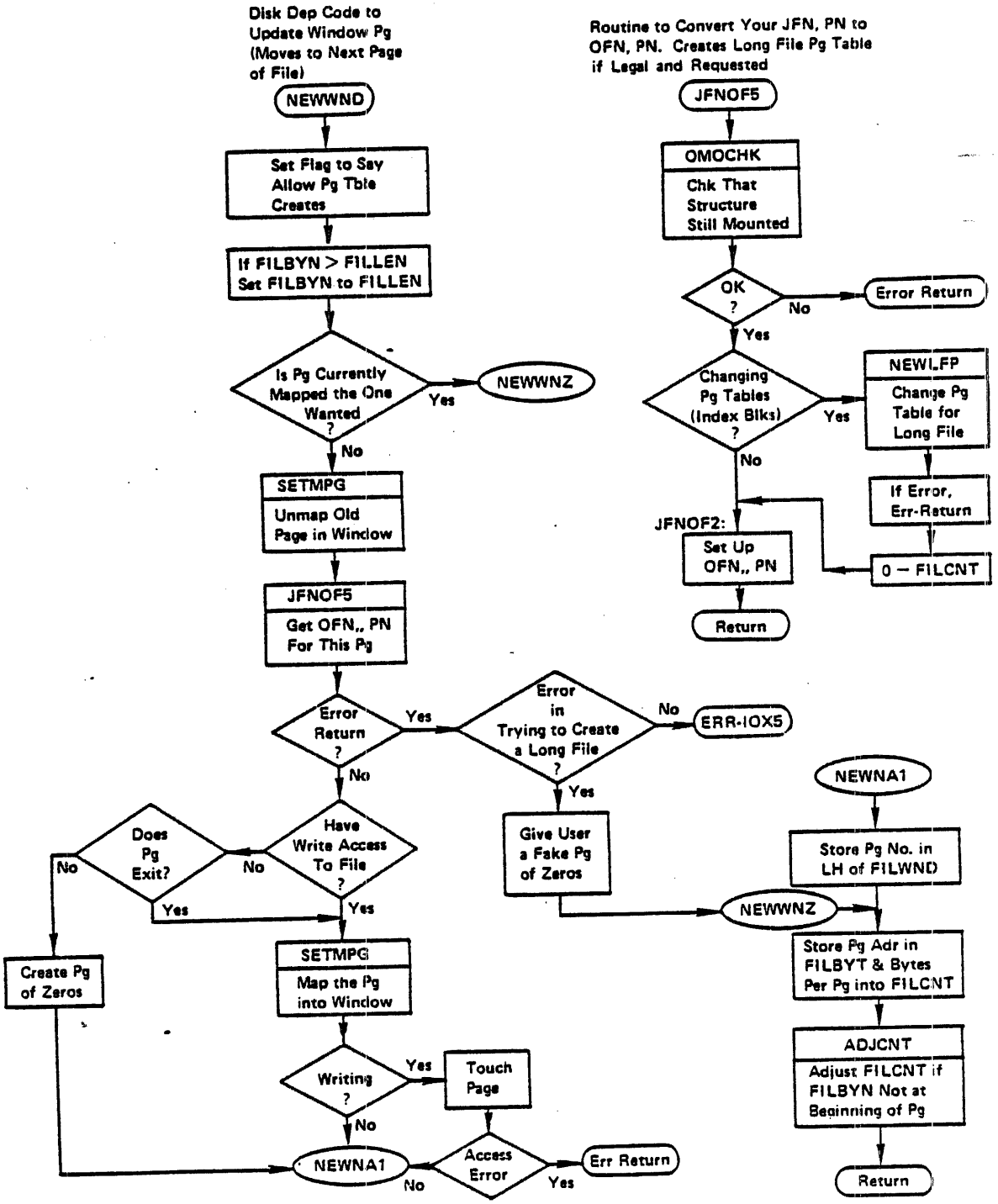

Assign OFN
OPENF - DISK (Cont)
ASFOFN
Structure No. Valid?
No
BUGHLT
Str. Bad
Yes
ASOFC:
Get Stor. Adr. for XB and Bits Caller Allowed to Specify
Disk Req ?
No
ERR-OPNX16
Yes
OFNSRC
Search OFN Portion of SPTH Table for Matching Str. No. & XB Adr.
Found in Table ?
No
ASOF4
Yes
When Rescheduled
OFN Locked ?
Yes
Set up Sched Test OFN,, OFNLKT — AC1 MDISMS
No
Lock OFN
New OPENF Request Consistent With OFN Access?
No
Unlock OFN
Error Ret
Yes
UPSHR
Update Shr Cnt in SPTO Tble
Unlock OFN
Return
Success Ret with OFN in AC1
ASOF4
Add new OFN to SPT Tble & Friends
Room For New OFN ?
No
ERR-OPNX10
Yes
Newly Assigned Index Blk?
Yes
Mark As New For SPT Entry
No
Access & Stor Adr → SPTH(OFN)
Stor Adr → SPT(OFN)
Str. No. → SPTO(OFN)
PGTCAL
If This Directory on This Str. Already Has Open Files, Store Its ALOC1 & ALOC2 Index Into the SPT Entry. If Not Make New ALOC1 & ALOC2 Tble Entries.
ASOF6A:
Increment OFN Use Cnt in RH of ALOC1
Update Shr Cnt for OFN in SPTO(OFN)
SETXB1
Map Index Blk Through PSB Slot
New Index Blk?
Yes
Mark OFNWRB bit in SPTH so DDMP will migrate the pg
No
Index Blk Chksum OK ?
No
ERR-OPNX16
Yes
RELCXB
Release (Unmap) Index Blk
RETURN

OD2

SEQUENTIAL I/O-DSK
(String & Byte Dev Dep Code)

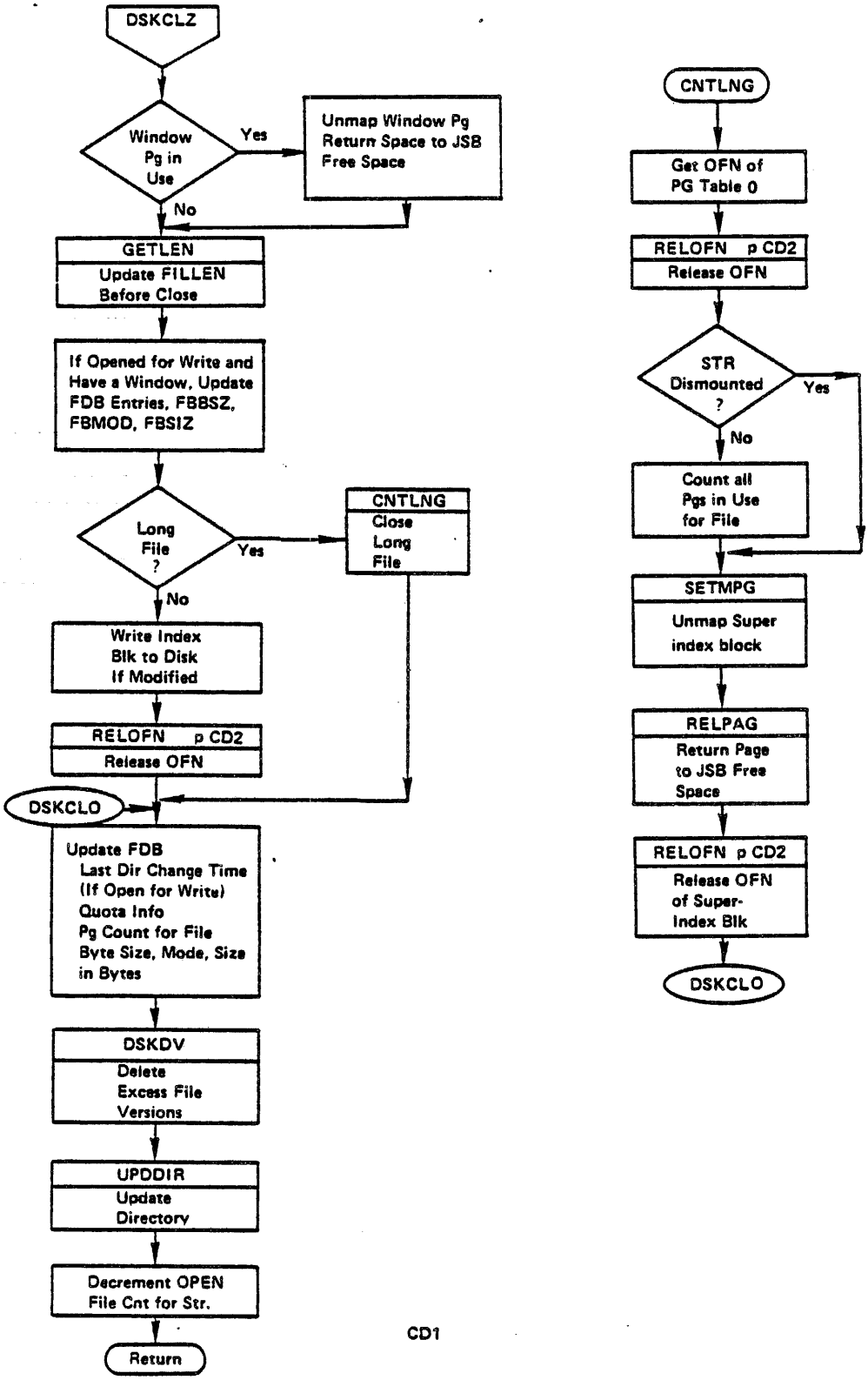


SD1


Disk Dep Code to Update Window Pg (Moves to Next Page of File)
NEWWND
Set Flag to Say Allow Pg Tble Creates
If FILBYN > FILLEN Set FILBYN to FILLEN
Is Pg Currently Mapped the One Wanted ?
Yes
NEWWNZ
No
SETMPG
Unmap Old Page in Window
JFNOF5
Get OFN,, PN For This Pg
Error Return ?
Yes
Error in Trying to Create a Long File ?
No
ERR-IOX5
Yes
Give User a Fake Pg of Zeros
No
Have Write Access To File ?
No
Does Pg Exit?
No
Yes
Yes
Create Pg of Zeros
SETMPG
Map the Pg into Window
Writing ?
Yes
Touch Page
No
NEWNA1
Access Error
No
Yes
Err Return
Routine to Convert Your JFN, PN to OFN, PN. Creates Long File Pg Table if Legal and Requested
JFNOF5
OMOCHK
Chk That Structure Still Mounted
OK ?
No
Error Return
Yes
Changing Pg Tables (Index Blks) ?
Yes
NEWLFP
Change Pg Table for Long File
No
If Error, Err-Return
JFNOF2:
Set Up OFN,, PN
0 — FILCNT
Return
NEWNA1
Store Pg No. in LH of FILWND
NEWWNZ
Store Pg Adr in FILBYT & Bytes Per Pg into FILCNT
ADJCNT
Adjust FILCNT if FILBYN Not at Beginning of Pg
Return

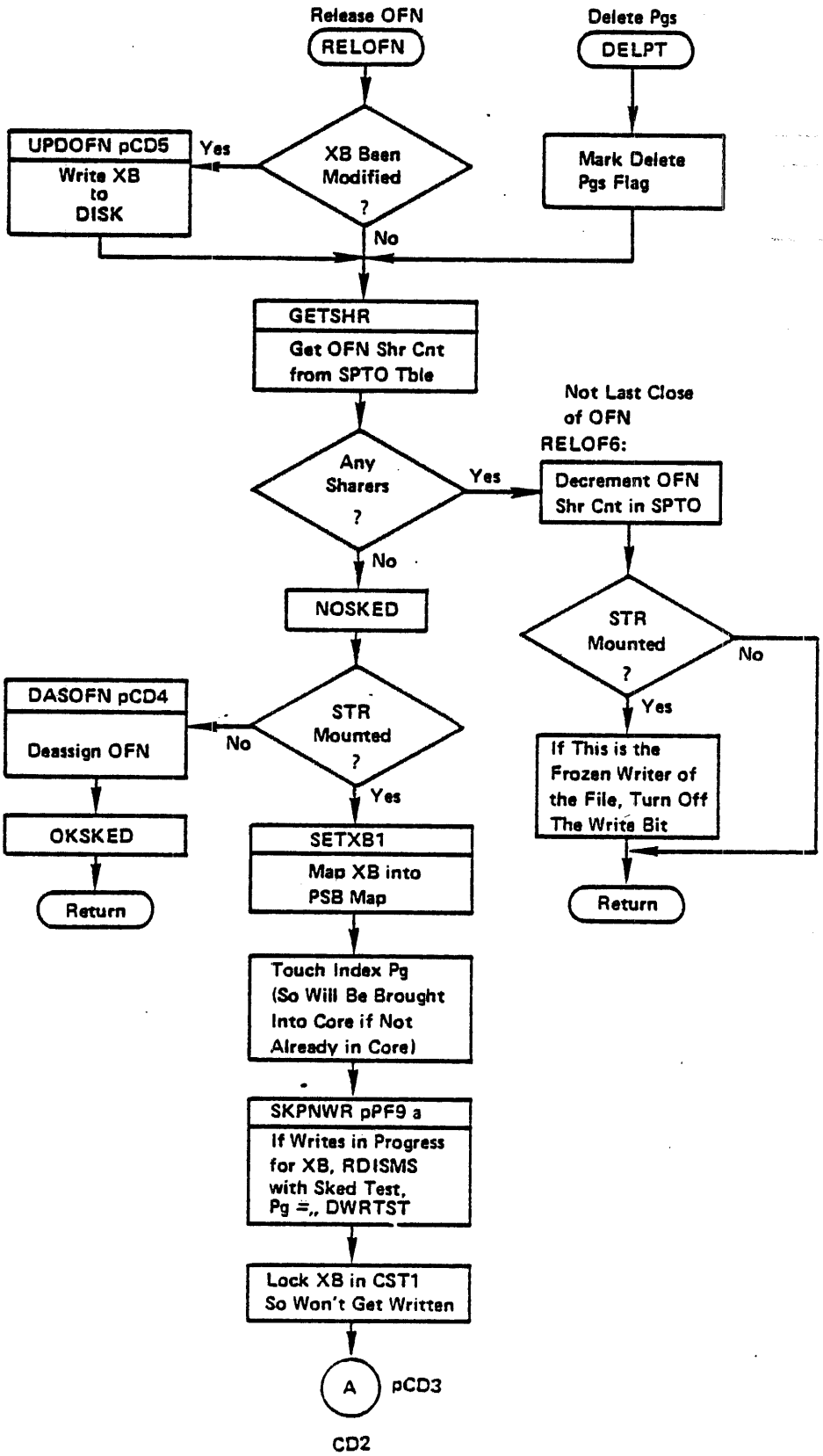
SD2

CLOSF – DSK
Dev. Dep. Code



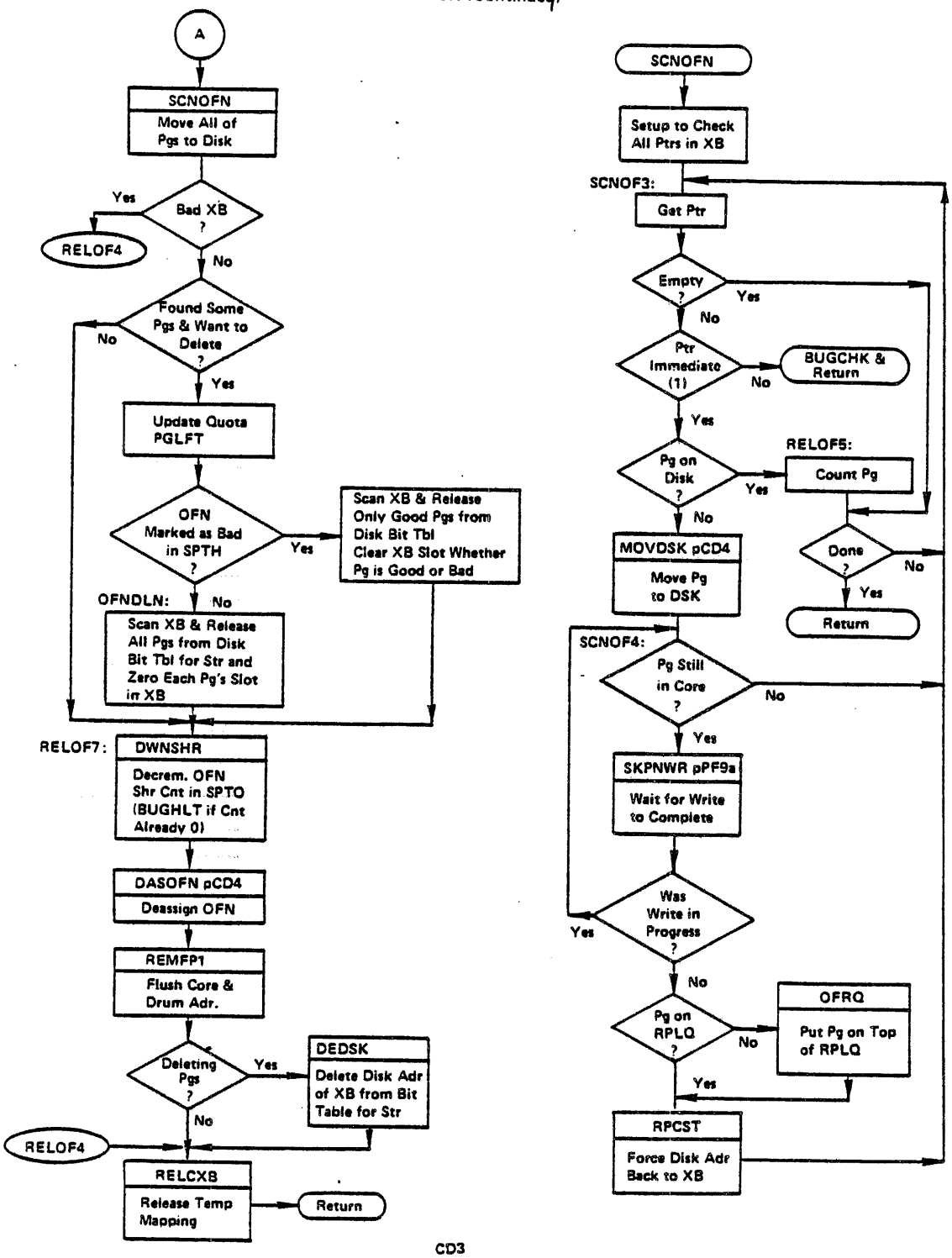
CD1

CLOSF-DSK (Continued)

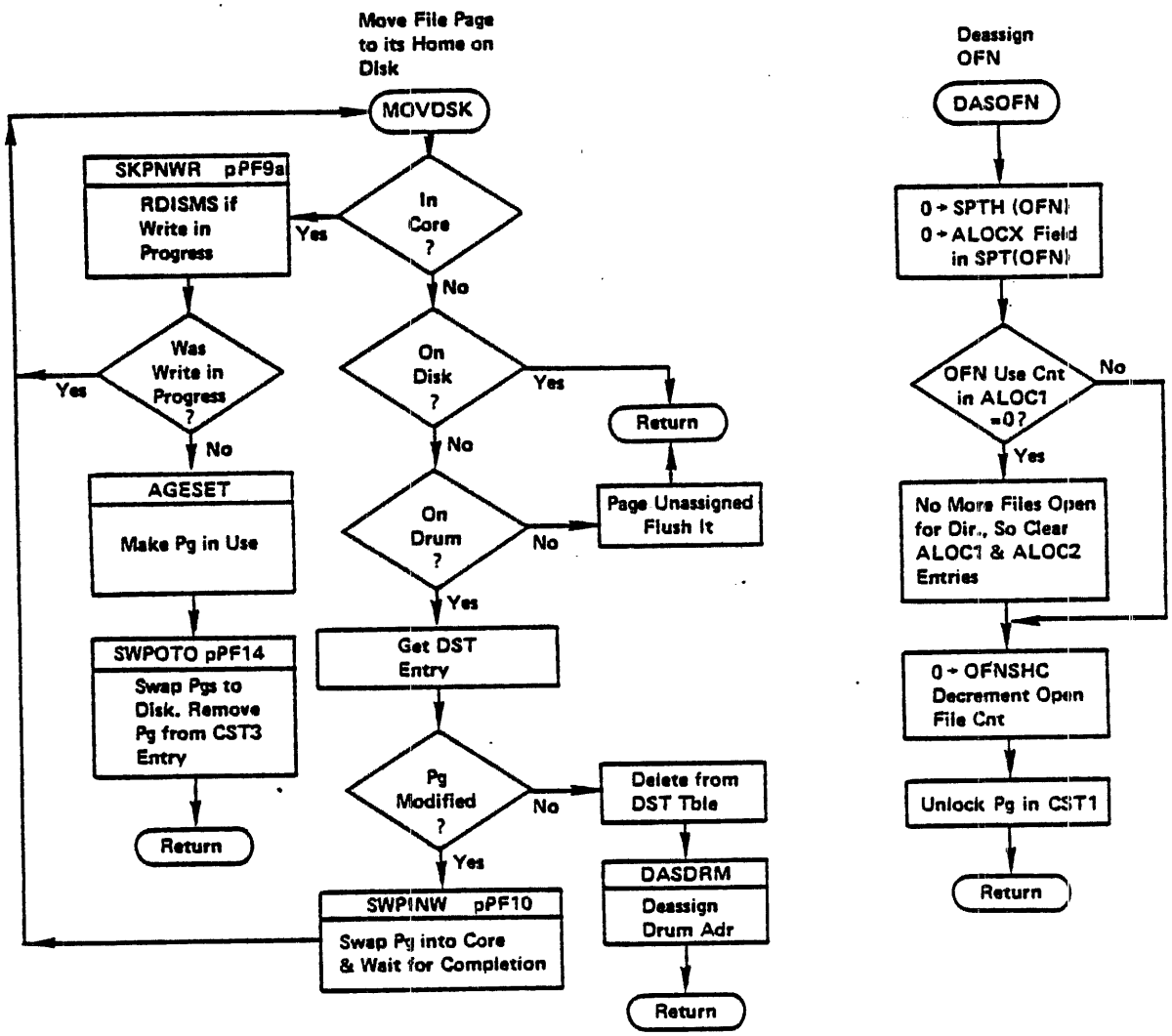


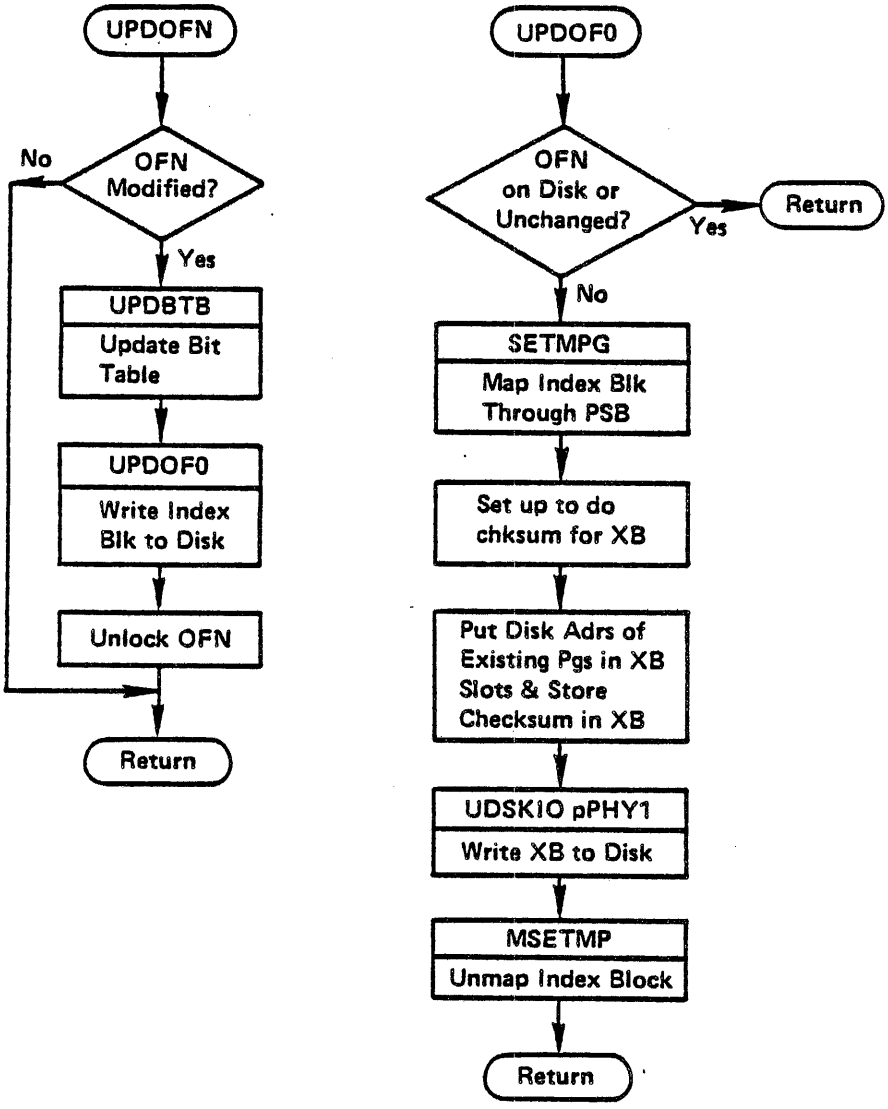
CD2

CLOSF-DSK (Continued)



CD3


Move File Page to its Home on Disk
MOVDSK
In Core ?
Yes
SKPNWR pPF9a
RDISMS if Write in Progress
No
On Disk ?
Yes
Return
No
On Drum ?
No
Page Unassigned Flush It
Yes
Get DST Entry
Pg Modified ?
No
Delete from DST Tble
Yes
SWPINW pPF10
Swap Pg into Core & Wait for Completion
DASDRM
Deassign Drum Adr
Return
Was Write in Progress ?
Yes
No
AGESET
Make Pg in Use
SWPOTO pPF14
Swap Pgs to Disk. Remove Pg from CST3 Entry
Return
Deassign OFN
DASOFN
0 → SPTH (OFN)
0 → ALOCX Field in SPT(OFN)
OFN Use Cnt in ALOC1 =0?
No
Yes
No More Files Open for Dir., So Clear ALOC1 & ALOC2 Entries
0 → OFNSHC
Decrement Open File Cnt
Unlock Pg in CST1
Return


UPDOFN
OFN Modified?
No
Yes
UPDBTB
Update Bit Table
UPDOF0
Write Index Blk to Disk
Unlock OFN
Return
UPDOF0
OFN on Disk or Unchanged?
Yes
Return
No
SETMPG
Map Index Blk Through PSB
Set up to do chksum for XB
Put Disk Adrs of Existing Pgs in XB Slots & Store Checksum in XB
UDSKIO pPHY1
Write XB to Disk
MSETMP
Unmap Index Block
Return

OPENF-DISK Comments

(1) OFN bits: 0=read, 10=write, 11=thawed, 01=restricted

(2) For a long file, the OFN of index block Ø is remembered in the JFN blk and used as the identity of the file by the ENQ/DEQ facility.

CLOSF-DISK Comments

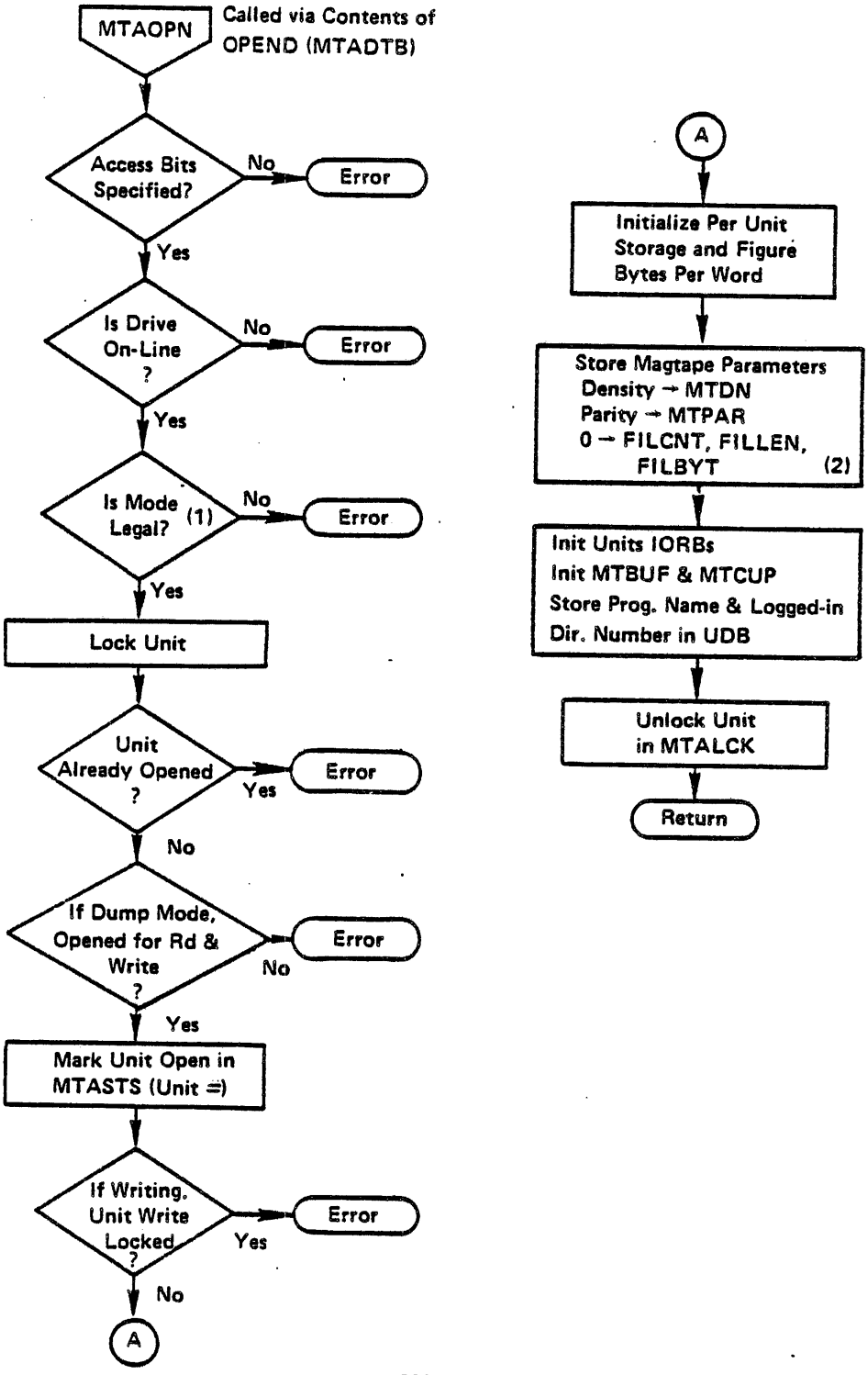
(1) All storage addresses placed in an index blk have the pointer type field set to immediate.

JSYS's CALLS

MTA DEPENDENT LEVEL

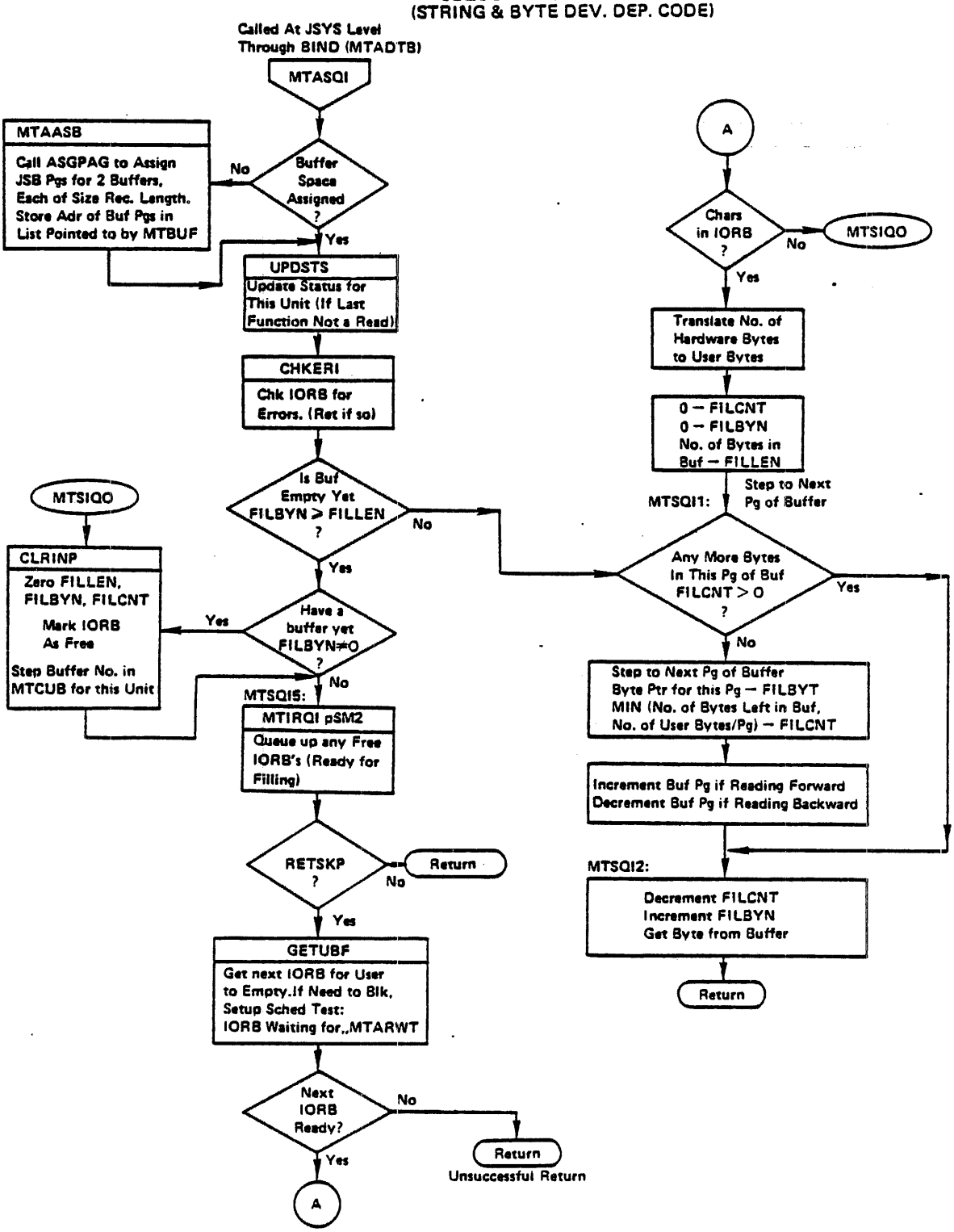
MTAOPN -	Magtape Opening of a File	OM1
MTASQI -	Magtape Sequential Input	SM1
	MTAIRQ - Queue Up Specified IORB	SM2
MTASQO -	Magtape Sequential Output	SM3
MTACLZ -	Magtape Closing of a File	CM1

OPENF – MAGTAPE
DEVICE DEPENDENT CODE

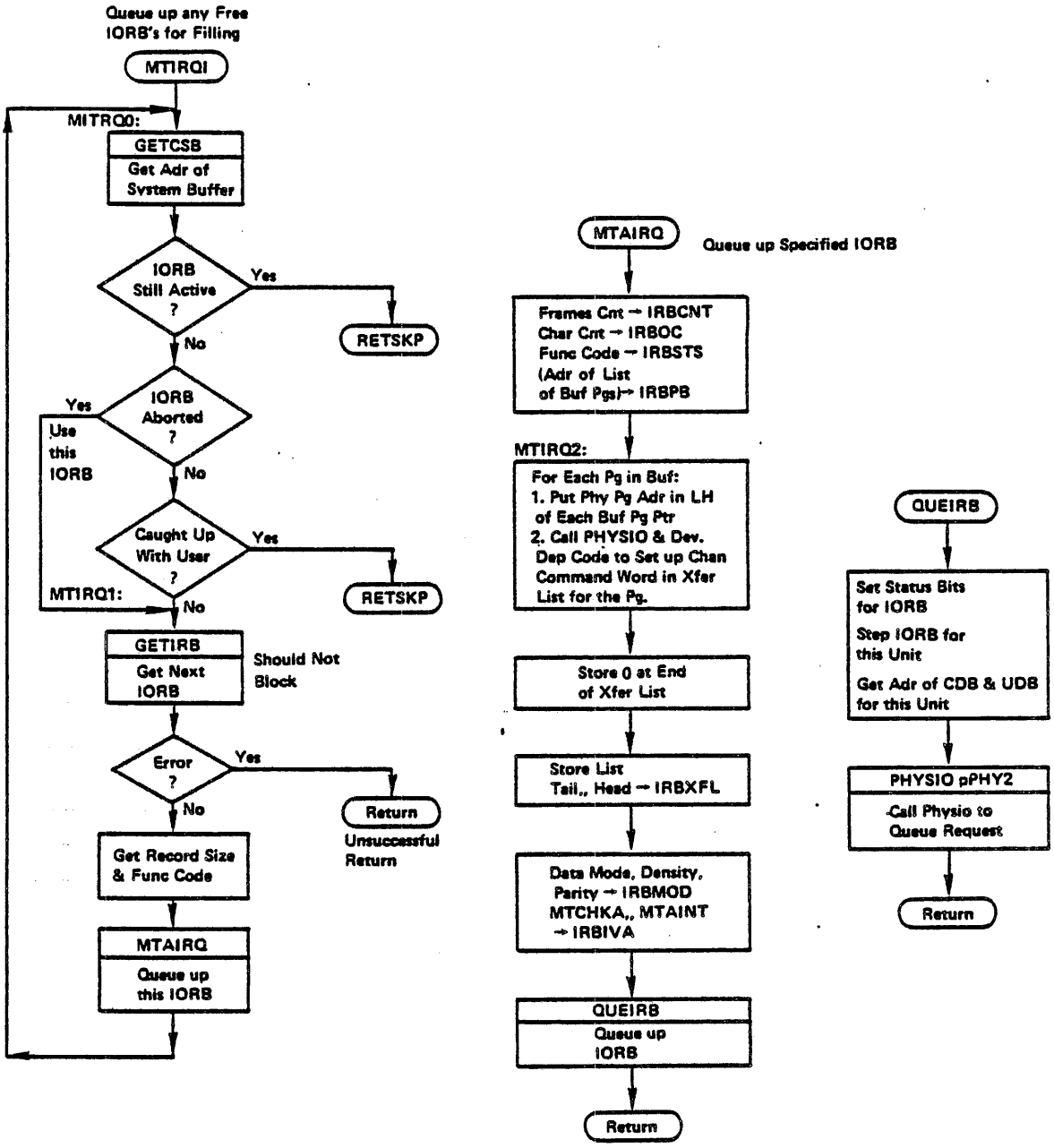


OM1

SEQUENTIAL INPUT – MTA
(STRING & BYTE DEV. DEP. CODE)

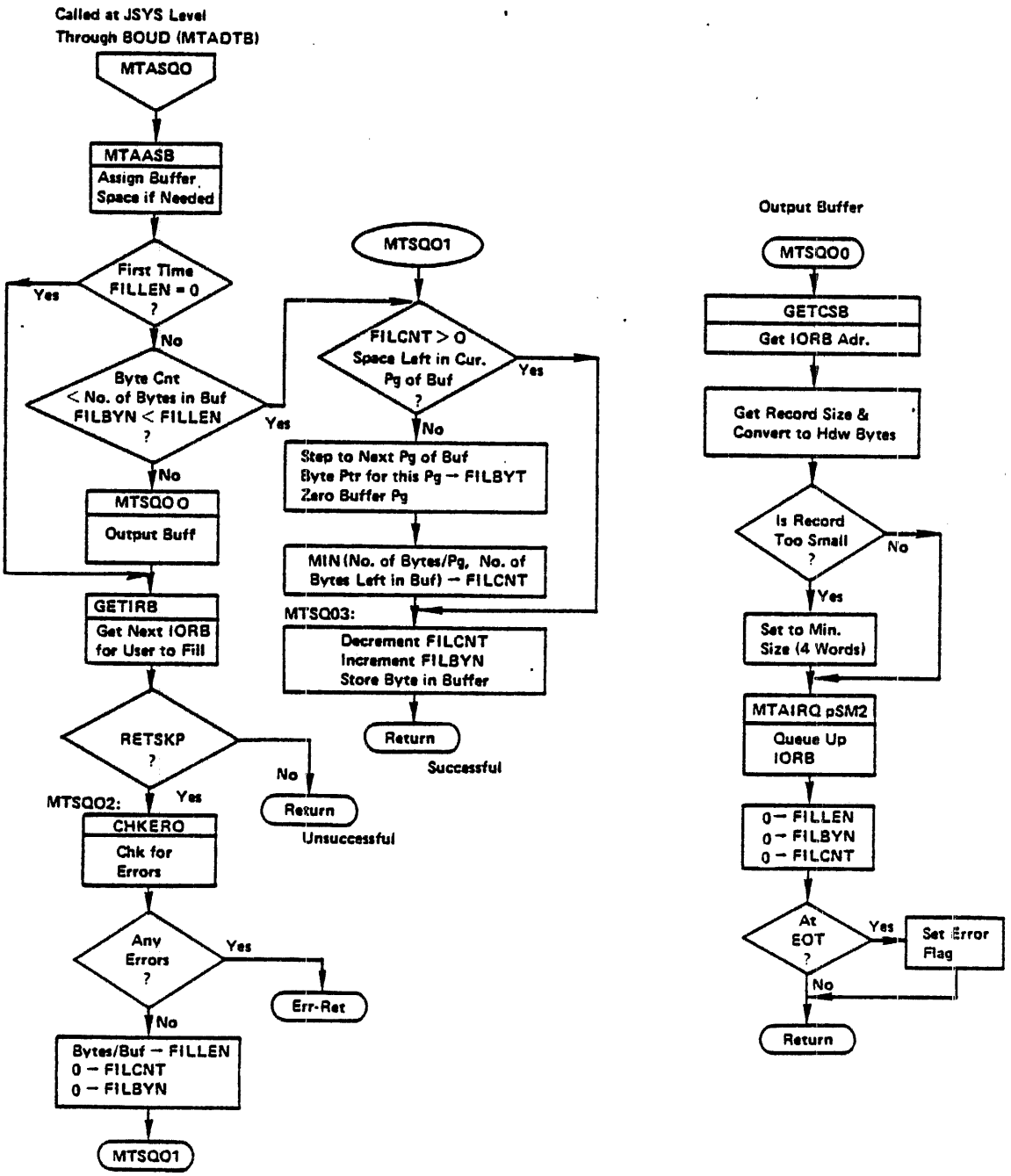


SM1


Queue up any Free IORB's for Filling
MTIRQI
MITRQ0:
GETCSB
Get Adr of Svstem Buffer
IORB Still Active ?
Yes
RETSKP
No
IORB Aborted ?
Yes
Use this IORB
No
Caught Up With User ?
Yes
RETSKP
MTIRQ1:
No
GETIRB
Get Next IORB
Should Not Block
Error ?
Yes
Return
Unsuccessful Return
No
Get Record Size & Func Code
MTAIRQ
Queue up this IORB
MTAIRQ
Queue up Specified IORB
Frames Cnt → IRBCNT
Char Cnt → IRBOC
Func Code → IRBSTS
(Adr of List of Buf Pgs)→ IRBPB
MTIRQ2:
For Each Pg in Buf:
1. Put Phy Pg Adr in LH of Each Buf Pg Ptr
2. Call PHYSIO & Dev. Dep Code to Set up Chan Command Word in Xfer List for the Pg.
Store 0 at End of Xfer List
Store List Tail,, Head → IRBXFL
Data Mode, Density, Parity → IRBMOD
MTCHKA,, MTAINT → IRBIVA
QUEIRB
Queue up IORB
Return
QUEIRB
Set Status Bits for IORB
Step IORB for this Unit
Get Adr of CDB & UDB for this Unit
PHYSIO pPHY2
Call Physio to Queue Request
Return

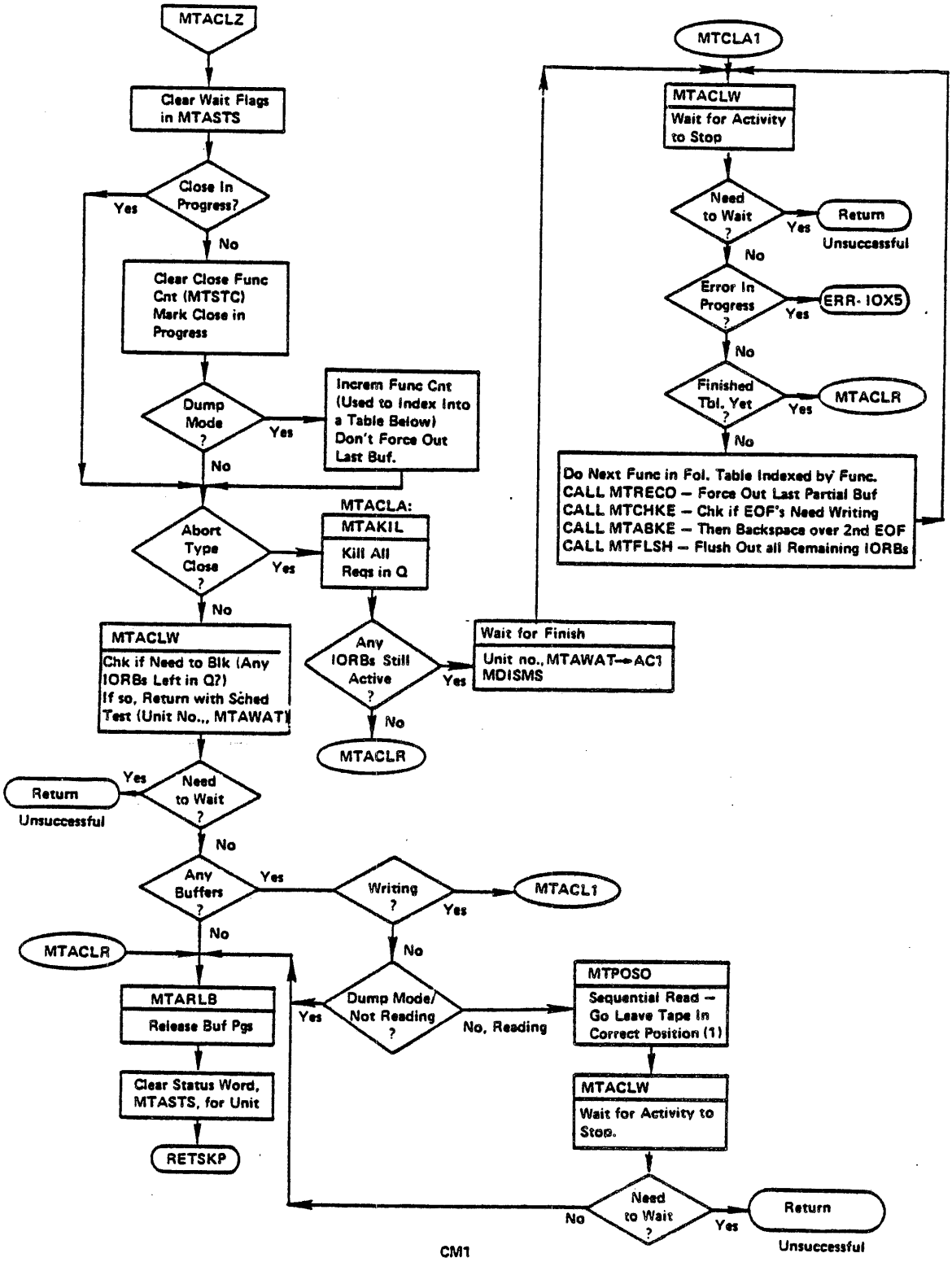
SM2

SEQUENTIAL OUTPUT – MTA
STRING & BYTE DEV. DEP. CODE



SM3

CLOSF – MAGTAPE



CM1

OPENF-MAGTAPE Comments

(1) One can open for read and write only in dump mode.

(2) FILCNT/Count of bytes left to use in current page of buffer.
FILLEN/Count of bytes in buffer.
FILBYN/Buffer byte number user is referencing.

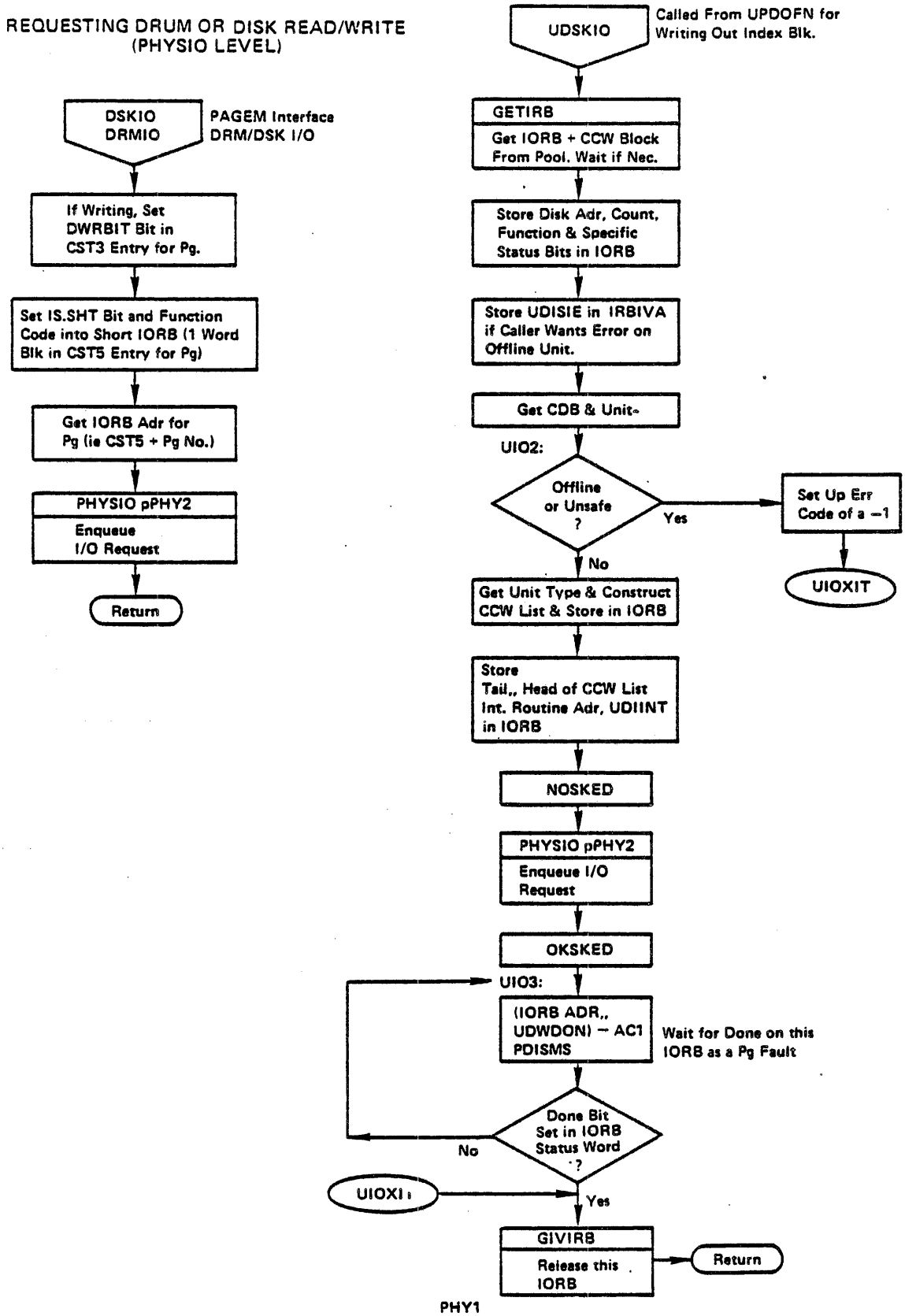
CLOSF - MAGTAPE Comments

(1) Since the monitor reads ahead, backspacing to just after last user record read may be necessary.

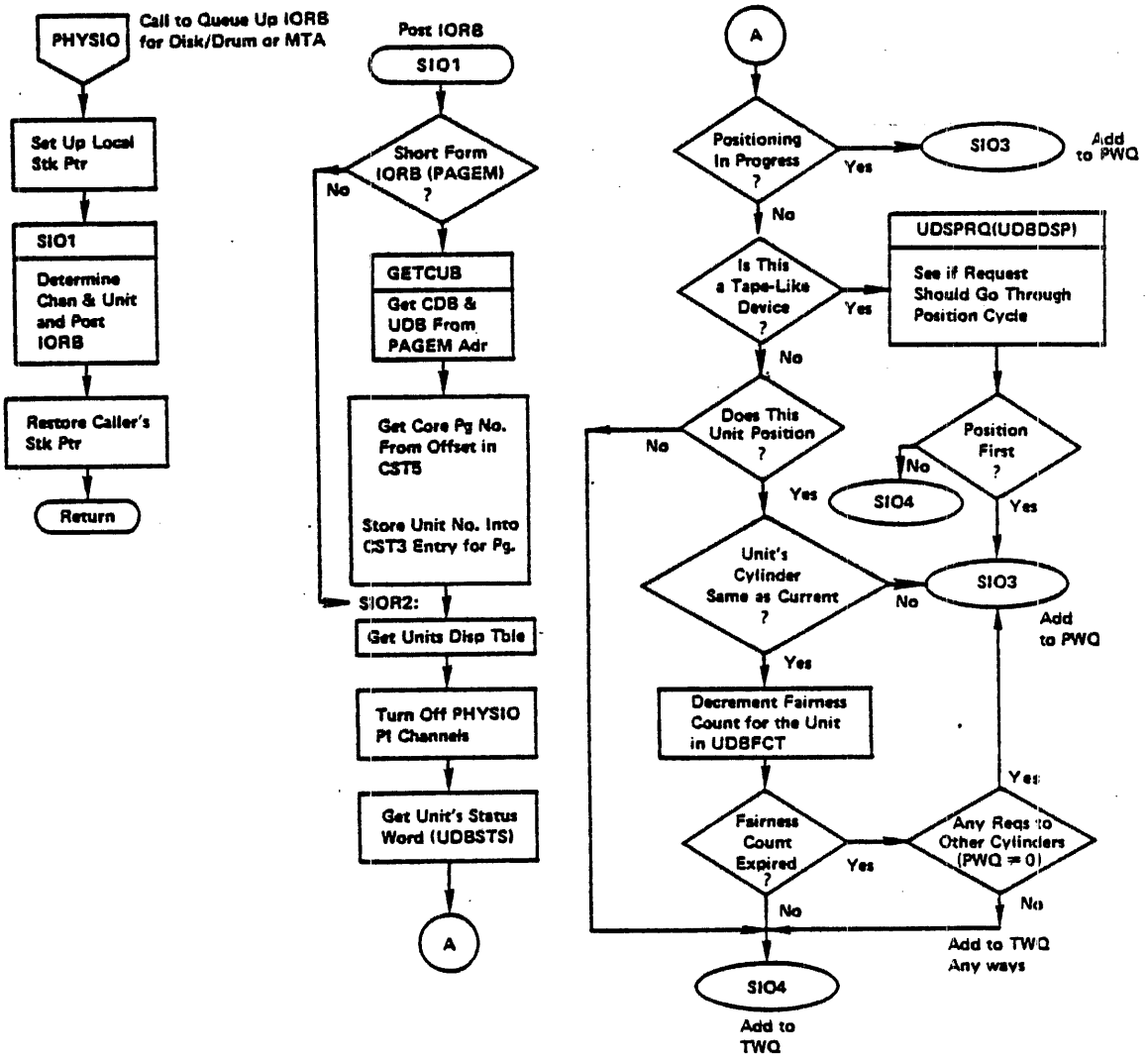
REQUESTING DISK/MTA I/O & INTERRUPT HANDLING FLOWCHARTS (PHYSIO LEVEL)

DRMIO/DSKIO/UDSKIO -	Requesting Drum or Disk Read/Write			PHY1
PHYSIO -	Queue Up IORB Request for Disk, Drum or Magtape			PHY2
	SIO1 -	Post IORB		PHY2
		STRTPS -	Start Unit Positioning	PHY3
		STRTIO -	Start Unit Transferring	PHY3a
PHYINT -	Disk and Magtape Interrupt Handler			PHY4
DONIRB -	Post IORB as Done			PHY5
	SWPDON/UDIINT -		Housekeep for Drum/Disk Done	PHY8
	MTAINT -		Housekeep for Magtape Done	PHY9
SCHSEK -	Schedule "Best" Seek Request			PHY6
SCHXFR -	Schedule "Best" Transfer Request			PHY7

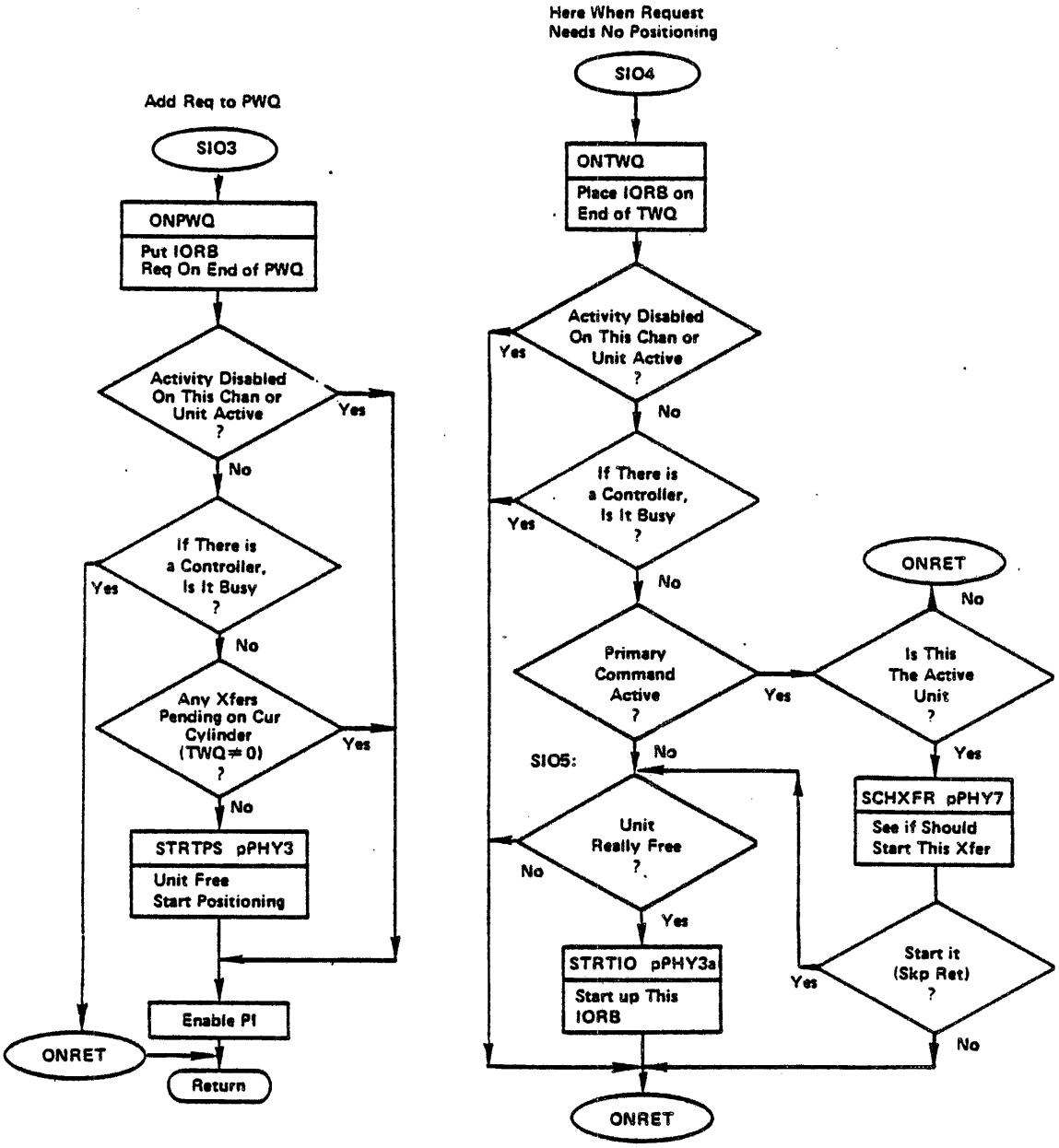
REQUESTING DRUM OR DISK READ/WRITE
(PHYSIO LEVEL)



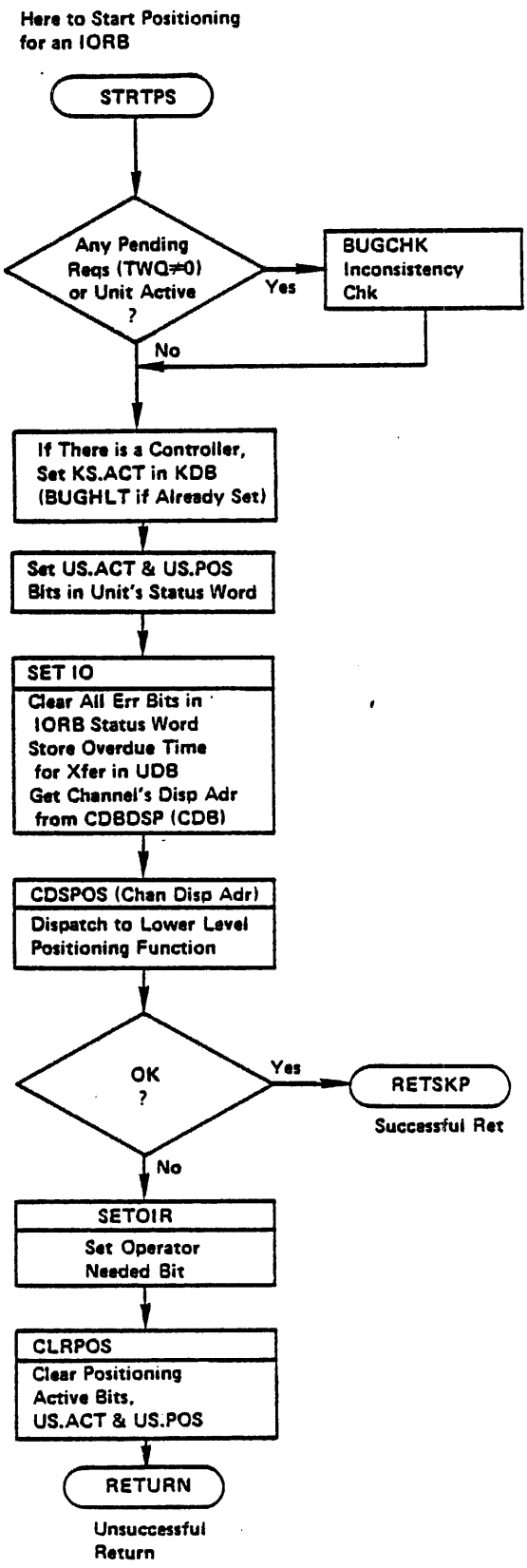
PHY1


PHYSIO
Call to Queue Up IORB for Disk/Drum or MTA
Set Up Local Stk Ptr
SIO1
Determine Chan & Unit and Post IORB
Restore Caller's Stk Ptr
Return
Post IORB
SIO1
Short Form IORB (PAGEM) ?
No
GETCUB
Get CDB & UDB From PAGEM Adr
Get Core Pg No. From Offset in CST5
Store Unit No. Into CST3 Entry for Pg.
SIOR2:
Get Units Disp Tble
Turn Off PHYSIO PI Channels
Get Unit's Status Word (UDBSTS)
A
A
Positioning In Progress ?
Yes
SIO3
Add to PWQ
No
Is This a Tape-Like Device ?
Yes
UDSPRQ(UDBDSP)
See if Request Should Go Through Position Cycle
No
Does This Unit Position ?
No
Yes
Position First ?
No
SIO4
Yes
Unit's Cylinder Same as Current ?
No
SIO3
Add to PWQ
Yes
Decrement Fairness Count for the Unit in UDBFCT
Fairness Count Expired ?
Yes
Any Reqs to Other Cylinders (PWQ ≠ 0)
Yes
No
No
Add to TWQ Any ways
SIO4
Add to TWQ

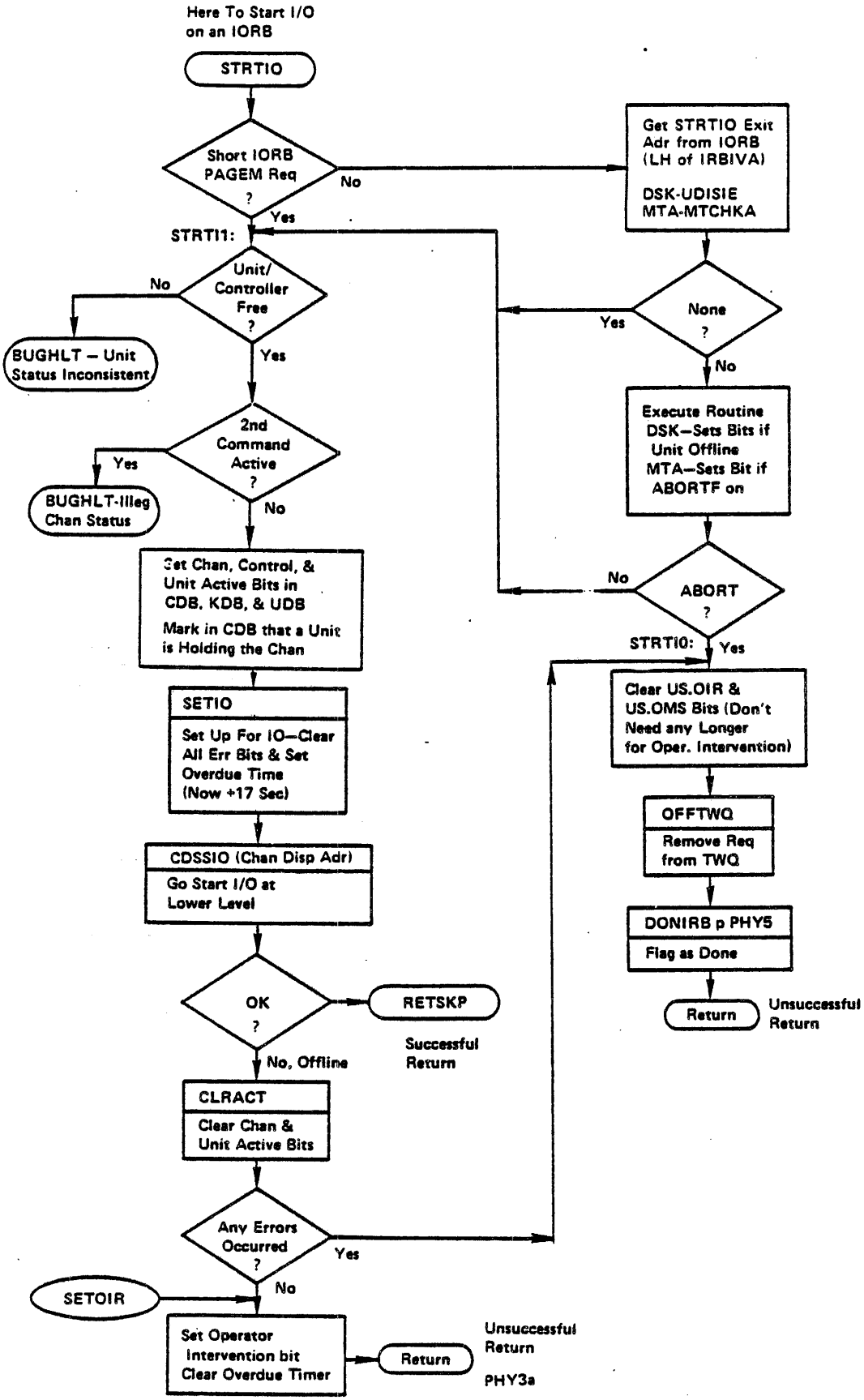
PHY2

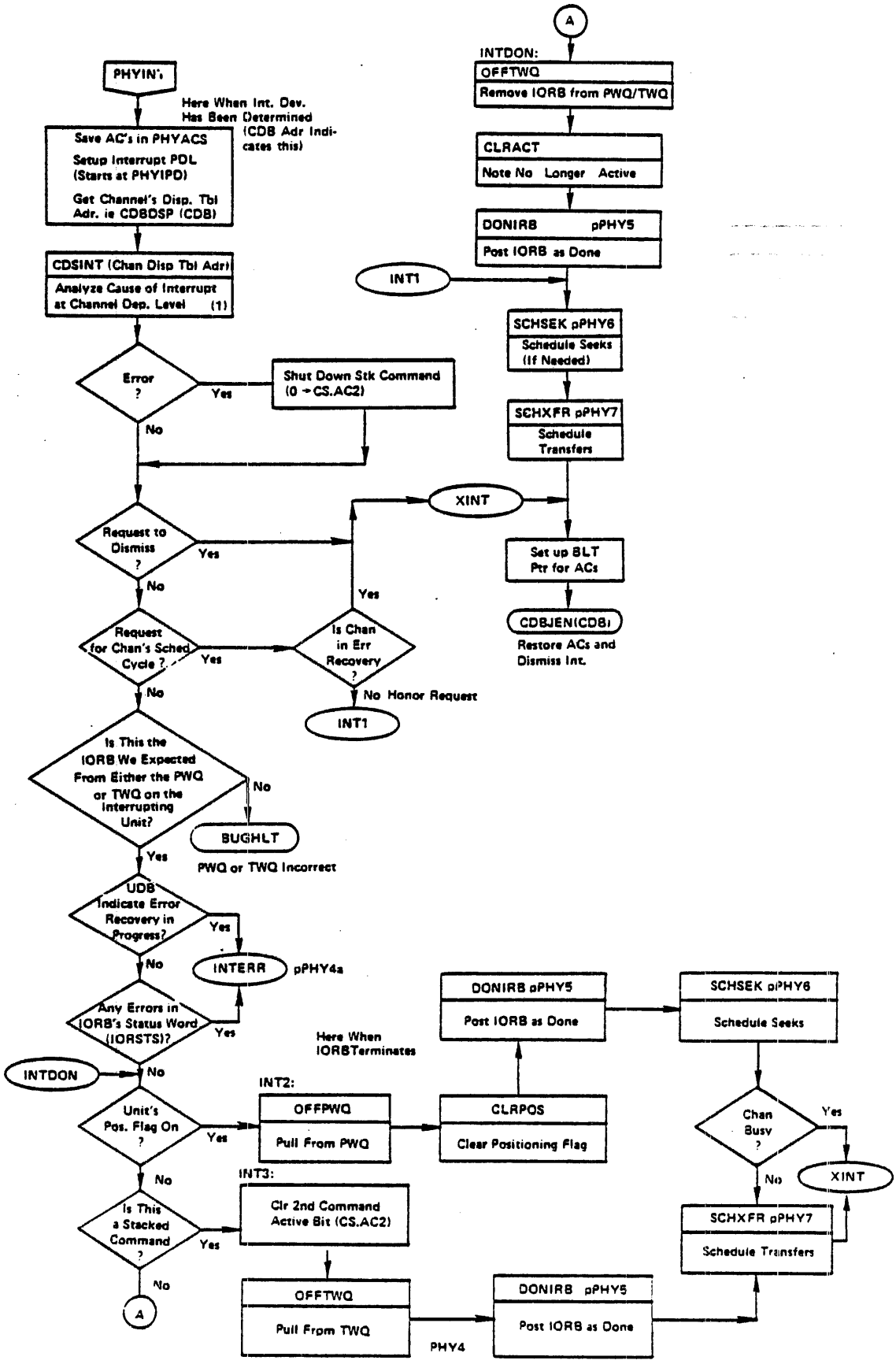

Add Req to PWQ
SIO3
ONPWQ
Put IORB
Req On End of PWQ
Activity Disabled
On This Chan or
Unit Active
?
Yes
No
If There is
a Controller,
Is It Busy
?
Yes
No
Any Xfers
Pending on Cur
Cylinder
(TWQ≠0)
?
Yes
No
STRTPS pPHY3
Unit Free
Start Positioning
Enable PI
ONRET
Return
Here When Request
Needs No Positioning
SIO4
ONTWQ
Place IORB on
End of TWQ
Activity Disabled
On This Chan or
Unit Active
?
Yes
No
If There is
a Controller,
Is It Busy
?
Yes
No
Primary
Command
Active
?
Yes
No
ONRET
Is This
The Active
Unit
?
No
Yes
SIO5:
Unit
Really Free
?
No
Yes
SCHXFR pPHY7
See if Should
Start This Xfer
Start it
(Skp Ret)
?
Yes
No
STRTIO pPHY3a
Start up This
IORB
ONRET

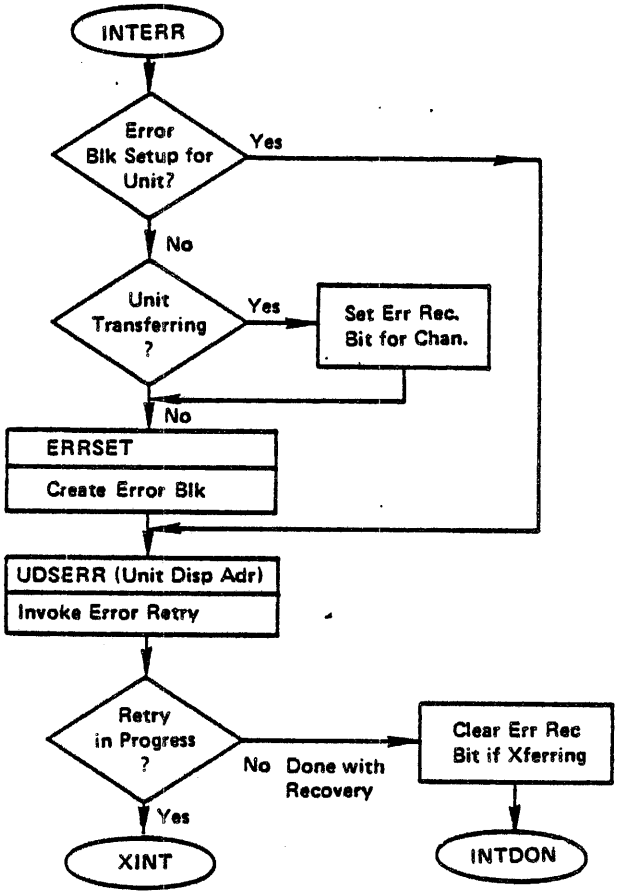
PHY2a


Here to Start Positioning for an IORB
STRTPS
Any Pending Reqs (TWQ≠0) or Unit Active ?
Yes
BUGCHK Inconsistency Chk
No
If There is a Controller, Set KS.ACT in KDB (BUGHLT if Already Set)
Set US.ACT & US.POS Bits in Unit's Status Word
SET IO
Clear All Err Bits in IORB Status Word
Store Overdue Time for Xfer in UDB
Get Channel's Disp Adr from CDBDSP (CDB)
CDSPOS (Chan Disp Adr)
Dispatch to Lower Level Positioning Function
OK ?
Yes
RETSKP
Successful Ret
No
SETOIR
Set Operator Needed Bit
CLRPOS
Clear Positioning Active Bits, US.ACT & US.POS
RETURN
Unsuccessful Return

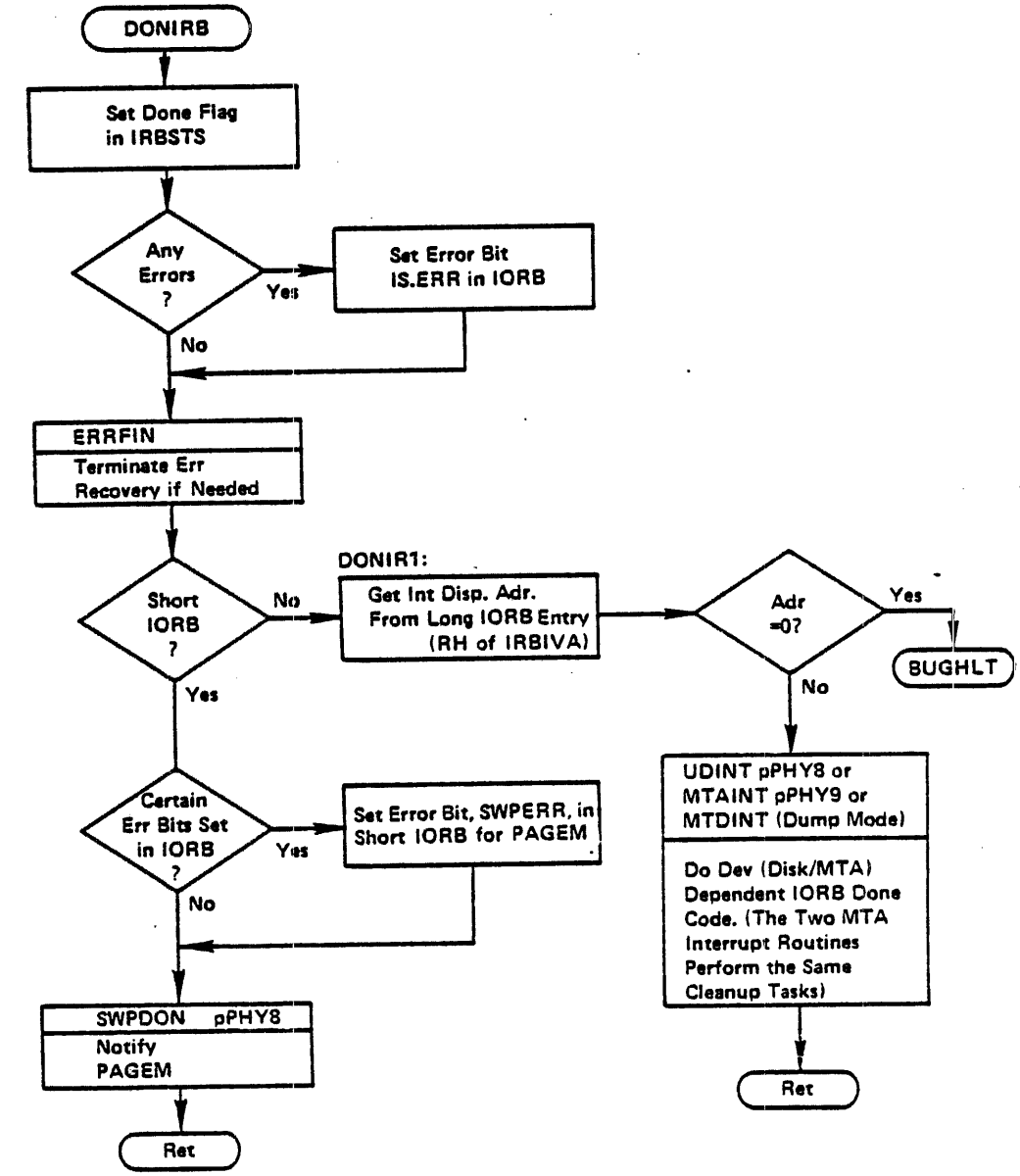
PHY3


Here To Start I/O on an IORB
STRTIO
Short IORB PAGEM Req ?
No
Yes
Get STRTIO Exit Adr from IORB (LH of IRBIVA)
DSK-UDISIE
MTA-MTCHKA
STRTI1:
Unit/ Controller Free ?
No
Yes
BUGHLT – Unit Status Inconsistent
None ?
Yes
No
2nd Command Active ?
Yes
No
BUGHLT-Illeg Chan Status
Execute Routine DSK–Sets Bits if Unit Offline MTA–Sets Bit if ABORTF on
Set Chan, Control, & Unit Active Bits in CDB, KDB, & UDB
Mark in CDB that a Unit is Holding the Chan
ABORT ?
No
Yes
STRTI0:
SETIO
Set Up For IO–Clear All Err Bits & Set Overdue Time (Now +17 Sec)
Clear US.OIR & US.OMS Bits (Don't Need any Longer for Oper. Intervention)
OFFTWQ
Remove Req from TWQ
CDSSIO (Chan Disp Adr)
Go Start I/O at Lower Level
DONIRB p PHY5
Flag as Done
OK ?
RETSKP
Successful Return
Return
Unsuccessful Return
No, Offline
CLRACT
Clear Chan & Unit Active Bits
Any Errors Occurred ?
Yes
No
SETOIR
Set Operator Intervention bit Clear Overdue Timer
Return
Unsuccessful Return
PHY3a

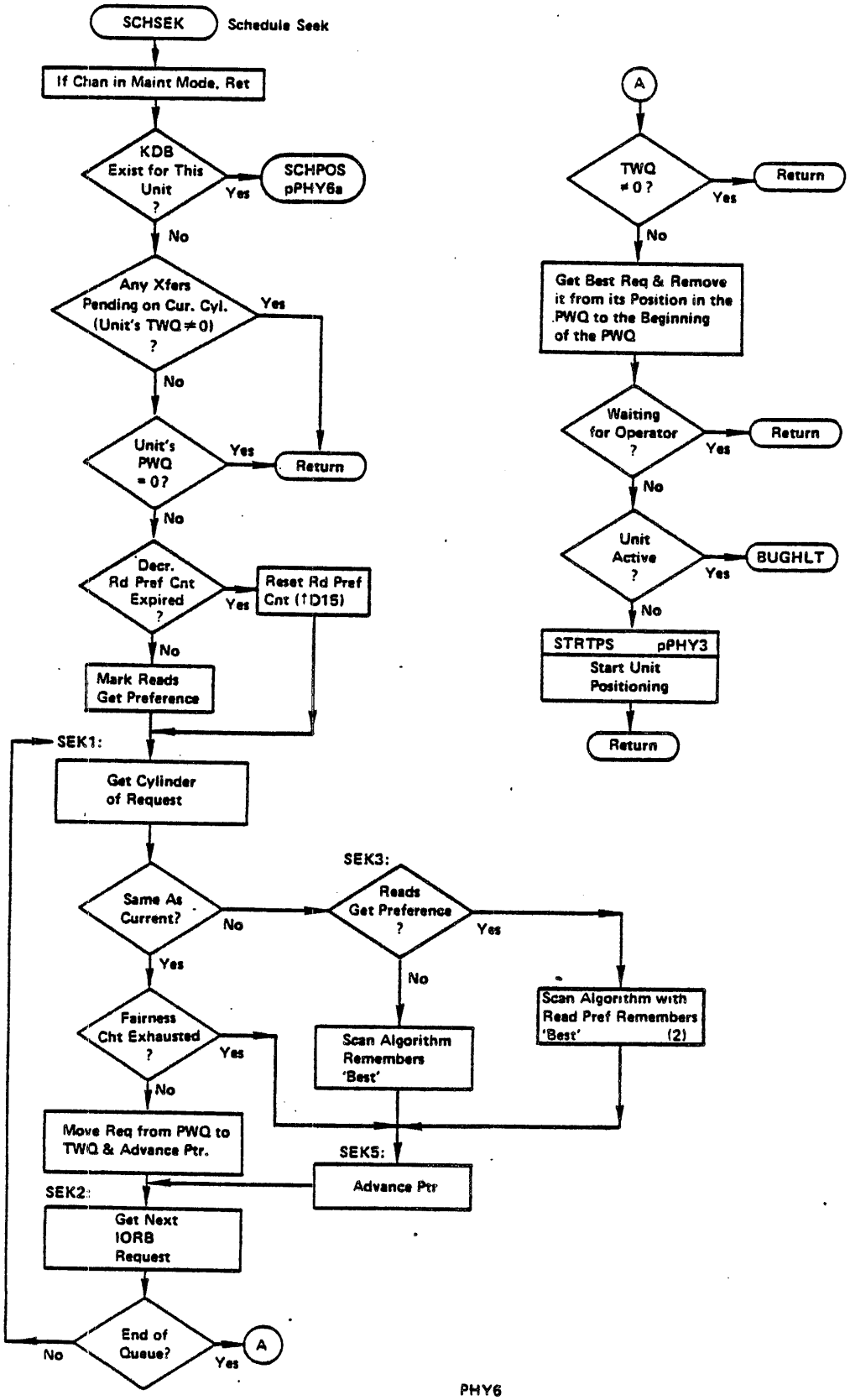

PHYIN:
Here When Int. Dev. Has Been Determined (CDB Adr Indicates this)
Save AC's in PHYACS
Setup Interrupt POL (Starts at PHYIPD)
Get Channel's Disp. Tbl Adr. ie CDBDSP (CDB)
CDSINT (Chan Disp Tbl Adr)
Analyze Cause of Interrupt at Channel Dep. Level (1)
Error ?
Yes
Shut Down Stk Command (0 → CS.AC2)
No
Request to Dismiss ?
Yes
No
Request for Chan's Sched Cycle ?
Yes
Is Chan in Err Recovery ?
No Honor Request
INT1
No
Is This the IORB We Expected From Either the PWQ or TWQ on the Interrupting Unit?
No
BUGHLT
PWQ or TWQ Incorrect
Yes
UDB Indicate Error Recovery in Progress?
Yes
INTERR
pPHY4a
No
Any Errors in IORB's Status Word (IORSTS)?
Yes
INTDON
No
Unit's Pos. Flag On ?
Yes
INT2:
OFFPWQ
Pull From PWQ
No
Is This a Stacked Command ?
Yes
INT3:
Clr 2nd Command Active Bit (CS.AC2)
No
A
OFFTWQ
Pull From TWQ
PHY4
A
INTDON:
OFFTWQ
Remove IORB from PWQ/TWQ
CLRACT
Note No Longer Active
DONIRB pPHY5
Post IORB as Done
INT1
SCHSEK pPHY6
Schedule Seeks (If Needed)
SCHXFR pPHY7
Schedule Transfers
XINT
Set up BLT Ptr for ACs
CDBJEN(CDB)
Restore ACs and Dismiss Int.
Here When IORBTerminates
DONIRB pPHY5
Post IORB as Done
CLRPOS
Clear Positioning Flag
DONIRB pPHY5
Post IORB as Done
SCHSEK pPHY6
Schedule Seeks
Chan Busy ?
Yes
XINT
No
SCHXFR pPHY7
Schedule Transfers


INTERR
Error Blk Setup for Unit?
Yes
No
Unit Transferring ?
Yes
Set Err Rec. Bit for Chan.
No
ERRSET
Create Error Blk
UDSERR (Unit Disp Adr)
Invoke Error Retry
Retry in Progress ?
No Done with Recovery
Clear Err Rec Bit if Xferring
Yes
XINT
INTDON

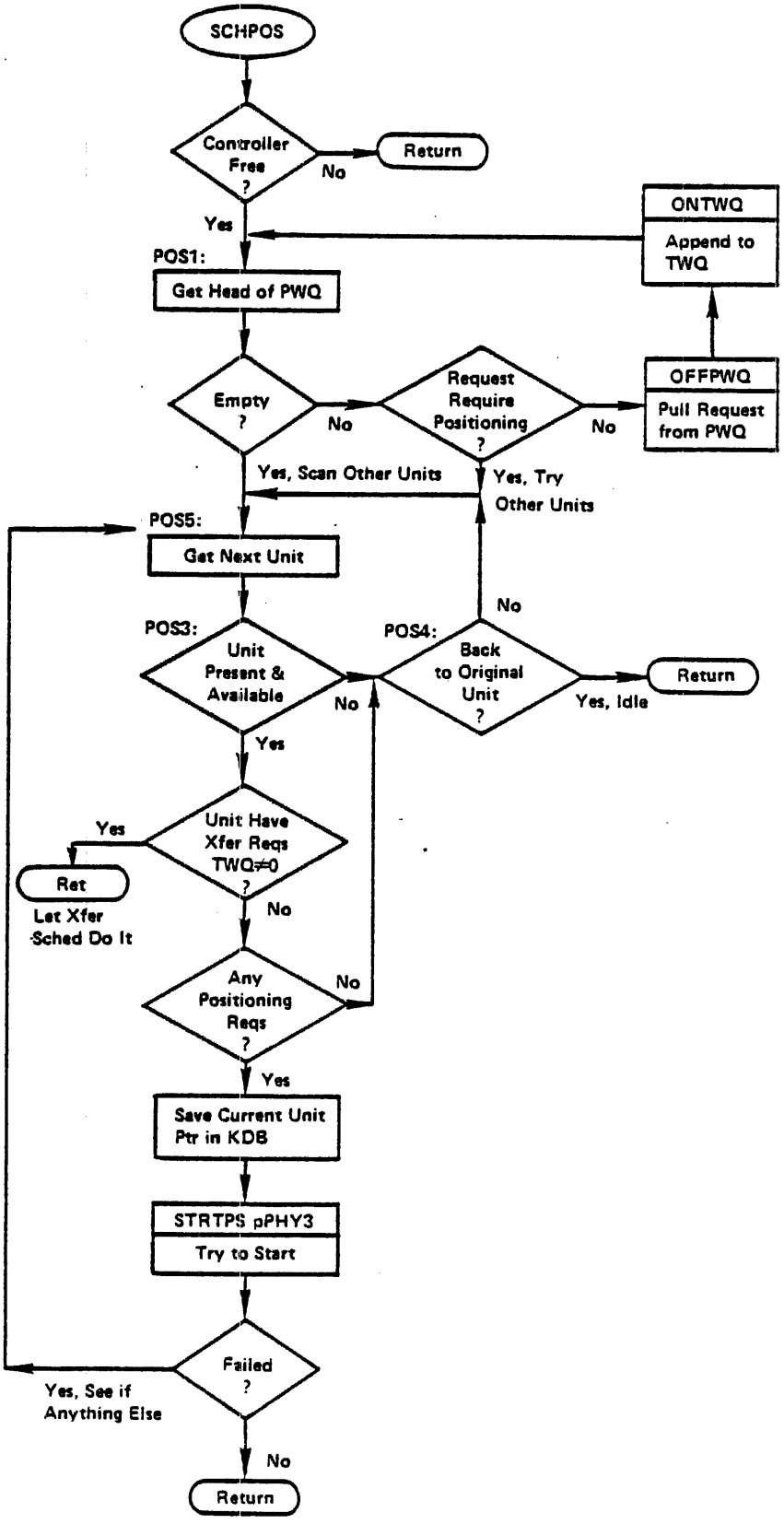
Here to Post an IORB Compiete



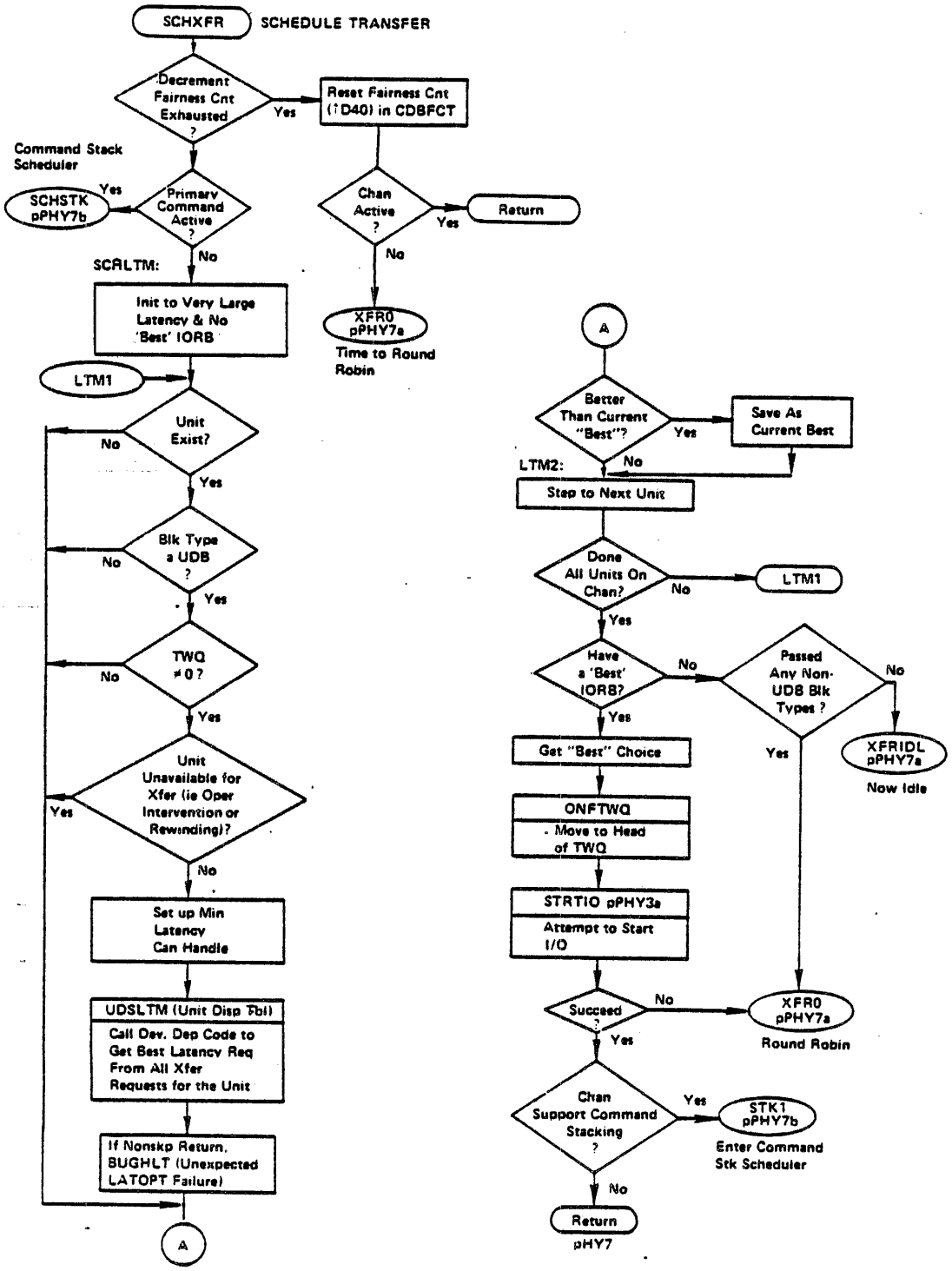
PHY5

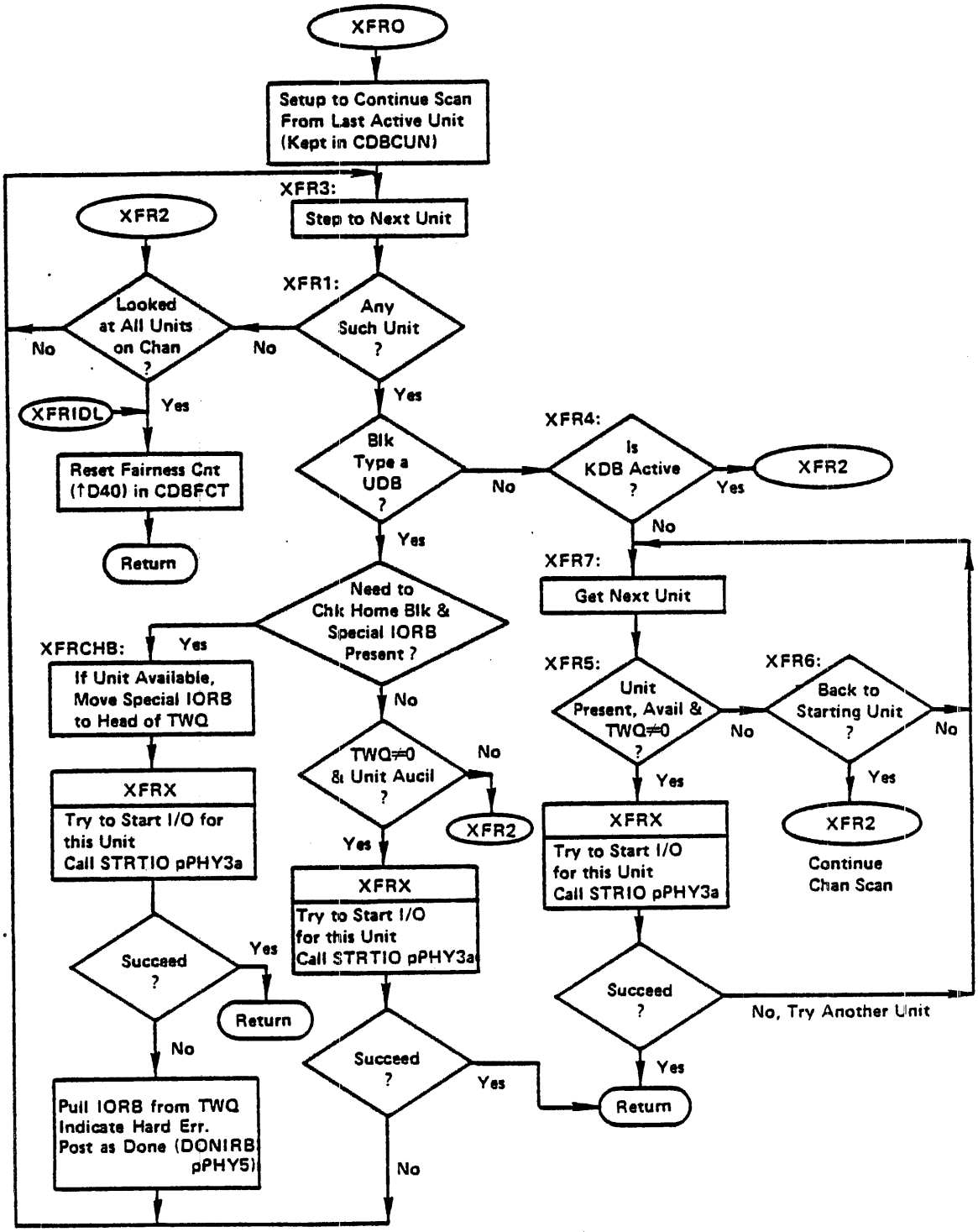

SCHSEK
Schedule Seek
If Chan in Maint Mode, Ret
KDB Exist for This Unit ?
Yes
SCHPOS pPHY6a
No
Any Xfers Pending on Cur. Cyl. (Unit's TWQ≠0) ?
Yes
No
Unit's PWQ = 0?
Yes
Return
No
Decr. Rd Pref Cnt Expired ?
Yes
Reset Rd Pref Cnt (↑D15)
No
Mark Reads Get Preference
SEK1:
Get Cylinder of Request
SEK3:
Same As Current?
No
Reads Get Preference ?
Yes
Yes
No
Fairness Cht Exhausted ?
Yes
No
Scan Algorithm with Read Pref Remembers 'Best' (2)
Scan Algorithm Remembers 'Best'
Move Req from PWQ to TWQ & Advance Ptr.
SEK5:
Advance Ptr
SEK2:
Get Next IORB Request
End of Queue?
No
Yes
A
A
TWQ ≠0?
Yes
Return
No
Get Best Req & Remove it from its Position in the PWQ to the Beginning of the PWQ
Waiting for Operator ?
Yes
Return
No
Unit Active ?
Yes
BUGHLT
No
STRTPS pPHY3
Start Unit Positioning
Return

PHY6

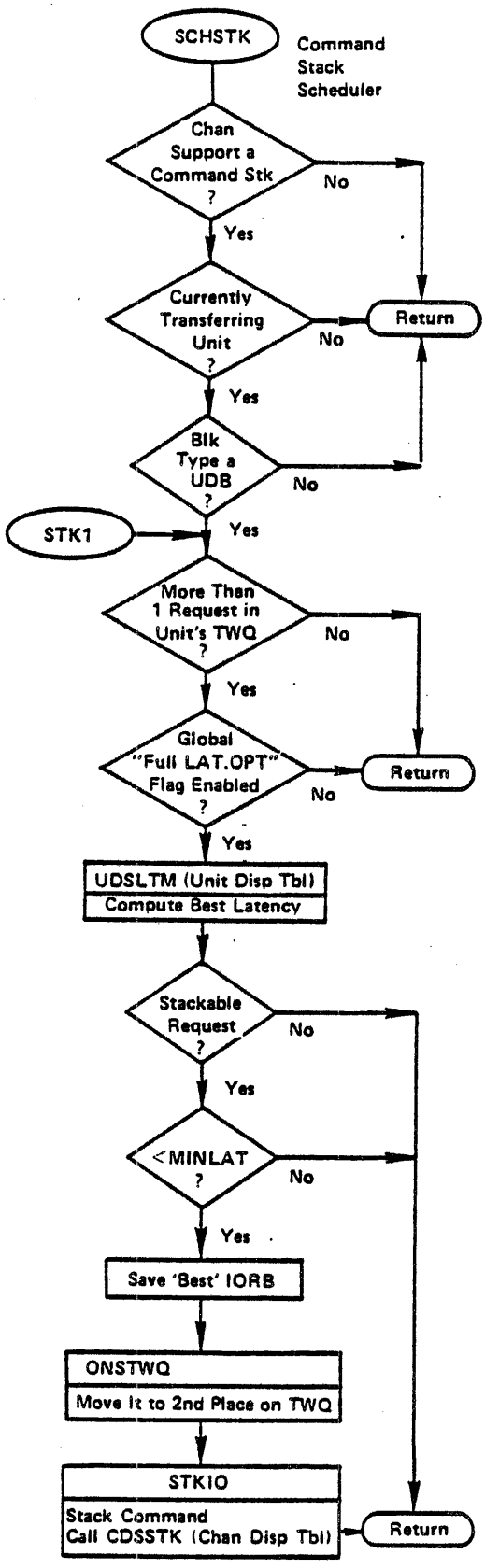

SCHPOS
Controller Free ?
No
Return
Yes
ONTWQ
Append to TWQ
POS1:
Get Head of PWQ
Empty ?
No
Request Require Positioning ?
No
OFFPWQ
Pull Request from PWQ
Yes, Scan Other Units
Yes, Try Other Units
POS5:
Get Next Unit
No
POS3:
Unit Present & Available
No
POS4:
Back to Original Unit ?
Yes, Idle
Return
Yes
Unit Have Xfer Reqs TWQ≠0 ?
Yes
Ret
Let Xfer Sched Do It
No
Any Positioning Reqs ?
No
Yes
Save Current Unit Ptr in KDB
STRTPS pPHY3
Try to Start
Failed ?
Yes, See if Anything Else
No
Return

PHY6a


SCHXFR
SCHEDULE TRANSFER
Decrement Fairness Cnt Exhausted ?
Yes
Reset Fairness Cnt (↑D40) in CDBFCT
Command Stack Scheduler
Primary Command Active ?
Yes
SCHSTK pPHY7b
No
Chan Active ?
Yes
Return
No
XFR0 pPHY7a
Time to Round Robin
SCALTM:
Init to Very Large Latency & No 'Best' IORB
LTM1
Unit Exist?
No
Yes
Blk Type a UDB ?
No
Yes
TWQ ≠ 0 ?
No
Yes
Unit Unavailable for Xfer (ie Oper Intervention or Rewinding)?
Yes
No
Set up Min Latency Can Handle
UDSLTM (Unit Disp Tbl)
Call Dev. Dep Code to Get Best Latency Req From All Xfer Requests for the Unit
If Nonskp Return, BUGHLT (Unexpected LATOPT Failure)
A
A
Better Than Current "Best"?
Yes
Save As Current Best
No
LTM2:
Step to Next Unit
Done All Units On Chan?
No
LTM1
Yes
Have a 'Best' IOR8?
No
Passed Any Non-UDB Blk Types ?
No
Yes
XFRIDL pPHY7a
Now Idle
Yes
Get "Best" Choice
ONFTWQ
- Move to Head of TWQ
STRTIO pPHY3a
Attempt to Start I/O
Succeed ?
No
XFR0 pPHY7a
Round Robin
Yes
Chan Support Command Stacking ?
Yes
STK1 pPHY7b
Enter Command Stk Scheduler
No
Return
pHY7

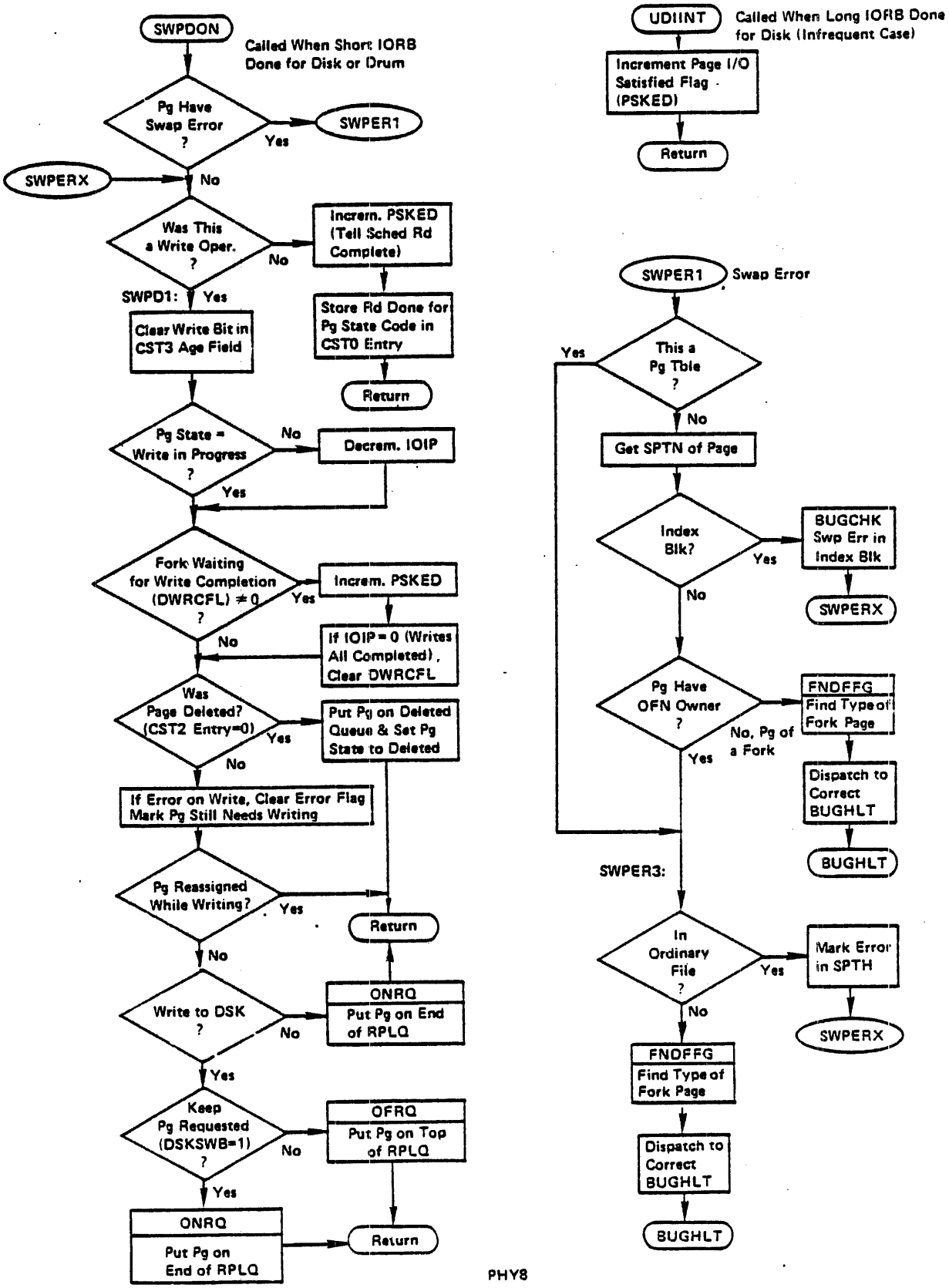

XFRO
Setup to Continue Scan From Last Active Unit (Kept in CDBCUN)
XFR3:
Step to Next Unit
XFR2
XFR1:
Any Such Unit ?
No
Looked at All Units on Chan ?
No
Yes
XFRIDL
Yes
Reset Fairness Cnt (↑D40) in CDBFCT
Return
Blk Type a UDB ?
No
XFR4:
Is KDB Active ?
Yes
XFR2
No
Yes
XFR7:
Get Next Unit
Need to Chk Home Blk & Special IORB Present ?
XFRCHB:
Yes
If Unit Available, Move Special IORB to Head of TWQ
XFRX
Try to Start I/O for this Unit
Call STRTIO pPHY3a
Succeed ?
Yes
Return
No
Pull IORB from TWQ
Indicate Hard Err.
Post as Done (DONIRB pPHY5)
No
TWQ≠0 & Unit Aucil ?
No
XFR2
Yes
XFRX
Try to Start I/O for this Unit
Call STRTIO pPHY3a
Succeed ?
Yes
Return
No
XFR5:
Unit Present, Avail & TWQ≠0 ?
No
XFR6:
Back to Starting Unit ?
No
Yes
XFR2
Continue Chan Scan
Yes
XFRX
Try to Start I/O for this Unit
Call STRIO pPHY3a
Succeed ?
No, Try Another Unit
Yes
Return

PHY7a


SCHSTK
Command Stack Scheduler
Chan Support a Command Stk ?
No
Yes
Currently Transferring Unit ?
No
Return
Yes
Blk Type a UDB ?
No
STK1
Yes
More Than 1 Request in Unit's TWQ ?
No
Yes
Global "Full LAT.OPT" Flag Enabled ?
No
Return
Yes
UDSLTM (Unit Disp Tbl)
Compute Best Latency
Stackable Request ?
No
Yes
<MINLAT ?
No
Yes
Save 'Best' IORB
ONSTWQ
Move It to 2nd Place on TWQ
STKIO
Stack Command
Call CDSSTK (Chan Disp Tbl)
Return

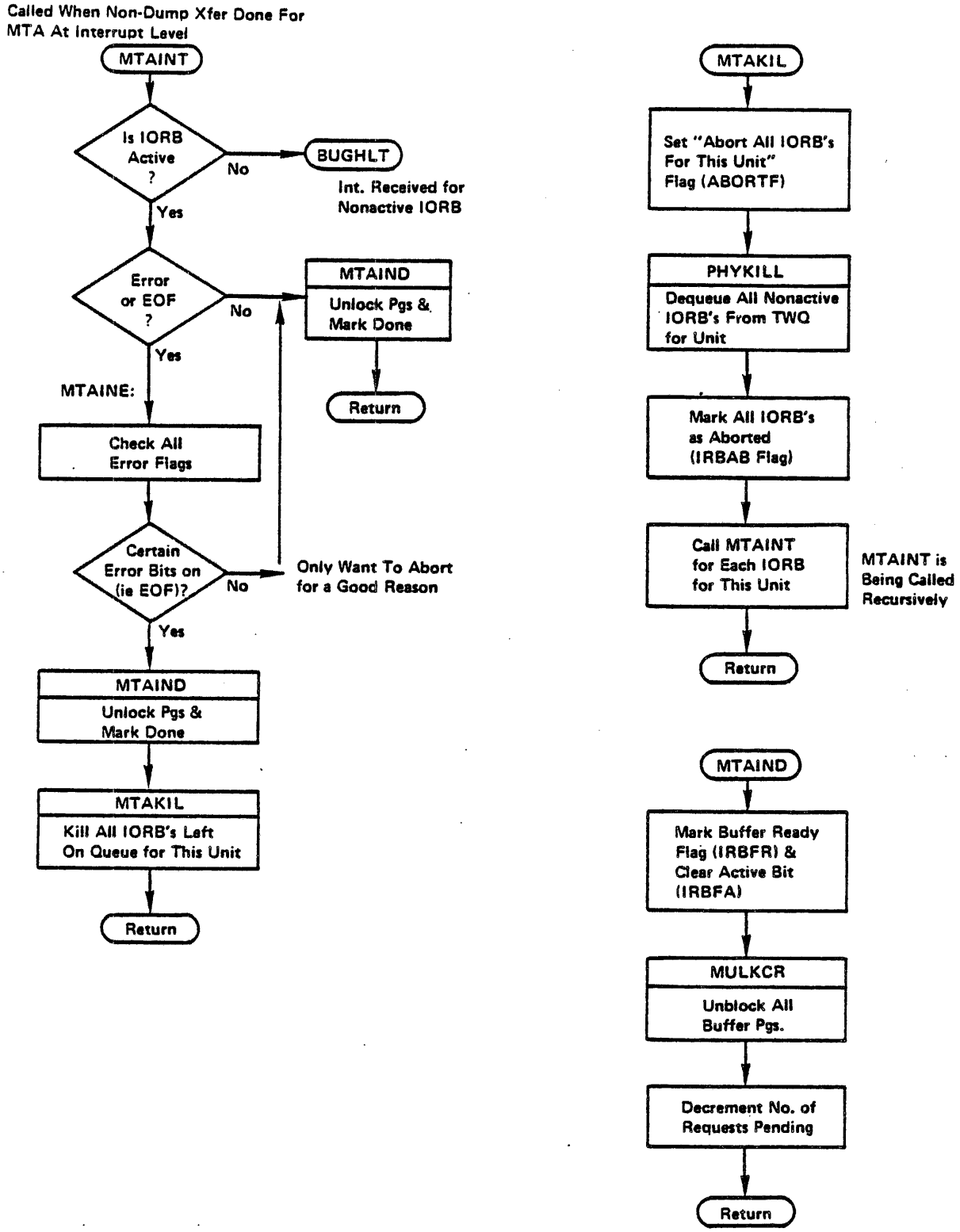
PHY7b

"INTERRUPT DONE" DSK/DRUM DEPENDENT CODE



PHY8

"INTERRUPT DONE" MAGTAPE DEPENDENT CODE



Requesting DISK/MTA I/O Comments

SIO1

(1) The algorithm for queuing up a MTA request is:

If the request requires positioning, append the request to the PWQ.

If the request requires no positioning (i.e., Read/Write Forward or Read Reverse) append the request to the TWQ only if the PWQ is empty. Otherwise, append it to the PWQ.

DSK/MTA Interrupt Handling Comments

PHYINT

(1) The channel dependent routine (RH2INT for RH2Øs) is called to analyze the interrupt. Lower level routines called by RH2INT (i.e., Unit dependent routines) return an argument in AC, P4, to PHYINT to indicate whether to dismiss the interrupt (P4 = Ø), to schedule another channel cycle right away (P4 < Ø) or to housekeep the current request (P4 > Ø) before scheduling another channel cycle. The channel dependent routine also records error information so that PHYINT can see if error recovery is in progress or should be started.

The request to dismiss (P4 = Ø) is invoked for example when the done flag is on and the channel is not occupied. The request for an immediate channel cycle (P4 < Ø) is made when a positioning done interrupt has occurred and there is no transfer in progress. Transfer Done requests will require further housekeeping (P4 > Ø) by PHYINT before scheduling another channel cycle.

SCHSEK

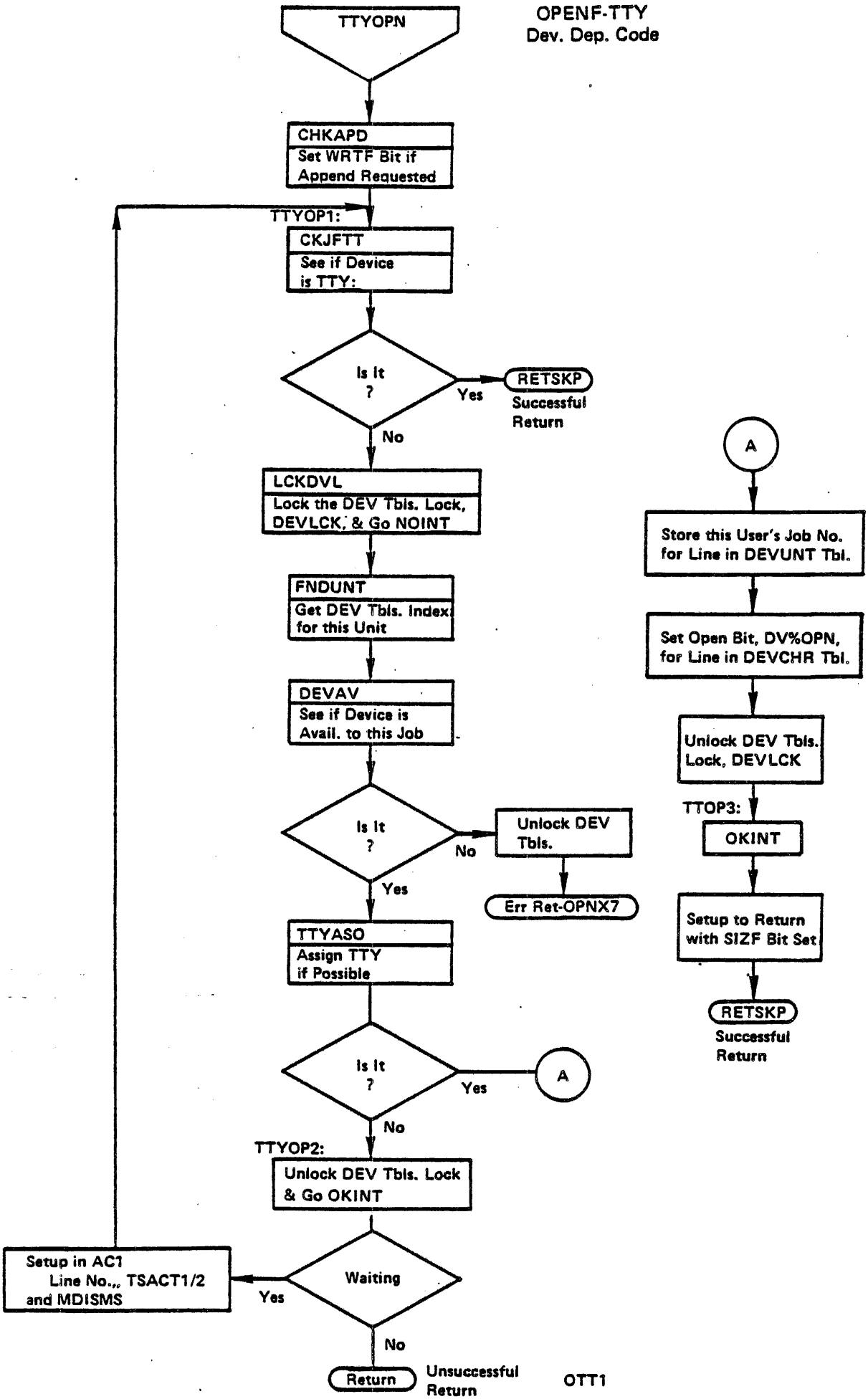
(2) The scan algorithm with read preference in effect performs as follows:

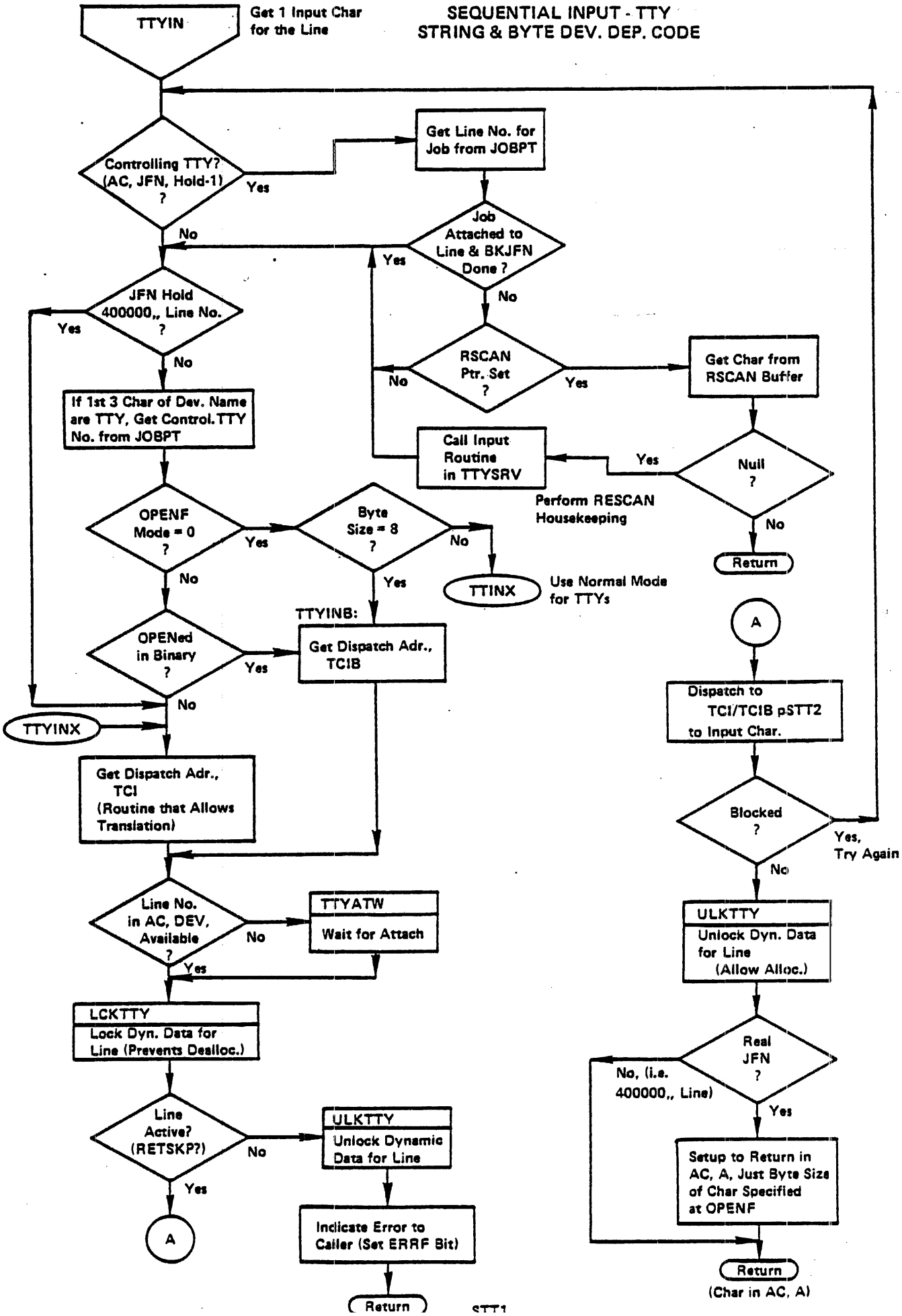
Take the next higher-numbered cylinder read request from the current cylinder. If none, take the next higher-numbered cylinder (write) request from the current cylinder.

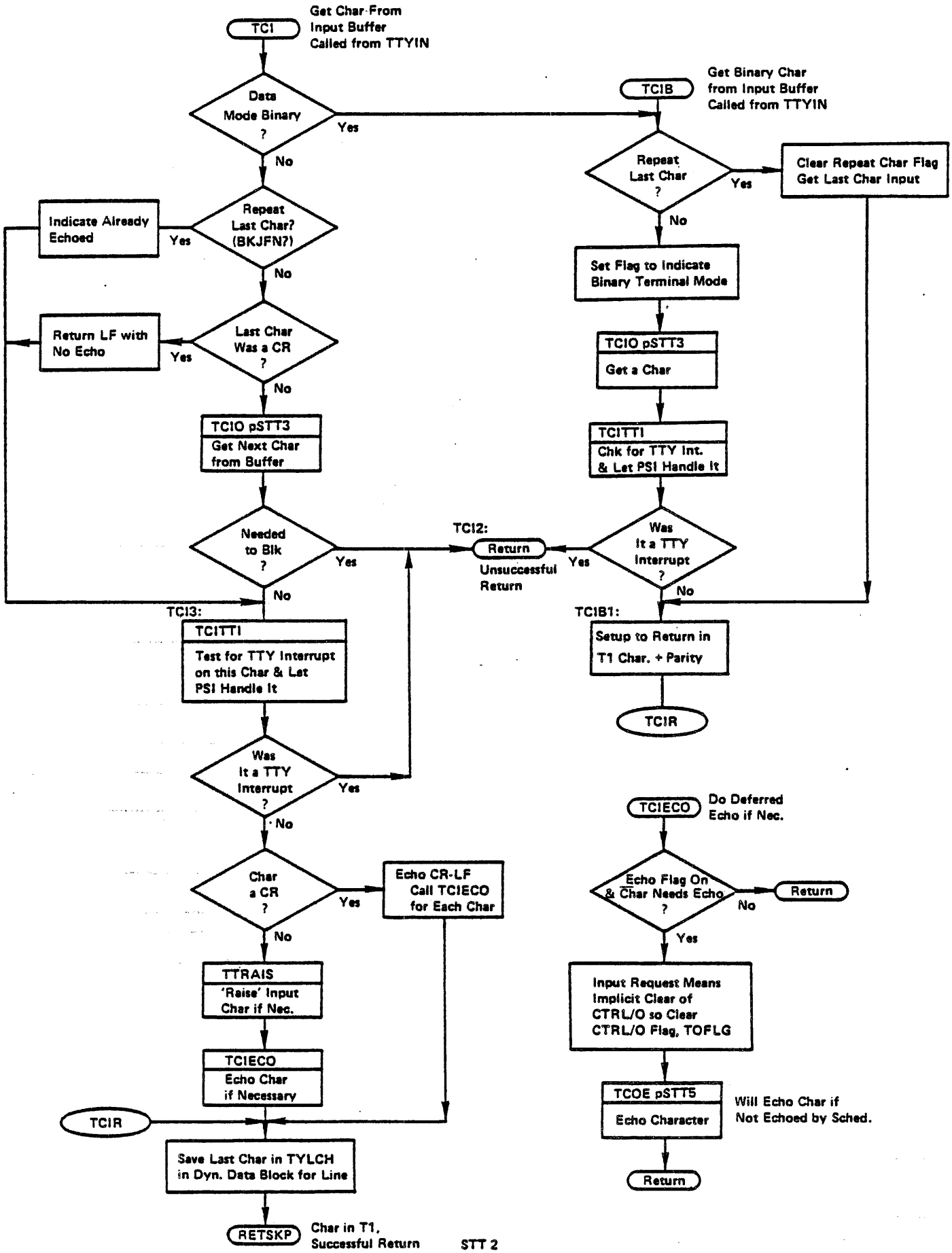
If none, take the lowest numbered cylinder read request from the current cylinder. If none, take the lowest numbered cylinder (write) request from the current cylinder.

JSYS CALL FLOWCHARTS

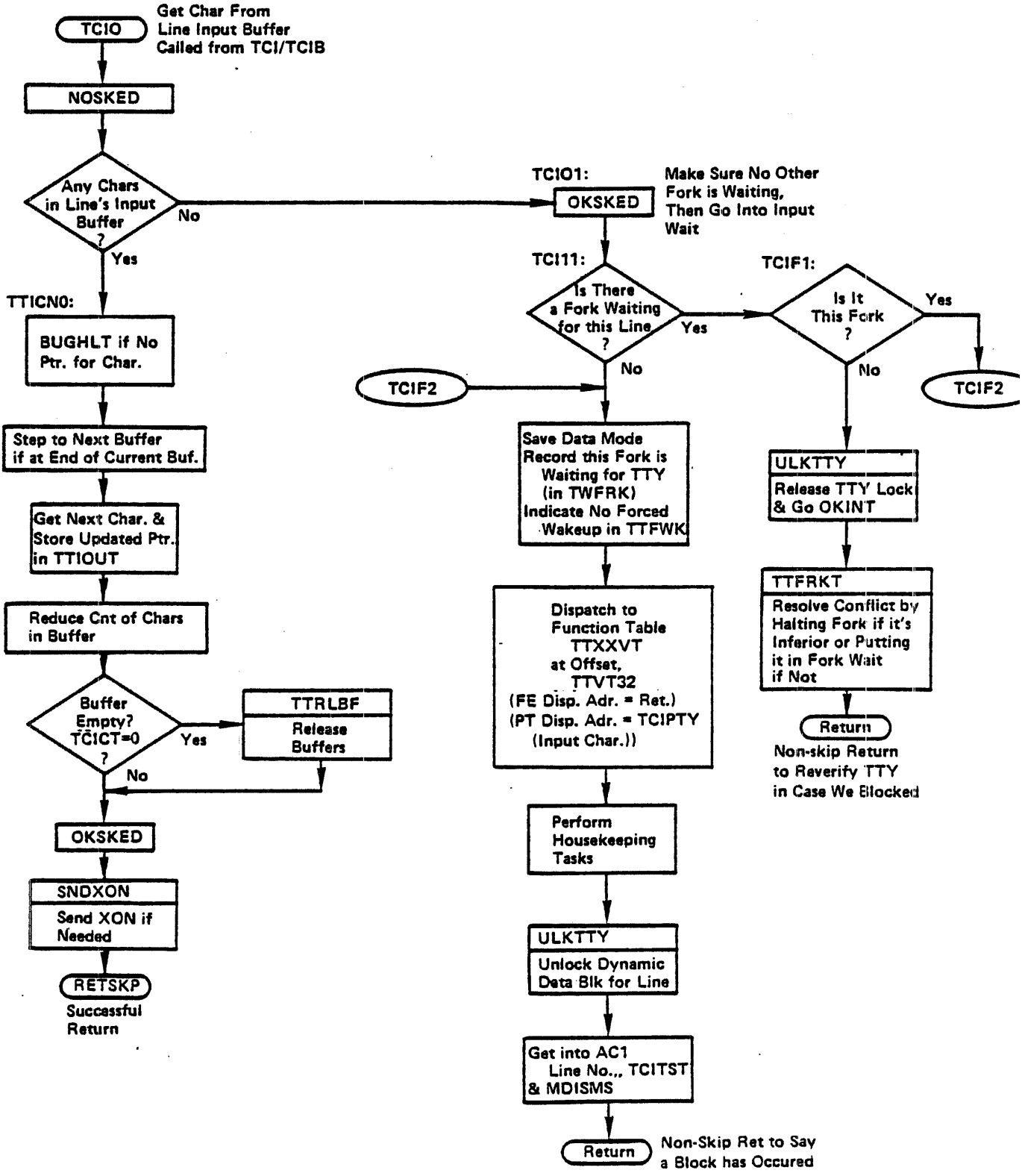
TTY DEPENDENT LEVEL

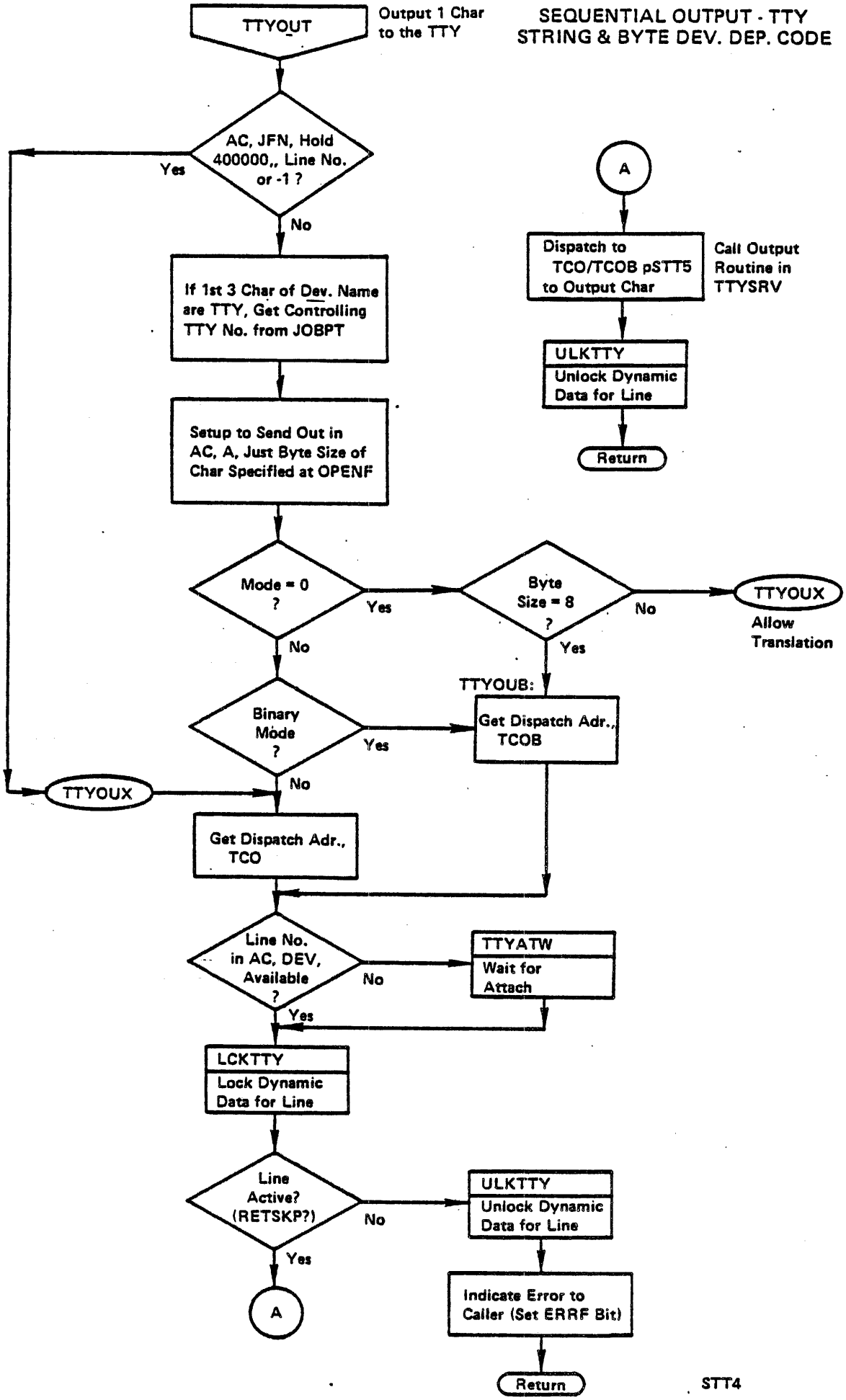

TTYOPN
OPENF-TTY
Dev. Dep. Code
CHKAPD
Set WRTF Bit if
Append Requested
TTYOP1:
CKJFTT
See if Device
is TTY:
Is It
?
Yes
RETSKP
Successful
Return
No
LCKDVL
Lock the DEV Tbls. Lock,
DEVLCK, & Go NOINT
FNDUNT
Get DEV Tbls. Index
for this Unit
DEVAV
See if Device is
Avail. to this Job
Is It
?
No
Unlock DEV
Tbls.
Err Ret-OPNX7
Yes
TTYASO
Assign TTY
if Possible
Is It
?
Yes
A
No
TTYOP2:
Unlock DEV Tbls. Lock
& Go OKINT
Setup in AC1
Line No.,, TSACT1/2
and MDISMS
Waiting
Yes
No
Return
Unsuccessful
Return
A
Store this User's Job No.
for Line in DEVUNT Tbl.
Set Open Bit, DV%OPN,
for Line in DEVCHR Tbl.
Unlock DEV Tbls.
Lock, DEVLCK
TTOP3:
OKINT
Setup to Return
with SIZF Bit Set
RETSKP
Successful
Return
OTT1


TTYIN
Get 1 Input Char for the Line
SEQUENTIAL INPUT - TTY
STRING & BYTE DEV. DEP. CODE
Controlling TTY? (AC, JFN, Hold-1) ?
Yes
Get Line No. for Job from JOBPT
No
Job Attached to Line & BKJFN Done ?
Yes
No
JFN Hold 400000,, Line No. ?
Yes
No
RSCAN Ptr. Set ?
No
Yes
Get Char from RSCAN Buffer
If 1st 3 Char of Dev. Name are TTY, Get Control.TTY No. from JOBPT
Call Input Routine in TTYSRV
Yes
Null ?
Perform RESCAN Housekeeping
No
Return
OPENF Mode = 0 ?
Yes
Byte Size = 8 ?
No
TTINX
Use Normal Mode for TTYs
No
Yes
TTYINB:
OPENed in Binary ?
Yes
Get Dispatch Adr., TCIB
No
TTYINX
Get Dispatch Adr., TCI (Routine that Allows Translation)
A
Dispatch to TCI/TCIB pSTT2 to Input Char.
Blocked ?
Yes, Try Again
No
Line No. in AC, DEV, Available ?
No
TTYATW
Wait for Attach
Yes
ULKTTY
Unlock Dyn. Data for Line (Allow Alloc.)
LCKTTY
Lock Dyn. Data for Line (Prevents Dealloc.)
Real JFN ?
No, (i.e. 400000,, Line)
Yes
Line Active? (RETSKP?)
No
ULKTTY
Unlock Dynamic Data for Line
Setup to Return in AC, A, Just Byte Size of Char Specified at OPENF
Yes
A
Indicate Error to Caller (Set ERRF Bit)
Return
Return
(Char in AC, A)
STT1

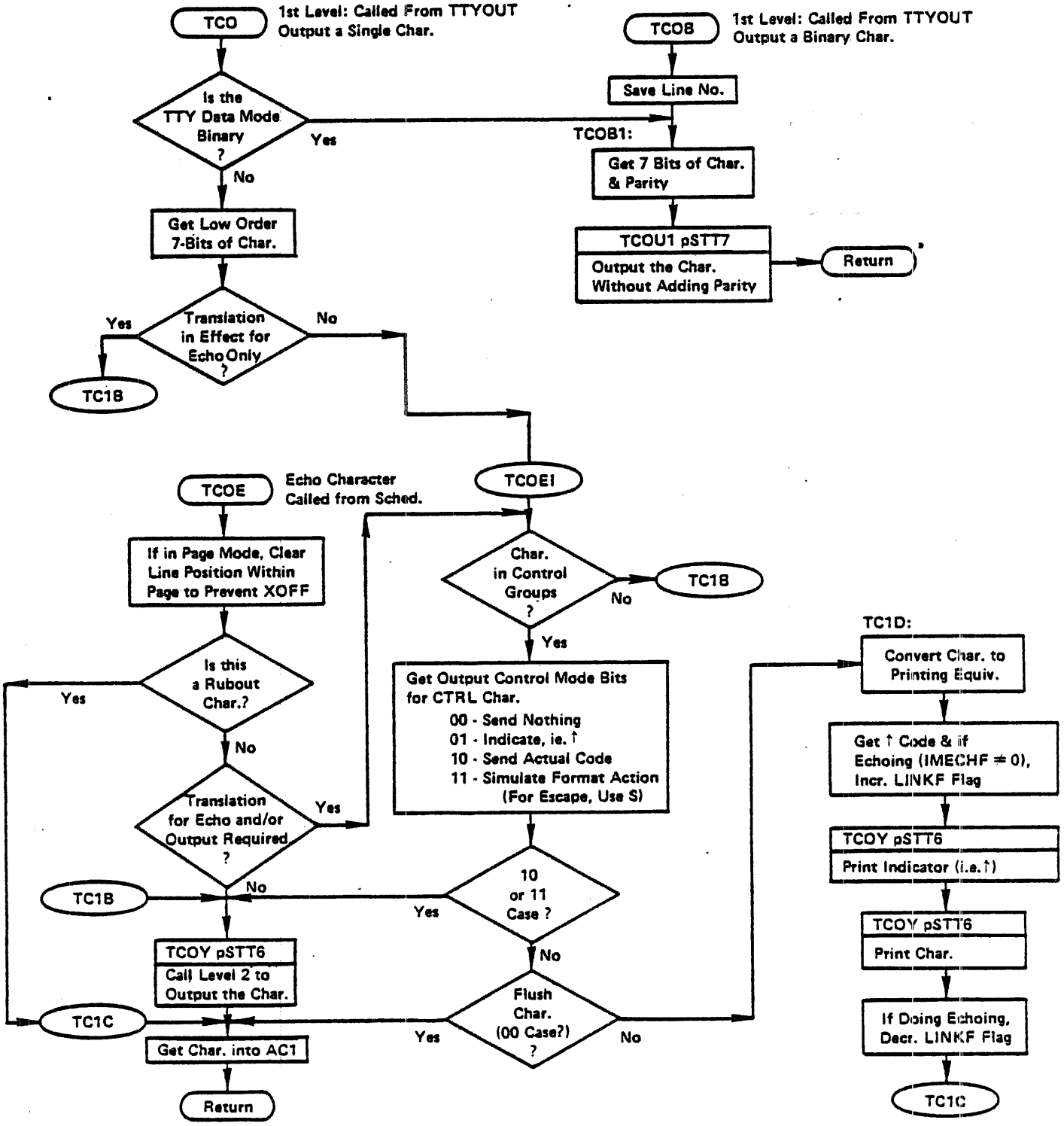

TC1
Get Char From Input Buffer Called from TTYIN
Data Mode Binary ?
Yes
No
TC1B
Get Binary Char from Input Buffer Called from TTYIN
Repeat Last Char ?
Yes
Clear Repeat Char Flag Get Last Char Input
No
Set Flag to Indicate Binary Terminal Mode
Indicate Already Echoed
Repeat Last Char? (BKJFN7)
Yes
No
Return LF with No Echo
Last Char Was a CR ?
Yes
No
TCIO pSTT3
Get Next Char from Buffer
TCIO pSTT3
Get a Char
TCITTI
Chk for TTY Int. & Let PSI Handle It
Needed to Blk ?
Yes
No
TC12:
Return
Unsuccessful Return
Was It a TTY Interrupt ?
Yes
No
TC13:
TCIB1:
TCITTI
Test for TTY Interrupt on this Char & Let PSI Handle It
Setup to Return in T1 Char. + Parity
TCIR
Was It a TTY Interrupt ?
Yes
No
TCIECO
Do Deferred Echo if Nec.
Char a CR ?
Yes
Echo CR-LF Call TCIECO for Each Char
No
Echo Flag On & Char Needs Echo ?
No
Return
Yes
TTRAIS
'Raise' Input Char if Nec.
Input Request Means Implicit Clear of CTRL/O so Clear CTRL/O Flag, TOFLG
TCIECO
Echo Char if Necessary
TCOE pSTT5
Echo Character
Will Echo Char if Not Echoed by Sched.
Return
TCIR
Save Last Char in TYLCH in Dyn. Data Block for Line
RETSKP
Char in T1, Successful Return

STT 2

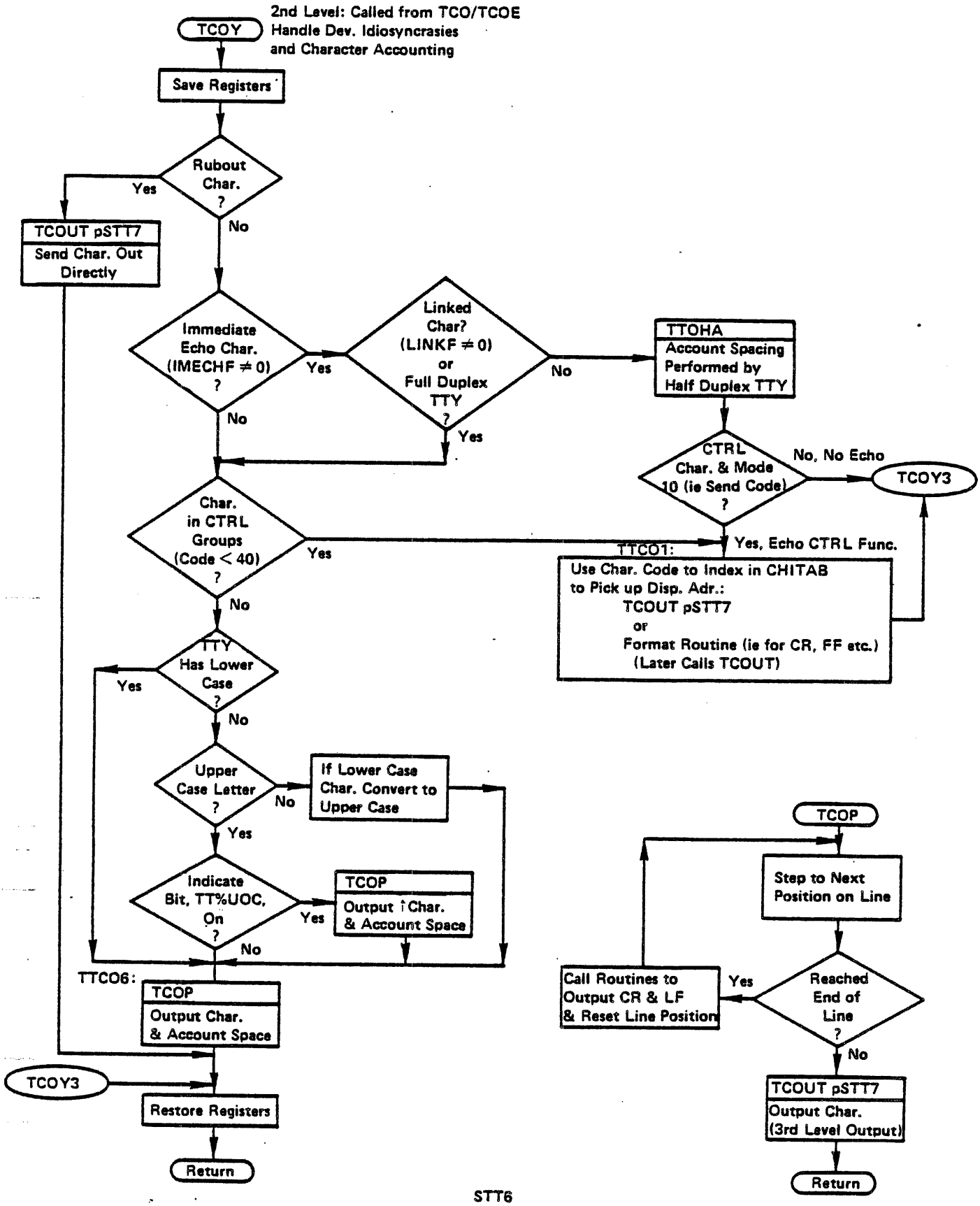

TCIO
Get Char From Line Input Buffer Called from TCI/TCIB
NOSKED
Any Chars in Line's Input Buffer ?
No
Yes
TTICN0:
BUGHLT if No Ptr. for Char.
Step to Next Buffer if at End of Current Buf.
Get Next Char. & Store Updated Ptr. in TTIOUT
Reduce Cnt of Chars in Buffer
Buffer Empty? TCICT=0 ?
Yes
TTRLBF
Release Buffers
No
OKSKED
SNDXON
Send XON if Needed
RETSKP
Successful Return
TCIO1:
OKSKED
Make Sure No Other Fork is Waiting, Then Go Into Input Wait
TCI11:
Is There a Fork Waiting for this Line ?
Yes
No
TCIF1:
Is It This Fork ?
Yes
No
TCIF2
TCIF2
Save Data Mode Record this Fork is Waiting for TTY (in TWFRK) Indicate No Forced Wakeup in TTFWK
Dispatch to Function Table TTXXVT at Offset, TTVT32
(FE Disp. Adr. = Ret.)
(PT Disp. Adr. = TCIPTY (Input Char.))
Perform Housekeeping Tasks
ULKTTY
Unlock Dynamic Data Blk for Line
Get into AC1 Line No.,, TCITST & MDISMS
Return
Non-Skip Ret to Say a Block has Occured
ULKTTY
Release TTY Lock & Go OKINT
TTFRKT
Resolve Conflict by Halting Fork if it's Inferior or Putting it in Fork Wait if Not
Return
Non-skip Return to Reverify TTY in Case We Blocked


TTYOUT
Output 1 Char to the TTY
SEQUENTIAL OUTPUT - TTY
STRING & BYTE DEV. DEP. CODE
AC, JFN, Hold 400000,, Line No. or -1 ?
Yes
No
If 1st 3 Char of Dev. Name are TTY, Get Controlling TTY No. from JOBPT
Setup to Send Out in AC, A, Just Byte Size of Char Specified at OPENF
Mode = 0 ?
Yes
No
Byte Size = 8 ?
No
Yes
TTYOUX
Allow Translation
TTYOUB:
Binary Mode ?
Yes
No
Get Dispatch Adr., TCOB
TTYOUX
Get Dispatch Adr., TCO
Line No. in AC, DEV, Available ?
No
Yes
TTYATW
Wait for Attach
LCKTTY
Lock Dynamic Data for Line
Line Active? (RETSKP?)
No
Yes
ULKTTY
Unlock Dynamic Data for Line
Indicate Error to Caller (Set ERRF Bit)
Return
A
A
Dispatch to TCO/TCOB pSTT5 to Output Char
Call Output Routine in TTYSRV
ULKTTY
Unlock Dynamic Data for Line
Return
STT4

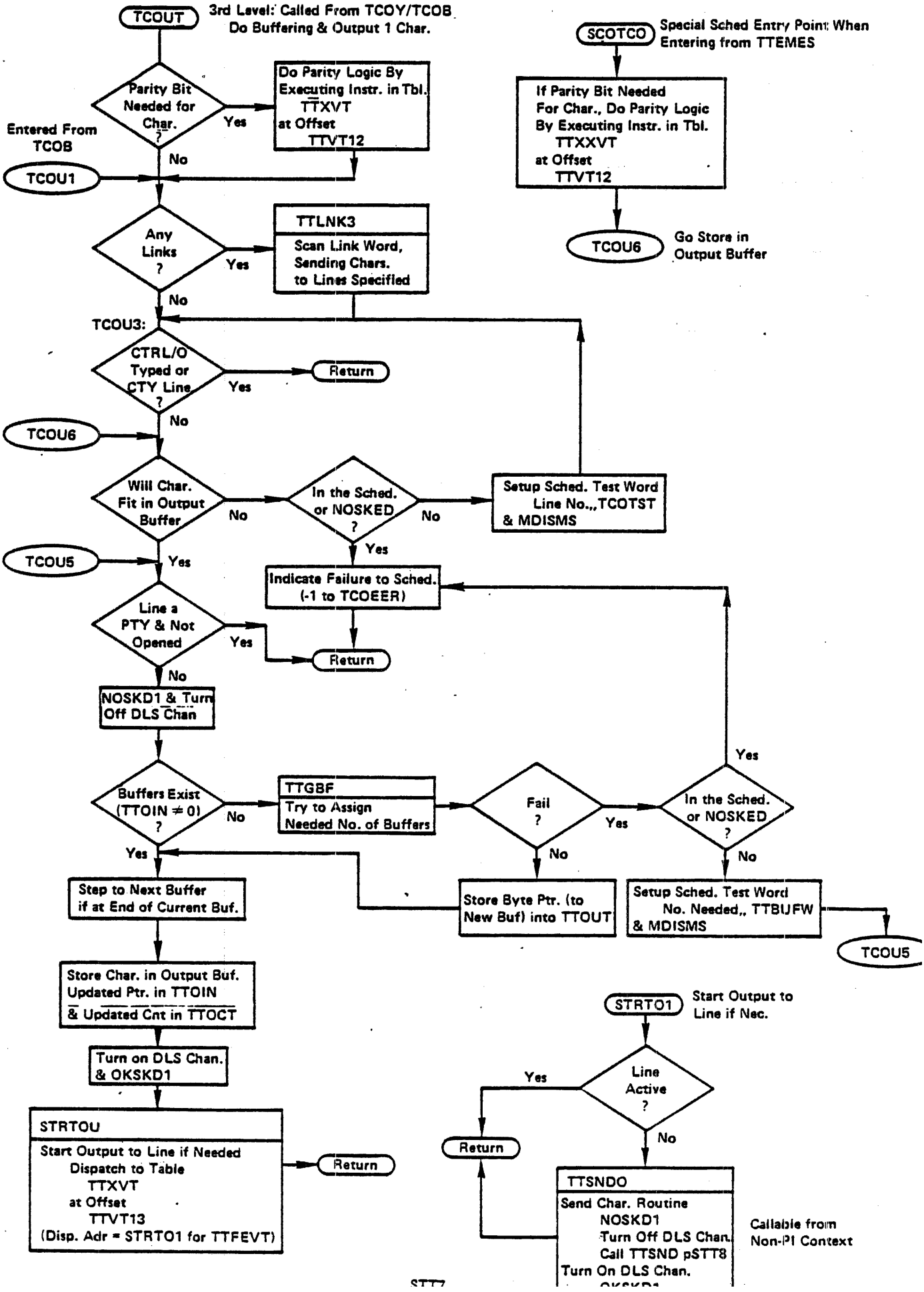
TCO - 1st Level - Translate According to Program's Desires
TCOY - 2nd Level - Do Links & Format for a Particular Device
TCOUT - 3rd Level - Do Buffering, etc.

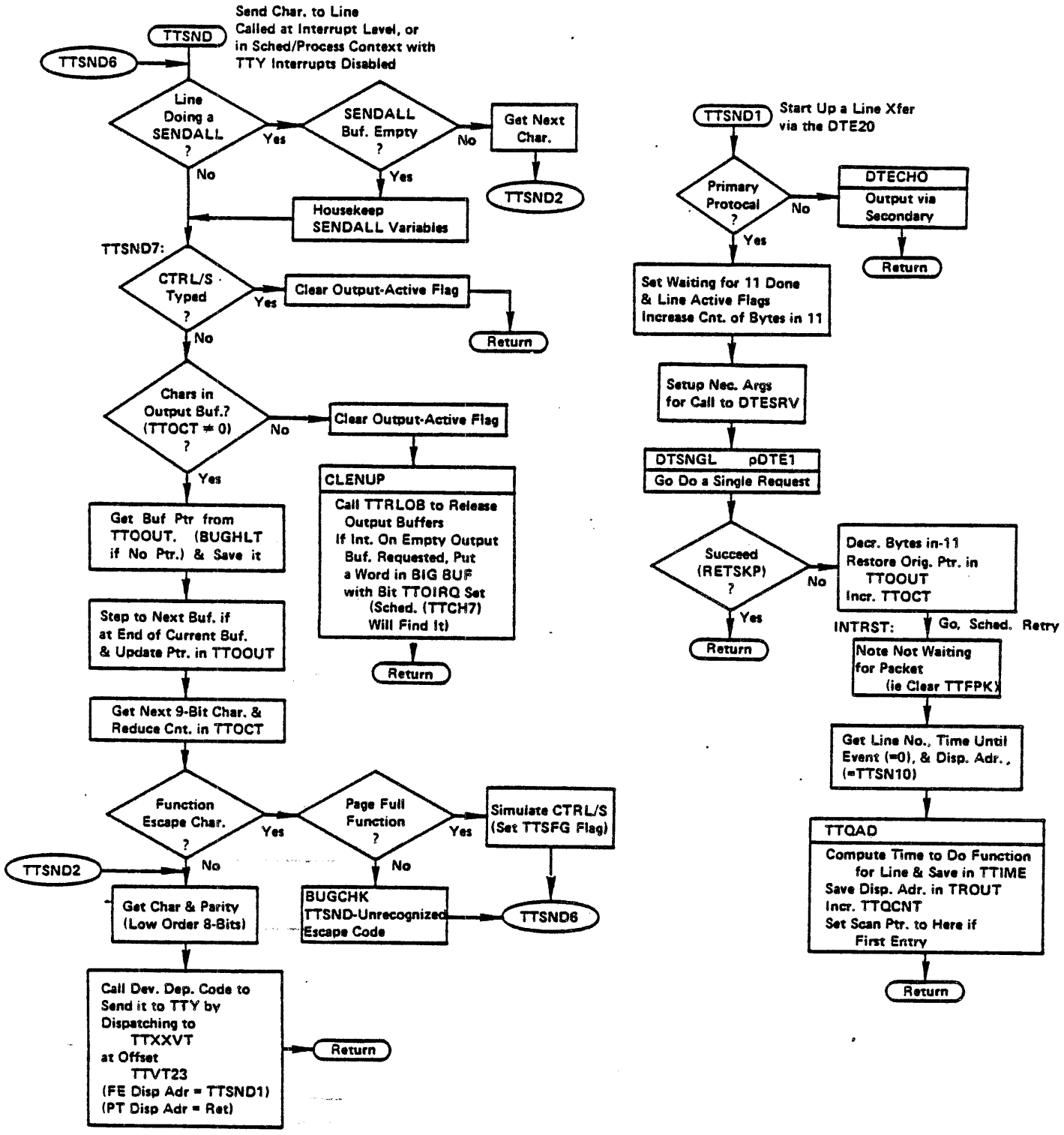


STT5

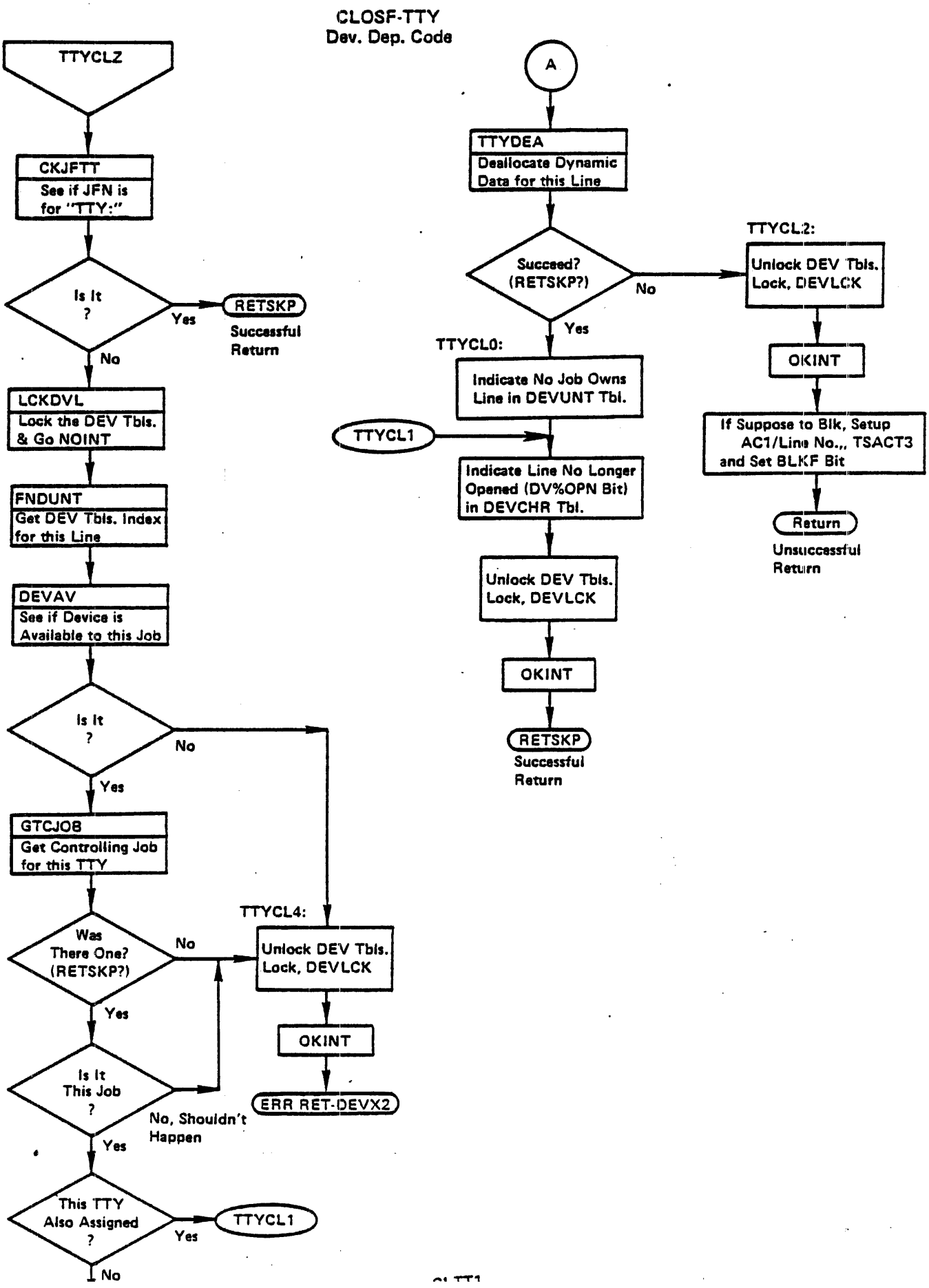

2nd Level: Called from TCO/TCOE
Handle Dev. Idiosyncrasies
and Character Accounting
TCOY
Save Registers
Rubout Char. ?
Yes
No
TCOUT pSTT7
Send Char. Out Directly
Immediate Echo Char. (IMECHF ≠ 0) ?
Yes
No
Linked Char? (LINKF ≠ 0) or Full Duplex TTY ?
No
Yes
TTOHA
Account Spacing Performed by Half Duplex TTY
CTRL Char. & Mode 10 (ie Send Code) ?
No, No Echo
TCOY3
Yes, Echo CTRL Func.
Char. in CTRL Groups (Code < 40) ?
Yes
No
TTCO1:
Use Char. Code to Index in CHITAB
to Pick up Disp. Adr.:
TCOUT pSTT7
or
Format Routine (ie for CR, FF etc.)
(Later Calls TCOUT)
TTY Has Lower Case ?
Yes
No
Upper Case Letter ?
No
If Lower Case Char. Convert to Upper Case
Yes
Indicate Bit, TT%UOC, On ?
Yes
TCOP
Output ↑Char. & Account Space
No
TTCO6:
TCOP
Output Char. & Account Space
TCOY3
Restore Registers
Return
TCOP
Step to Next Position on Line
Reached End of Line ?
Yes
Call Routines to Output CR & LF & Reset Line Position
No
TCOUT pSTT7
Output Char. (3rd Level Output)
Return

STT6


TCOUT
3rd Level: Called From TCOY/TCOB
Do Buffering & Output 1 Char.
Parity Bit Needed for Char. ?
Yes
Do Parity Logic By Executing Instr. in Tbl. TTXVT at Offset TTVT12
Entered From TCOB
No
TCOU1
Any Links ?
Yes
TTLNK3
Scan Link Word, Sending Chars. to Lines Specified
No
TCOU3:
CTRL/O Typed or CTY Line ?
Yes
Return
No
TCOU6
Will Char. Fit in Output Buffer
No
In the Sched. or NOSKED ?
No
Setup Sched. Test Word Line No.,,TCOTST & MDISMS
Yes
TCOU5
Yes
Indicate Failure to Sched. (-1 to TCOEER)
Line a PTY & Not Opened
Yes
Return
No
NOSKD1 & Turn Off DLS Chan
Buffers Exist (TTOIN ≠ 0) ?
No
TTGBF
Try to Assign Needed No. of Buffers
Fail ?
Yes
In the Sched. or NOSKED ?
Yes
No
No
Yes
Step to Next Buffer if at End of Current Buf.
Store Byte Ptr. (to New Buf) into TTOUT
Setup Sched. Test Word No. Needed,, TTBUFW & MDISMS
TCOU5
Store Char. in Output Buf. Updated Ptr. in TTOIN & Updated Cnt in TTOCT
Turn on DLS Chan. & OKSKD1
STRTOU
Start Output to Line if Needed
Dispatch to Table TTXVT at Offset TTVT13
(Disp. Adr = STRTO1 for TTFEVT)
Return
SCOTCO
Special Sched Entry Point When Entering from TTEMES
If Parity Bit Needed For Char., Do Parity Logic By Executing Instr. in Tbl. TTXXVT at Offset TTVT12
TCOU6
Go Store in Output Buffer
STRTO1
Start Output to Line if Nec.
Line Active ?
Yes
No
Return
TTSNDO
Send Char. Routine
NOSKD1
Turn Off DLS Chan.
Call TTSND pSTT8
Turn On DLS Chan.
Callable from Non-PI Context


Send Char. to Line
Called at Interrupt Level, or
in Sched/Process Context with
TTY Interrupts Disabled
TTSND
TTSND6
Line Doing a SENDALL ?
Yes
SENDALL Buf. Empty ?
No
Get Next Char.
TTSND2
Yes
No
Housekeep SENDALL Variables
TTSND7:
CTRL/S Typed ?
Yes
Clear Output-Active Flag
Return
No
Chars in Output Buf.? (TTOCT ≠ 0) ?
No
Clear Output-Active Flag
CLENUP
Call TTRLOB to Release Output Buffers
If Int. On Empty Output Buf. Requested, Put a Word in BIG BUF with Bit TTOIRQ Set (Sched. (TTCH7) Will Find It)
Return
Yes
Get Buf Ptr from TTOOUT. (BUGHLT if No Ptr.) & Save it
Step to Next Buf. if at End of Current Buf. & Update Ptr. in TTOOUT
Get Next 9-Bit Char. & Reduce Cnt. in TTOCT
Function Escape Char. ?
Yes
Page Full Function ?
Yes
Simulate CTRL/S (Set TTSFG Flag)
TTSND2
No
No
Get Char & Parity (Low Order 8-Bits)
BUGCHK
TTSND-Unrecognized Escape Code
TTSND6
Call Dev. Dep. Code to Send it to TTY by Dispatching to
TTXXVT
at Offset
TTVT23
(FE Disp Adr = TTSND1)
(PT Disp Adr = Ret)
Return
TTSND1
Start Up a Line Xfer via the DTE20
Primary Protocal ?
No
DTECHO
Output via Secondary
Return
Yes
Set Waiting for 11 Done & Line Active Flags
Increase Cnt. of Bytes in 11
Setup Nec. Args for Call to DTESRV
DTSNGL pDTE1
Go Do a Single Request
Succeed (RETSKP) ?
No
Decr. Bytes in-11
Restore Orig. Ptr. in TTOOUT
Incr. TTOCT
INTRST:
Go, Sched. Retry
Note Not Waiting for Packet
(ie Clear TTFPKX)
Yes
Return
Get Line No., Time Until Event (=0), & Disp. Adr., (=TTSN10)
TTQAD
Compute Time to Do Function for Line & Save in TTIME
Save Disp. Adr. in TROUT
Incr. TTQCNT
Set Scan Ptr. to Here if First Entry
Return

STT8

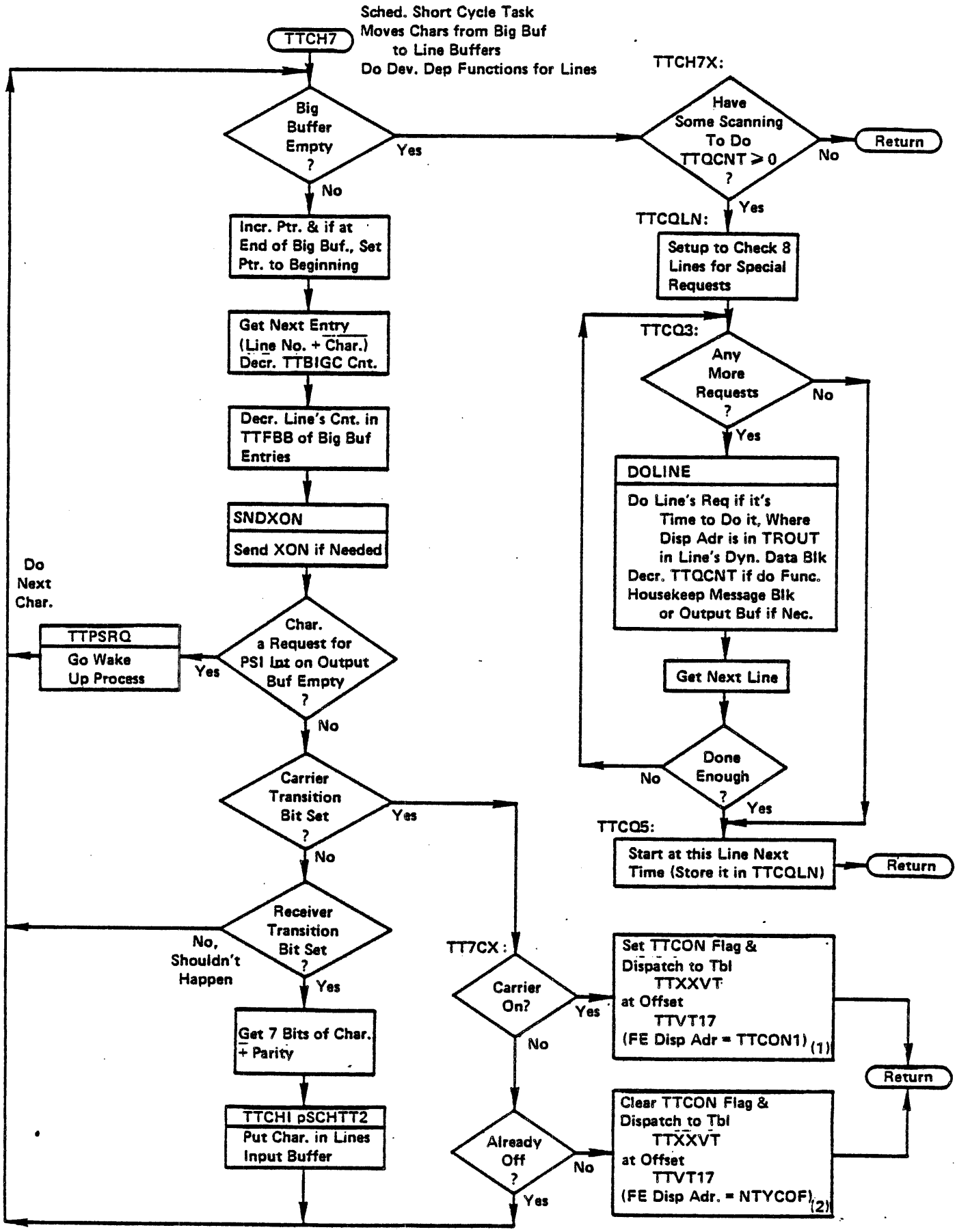

CLOSF-TTY
Dev. Dep. Code
TTYCLZ
CKJFTT
See if JFN is for "TTY:"
Is It ?
Yes
RETSKP
Successful Return
No
LCKDVL
Lock the DEV Tbls. & Go NOINT
FNDUNT
Get DEV Tbls. Index for this Line
DEVAV
See if Device is Available to this Job
Is It ?
No
Yes
GTCJOB
Get Controlling Job for this TTY
TTYCL4:
Was There One? (RETSKP?)
No
Unlock DEV Tbls. Lock, DEVLCK
Yes
OKINT
Is It This Job ?
ERR RET-DEVX2
No, Shouldn't Happen
Yes
This TTY Also Assigned ?
TTYCL1
Yes
No
A
TTYDEA
Deallocate Dynamic Data for this Line
Succeed? (RETSKP?)
No
TTYCL2:
Unlock DEV Tbls. Lock, DEVLCK
OKINT
If Suppose to Blk, Setup AC1/Line No.,, TSACT3 and Set BLKF Bit
Return
Unsuccessful Return
Yes
TTYCL0:
Indicate No Job Owns Line in DEVUNT Tbl.
TTYCL1
Indicate Line No Longer Opened (DV%OPN Bit) in DEVCHR Tbl.
Unlock DEV Tbls. Lock, DEVLCK
OKINT
RETSKP
Successful Return

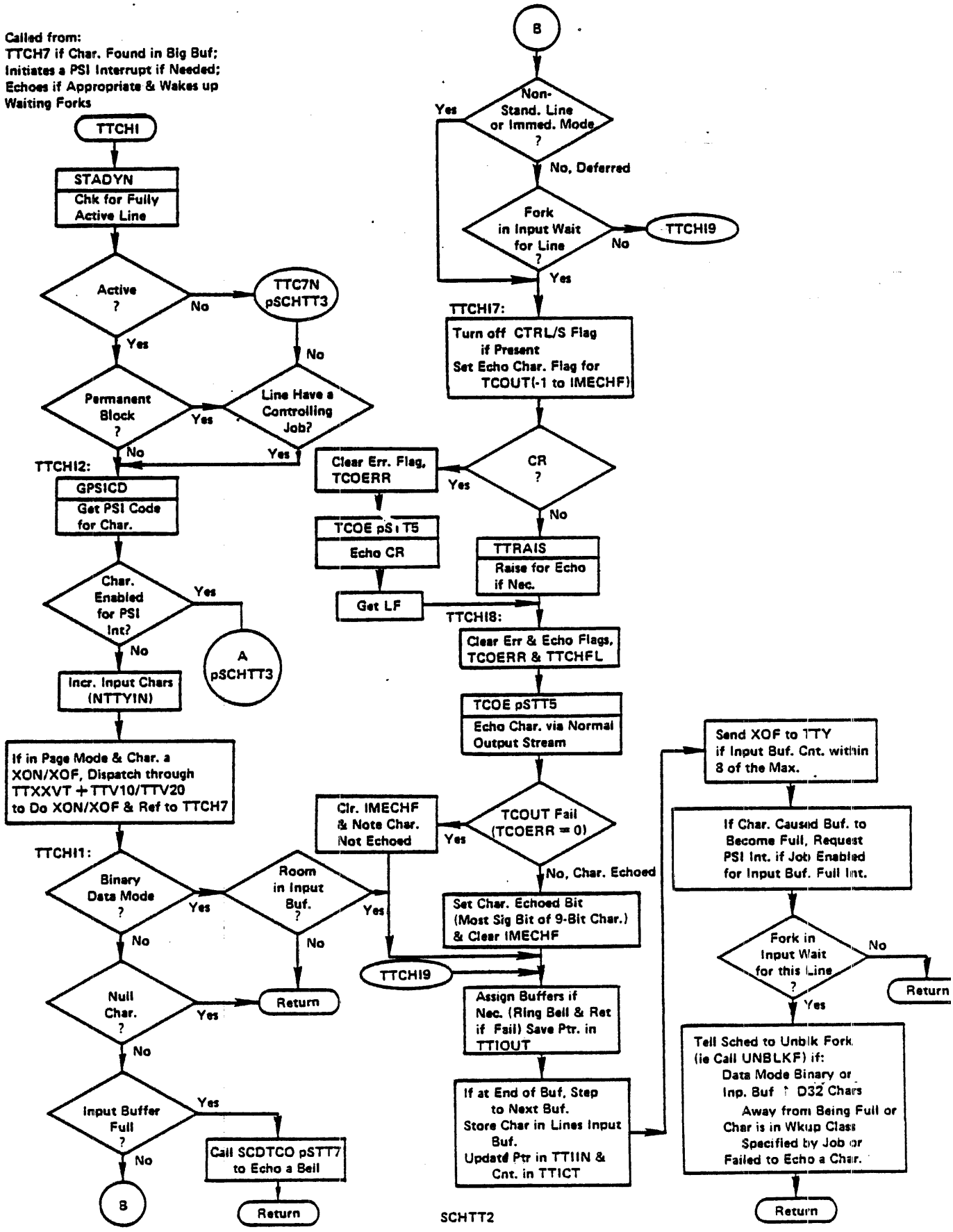
SCHEDULER TTY INPUT ANALYSIS & STORAGE

TTCH7 - Moves Characters from the Big Buffer to Line Buffers — SCHTT1

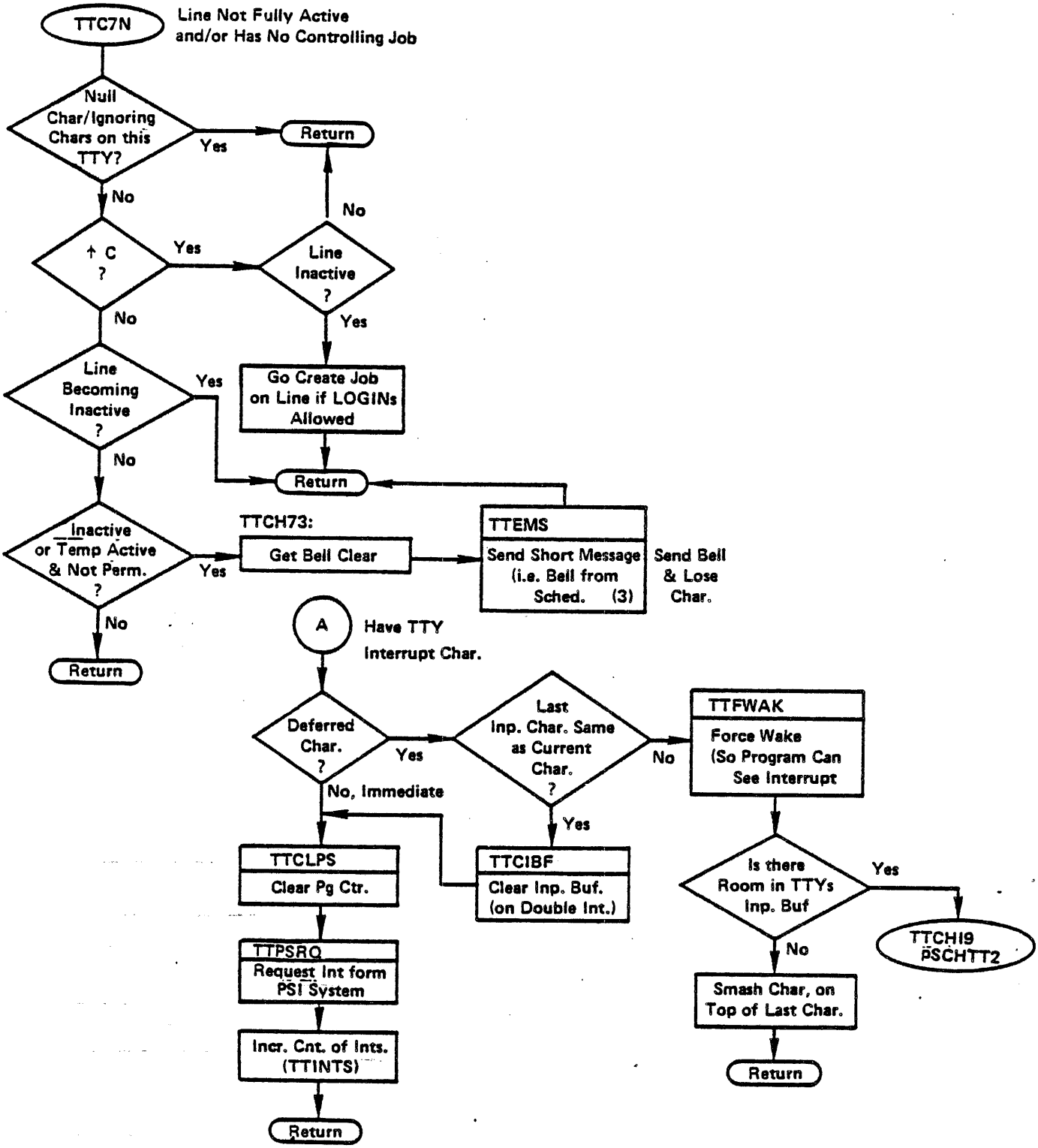
TTCHI - Initiates a PSI Interrupt if Needed, Echoes if Appropriate & Wakes Up Waiting Forks — SCHTT2

SCHEDULER TTY INPUT ANALYSIS & STORAGE




Called from:
TTCH7 if Char. Found in Big Buf;
Initiates a PSI Interrupt if Needed;
Echoes if Appropriate & Wakes up
Waiting Forks
TTCHI
STADYN
Chk for Fully Active Line
Active ?
No
TTC7N pSCHTT3
Yes
Permanent Block ?
Yes
Line Have a Controlling Job?
No
Yes
No
TTCHI2:
GPSICD
Get PSI Code for Char.
Char. Enabled for PSI Int?
Yes
A pSCHTT3
No
Incr. Input Chars (NTTYIN)
If in Page Mode & Char. a XON/XOF, Dispatch through TTXXVT + TTV10/TTV20 to Do XON/XOF & Ref to TTCH7
TTCHI1:
Binary Data Mode ?
Yes
Room in Input Buf. ?
Yes
No
No
Null Char. ?
Yes
Return
No
Input Buffer Full ?
Yes
Call SCDTCO pSTT7 to Echo a Bell
Return
No
B
B
Non-Stand. Line or Immed. Mode ?
Yes
No, Deferred
Fork in Input Wait for Line ?
No
TTCHI9
Yes
TTCHI7:
Turn off CTRL/S Flag if Present
Set Echo Char. Flag for TCOUT(-1 to IMECHF)
CR ?
Yes
Clear Err. Flag, TCOERR
TCOE pSTT5
Echo CR
Get LF
No
TTRAIS
Raise for Echo if Nec.
TTCHI8:
Clear Err & Echo Flags, TCOERR & TTCHFL
TCOE pSTT5
Echo Char. via Normal Output Stream
TCOUT Fail (TCOERR = 0)
Yes
Clr. IMECHF & Note Char. Not Echoed
No, Char. Echoed
Set Char. Echoed Bit (Most Sig Bit of 9-Bit Char.) & Clear IMECHF
TTCHI9
Assign Buffers if Nec. (Ring Bell & Ret if Fail) Save Ptr. in TTIOUT
If at End of Buf, Step to Next Buf.
Store Char in Lines Input Buf.
Updatd Ptr in TTIIN & Cnt. in TTICT
Send XOF to TTY if Input Buf. Cnt. within 8 of the Max.
If Char. Caused Buf. to Become Full, Request PSI Int. if Job Enabled for Input Buf. Full Int.
Fork in Input Wait for this Line ?
No
Return
Yes
Tell Sched to Unblk Fork (ie Call UNBLKF) if:
Data Mode Binary or
Inp. Buf ↑ D32 Chars Away from Being Full or
Char is in Wkup Class Specified by Job or
Failed to Echo a Char.
Return

SCHTT2


TTC7N
Line Not Fully Active and/or Has No Controlling Job
Null Char/Ignoring Chars on this TTY?
Yes
Return
No
↑ C ?
Yes
Line Inactive ?
No
Yes
Line Becoming Inactive ?
Yes
Go Create Job on Line if LOGINs Allowed
No
Return
Inactive or Temp Active & Not Perm. ?
Yes
TTCH73:
Get Bell Clear
TTEMS
Send Short Message (i.e. Bell from Sched. (3)
Send Bell & Lose Char.
No
Return
A
Have TTY Interrupt Char.
Deferred Char. ?
Yes
Last Inp. Char. Same as Current Char. ?
No
TTFWAK
Force Wake (So Program Can See Interrupt
No, Immediate
Yes
TTCLPS
Clear Pg Ctr.
TTCIBF
Clear Inp. Buf. (on Double Int.)
Is there Room in TTYs Inp. Buf
Yes
TTCH19 PSCHTT2
No
Smash Char, on Top of Last Char.
Return
TTPSRQ
Request Int form PSI System
Incr. Cnt. of Ints. (TTINTS)
Return

Scheduler TTY Input Comments

TTCH7

(1) The carrier-on routine for the FE device is TTCON1. If the line is in use or a job is being created, it just returns. Otherwise, it creates a job by the CTRL/C mechanism (i.e., putting a request in Scheduler's Request Queue, SCDRQB) before returning.

(2) The carrier-off routine for the FE device is NTYCOF. It flushes outputs and issues an interrupt via the PSI system if process has enabled for carrier-off interrupt. It then issues a monitor-internal interrupt via routine, PSIR4, which causes the top fork to go to JOBCOF in MEXEC to cause the job to be detached.

TTC7N

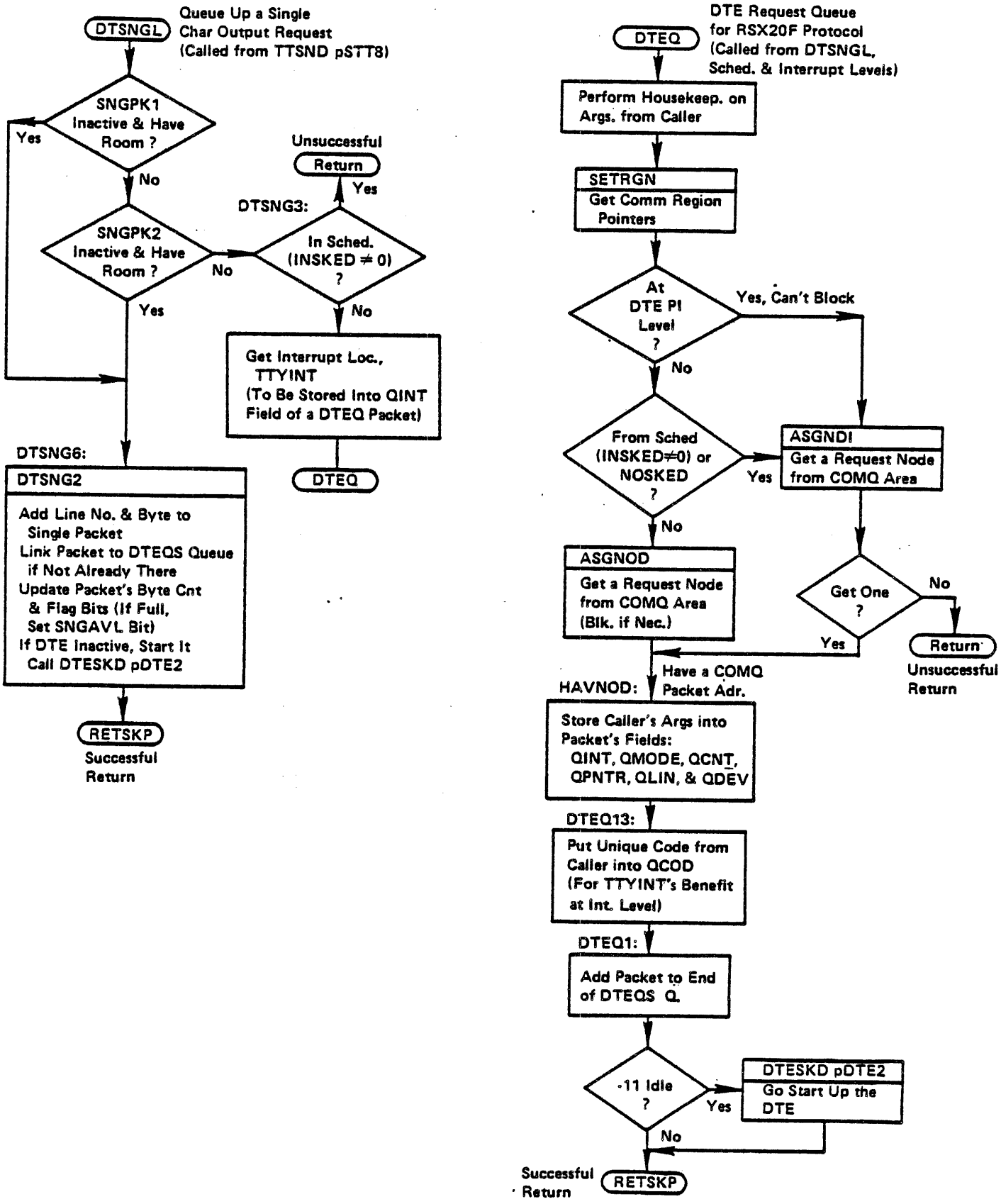
(3) TTEMES is called at Scheduler Level to send a short message to a line. If the line is active, it appends characters to the line's output buffer. If the line is not active, it creates a message-length dynamic block for the line and puts the characters into this block.

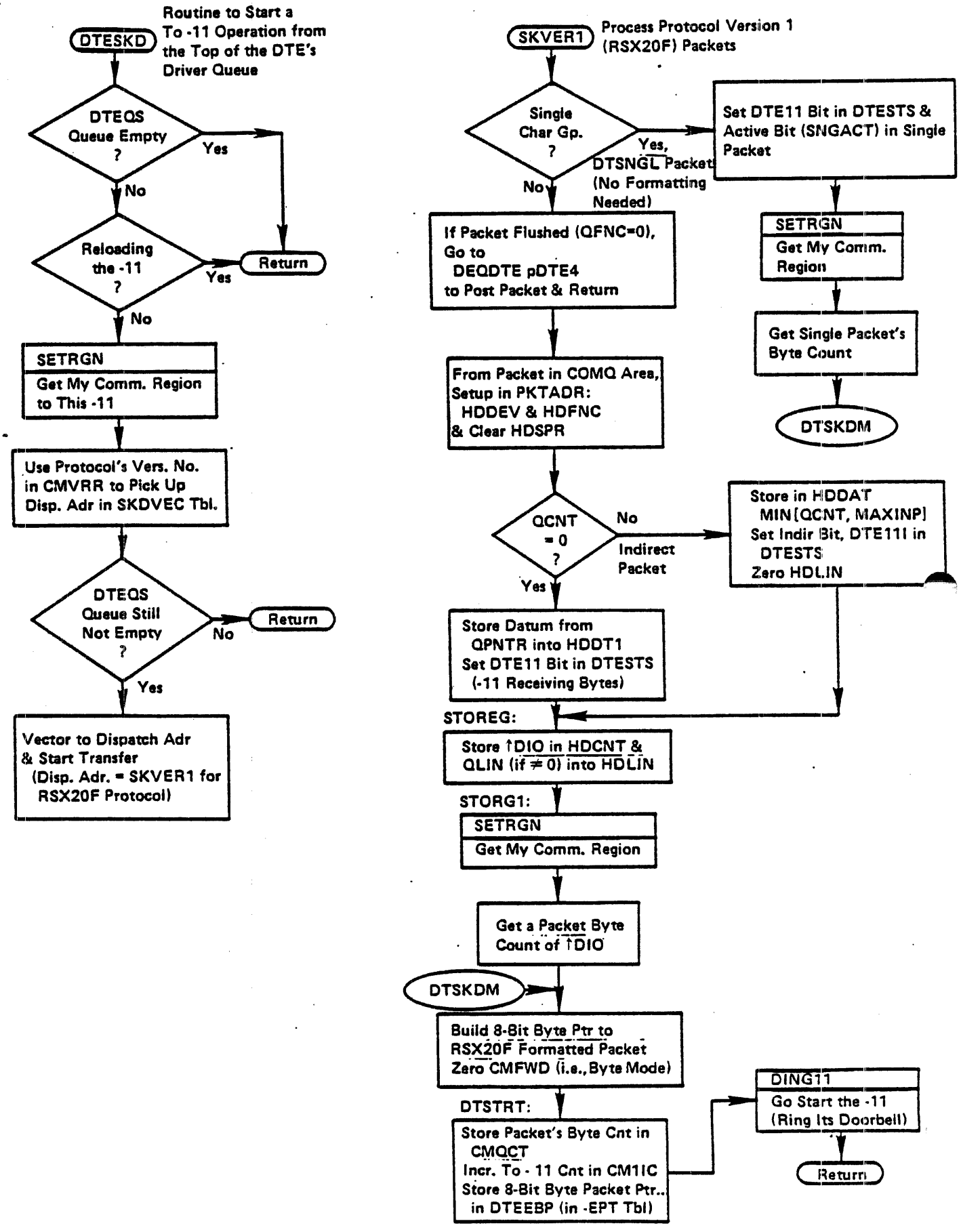
TTEMES calls SCDTCO (pSTT7) to output each character via TCOUT to the buffer or message block.

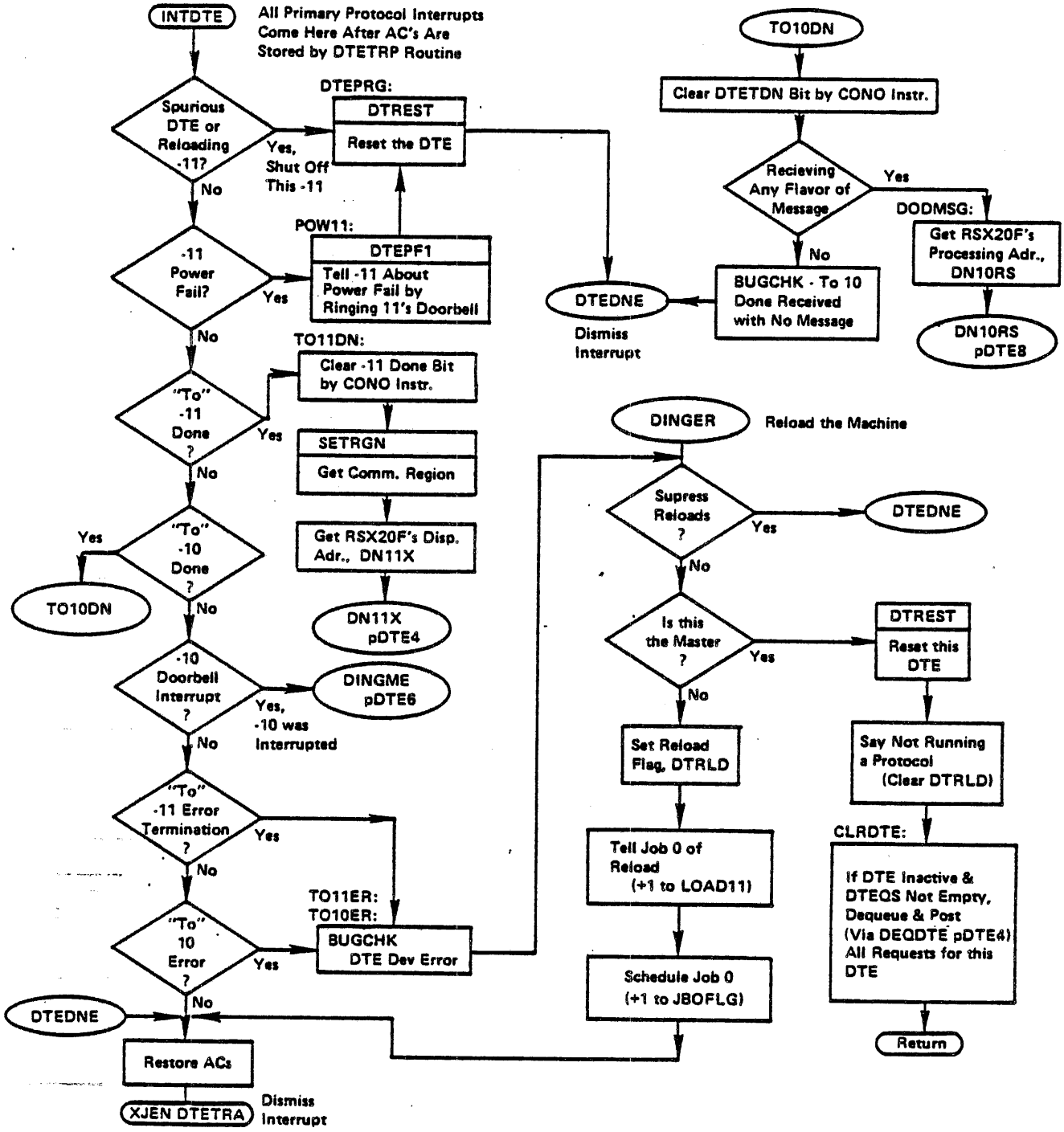
REQUESTING DTE OUTPUT & DTE INTERRUPT HANDLING FLOWCHARTS

(DTE PROTOCOL HANDLER)

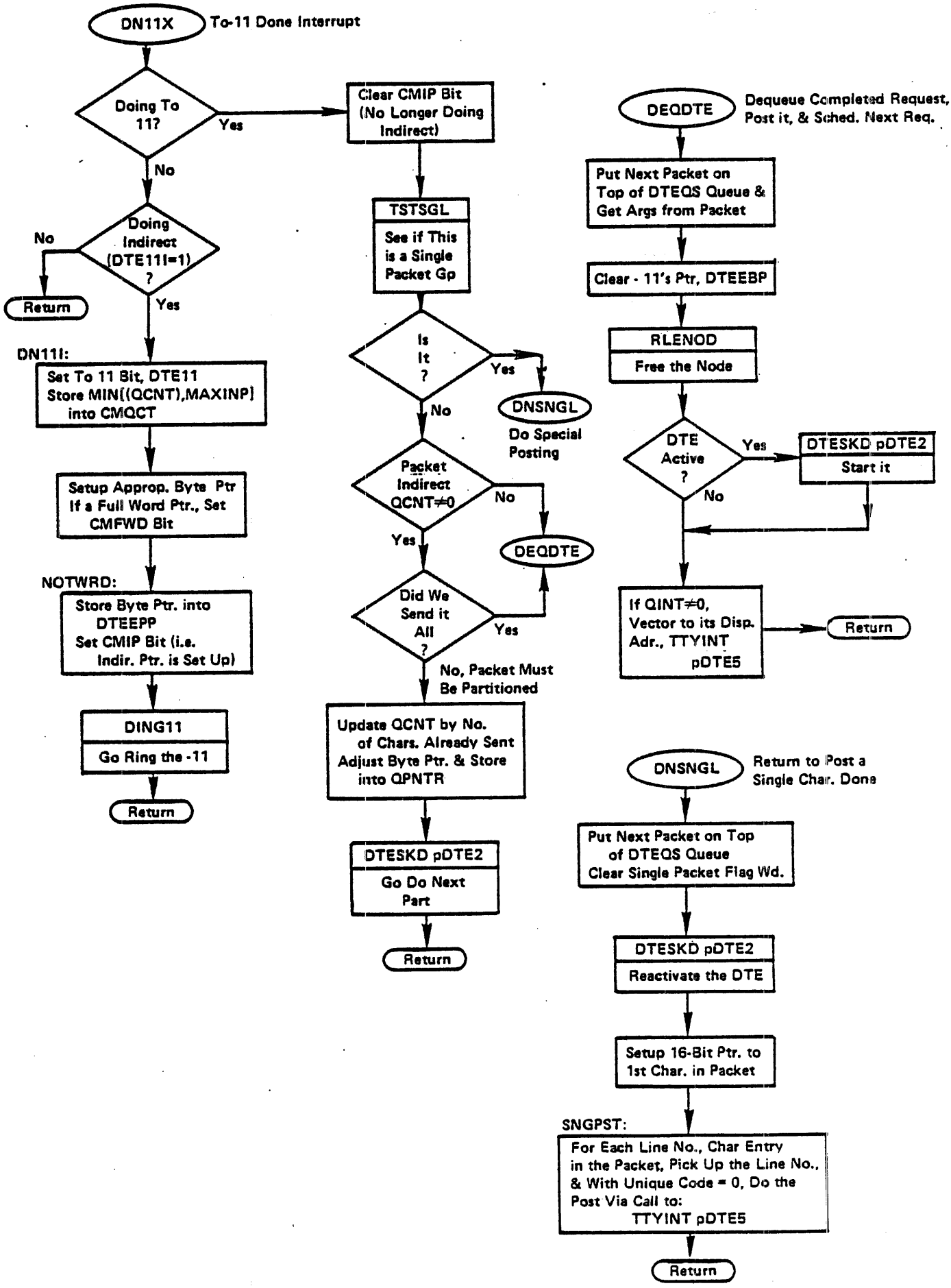
REQUESTING DTE OUTPUT
Called From Interrupt, JSYS
& Scheduler Levels

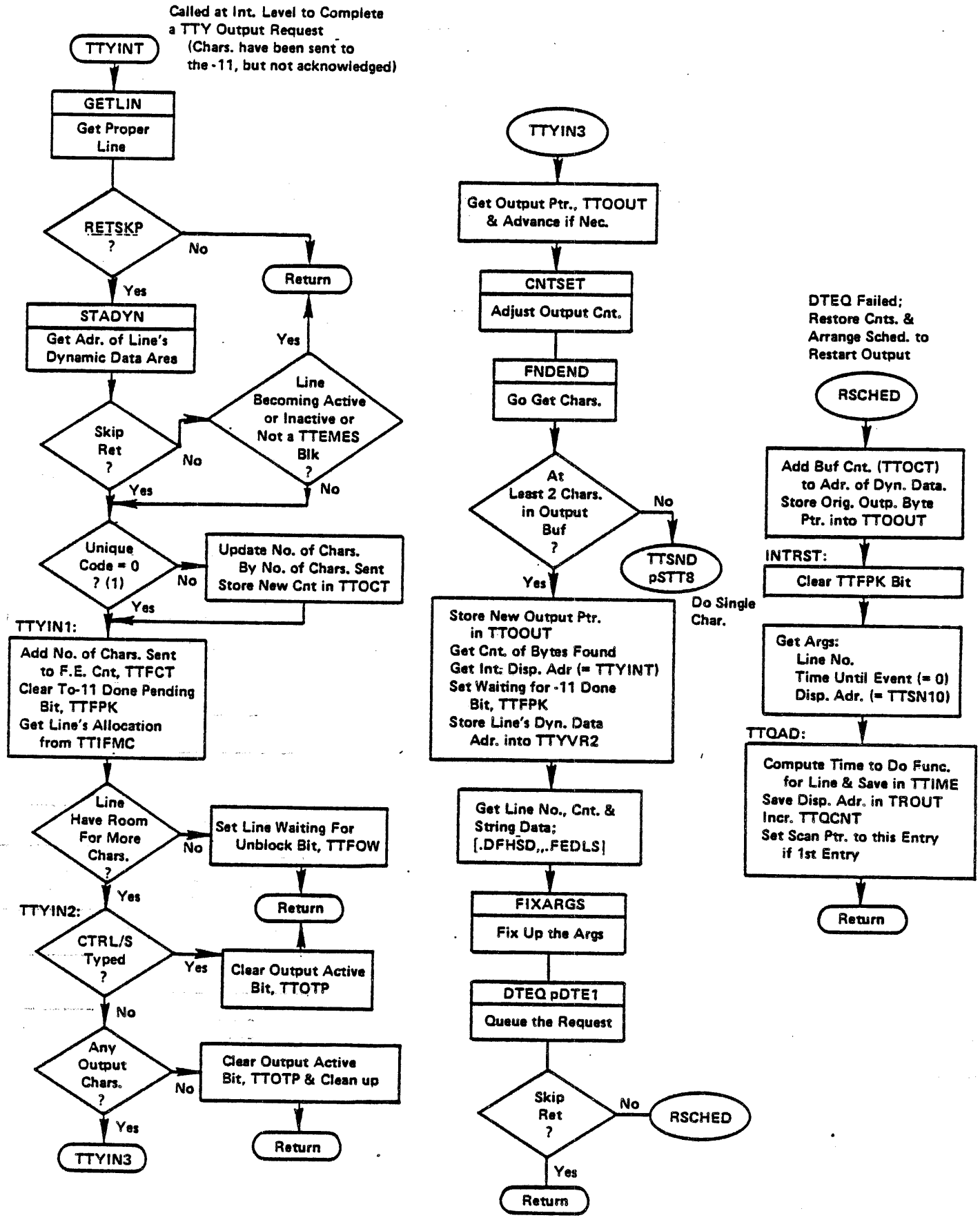


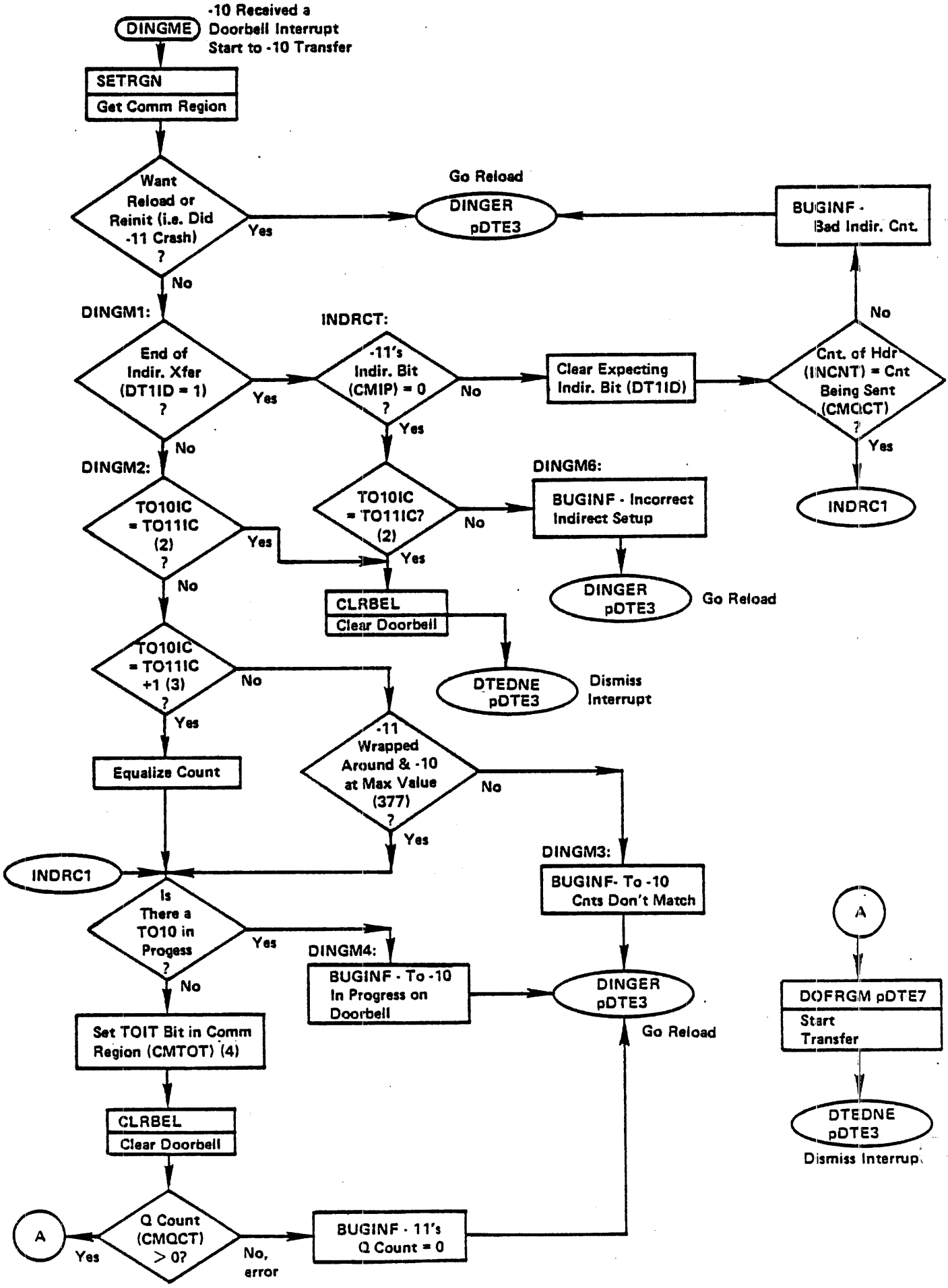

Routine to Start a To -11 Operation from the Top of the DTE's Driver Queue
DTESKD
DTEQS Queue Empty ?
Yes
No
Reloading the -11 ?
Yes
Return
No
SETRGN
Get My Comm. Region to This -11
Use Protocol's Vers. No. in CMVRR to Pick Up Disp. Adr in SKDVEC Tbl.
DTEQS Queue Still Not Empty ?
No
Return
Yes
Vector to Dispatch Adr & Start Transfer (Disp. Adr. = SKVER1 for RSX20F Protocol)
SKVER1
Process Protocol Version 1 (RSX20F) Packets
Single Char Gp. ?
Yes, DTSNGL Packet (No Formatting Needed)
Set DTE11 Bit in DTESTS & Active Bit (SNGACT) in Single Packet
No
If Packet Flushed (QFNC=0), Go to DEQDTE pDTE4 to Post Packet & Return
SETRGN
Get My Comm. Region
Get Single Packet's Byte Count
DTSKDM
From Packet in COMQ Area, Setup in PKTADR: HDDEV & HDFNC & Clear HDSPR
QCNT = 0 ?
No
Indirect Packet
Store in HDDAT MIN[QCNT, MAXINP] Set Indir Bit, DTE11I in DTESTS Zero HDLIN
Yes
Store Datum from QPNTR into HDDT1 Set DTE11 Bit in DTESTS (-11 Receiving Bytes)
STOREG:
Store ↑DIO in HDCNT & QLIN (if ≠ 0) into HDLIN
STORG1:
SETRGN
Get My Comm. Region
Get a Packet Byte Count of ↑DIO
DTSKDM
Build 8-Bit Byte Ptr to RSX20F Formatted Packet Zero CMFWD (i.e., Byte Mode)
DTSTRT:
Store Packet's Byte Cnt in CMQCT Incr. To - 11 Cnt in CM1IC Store 8-Bit Byte Packet Ptr.. in DTEEBP (in -EPT Tbl)
DING11
Go Start the -11 (Ring Its Doorbell)
Return

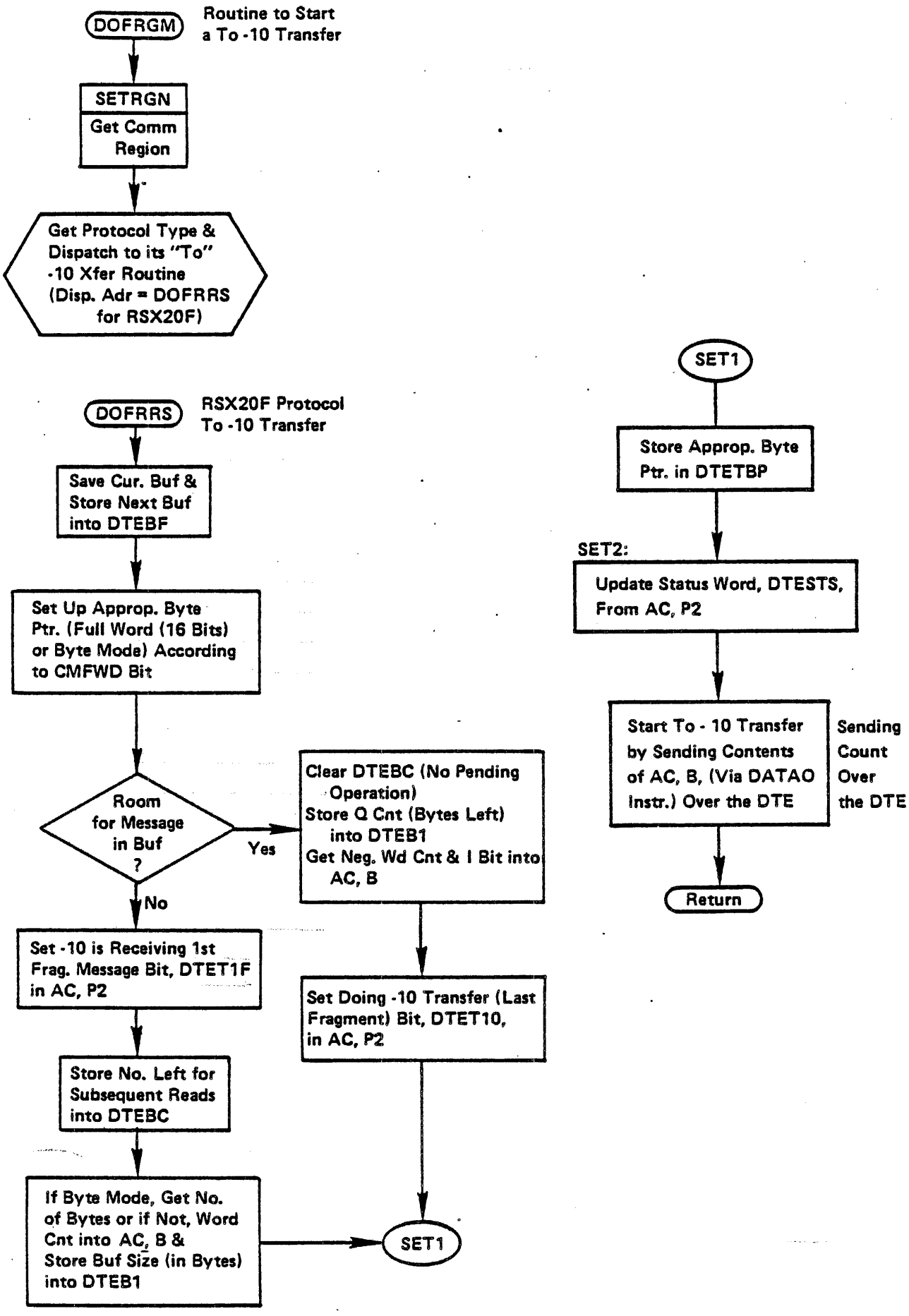

INTDTE
All Primary Protocol Interrupts Come Here After AC's Are Stored by DTETRP Routine
Spurious DTE or Reloading -11?
Yes, Shut Off This -11
No
DTEPRG:
DTREST
Reset the DTE
-11 Power Fail?
Yes
POW11:
DTEPF1
Tell -11 About Power Fail by Ringing 11's Doorbell
No
"To" -11 Done ?
Yes
TO11DN:
Clear -11 Done Bit by CONO Instr.
No
SETRGN
Get Comm. Region
"To" -10 Done ?
Yes
TO10DN
No
Get RSX20F's Disp. Adr., DN11X
DN11X pDTE4
-10 Doorbell Interrupt ?
Yes, -10 was Interrupted
DINGME pDTE6
No
"To" -11 Error Termination ?
Yes
No
"To" 10 Error ?
Yes
TO11ER:
TO10ER:
BUGCHK DTE Dev Error
No
DTEDNE
Restore ACs
XJEN DTETRA
Dismiss Interrupt
TO10DN
Clear DTETDN Bit by CONO Instr.
Recieving Any Flavor of Message
Yes
DODMSG:
Get RSX20F's Processing Adr., DN10RS
DN10RS pDTE8
No
BUGCHK - To 10 Done Received with No Message
DTEDNE
Dismiss Interrupt
DINGER
Reload the Machine
Supress Reloads ?
Yes
DTEDNE
No
Is this the Master ?
Yes
DTREST
Reset this DTE
No
Set Reload Flag, DTRLD
Tell Job 0 of Reload (+1 to LOAD11)
Schedule Job 0 (+1 to JBOFLG)
Say Not Running a Protocol (Clear DTRLD)
CLRDTE:
If DTE Inactive & DTEQS Not Empty, Dequeue & Post (Via DEQDTE pDTE4) All Requests for this DTE
Return

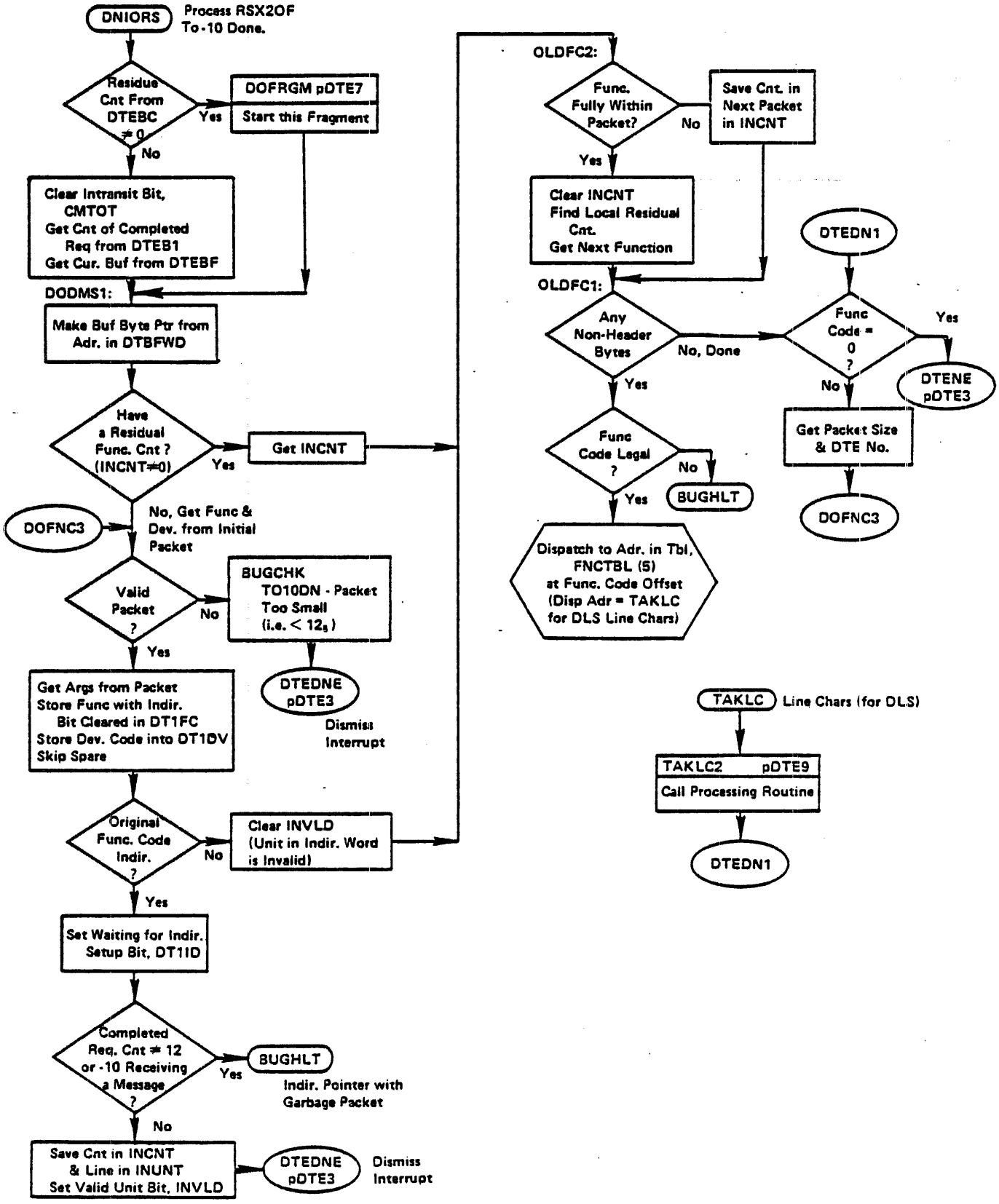
DTE3


DN11X
To-11 Done Interrupt
Doing To 11?
Yes
No
Clear CMIP Bit (No Longer Doing Indirect)
Doing Indirect (DTE11I=1) ?
No
Return
Yes
DN11I:
Set To 11 Bit, DTE11
Store MIN[(QCNT),MAXINP]
into CMQCT
Setup Approp. Byte Ptr
If a Full Word Ptr., Set
CMFWD Bit
NOTWRD:
Store Byte Ptr. into
DTEEPP
Set CMIP Bit (i.e.
Indir. Ptr. is Set Up)
DING11
Go Ring the -11
Return
TSTSGL
See if This is a Single Packet Gp
Is It ?
Yes
DNSNGL
Do Special Posting
No
Packet Indirect QCNT≠0
No
DEQDTE
Yes
Did We Send it All ?
Yes
No, Packet Must Be Partitioned
Update QCNT by No.
of Chars. Already Sent
Adjust Byte Ptr. & Store
into QPNTR
DTESKD pDTE2
Go Do Next Part
Return
DEQDTE
Dequeue Completed Request, Post it, & Sched. Next Req.
Put Next Packet on Top of DTEQS Queue & Get Args from Packet
Clear - 11's Ptr, DTEEBP
RLENOD
Free the Node
DTE Active ?
Yes
DTESKD pDTE2
Start it
No
If QINT≠0,
Vector to its Disp.
Adr., TTYINT
pDTE5
Return
DNSNGL
Return to Post a Single Char. Done
Put Next Packet on Top
of DTEQS Queue
Clear Single Packet Flag Wd.
DTESKD pDTE2
Reactivate the DTE
Setup 16-Bit Ptr. to
1st Char. in Packet
SNGPST:
For Each Line No., Char Entry
in the Packet, Pick Up the Line No.,
& With Unique Code = 0, Do the
Post Via Call to:
TTYINT pDTE5
Return

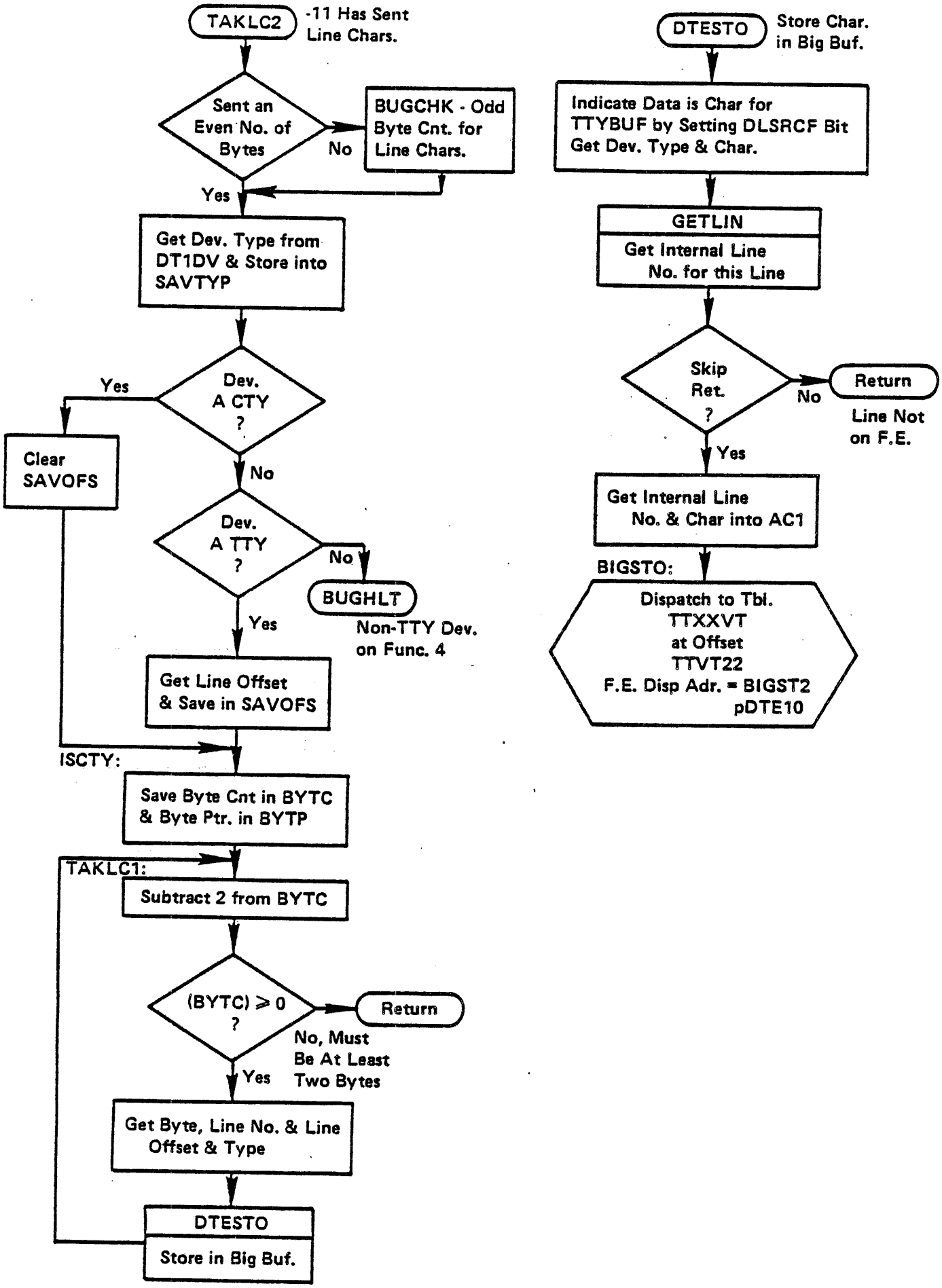

Called at Int. Level to Complete a TTY Output Request (Chars. have been sent to the -11, but not acknowledged)
TTYINT
GETLIN
Get Proper Line
RETSKP ?
No
Return
Yes
STADYN
Get Adr. of Line's Dynamic Data Area
Yes
Line Becoming Active or Inactive or Not a TTEMES Blk ?
Skip Ret ?
No
No
Yes
Unique Code = 0 ? (1)
No
Update No. of Chars. By No. of Chars. Sent Store New Cnt in TTOCT
Yes
TTYIN1:
Add No. of Chars. Sent to F.E. Cnt, TTFCT
Clear To-11 Done Pending Bit, TTFPK
Get Line's Allocation from TTIFMC
Line Have Room For More Chars. ?
No
Set Line Waiting For Unblock Bit, TTFOW
Return
Yes
TTYIN2:
CTRL/S Typed ?
Yes
Clear Output Active Bit, TTOTP
Return
No
Any Output Chars. ?
No
Clear Output Active Bit, TTOTP & Clean up
Return
Yes
TTYIN3
TTYIN3
Get Output Ptr., TTOOUT & Advance if Nec.
CNTSET
Adjust Output Cnt.
FNDEND
Go Get Chars.
At Least 2 Chars. in Output Buf ?
No
TTSND pSTT8
Do Single Char.
Yes
Store New Output Ptr. in TTOOUT
Get Cnt. of Bytes Found
Get Int. Disp. Adr (= TTYINT)
Set Waiting for -11 Done Bit, TTFPK
Store Line's Dyn. Data Adr. into TTYVR2
Get Line No., Cnt. & String Data; [.DFHSD,,.FEDLS]
FIXARGS
Fix Up the Args
DTEQ pDTE1
Queue the Request
Skip Ret ?
No
RSCHED
Yes
Return
DTE5
DTEQ Failed; Restore Cnts. & Arrange Sched. to Restart Output
RSCHED
Add Buf Cnt. (TTOCT) to Adr. of Dyn. Data. Store Orig. Outp. Byte Ptr. into TTOOUT
INTRST:
Clear TTFPK Bit
Get Args:
Line No.
Time Until Event (= 0)
Disp. Adr. (= TTSN10)
TTQAD:
Compute Time to Do Func. for Line & Save in TTIME
Save Disp. Adr. in TROUT
Incr. TTQCNT
Set Scan Ptr. to this Entry if 1st Entry
Return

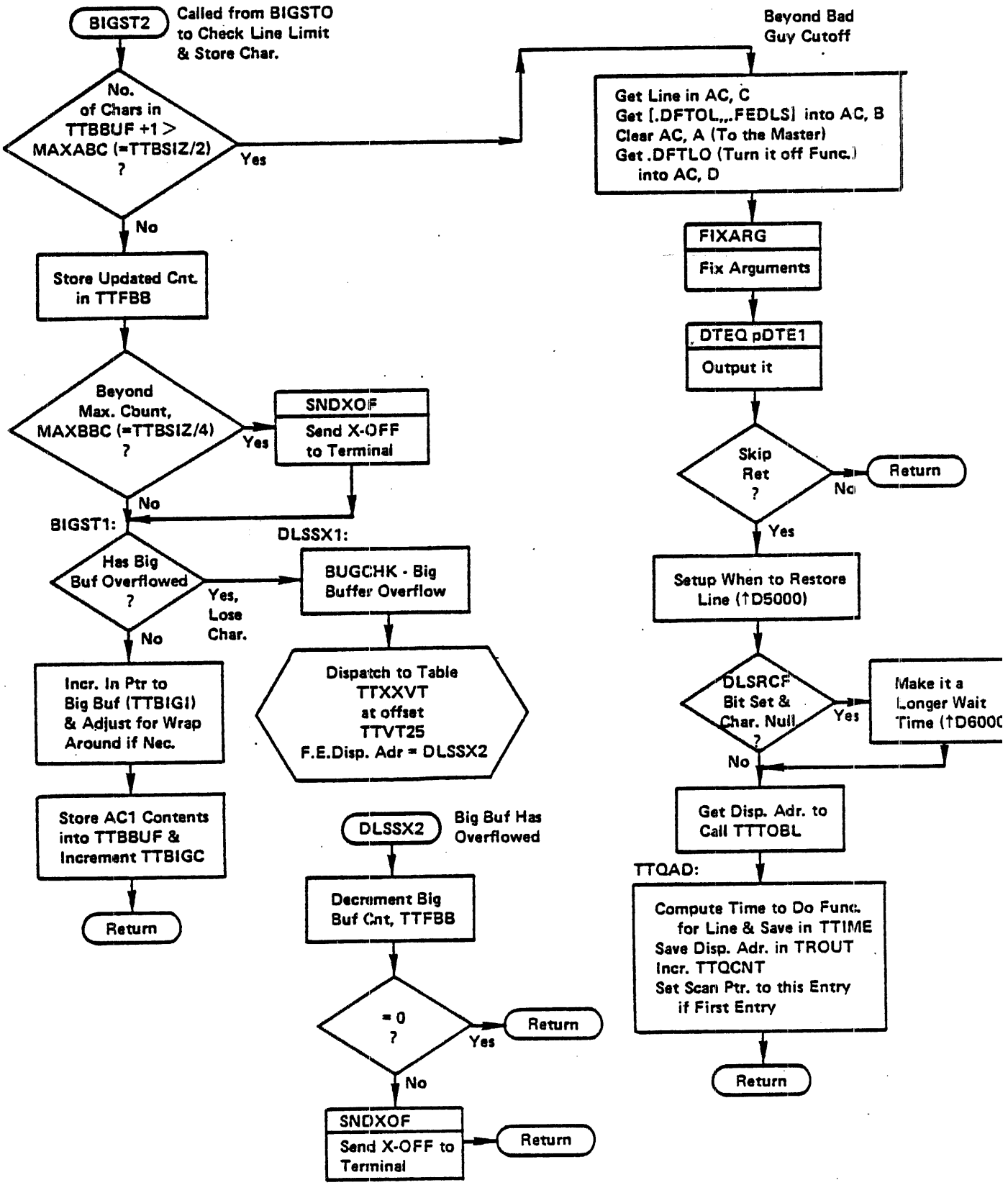

DINGME
-10 Received a Doorbell Interrupt Start to -10 Transfer
SETRGN
Get Comm Region
Want Reload or Reinit (i.e. Did -11 Crash) ?
Yes
Go Reload
DINGER pDTE3
BUGINF - Bad Indir. Cnt.
No
DINGM1:
End of Indir. Xfer (DT1ID = 1) ?
Yes
INDRCT:
-11's Indir. Bit (CMIP) = 0 ?
No
Clear Expecting Indir. Bit (DT1ID)
Cnt. of Hdr (INCNT) = Cnt Being Sent (CMQCT) ?
No
Yes
INDRC1
No
Yes
DINGM2:
TO10IC = TO11IC (2) ?
Yes
TO10IC = TO11IC? (2)
No
DINGM6:
BUGINF - Incorrect Indirect Setup
DINGER pDTE3
Go Reload
No
Yes
CLRBEL
Clear Doorbell
DTEDNE pDTE3
Dismiss Interrupt
TO10IC = TO11IC +1 (3) ?
No
Yes
-11 Wrapped Around & -10 at Max Value (377) ?
No
Yes
Equalize Count
INDRC1
DINGM3:
BUGINF- To -10 Cnts Don't Match
Is There a TO10 in Progess ?
Yes
DINGM4:
BUGINF - To -10 In Progress on Doorbell
DINGER pDTE3
Go Reload
No
Set TOIT Bit in Comm Region (CMTOT) (4)
CLRBEL
Clear Doorbell
A
Q Count (CMQCT) > 0?
Yes
No, error
BUGINF - 11's Q Count = 0
A
DOFRGM pDTE7
Start Transfer
DTEDNE pDTE3
Dismiss Interrup


DOFRGM
Routine to Start a To -10 Transfer
SETRGN
Get Comm Region
Get Protocol Type & Dispatch to its "To" -10 Xfer Routine (Disp. Adr = DOFRRS for RSX20F)
DOFRRS
RSX20F Protocol To -10 Transfer
Save Cur. Buf & Store Next Buf into DTEBF
Set Up Approp. Byte Ptr. (Full Word (16 Bits) or Byte Mode) According to CMFWD Bit
Room for Message in Buf ?
Yes
No
Clear DTEBC (No Pending Operation)
Store Q Cnt (Bytes Left) into DTEB1
Get Neg. Wd Cnt & I Bit into AC, B
Set -10 is Receiving 1st Frag. Message Bit, DTET1F in AC, P2
Store No. Left for Subsequent Reads into DTEBC
Set Doing -10 Transfer (Last Fragment) Bit, DTET10, in AC, P2
If Byte Mode, Get No. of Bytes or if Not, Word Cnt into AC, B & Store Buf Size (in Bytes) into DTEB1
SET1
SET1
Store Approp. Byte Ptr. in DTETBP
SET2:
Update Status Word, DTESTS, From AC, P2
Start To - 10 Transfer by Sending Contents of AC, B, (Via DATAO Instr.) Over the DTE
Sending Count Over the DTE
Return


DNIORS
Process RSX2OF To-10 Done.
Residue Cnt From DTEBC ≠ 0
Yes
DOFRGM pDTE7
Start this Fragment
No
Clear Intransit Bit, CMTOT
Get Cnt of Completed Req from DTEB1
Get Cur. Buf from DTEBF
DODMS1:
Make Buf Byte Ptr from Adr. in DTBFWD
Have a Residual Func. Cnt ? (INCNT≠0)
Yes
Get INCNT
No, Get Func & Dev. from Initial Packet
DOFNC3
Valid Packet ?
No
BUGCHK TO10DN - Packet Too Small (i.e. < 12s)
DTEDNE pDTE3
Dismiss Interrupt
Yes
Get Args from Packet
Store Func with Indir. Bit Cleared in DT1FC
Store Dev. Code into DT1DV
Skip Spare
Original Func. Code Indir. ?
No
Clear INVLD (Unit in Indir. Word is Invalid)
Yes
Set Waiting for Indir. Setup Bit, DT1ID
Completed Req. Cnt ≠ 12 or -10 Receiving a Message ?
Yes
BUGHLT
Indir. Pointer with Garbage Packet
No
Save Cnt in INCNT & Line in INUNT
Set Valid Unit Bit, INVLD
DTEDNE pDTE3
Dismiss Interrupt
OLDFC2:
Func. Fully Within Packet?
No
Save Cnt. in Next Packet in INCNT
Yes
Clear INCNT
Find Local Residual Cnt.
Get Next Function
OLDFC1:
Any Non-Header Bytes
No, Done
Yes
Func Code = 0 ?
Yes
DTENE pDTE3
No
Get Packet Size & DTE No.
DOFNC3
DTEDN1
Func Code Legal ?
No
BUGHLT
Yes
Dispatch to Adr. in Tbl, FNCTBL (5) at Func. Code Offset (Disp Adr = TAKLC for DLS Line Chars)
TAKLC
Line Chars (for DLS)
TAKLC2 pDTE9
Call Processing Routine
DTEDN1

DTE8


TAKLC2
-11 Has Sent Line Chars.
Sent an Even No. of Bytes
No
BUGCHK - Odd Byte Cnt. for Line Chars.
Yes
Get Dev. Type from DT1DV & Store into SAVTYP
Dev. A CTY ?
Yes
Clear SAVOFS
No
Dev. A TTY ?
No
BUGHLT
Non-TTY Dev. on Func. 4
Yes
Get Line Offset & Save in SAVOFS
ISCTY:
Save Byte Cnt in BYTC & Byte Ptr. in BYTP
TAKLC1:
Subtract 2 from BYTC
(BYTC) ≥ 0 ?
Return
No, Must Be At Least Two Bytes
Yes
Get Byte, Line No. & Line Offset & Type
DTESTO
Store in Big Buf.
DTESTO
Store Char. in Big Buf.
Indicate Data is Char for TTYBUF by Setting DLSRCF Bit Get Dev. Type & Char.
GETLIN
Get Internal Line No. for this Line
Skip Ret. ?
No
Return
Line Not on F.E.
Yes
Get Internal Line No. & Char into AC1
BIGSTO:
Dispatch to Tbl. TTXXVT at Offset TTVT22
F.E. Disp Adr. = BIGST2
pDTE10


BIGST2
Called from BIGSTO to Check Line Limit & Store Char.
No. of Chars in TTBBUF +1 > MAXABC (=TTBSIZ/2) ?
Yes
No
Store Updated Cnt. in TTFBB
Beyond Max. Count, MAXBBC (=TTBSIZ/4) ?
Yes
SNDXOF
Send X-OFF to Terminal
No
BIGST1:
Has Big Buf Overflowed ?
Yes, Lose Char.
No
DLSSX1:
BUGCHK - Big Buffer Overflow
Dispatch to Table TTXXVT at offset TTVT25 F.E.Disp. Adr = DLSSX2
Incr. In Ptr to Big Buf (TTBIGI) & Adjust for Wrap Around if Nec.
Store AC1 Contents into TTBBUF & Increment TTBIGC
Return
DLSSX2
Big Buf Has Overflowed
Decrement Big Buf Cnt, TTFBB
= 0 ?
Yes
Return
No
SNDXOF
Send X-OFF to Terminal
Return
Beyond Bad Guy Cutoff
Get Line in AC, C
Get [.DFTOL,,.FEDLS] into AC, B
Clear AC, A (To the Master)
Get .DFTLO (Turn it off Func.) into AC, D
FIXARG
Fix Arguments
DTEQ pDTE1
Output it
Skip Ret ?
No
Return
Yes
Setup When to Restore Line (↑D5000)
DLSRCF Bit Set & Char. Null ?
Yes
Make it a Longer Wait Time (↑D600
No
Get Disp. Adr. to Call TTTOBL
TTQAD:
Compute Time to Do Func. for Line & Save in TTIME
Save Disp. Adr. in TROUT
Incr. TTQCNT
Set Scan Ptr. to this Entry if First Entry
Return

DTE10

DTE Interrupt Handling Comments

TTYINT

(1) The Unique Code argument of form (Ø, count) tells TTYINT the number of characters that have been sent to the -11 in some call to DTEQ that specified TTYINT as its return address.

Count = Ø implies this was a single character (DTSNGL was called) and buffer counts have already been updated.

Count ≠ Ø implies this was multiple characters and the count must be updated.

DINGME

(2) TO1ØIC and TO11IC are wrap-around counters of Indirect Transfers where TO1ØIC is maintained by the -11 and TO11IC is maintained by the -1Ø. If the two wrap-around counters are equal, it means the transfer finished correctly.

(3) If the difference between the wrap-around counters is greater than 1, the -11 has tried to send a direct transfer before the last indirect transfer finished or a doorbell has been lost in a previous transaction.

(4) Receiver sets TOIT equal to 1 in Sender's section of Receiver's communication region after Sender sets @ or increments Q count and rings the doorbell; Receiver clears TOIT upon getting To-Receiver Done (This assures that the Receiver doesn't lose an interrupt).

(5) The function table has dispatches for such features as:

- F.E. telling about the CTY
- String data for the CDR
- Line characters (for DLS)
- -11 Sending error information
- -11 wants or is sending time of day
- Line dialed up, hung up or line buffer empty
- Set line speed or allocation
- Take -11 reload information
- Acknowledge all devices and units
- Take KLINIK data.

CHAPTER 1

EXTENDED ADDRESSING IN TOPS-20

1.1 INTRODUCTION

The information in this section is of a preliminary nature, and should not be construed as a commitment by Digital Equipment Corp. TOPS-20 does NOT support user-mode extended addressing at this time. The Jsys described herein is available but not documented nor supported. The purpose of this information is to inform the student of the eventual availability of this feature to user mode programs, as we have already learned about the monitor's use of this hardware functionality of the Model B KL processor.

Reference:

DECsystem-10/DECSYSTEM-20 Hardware Reference Manual, Vol. 1 Chapter 3, section 4, page 3-23: Tops-20 Paging and Process Tables

Release 4 of TOPS-20 does NOT support user mode extended addressing. Extended addressing is a feature of the current KL (sometimes referred to as "Model B"). With this feature, TOPS-20 itself will use extended addressing to expand its own address space. This is true to different extents depending on whether the system is running version 3A or version 4, and also whether the machine is running a 2050 or 2060 monitor.

The main purpose of extended addressing is to allow expansion of the previous 18 bit virtual address on the KI, KS and Model A KL processors to 23 bits. Actually, the software is capable of supporting a processor with 27 bit virtual addresses. The design as described herein is intended to support a 30 bit virtual address space, making our high-end products support larger address spaces than VAX/VMS. (VAX supports a 30 bit address space as well, but VAX is byte-addressed). Since the code to handle the extended capabilities exists within the monitor (for the KL), the monitor is the first program built to take advantage of the firmware features.

The implementation of extended addressing calls for the "sectioning" of the large virtual address space. Each section contains 512 pages, and corresponds to the traditional 18 bit address spaces we are used to. The number of sections available depends on the software and firmware capabilities. Current support is for the KL, meaning 5 more bits for the section number. Therefore, each process can have 32 address spaces each containing 512 pages. All sections work the same, with the exception of section 0. Section 0 references are always in the traditional 18 bit format. Non-section 0 references can be local or global. With local addressing, instructions executed in that section can reference other locations within the section, using the same old 18 bit addresses and instruction format as always. Global references need to supply a full virtual address, including a section number.

1.1.1 Current Implementation Objectives

The current implementation of user mode extended addressing (available but not supported by Digital) intends to provide the capability for programs to create and use multiple sections for data storage. Also, to allow programs to be run in any section, or in multiple sections, with certain restrictions. These restrictions are that addresses passed to most Jsys's as arguments must specify arguments in the same section as the call itself.

1.1.2 JSYS Interface

Each section has a page map, and the page maps for a given process are pointed to by the process' section map, which exists in the User Process Table starting at location USECTB (540). To find the page map for section n, we simply look at USECTB plus n, where we find a section pointer to the page map. The Jsys's which deal with user mode extended addressing manipulate these maps. The SMAP% Jsys sets the map for a section or a virtually contiguous series of sections. The SMAP% Jsys does with sections what the PMAP% Jsys does with pages. This includes the possibilty of specifying whether the section is private, shared with a file (files don't really have sections, so this specifies any group of 512 contiguous pages in the file), or shared with some other fork, or some other section within the current fork. This indirect relationship is analogous to the indirect mapping available on a page basis with the PMAP% call. This means that when the source mapping is changed, the destination will see the change in the section's map.

The RSMAP% Jsys (currently unavailable) will be used to obtain information about the current mapping of sections. This corresponds to the RMAP% Jsys used to obtain information on pages

within a section. Some other Jsys's will be added or changed to allow certain functions to occur within the extended environment, and usually these have the letter X prefixed to the name of the corresponding Jsys. For example, XSIR and XRIR are used to specify channel and level tables with full 30 bit addresses, so the software interrupt system can be section independent. Note that most Jsys calls which allow the program to supply an argument block address will currently not work if executed in a non-zero section.

1.2 PROGRAMMING IMPLICATIONS AND METHODOLOGY

The basic architecture of the DECsystem-10 family of processors up through the KL10 provided an 18-bit, 256K-word addressing space. This means in particular that:

1. The Program Counter register is 18 bits;

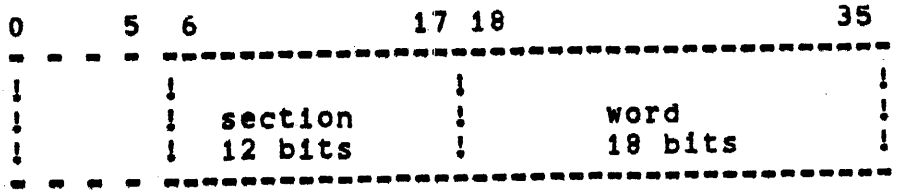
2. Each and every instruction executed computes an 18-bit effective address. The contents of this address may or may not be referenced depending on the actual instruction, but the algorithm for calculating it (including indexing and indirecting rules) is the same for all instructions.

The above is true regardless of the size of the physical core memory available on any particular configuration. Note also that paging (or relocation on earlier processors) will determine if a particular virtual address corresponds to an existing physical word in memory, but the fundamental size of the virtual address space is a constant.

During the design of the KL10 processor, the need for a larger virtual address space was recognized. A design was developed which provides an extended address space to new programs while still allowing existing unmodified programs to execute correctly on the same processor. Although most of the essential data paths were provided in the original KL10 implementation, various design changes caused actual availability of extended addressing operation as described here to be deferred to the "model B" KL10 CPU.

1.2.1 Virtual Address Space

Under the extended addressing design, the virtual address space of the machine is now 30 bits, or 1,073,741,824 words. Although one can think of this as a single homogeneous space, it is generally more useful to consider an address as consisting of two components, the section number, and the word number.



The word (more precisely word-within-section) field consists of 18 bits and thus represents a 256K-word address space similar to the single address space on earlier machines. The section number field is 12 bits and thus provides 4096 separate sections, each of 256K words.

Each section is further divided into pages of 512 words each just as on earlier machines. The paging facilities allow the monitor to independently establish the existence and protection of each section as a unit.

In order to implement this extended 30-bit address space, the PC must be extended to hold a full address. The PC is always considered to consist of a section field and a word field, and the incrementing of the PC never allows a carry from the word field into the section field. That is, if a program allows flow of control to reach the last word of a section and the instruction in that location is not a jump, then the PC will "wrap-around", and the next instruction fetched will be word 0 of the same section. This will in fact be AC 0 as described below. The consequence of this is that flow of control of a program cannot inmplicitly cross section boundaries. In general, it would be a programming error to allow the PC to reach the last location of a section and execute a non-jump instruction or to execute a skipping instruction in either of the last two locations of a section.

1.2.2 Compatibility

In order to allow efficient use of the extended address space, it was necessary to modify the operation of various machine instructions and to change the algorithm for the calculation of

effective addresses. Because these changes have a high probability of causing any existing program to malfunction, the following convention was adopted:

> If a program is executing in section 0, all instructions are executed exactly as they would be on a non-extended machine. If a program is executing in any section other than 0, the extended addressing algorithms are used for effective address calculation and instruction execution.

A program is said to be executing in section 0 when the section field of the PC contains 0. Effective address calculations and instruction executions are performed exactly as on a non-extended machine; hence any existing program will work correctly if run in section 0. Note however, that this also implies that no addresses outside of section 0 can be generated, either for data references or for jumps. That is, a program executing in section 0 cannot leave section 0 except via a monitor call. The only exception to this is the XJRSTF instruction.

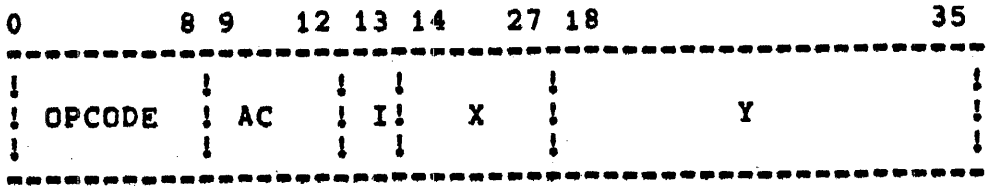
It is easy however to write or generate code which is compatible with both extended and non-extended execution. This was a specific goal of the design, and in general requires only that certain precautions must be taken regarding previously unused fields. Since these precautions must be taken however, one cannot generally assume that any existing program will have observed them and so execute correctly in an extended section.

1.2.3 Effective Address Calculation

Unless explicitly stated otherwise, everything in the following discussion refers to execution of instructions with the PC in a non-0 section; nothing applies to instructions executed from section 0.

1. Instruction format

   The format of a machine instruction is the same as on a non-extended machine. In particular, the effective address computation is dependent on three quantities from the instruction, the Y (address) field, the X (index) field, and the I (indirect) field. These are 18 bits, 4 bits, and 1 bit respectively.



Depending on the format of the index and indirect words, the effective address algorithm will perform either 18-bit or 30-bit address computations. When a 30-bit quantity is indicated, an explicit section number is being specified and the address is called a global address. When an 18-bit quantity is indicated, the section field is defaulted from some other quantity (e.g., the PC), and the address is thus local to the default section and is called a local address.

In the simplest case, consider an instruction which specifies no indexing or indirection. E.g.,

```
3,,400/ MOVE T,1000
```

Here the effective address computation yields a local address 1000, and the section used for the reference is 3, i.e., the section from which the instruction itself was fetched. Precisely stated:

> Whenever an instruction or indirect word specifies a local address, the default section is the section from which the word containing that address was taken.

This means that the default section will change during the course of an effective address calculation which uses indirection. The default section will always be the section from which the last indirect word was fetched.

2. Indexing

The first step in the effective address calculation is to perform indexing if specified by the instruction. The calculation performed depends on the contents of the specified index register:

1. If the left half of the contents of X is negative or 0, the right half of X is added to Y (from the instruction word) to yield an 18-bit local address.

   This is consistent with indexing on a non-extended machine, and means, for example, that the usual AOBJN and stack pointer formats can be used for

tables and stacks which are in the same section as the program.

2. If the left half of the contents of X is positive and non-0, the entire contents of X are added to Y (sign extended) to yield a 30-bit global address.

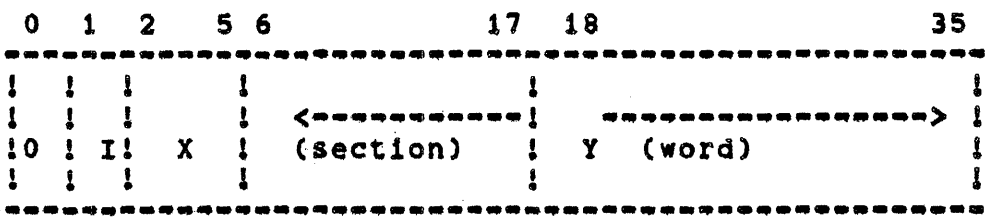
   This means that index registers may be used to hold complete addresses which are referenced via indexed instructions. A Y field of 0 will commonly be used to reference exactly the address contained in X. Small positive or negative offsets (magnitude less than 2**17) may also be specified by the Y field, e.g., for referencing data structure items in other sections.

3. Indirection

If indirection is specified by the instruction, an indirect word is fetched from the address determined by Y and indexing (if any). The indirect word is considered to be "instruction format" if bit 0 is a 1, and "extended format" if bit 0 is a 0.

1. Instruction format indirect word (IFIW): This word contains Y, X, and I fields of the same size and in the same position as instructions, i.e., in bits 13-35. Bit 1 must be 0 (its use is reserved for future hardware); bits 2-12 are not used. The effective address computation continues with the quantities in this word just as for the original instruction. That is, indexing may be specified and may be local or global depending on the left half of the index, and further indirection may be specified. Note that the default section for any local addresses produced from this indirect word will be the section from which the word itself was fetched.

2. Extended Format Indirect Word (EFIW): This word contains Y, X, and I fields also, but in a different format such as to allow a full 30-bit address field.



If indexing is specified in this indirect word, the entire contents of X are added to the 30-bit Y to produce a global address. A local address is never produced by this operation, and the type of operation is not dependent on the contents of X. Hence, either Y or C(X) may be used as an address or an offset within the extended address space just as is done in the 18-bit address space.

If further indirection is specified, the next indirect word is fetched from Y as modified by indexing if any. The next indirect word may be instruction format or extended format, and its interpretation does not depend on the format of the previous indirect word.

4. Some examples

1. Simple instruction reference within the current PC section:

```
3,,400/ MOVE T,1000 ;fetches from 3001000
        JRST 2000 ;jumps to 3002000
```

2. Local tables may be scanned with standard AOBJN loops:

```
        MOVSI X,-SIZ
LP:  CAMN T,TABL(X) ;TABL in current section
        JRST FOUND
        AOBJN X,LP
```

3. Global tables may be scanned with full address and index:

```
        MOVEI X,0
LP:  CAMN T,@[EFIW TABL,X]  ;TABL(X) in ext
fmt
        JRST FOUND
        CAIGE X,SIZ-1
        AOJA X,LP
```

4. Subroutine argument pointer may be passed to subroutine in another section:

word in arglist:

```
IFIW @VAR(X)    ;indexing and indirecting
                ;if specified will be relative
                ;to the section in which this
                ;pointer resides, i.e., the
```

;section of the caller

1.2.4 Immediate Instructions

All effective address computations yield a 30-bit address defaulting the section if necessary, as described above. Immediate instructions use only the low-order 18-bits of this as their operand however, setting the high-order 18 bits to 0. Hence, instructions such as MOVEI, CAI, etc. produce identical results regardless of the section in which they are executed.

Two immediate instructions are implemented which do retain the section field of their effective addresses.

1. XMOVEI (opcode 415) Extended Move Immediate:

   This instruction loads the entire 30-bit effective address into the designated AC, setting bits 0-5 to 0. If no indexing or indirection is specified, the current PC section will appear in the section field of the result. This instruction would replace MOVEI in those cases where an address (rather than a small constant) is being loaded, and the full address is needed.

   Example: calling a subroutine in another section (assuming arglist in same section as caller):

```
XMOVEI AP,ARGLIST

PUSHJ P,@[EFIW SUBR]
```

   The subroutine could reference arguments as:

```
MOVE T,@1(AP)
```

   or could construct argument addresses by:

```
XMOVEI T,@2(AP)
```

   In both cases, the arglist pointer would be found in the caller's section because of the global address in AP. The actual section of the effective address is determined by the caller, and is implicitly the same as the caller if an IFIW is used as the arglist pointer, or is explicitly given if an EFIW is used.

2. XHLLI (opcode 501)

   This instruction replaces the left half of the designated accumulator with the section number of its

effective address. It is convenient where global addresses must be constructed.

1.2.5 AC References

Any reference to a local address in the range 0-17(8) will be made to the hardware ACs. Also, any gobal reference to an address in section 1 in the range 0-17(8) (i.e., 1000000-1000017) will be made to the hardware ACs. Global references to locations 0-17(8) in any section other than section 1 will reference memory. Thus:

1. Simple addresses in the usual AC range will be reference ACs as expected, e.g., MOVE 2,3 will fetch from hardware AC 3 regardless ot the current section.

2. To pass a global pointer to an AC, a section number of 1 must be included.

3. Very large arrays may lie across section boundaries; they will be referenced with global addresses which will always go to memory, never to the hardware ACs.

4. PC references are always considered local references; hence a jump instruction which yields an effective address of 0-17 in any section will cause code to be executed from the ACs.

1.2.6 Special Case Instructions

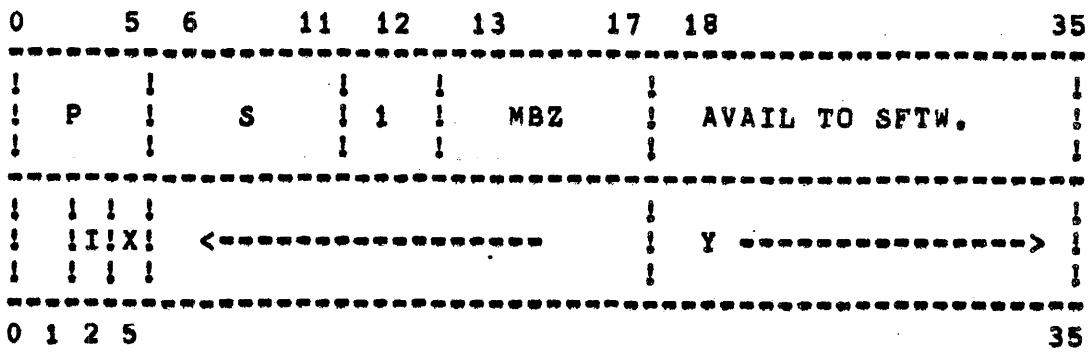
In addition to the differences in effective address calculation, certain instructions are affected in other ways by extended addressing.

1. Instructions which store/load the PC; PUSHJ, POPJ, JSR, JSP. These instructions store a 30-bit PC without flags and with bits 0-5 of the destination word set to 0. POPJ restores the entire 30-bit PC from the stack word. JSA and JRA are not affected by extended addressing and store/load only 18 bits of PC. Hence they are not useful for inter-section calls.

2. Stack instructions (PUSHJ, POPJ, PUSH, POP, ADJSP) use a local or global address for the stack according to the contents of the stack register following the same convention as for indexing. That is, if the left half of the stack pointer is 0 or negative (prior to

incrementing or decrementing), a local address using the right half of the stack pointer is computed and used for the stack reference. If the left half of the stack pointer is positive and non-0, the entire word is taken as a global address. In the latter case, incrementing and decrementing of the stack pointer is done by adding of subtracting 1, not 1000001. Hence, a global stack for routines in many sections may be used in a similar manner to present stacks. Stack overflow and underfow protection would be done by making the pages before and after stack inaccessible since a space count field is not present in a gobal stack pointer.

3. Byte instructions. Two formats of byte pointer are recognized by the byte instructions. The non-extended format is identical to the present standard byte pointer format and is recognized if bit 12 (previously unused) is 0. If bit 12 is 1, a two-word extended byte pointer format is recognized which contains the fields as shown:



Note that the second word is identical to the Extended Format Indirect Word (EFIW). The right half of the first word is specifically reserved to software for byte counts, etc. Incrementing of this format of byte pointer is consistent with non-extended format; the P field is reduced until the end of the word is reached, whereupon the address in the second word in incremented. Incrementing may cause a carry from the word field to the section field of the address; hence extended byte arrays may lie across section boundaries.

4. Other new or modified instructions are LUUOs, BLT, XBLT, XCT, XJRSTF, XJEN, XPCW, XSFM. Some of these are valid only in exec mode. Consult the System Reference Manual or chapter 2.2 of the KL10 Functional Specifications for details.

1.2.7 Compatible Programming

It is possible to generate code which works correctly in both extended and non-extended environments. Such code may even utilize inter-section references when running in an extended environment; it is not limited to local addressing.

The basic rule is to observe the extended addressing rules in the construction or computation of:

1. Index words: be sure the left half is cleared or contains a negative quantity (e.g., a count) when setting up and using an index register. This will cause a local reference in the extended environment which will produce the same result as on the non-extended machine.

2. Indirect words: always set bit 0 of indirect words used in argument lists, etc. so as to produce local addresses.

3. Stack pointers: the most common present format (negative count in left half) works consistently under extended addressing; modifying a program to use an extended stack should require no change to stack instructions except if the code expects to find the processor flags in the stacked PC words.

When generating addresses to be passed to subroutines or used by other code, XMOVEI and XHLLI instructions may be used. In the non-extended environment, these opcodes are SETMI (equivalent to MOVEI) and HLLI respectively, which always load 0 into the left half.

1.2.8 Program Architecture And Facilities

Ultimately, the extended addressing hardware in conjunction with the monitor should provide some of the power of general segmentation and some other useful conventions and facilities. The following are some of the ideas which have been advanced.

1. The fact that instructions are generally executed relative to the current sections suggests that subroutine libraries and facilities packages can be written which can be dynamically loaded into any available section and used by many programs. An important point to observe in connection with this and most of the following conventions is that programs must not be compiled with fixed section numbers built into

the code. Programs should be built so as to be loadable into any section and to request additional section allocations from the monitor as necessary. This is the only reasonable way to ensure that conflicting use of particular sections is avoided.

2. An entire section can be mapped by the monitor as quickly as a single page. Hence, an entire file (up to 256K) can be mapped into a section, and the data therein manipulated with ordinary instructions.

3. Programs can greatly reduce memory management logic by merely assuming very large sizes for all data bases. It is not however, very efficient to use many sections, each with only a small amount of data. A reasonable middle ground should be chosen.

CHAPTER 2

BIAS CONTROL

.1 BIAS CONTROL

he bias control knob provides for administrative control over ome scheduling algorithms. The knob allows a system dministrator to bias the system according to the needs of the nstallation, to establish degrees of "fairness" among n[illegible]active and compute-bound users. This logical knob has etings from 1 to 20. The lower the setting of the knob, the ore interactive jobs are favored. The higher the setting of the nob, the more compute-bound jobs are favored.

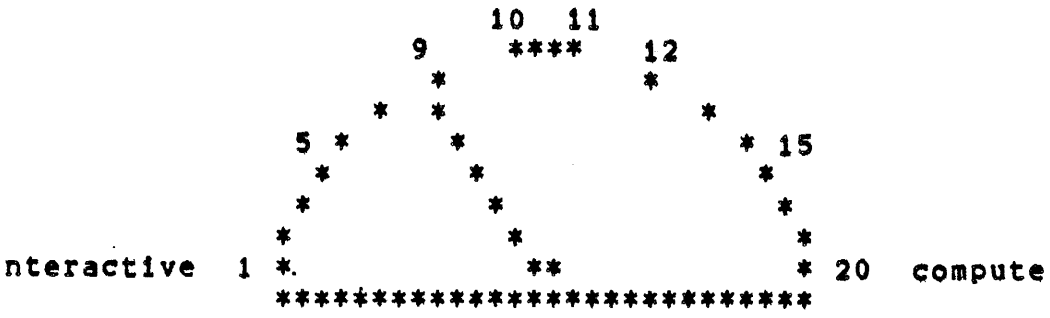


Figure of the bias control knob

Figure

2.1.1 USER INFORMATION

The bias control knob can be set through the use of a command to OPR, the operator interface. The command has the form:

```
OPR>SET SCHEDULER BIAS-CONTROL (to) n
```

where n is a decimal number from 1 to 20.

The bias knob can also be set through the use of a CONFIG command. The following command can be placed in 4-CONFIG.CMD:

```
BIAS n
```

where n is a decimal number from 1 to 20.

2.1.2 PROGRAMMING INFORMATION

Bias control can also be changed under program control through the use of the new SKED% JSYS. The .SKRBC function of this JSYS reads the bias control knob setting; the .SAKNB function sets the bias control knob setting.

2.1.3 IMPLEMENTATION

The value of the bias control knob is a displacement into a table of twenty entries. Each table entry represents a set of binary switches controlling a scheduler algorithm. Any binary value that causes an "on" setting (1) causes a bias in favor of interactive jobs while any binary switch that is "off" (0) is biased in favor of compute-bound jobs.

2.1.4 SWITCH SPECIFICS

The following is a graph of the current table (which starts at location SKFLGV) being used for the bias control knob.

Graph of Bias Settings

The following describes the current switches and their actions:

SK%CYT Bit 18, default setting is on.

Used for adjusting the length of the cycle times for the short and long cycle clocks. If 1, standard values for the cycle times are used (20 ms and 100 ms). If 0, the cycle times used are four times the standard cycle times (80 ms and 400 ms). This switch favors compute-bound jobs if off, since longer cycle times should reduce overhead, interactive jobs may not be allowed in during the longer cycles (they wait longer to be chosen), and a compute-bound job can hold the processor longer during longer cycles (which reduces context switching).

SK%IOC Bit 19, default setting is off; does not change.

Used to decide if a fork's quantum time should receive a charge for I/O. This is not really a bias between interactive and compute-bound processes in the strict sense. Jobs that do a large amount of disk file I/O are affected. Off indicates the charge is to be made; on eliminates this charge.

SK%HT1 Bit 20, default setting is on.

The SK%HT1 switch controls the number of processes with balance set hold time. If the switch is on, there is no more than one other process with balance set hold time, and less than half of memory is allocated, the process under consideration is given hold time. If the switch is off or the other conditions do not hold, the process is given hold time. The switch favors interactive processes if on by controlling the amount of hold time for a process. This keeps the balance set membership more dynamic which helps interactive processes back into the balance set.

SK%HT2 Bit 21,default setting is off.

If on, a process that enters the balance set after a process of higher priority has been skipped receives no balance set hold time. This favors interactive processes by not allowing processes of lower priority to run for long periods when processes of higher priority cannot fit into the balance set.

SK%HQR Bit 22, default setting is off.

Used to control "HQ disaster avoidance." If, during system

operation, the load average reaches a certain level (a load average between 4 and 6) low queue jobs (which tend to be compute-bound jobs) are no longer allowed into the balance set. If the load average goes higher (to between 6 and 9) low queue jobs are forced out of the balance set. If this switch is off, the code that causes the above to happen is skipped. Having this switch off favors compute-bound jobs since they can still get machine time when the load average is high.

)its 23 and 24 are currently not used.

)K%RSQ Bit 25, default setting is on.

The SK%RSQ switch determines whether an overhead cycle is initiated when a process with higher priority than the running process becomes runnable as the result of disk I/O completion. If the switch is on and the conditions are met, the current process is dismissed and the higher priority one selected to run during the overhead cycle. This favors interactive processes since they can be run immediately after unblocking. If the switch is off, the dismissal occurs in the next normal overhead cycle.

K%RQ1 Bit 26, default setting is on.

Used for controlling the use of the interactive queues. If this switch is off when setting the queue level for an unblocking process, the interactive queues are not used. This favors compute-bound jobs if this switch is off since the interactive jobs do not get the increased priority and the extra quantum of the interactive queues.

K%TTP Bit 27, default setting is on.

Used to control the moving of processes to the interactive queues for TTY wait. If on, the added boost of moving the process to Q1 is given, which favors interactive processes with an increased priority.

K%WCF Bit 28, default setting is off.

Used to decide whether to decrease the wait credit of a previously blocked process based on the load average in determining the new queue. When off, the wait time is divided by the short-term load average to maintain fairness (that is, decrease the advantage gained for a wait if the load average is high). Other processes on the system have not fared that well while the process was blocked. When on,

interactive jobs are favored since they are not penalized for a high load average.

Bits 29 to 35 are not used.

It is interesting to see how values change from one setting of the knob to the next. Some changes of the knob may cause no change in the scheduler. For example, changing the knob from 1 to 4 currently causes no change in the scheduler. Changing from 16 to 17, however, causes three specific switches to be changed. Also, notice that the design of the table lends itself to changes in the future.

CHAPTER 3

CLASS SCHEDULING

.1 CLASS SCHEDULING

lass scheduling enables the administrator of a system to llocate the system according to classes of users.

n a particular system there may be classes of users that need arge amounts of resources at a given time. There may also be a l[illegible] of users that should not be given a large amount of esources. Class scheduling permits the administrator of a ystem to allocate system resources (specifically, CPU resources, hich implies other resources such as I/O and memory) on the asis of class.

NOTE

Class scheduling is not required. If it is not enabled, its parameters are ignored.

onsider the following example:

DECSYSTEM-20 is used for computer courses at a university. his system has three definite classes of users:

1. Students that are taking the computer science courses
2. Faculty members that give the courses
3. Administrative users that do some processing during free time on the machine

e[illegible]y wish to divide up the machine as shown below:

Figure 2-3

Using class scheduling, the administrator would define three classes on the system. Each class would be defined to receive the indicated percentage and each user would be assigned to a class.

An important point should be realized concerning the divisions made in the example above. The machine is divided using class scheduling. If there is one student, three faculty members, and five administrators on the machine at a given time, the system would try to give the one student 75% of the machine, each of the faculty would receive 5% of the machine, and each of the administrators would receive only 2% of the machine. Or, another situation might have 25 students and only one member of the faculty and administrative groups at a given time. In the latter case, even though the students were allocated a larger percentage of the machine, each student would receive only 3% while the faculty user and the administrative user would receive 15% and 10%, respectively. It is therefore important to consider the size of a class of users when defining the percentage of the machine for that class.

As in the case where one student was logged on the system, it is possible that a particular class may sometimes not be able to use all of its allocated percentage. The amount of unused CPU resource in such a case is known as **windfall**. Another possible case that creates windfall is when all of the machine is not

allocated with class percentages. In either case, windfall can be either withheld or allocated. If withheld, the system runs idle rather than give a class more than its share. If windfall is allocated, the system distributes the excess CPU resource proportionately among the active processes.

NOTE

Withholding of windfall is not recommended. Its use should be restricted for at least two reasons:

1. If withholding windfall is enabled before all of the class percentages are defined correctly, a condition could exist where no one could do any useful work.

2. It wastes system resources.

3.1.1 USER CHARACTERISTICS

The first step in implementing class scheduling is to determine whether it is really needed. Class scheduling should not be used only because it is there. There should be a definite need for dividing up the machine. It is important to remember that while class scheduling may improve throughput for a particular class, it probably decreases throughput for the system as a whole. There is overhead involved with class scheduling and for this reason its use is not recommended with small machines (2020s and 2050s with less than 256K). It is not that class scheduling does not work with these machines -- the expense in overhead of using class scheduling on small machines probably outweighs the possible benefits.

3.1.2 CLASS ASSIGNMENT

Once a need is established to use the class scheduler, the class members must be identified and assigned. Identifying the members requires grouping them according to characteristics. Assigning users to a class on the system requires using one of two

available methods.

To identify the classes requires some considerations of the mechanisms that the system uses. The system only supports eight classes, each specified on the system with a number from 0 to 7.

The characteristics of a class can be determined in a number of ways. In the previous example, the groupings were based on the obvious distinctions between students, faculty, and administrators. In the case of a timesharing bureau, the divisions could be made based on specific customers, with the end result being the sale of specified portions of the machine.

After the classes are defined, they need to be assigned on the system. There are two methods for accomplishing this.

The recommended method of assigning class members is to use the accounting system. Using this method, class membership becomes an attribute of a particular account. As the user logs in and is assigned an account, the user is also placed in the class that is identified as an attribute of that account. If the user is able to change accounts with the SET ACCOUNT command, the action of changing accounts may also change the class of the user.

NOTE

Using the accounting method requires that account validation be enabled.

An alternate method of assigning class membership is to use the access control job. By using this method, the user's class can be determined dynamically by a user-written policy program. This enables class assignments to be based on the current state of the system.

If the class is not assigned by one of these methods, the system assigns a user to a default class as the user logs in.

3.1.3 CLASS PERCENTAGES

The next step in implementing class scheduling is to determine the class percentages. The number of members in the class needs to be considered. As seen previously, the more members of a class with active jobs on the system, the less the percentage each member of the class receives. Factors unique to the installation, such as time as day, need to be considered. In the

iniversity example, students may receive a very large percentage of the machine during the day, when they are doing a majority of their work. At night, however, the administration may need a large percentage of the machine for administrative work.

Finally, the mechanics of the implementation need to be considered. In assigning percentages to classes, the total percentage can be less than 100 % of the machine but not more than 100 %.

The percentages for classes can be defined in one of three ways: by using CONFIG commands, by OPR commands or by a policy program.

To define the class percentages using CONFIG commands, a command of the following form must be placed in 4-CONFIG.CMD:

```
CREATE n frac
```

where: n is the class number (0-7).

frac is the class percentage expressed as a fraction of 1 (0.00-.99).

There should be as many CREATE commands as there are classes on the system.

To define the class percentages using OPR commands, a command of the following form must be used:

```
OPR>SET SCHEDULER CLASS (number) n (to percent) per
```

where: n is the class number (0-7).

per is the class percentage (0-99) (note that per is actual percentage in this case).

This command enables the operator to change class percentages during normal system operations.

The class percentages can also be changed using a policy program. This enables the percentages to be changed dynamically as based on administrative policy.

3.1.4 Batch Class

For added control of batch jobs, the administrator can require that batch jobs be placed in a specific class on the system. Using this option, batch jobs are placed in the specified batch class. This overrides the class assignment that would be

received if the user were to log in normally. This option is indicated either through a CONFIG command, an OPR command, or through the use of a policy program.

NOTE

This option requires class scheduling to be implemented using the accounting method. If the classes are assigned with the policy program, the batch class commands have no effect.

The CONFIG command has the form:

BATCH-CLASS n

where n is the class to be used for batch jobs (0-7).

The OPR command has the form:

OPR>SET SCHEDULER BATCH-CLASS n

where n is the class to be used for batch jobs (0-7).

3.1.5 TURNING CLASS SCHEDULING ON/OFF

Class scheduling can be turned on and off using either CONFIG commands, OPR commands, or a policy program. The CONFIG and OPR commands are described below.

Starting class scheduling with a CONFIG command requires two additional pieces of information. First, the disposition of the windfall must be determined by the requirements of the system. For example, if the requirements are such that no user is to receive more than his allocated share, windfall should be withheld.

NOTE

If windfall is withheld and there are errors in the CONFIG file, a situation could occur where no one has processor time allocated to

them.

Second, the method for determining class assignments, either by accounting or by policy program, should be identified. The CONFIG command has the form:

```
ENABLE CLASS-SCHEDULING ACCOUNTS          WITHHELD
                        POLICY-PROGRAM    ALLOCATED
```

NOTE

This command must be the last class scheduling command in the CONFIG file. That is, it must follow all of the CREATE and BATCH commands. If it does not, all CREATE or BATCH commands that follow the ENABLE command will have no effect. If windfall is withheld and all of the CREATE commands follow the ENABLE command, no class will have percentages and no one will be able to do any useful work.

The OPR command for starting the class scheduler requires the same information that the CONFIG command requires. Both the method for assigning classes and the disposition of the windfall should be indicated. The OPR command has the form:

```
OPR>ENABLE CLASS-SCHEDULER
        /CLASS-ASSIGNMENTS: ACCOUNTS
                            POLICY-PROGRAM
        /WINDFALL: ALLOCATED
                   WITHHELD
```

OPR has an additional command for disabling the class scheduler:

```
OPR>DISABLE CLASS-SCHEDULER
```

3.1.6 IMPLEMENTATION

Class scheduling is used to influence the priority of a process. A process that is ahead of its percentage will tend to have its priority reduced and a process that is behind its percentage will tend to have its priority increased. Therefore, a process'

relation to its "target" percentage is a new metric that becomes part of the heuristic that dictates the transition rules among the queues. The following is a description of the new metrics and how they are employed.

A job's "utilization" is periodically computed by the process controller as a decaying average of the CPU time used by the processes of the job. The computation performed is

$$U(I+1)=U(I)*e^{(-t/C)}+F(1-e^{(-t/C)})$$

where: U(I) is the current process utilization.

F is the fraction of the CPU used by the process in the recent interval "t".

t is the time since the last computation of U.

C is the "decay" interval. After time C, the utilization decays by e. This number represents the amount of "history" that the monitor uses in determining a process' behavior. The number is chosen to insure as uniform a behavior as possible.

The utilization function for each of the classes is the sum of the utilizations of the jobs in the class. The class utilization, CU, is used to compute the class "distance" (CD) from its ideal (target) utilization as follows:

$$CD = \frac{CP - CU}{CP}$$

where: CP is the class' ideal utilization. CP is the class percentage assigned when the class is defined.

Each job also has a job distance (JD) calculated as follows:

$$JD = \frac{\frac{CP}{N} - JU}{\frac{CP}{N}}$$

where: JU is the job utilization.

N is the "population" of the class of which this job is a member. N is currently the number of logged-in jobs belonging to the class.

The class distance and the job distance serve as a two-key sort for the GOLST for assigning priorities. CD is the primary key

and JD is the secondary key. In addition to this sorting, "specials" in the assigning of priorities ensure that certain critical and interactive processes are scheduled promptly (for example, NOSKED, CRSKED, high priority queue processes).

CHAPTER 4

EXECUTE-ONLY

.1 EXECUTE-ONLY

xecute-only provides the capability of protecting files from eing changed or examined, while allowing the same files to be xecuted.

.1.1 DEEINITION

n execute-only file is one that cannot be copied or read in a ormal manner, but can be run as a program. In order to provide his in TOPS-20, the following constraints must be placed on a ile in order for it to be considered an execute-only file:

1. The file must be protected with EXECUTE access allowed but without READ access allowed.

2. The file cannot be read or written using any of the file-oriented monitor calls (i.e., SIN, SOUT, BIN, PMAP in referencing a file, etc.).

3. The file can be mapped into a process (via GET), but only in its entirety and only into a virgin process.

n execute-only process is a process that is created by erforming a GET on an execute-only file. To insure security for he execute-only file and process, the execute-only process must lso be restricted:

1. No other process can read from an execute-only process' address space or accumulators.

2. No other process can change any part of an execute-only process' context in such a way as to cause the execute-only process to reveal any part of its address space unintentionally.

3. An execute-only process must start at either its START or REENTER entry point (ENTRY VECTOR). Allowing a process to start elsewhere could cause it to reveal itself.

4. An inferior fork that is created from an execute-only process with the same map should be execute-only also.

Some related definitions are

Virgin process -- A process that has just been created (using CFORK) with none of its pages having been mapped and no operations having yet changed its context.

Context of a process -- The context of a process includes its address space, PC, ACs, interrupt system, traps, etc.

4.1.2 USER CHARACTERISTICS

An execute-only file is created by modifying the protection attribute of the file. This can be done by setting the protection field for the the desired class of users (owner, group or world) to FP%EX+FP%DIR, or 12 (octal). For example, to make a file execute-only for everyone except the owner of the file, set the protection to 771212. This indicates that the file can only be seen with the directory command and executed; the file cannot be read or written.

NOTE

Any file can be made execute-only. This can include data files, for example, but the results of such an exercise prevents any use of the file. It does not indicate that the file can only be read by an executing process.

An execute-only process is created by performing a GET on an execute-only file -- using a virgin process' address space. This prevents merging the execute-only file into an address space

'hich may already contain code that could reveal the address pace of the execute-only file.

.1.3 WHAT CAN AND CANNOT BE EXECUTE-ONLY

ue to the characteristics of execute-only, there are some estrictions as to where it can and cannot be used.

ost programs (.EXE files) can be protected execute-only. For xample, saved COBOL, FORTRAN, and BASIC programs can be xecute-only. Some utilities that can be execute-only include: UMPER, TV, EDIT, etc.

NOTE

Although COBOL and FORTRAN programs can be made execute-only, care must be exercised. The object time system (OTS) must be included in the EXE file. Otherwise, users can make their own "OTS" and define SYS: to cause that OTS to be used and to cause the execute-only process to reveal itself.

here are some programs that cannot be execute-only. Some of the ajor ones are listed here along with an explanation of why they annot be execute-only:

1. Any object time system such as FOROTS -- These are merged into an address space. This violates the restriction that confines the reading of an execute-only file into a virgin address space. Note: This does not imply that an execute-only process cannot bring in an OTS, but only that the OTS cannot be execute-only.

2. The TOPS-10 Compatablilty Package (PA1050) -- This cannot be execute-only for the same reason that an OTS cannot.

3. Any program that is brought in with the TOPS-10 UUO's RUN and GETSEG -- These UUOs require the program to be mapped into nonvirgin address space.

4. Any program that needs to be started at any location except its entry vector (START or REENTER address) -- To start elsewhere may cause an execute-only process to reveal itself.

5. Any program that uses TOPS-10-style "CCL starts" (starting at the start address plus one) -- Again, the program cannot start at a location other than that specified in the entry vector.

6. A compiler or linker invoked through the COMPILE/LOAD/EXECUTE/DEBUG commands -- These use the CCL start.

4.1.4 SOME OTHER RESTRICTIONS ON THE USER

Some other restrictions on using an execute-only file or process are extensions of the previous definitions.

EXAMINE, DEPOSIT, MERGE, DDT, SET ENTRY-VECTOR, SET PAGE-ACCESS, SET ADDRESS-BREAK commands do not work for execute-only processes. These commands would either cause the process to reveal itself or allow a modification to the process that could eventually cause the process to reveal itself.

The START command cannot be used with a start address argument for an execute-only process. The process must be started using its defined entry points (entry vector).

The INFORMATION (ABOUT) VERSION will only return the name of the program. The version information is part of the process address space that is not to be revealed.

It is important to note that the use of execute-only protection does not guarantee the security of the file. It is the programmer's responsibility to insure that the program does not reveal itself through programming mistakes that would, for example, allow the program to map itself to unsecured forks or use noncertified libraries.

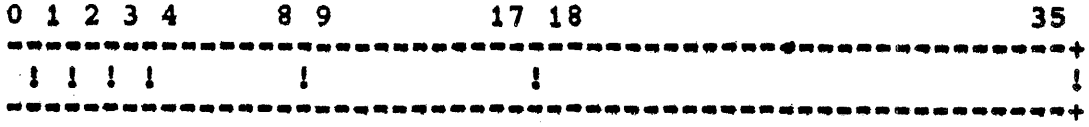
WHEEL privileges can affect the use of execute-only files. An enabled WHEEL cannot create an execute-only process. It is when the process is initialized that it becomes execute-only. If the user has WHEEL privileges and they are enabled, the execute-only code in the initialization is bypassed since the enabled WHEEL has read access as well as execute access. For a user with WHEEL privileges enabled to create an execute-only process, privileges must first be disabled, the GET performed, and the privileges reenabled if needed.

.1.5 IMPLEMENTATION

'he implementation of execute-only involves some data in the JSB, 'hecks in a number of JSYSs to see if the user is allowed to :xamine a particular process, and steps in a few JSYSs to set up :xecute-only processes.

.1.5.1 DATA STRUCTURE -

n the JSB is a SYSFK area that has an entry for each process of he job. Three flags contain information about the process eeded for execute-only. An entry appears as follows:



its	Pointer	Contents
		If set, indicates entry not in use, fork has been deleted
	SFEXO	If set, fork is execute-only
	SFNVG	If set, fork is not virgin
	SFGXO	If set, indicates fork can PMAP into execute-only forks because it is doing an execute-only GET
-8		Not used
-17	FKHCNT	Count of handles on a given fork
8-35		System fork number

.1.5.2 ROUTINES -

ost of the use of the flags in the SYSFK area described above akes place in general routines. These routines are described elow.

HKNXS - CHeck if Not eXecute-only or Self -- Routine CHKNXS is a eneral test routine to determine if the process specified is ither SELF or not an execute-only process. Otherwise, an llegal instruction trap occurs, returning the error FRKHX8 -- Illegal to manipulate an execute-only process". If the pe fied process is not execute-only, it will be declared onvirgin by clearing the virgin process bit SFNVG.

ETEXO - SET process EXecute-Only -- This routine in FORK will

cause the selected process to become execute-only. If the process is not virgin, this does not succeed.

SETGXO/CLRGXO - SET/CLeaR GET eXecute-Only status -- These routines set and clear the execute-only GET bit (SFGXO) in the current process. These routines are called by GET to allow mapping into an execute-only process.

CLRVGN - CLeaR VirGin flag -- This routine sets SFNVG to indicate the process is not virgin.

SREADF - Set READF, read access and restricted-access -- This routine will set the read access bit (READF) and the restricted-access bit (FRKF) in the status word for the selected JFN. Also, the previous state of the FRKF flag is returned. This routine is required to allow GET to use BIN, SIN, PMAP, etc. to a file opened for execute-only access.

CREADF - Clear READF -- This routine undoes what SREADF did.

4.1.5.3 Major JSYSs -

Certain JSYSs have a major role in implementing execute-only. These JSYSs are presented below.

CFORK

CFORK creates a virgin process if CR%ST (start process) and CR%MAP (give process same map as creating process) are not set. Note that loading parameters in the ACs using CR%ACS does not make this a nonvirgin process. Setting CR%ST and either CR%ACS or CR%MAP allows the process to execute code and, therefore, makes the process nonvirgin. Setting CR%ST without CR%MAP or CR%ACS seems rather useless.

CFORK creates an execute-only process if bit CR%MAP is set and the creating process is an execute-only process. This is the only way (besides GET) to create an execute-only process.

SFORK

SFORK has a switch to indicate that a process is to be continued, ignoring any PC change that may be specified in the right half of AC2. This is to insure that an execute-only process is not halted and continued from another location that could cause the process to reveal itself.

PMAP

It is illegal to specify an execute-only process as either the source or the destination in a PMAP call unless that execute-only process is executing the PMAP. If the executing process is doing a GET of an execute-only file (that is, if SFGXO is set), the process may map pages into any execute-only process.

GET

A GET call that addresses an execute-only process is illegal unless the calling process is the same execute-only process (SELF).

If the JFN specified in the GET call refers to a file for which the user only has execute access, the process specified must be a virgin process. GET must overcome two protection features to GET an execute-only file:

1. Reading the file without READ access (and not allowing others access at the same time).
2. Mapping pages from the file into an execute-only process.

A GET must then perform the following steps:

1. Perform OPENF on the file for READ and EXECUTE (as in the past).
2. If the OPENF succeeds, proceed as usual since the file is not execute-only.
3. If the OPENF for READ and EXECUTE fails and either the specified process is not virgin or GT%ADR (address limits) was specified, return the error from the OPENF.
4. Perform OPENF for only EXECUTE access.
5. If Step 4 fails, return the error from the OPENF.
6. Lock the process structure.
7. Set the execute-only bit (SFEXO) in the destination process by calling SETEXO. If the destination process is not virgin, the execute-only bit will not be set and:
   a. Unlock the process structure.
   b. Mark the process as not virgin.
   c. Return to Step 1. Since the process is not virgin, Step 3 will fail.

8. Unlock the process structure.

9. Remember that this will be an execute-only GET.

10. Disable interrupts within this process (NOINT). This is to protect the use of READ access to the file and the use of the execute-only GET bit (SFGXO).

11. Set READ access and restricted access in the JFN status for the selected JFN by calling SREADF.

12. Set the execute-only GET bit (SFGXO) in the executing process by calling SETGXO.

13. Perform the normal operations required to GET the file into the process.

14. If any errors occur, clean up but return error.

15. If this was an execute-only GET, clear the READ access and the restricted access in the JFN status by calling CREADF, clear the execute-only GET bit (SFGXO) in the executing process by calling CLRGXO, and enable process interrupts (OKINT).

16. Close the file (if possible) using CLOSF. Note that if pages are mapped from the file, it will not be closed, but will be left open with only EXECUTE access.

4.1.5.4 Minor JSYSs -

There are a number of JSYSs that either change a process' context or allow access to a process' address space. Since many of these functions are disallowed if the process involved is execute-only, calls to CHKNXS are contained in these JSYSs. Routines that use CHKNXS either directly or indirectly and return the error "FRKHX8: Illegal to manipulate an execute-only process" include:

ADBRK	AIC	DIC	DIR	EIR	IIC
RFACS	SAVE	SCVEC	SDVEC	SETER	SEVEC
SFACS	SFRKV	SIR	SIRCM	SPACS	SSAVE
STIW	TFORK	UTFRK			

OPENF returns fail if a attempt is made to open an execute-only file with illegal access specified.

CRJOB creates a virgin process as the top-level process. Thus, an execute-only program can be run as the top-level fork.

.1.6 RESTRICTIONS AND LIMITATIONS

'his section summarizes some restrictions and limitations of the 'urrent implementation of execute-only:

1. There is no hardware concealment of process pages. The KL10 hardware has the capability to conceal pages within a process from other parts of the same process. This feature, if used, would provide an additional feature that would allow non-execute-only programs to load execute-only programs (such as an execute-only object time system) into their address space.

2. There is no protection from a process revealing itself through its own carelessness.

3. Compilers and object time systems cannot be execute-only.

4. There is no page-by-page protection. Only an entire file can be execute-only.

5. Only disk files can be execute-only.

6. PA1050 must exist on physical disk for execute-only programs. A user cannot use his own version of PA1050 with execute-only programs.

7. An enabled WHEEL cannot create an execute-only process.

8. The version of an execute-only program cannot be read, since that information is stored only in the program's address space.

CHAPTER 5

MONITOR ADDRESS SPACE

.1 MONITOR ADDRESS SPACE

uring the development of Release 4, the TOPS-20 monitor ran out f address space. Extended addressing could not be used to solve he problem. Even though 2060 and ARPA monitors require extended ddressing, there are DECSYSTEM-20s that do not support extended d ssing (2020s and older 2040s and 2050s).

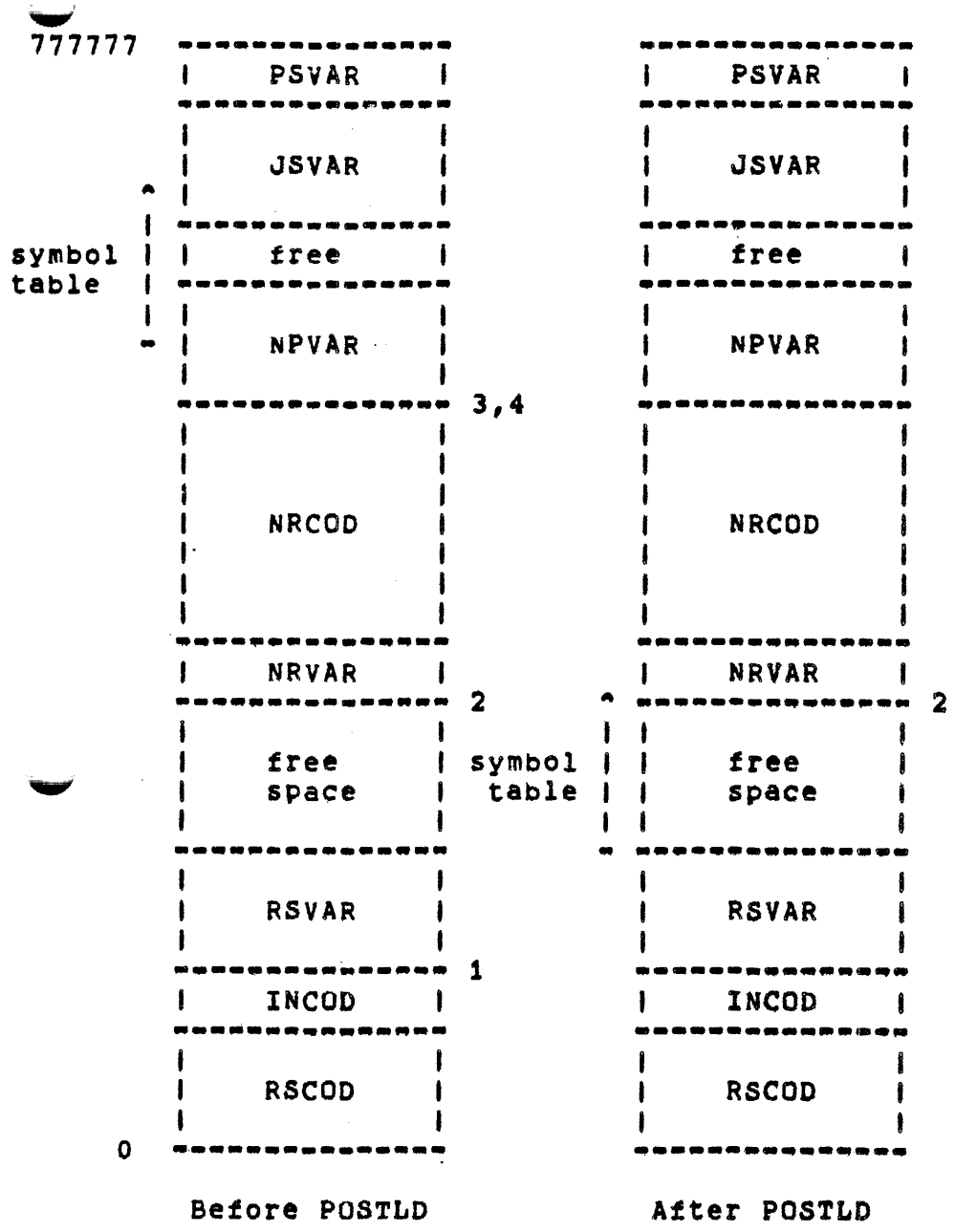
.1.1 ADDRESS SPACE REARRANGEMENT

o solve the shortage of address space, the symbol table was oved out of the monitor address space and placed in an alternate ddress space. The symbol table was selected since its removal rees a large amount of space and it is used only infrequently nder normal conditions. Making the symbol table harder to ccess does not degrade normal operations. In addition to the emoval of the symbol table, other psects of the monitor have een reorganized.

n order to make the changes easier to understand, the following hort description of the address space layout for Release 3A is resented.

5.1.1.1 ADDRESS SPACE IN RELEASE 3A -

The address space in Release 3A of TOPS-20 had two layouts (see Figure 1) because of two different phases that occured in the loading of the monitor. The difference between the two layouts was the location of the symbol table. The symbol table was moved by POSTLD from its original location to immediately after RSVAR.



Notes:

1. POSTCD 2. PPVAR 3. PPVAR 4. BGPTR

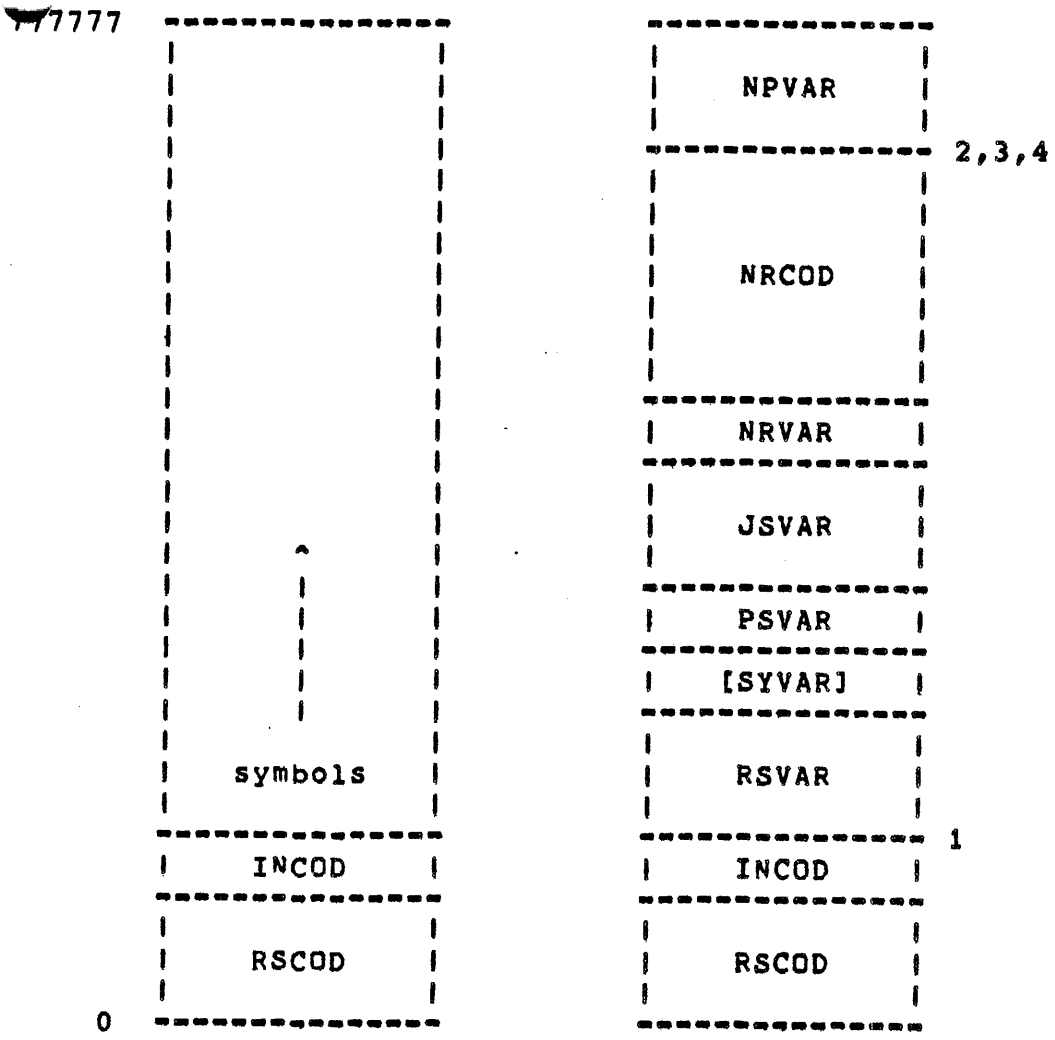
Figure 1 Release 3A Address Space

5.1.1.2 ADDRESS SPACE IN RELEASE 4 -

In order for the symbols to be available from monitor startup time, they must reside in the MONITR.EXE file and be read in by BOOT. Any scheme to put them into a different file would mean that they would not be available until after all of the monitor's disk mounting and other startup code had been executed. This would mean that EDDT would have no symbols for debugging. To put the symbols into the EXE file without overwriting code or data, an all-zero area big enough to hold them must be used.

Based on this, the monitor's address space is laid out as indicated in Figure 2. BOOT reads in RSCOD (resident monitor) and INCOD (which contains initialization code and EDDT) and the symbols. The symbols are initially read into the area wnere the psects PPVAR and RSVAR will ultimately reside. STG then moves the symbol table up to immediately following the SYVAR psect in physical memory. STG does this to free up the RSVAR and PPVAR psects so that MMAP and the CSTs can be set up to turn on paging.

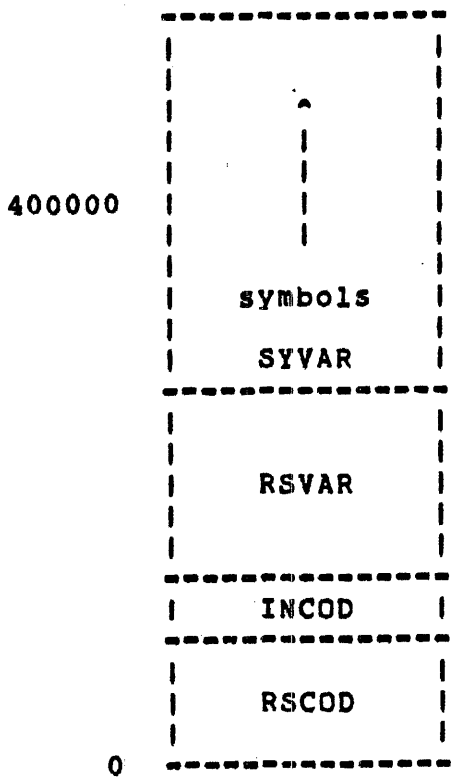
Later, PGRINI sets up SUMMAP to point to an alternate address space containing EDDT, the symbols, and parts of the monitor. EDDT and the symbols live at the same addresses in this virtual address space as they do in physical memory. (See Figure 3.) The symbols do not appear in the monitor's main address space at all.



Notes:

1. PPVAR 2. BGSTR 3. BGPTR 4. POSTCD

Figure 2 Release 4 Address Space



Notes:

1. PPVAR 2. BGSTR 3. BGPTR 4. POSTCD

Figure 3 Alternate Address Space

5.1.2 MODULE CHANGES

With the changes to implement the alternate address space (which is also often referred to as hidden symbol processing) some modules now function differently.

BOOT

BOOT is read in by the front end at a predetermined physical address (currently on top of the RSCOD psect) and started. The first thing BOOT does is find the highest 20 pages in section 0 of physical memory, and move itself there. It then sets up a mapping that is straight physical to virtual, except that it always maps itself at the end of virtual memory (pages 760000 and beyond).

:DDT

/ith Release 4 EDDT now has two conditions. In the first :ondition, when the monitor is loaded by BOOT but not started, :he symbols are in the monitor's virtual address space and EDDT :ccesses them much as it did in past releases. In the second :ondition, the monitor has been started and EDDT must reference :he symbols using the alternate mapping.

'he alternate mapping is accomplished using an alternate page able. To access the symbols, EDDT changes the page table ointers at MSECTB and MSECTB+1 (for sections 0 and 1). The old age table pointers are placed in OSECTB and OSECTB+1. New page able pointers are retrieved from SSECTB and SSECTB+1.

DDT

DDT accesses symbols when it needs them by mapping needed pages f the alternate address space. This is accomplished through the se of a new monitor subroutine called .IMOPR (Internal Monitor PeRation).

NOOP JSYS

he SNOOP JSYS accesses symbols in the alternate address space by sing .IMOPR in the same fashion as MDDT.

.1.3 LIST OE MONITOR PSECTS

he following is a list of all of the TOPS-20 monitor's psects, long with a short description of each. Note that each psect hose name ends in VAR is all-zero, and is not filled in until he monitor starts running.

1. Page 0 (and 1 on 2020s) -- Not really a psect; these pages are loaded with LOC statements and are full of miscellaneous communication areas, flags, etc.

2. RSCOD (Resident code) -- This psect contains the code and data that can never be swapped out. This psect is hard-wired into the monitor as the first one, and the location MONCOR contains the last page number in it.

3. INCOD (Initialization code) -- This psect contains some routines used only during monitor initialization (including the 143$G dialog). It also contains Exec DDT. This psect is locked when the monitor is started, but gets unlocked at GETSWM unless EDDT is needed.

4. PPVAR (Per-processor variables) -- This psect contains nothing. It is used to reserve a few slots in MMAP for use in setting up temporary mappings to memory pages. APRSRV uses these map slots to recover from parity errors, for example.

5. RSVAR (Resident variables) -- This psect contains the monitor's variables that must always remain resident, including the EPT, MMAP, and the CSTs.

6. SYVAR (Symbol variables) -- This psect contains everything that should appear only in EDDT's alternate address space (for example, symbols), and not in the monitor's normal address space. The symbols are appended to this psect early in the monitor's initialization, and then, both the symbols and the psect are "hidden." This psect is of zero size if the HIDSYM conditional is not set.

7. PSVAR (PSB variables) -- This psect contains the PSB.

8. JSVAR (JSB variables) -- This psect contains the JSB.

9. NRVAR (Nonresident variables) -- This psect contains monitor data locations that can be swapped.

10. NRCOD (Nonresident code) -- This psect contains the monitor code that can be swapped, including the processing routines for all the JSYSs. This psect is usually write-locked.

11. BGSTR (Bugstrings) -- This psect contains ASCIZ strings that describe each BUGINF, BUGCHK and BUGHLT. Like NRCOD, this psect is swappable and write-locked.

12. BGPTR (Bugpointers) -- This psect contains a few words for each BUGxxx, including the additional arguments and pointer to the corresponding Bugstring. This psect is also swappable and write-locked.

13. NPVAR (Nonresident page variables) -- This psect contains monitor variables that are allocated a page at a time and can be swapped. One of the main features of this psect is the resident free pool (RESFRP), whose pages are locked in memory one at a time as they are allocated.

5.1.4 HINTS ON MONITOR BUILDING

The following hints appear below as an aid to those who must rebuild monitors.

WHY THE PSECTS ARE WHERE THEY ARE

Several rules must be followed when rearranging the psects:

1. BOOT reads the monitor in around itself in virtual memory. Since BOOT needs to remain mapped and functioning until the swappable monitor has been read in and started, it must not lie in any part of the monitor's virtual address space that will be used by the monitor's initialization code. For this reason, the only three areas that can be used are a gap between psects, the NRVAR psect, or the last part of the NPVAR psect (the first part contains the resident free pool that is used by the

disk mounting code and the swapper). The last area is the one currently used.

2. RSCOD must be first; it checks to see if an address lies in RSCOD are just CAMLE MONCOR, which will not work if any psect falls below RSCOD.

3. All of the psects in the part of the monitor that is started first and that reads in the swappable monitor must be first. In addition, the last item in that part of the monitor must be the symbol table, which includes the RSCOD, INCOD, PPVAR, RSVAR, and SYVAR psects. The code in these psects must work before any swapping or paging can occur; therefore, they must all be low enough to fit in physical memory on the smallest configuration supported. Also, BOOT stops reading in the resident monitor when it hits the end of the symbol table, so the symbol table must be the last psect in this group.

4. The group of nonzero psects that swap are treated as a unit by certain parts of the monitor, and should therefore be together. These psects are NRCOD, BGSTR, and BGPTR.

5. PSVAR, JSVAR, POSTCD, NRVAR, and NPVAR can generally be put anywhere. They have been moved in the past with success.

5.1.5 WHICH PSECTS CAN OVERLAP

The POSTCD psect can overlap any xxVAR psect, since it will be gone by the time MONITR.EXE is generated. POSTCD is currently allocated its own three pages to avoid psect overlap warnings from LINK (a cosmetic precaution only).

With hidden symbol processing, the SYVAR psect and the symbol table can overlap any other xxVAR psects. The SYVAR psect is currently allocated its own page to avoid psect overlap warning messages from LINK.

If BUGSTF is not set and the bugstrings and bugpointers are not present in the running monitor, the BGSTR and BGPTR psects can overlap any xxVAR psect. In the current monitor, they would probably be overlapped with the NPVAR psect, which immediately follows them.

No other psect overlaps can be allowed without breaking the monitor.

5.1.6 HOW TO CONTROL THE SYMBOL TABLE ORIGIN

The psect origins of all the real psects are controlled by /SET switches in the LINK .CCL file. The symbol table, however, must be controlled in a more indirect way, via the LINK switches /SYMSEG and /UPTO.

The /SYMSEG:PSECT:name switch directs LINK to append the symbol table to the named psect. If symbols are hidden, the symbols should be appended to INCOD; if not, they should be appended to SYVAR. The symbols normally start 200 words after the end of the preceding psect. The extra 200 words are the PAT.. area.

The /UPTO:address switch tells LINK the highest legal address for the symbol table. If there are enough symbols to make the symbol table attempt to exceed the specified address, LINK will output a warning message and truncate the symbol table. (The resulting monitor should still run if this occurs.) The address should be set to one less than the base of the first psect that the symbol table cannot overlap.

5.1.7 SAMPLE OUTPUT FROM LINKING A MONITOR

The following is part of the output from linking a 2020 monitor; it shows both a new format for the output and the new address layout.

Monitor address space:

Psect	Start	End	Length	Free	Limit
RSCOD	2000	63633	61634	6144	
INCOD	72000	112634	20635	1143	
PPVAR	114000	117777	4000	0	
RSVAR	120000	214226	74227	13551	
SYVAR	230000	230777	1000	7000	
PSVAR	240000	325777	66000	2000	
JSVAR	330000	417777	70000	0	
NRVAR	420000	441426	21427	2351	
NRCOD	444000	660347	214350	13430	
BGSTR	674000	702602	6603	1175	
BGPTR	704000	704714	715	1063	
POSTCD	706000	710217	2220	560	
NPVAR	711000	772777	62000	5000	777777
Loaded symbols	112735	251750	137014	172027	NRCOD
Runtime symbols	231000	370013	137014	320764	NPVAR

The symbols will be moved to their runtime area right after the SYVAR psect by STG early in the monitor's initialization.

There are 57 (octal) free pages, not counting symbols.

% Runtime symbols must end by 360000 in order to run on a 128K system.

Writing sorted bug list to file BUGSTRINGS.TXT.1
Saving monitor as SMONITR.EXE.1

CHAPTER 6

MONITOR MODULES

ONITOR MODULES

he source code for the TOPS-20 monitor is in the form of macro iles. Each file contains code pertaining to a particular unction or device. These files are assembled into monitor odules. Some of the modules are assembled from only one file, hich may include searching certain universal files. But some of h modules are a combination of files. For example, consider he module TTYSRV. The file TTYSRV.MAC contains the terminal ervice functions. The module TTYSRV, however, is built of four iles: TTYSRV.MAC, KLPRE.MAC which indicates the terminal ervice is for a KL, TTFEDV.MAC which contains information about he lines being connected to a front-end, and TTPTDV.MAC which ontains the code required to support pseudo terminals.

ll modules are not contained in every TOPS-20 monitor. Some odules are built for certain hardware and/or software. For xample, the TTYSRV module described above is contained in a onitor built for a KL10, but the TTYSSM module contains the erminal information needed for a monitor that is to run on a 020.

he following describes the modules contained in the Release 4 onitor.

PRSRV This is the processor dependent service module for the KL10. It contains the initialization code for paging, MUUO handlers, and the priority interrupt system as well as for the clocks, APR, and DTE devices. Interrupt handling for these devices, pager control routines, and pre and post JSYS handling is also performed here.

PM This is the processor dependent service module for the KS10. It performs the same functions as APRSRV (excluding service for devices that do not exist on the KS10 such as

DTE service).

CDRSRV Card punch service.

CDRSM Card reader service for the KS10 processor.

CDRSRV Card reader service for the KL10 processor.

COMND Code for the COMND JSYS.

DATIME Code for the date and time conversion JSYSs.

DEVICE Device initialization and lookup code.

DIAG This module contains the code to support the DIAG JSYS for the KL10.

DIRECT Directory management code.

DISC This module contains the pre-PHYSIO disk dependent routines for I/O JSYSs and a dispatch table of vectored addresses, DSKDTB, which points to them.

DSKALC Drive type independent code for disk block allocation, including swapping space allocation, front-end file system definition, and structure definition.

DTESM Dummy replacement code for DTESRV for the KS10.

DTESRV DTE service driver; protocol handler for requests to and from the front-end.

EFILIN This module performs the same functions as FILINI using extended directory support.

ENQ This module implements the ENQ/DEQ facility to control simultaneous access to user specified sharable resources.

EFSRV Device code for FE devices. This code contains the device-dependent routines for the FE pseudo devices FE0-FE3.

FILINI This module contains code to initialize the file system at system startup.

FILMSC This module contains miscellaneous routines for the PTY, TTY string (includes break mask and field width support) and null I/O devices and also includes a device dispatch table for each of these devices.

FILNSP This module contains the filesystem interface to NSP. It includes the device dispatch tables for SRV: and DCN:.

FORK Contains the fork controlling JSYSs and support code.

ERLL Job storage free area management.

EUTILI Utility module which contains routines to copy strings to/from JSBs, routines to retrieve/change connected structure and directory information and routines to get a yes/no answer from CTY.

GTJFN Contains the code for GTJFN, and the JSYSs which support lookup, recognition, and creation of file names.

IMAN22 AN22 driver for the 2020.

IMPANX IMP driver for AN10.

IMPDV This module contains the Interface-Message-Protocol (IMP) device independent code. It runs cyclically as a separate fork (i.e., under JOB0) and handles the interface to the ARPA network by monitoring network activity and managing the message queues.

IMPPAR This is the parameter file for the IMP modules.

IO Contains most of the device-independent sequential, random,and dump input/output routines for BIN, BOUT, SIN, SOUT, DUMPI, and DUMPO.

IPCE Code for the system interprocess communications facility.

ISYSA Random JSYSs for system and directory access, device allocation, job parameter settings, system accounting and file/fork mapping.

ISYSE Contains code which implements various file system JSYSs.

LDRSRV This module provides support for DUP11's with a KMC11 for NSPSRV on a 2020.

LINIT This module at load time defines storage PCs for the JSYS dispatch table, JSTAB.

LNPRR Lineprinter service for the KL10.

LNPSM Lineprinter service for the KS10.

LOGNAM Contains the logical name definition and recognition JSYSs and routines.

LOOKUP Device independent file name lookup.

MGTAP This module contains the pre-PHYSIO magtape-dependent routines for I/O JSYSs and a dispatch table of vectored addresses, MTADTB, which points to them.

MEXEC This module contains the MINI-EXEC (MX) which is a limited

command interpreter for certain system loading/maintenence functions, and swappable monitor bootstrap procedures. It is part of the swappable monitor and also contains many JSYS routines.

MFILN Floating point input and conversion JSYSs.

MFLOUT Floating point output and conversion JSYSs.

MSTR Contains the code to implement the mountable structure JSYS, MSTR.

NETWRK This module contains the interface for all standard I/O JSYSs that communicate with the ARPA-network. It also provides a finite state machine of various events associated with a connection for the network control program (NSP).

NSPINT Network Services Protocol Internal Interface.

NSPPAR This parameter module contains data structures and symbol definitions required by NSPSRV. In particular, NSPPAR contains the definitions of the logical link blocks.

NSPSRV This module contains the control routines and JSYS interfaces for the message level protocol of DECNET known as NSP (Network Services Protocol), which allows communication between processes on hosts by means of logical links.

PAGEM Page management code; core management routines, swapper routines, pager trap logic, OFN control, and CST and SPT initialization.

PHYH11 Channel dependent code for RH11 controller.

PHYH2 Channel dependent code for RH20 controller at direct I/O level.

PHYM2 Device dependent code for TM02/TM45 magtapes at direct I/O level.

PHYP4 Device dependent code for RP04/RP06 disks at direct I/O level.

PHYPAR Universal file for PHYSIO and associated modules. It contains the definitions for the Channel Data Block, Channel Dispatch Table, Unit Data Block, Unit Dispatch TAble, and the Input/Output Request Block.

PHYSIO This module handles the channel and driver I/O routines. It is responsible for queueing I/O requests into their proper queue, choosing the "best" request for seeking and/or transferring and starting I/O.

PHYX2 Device dependent code for DX20.

PLT Plotter service.

POSTLD This code runs immediately following the loading of the monitor, and performs functions outside the capabilities of LINK. It builds the MONITR.EXE file, writes a BUGSTR text file and deletes itself from core.

PROKL Parameter file indicating KL10.

PROKS Parameter file indicating KS10.

PROLOG This is a file of parameters, storage assignments, and macro definitions. The major regions of the monitor address space are defined as well as macros affecting PI bug strings, pseudo-interrupts, and scheduling. All PSB and JSB storage defined by the monitor at assembly time is specified here. All of the BUGXXX definitions are included in this module from the file BUGS.MAC.

PTP Paper tape punch service.

PTR Paper tape reader service.

SCHED This module contains the Channel 7 Interrupt routine (which performs context switching), the process controller, the working set manager, the job/fork initialization/dismiss routines, and the Program Software Interrupt (PSI) analysis and resolution routines.

SERCOD This module contains the error codes and fields for SYSERR, a program which produces hardware performance reports for field service personnel.

STG The bulk of the monitor storage, both resident and non-resident, is defined in this module.

SWPACL This is the swapping space allocator which handles a device of some number (SWPSEC) of sectors, and some number (DRMMXB) of tracks. It has a resident bit table which is used to allocate swapping storage.

SYSERR Error reporting module for field service (not to be confused with the SYSERR program used to read the error information).

TAPE This module contains the tape handler and record processor.

TIMER This module implements the TIMER JSYS and all of its support.

TTYSRV This is the terminal service module for a KS10 ARPA monitor. See TTYSRV.

TTYSNV This is the terminal service module for a KL10 ARPA

monitor. See TTYSRV.

TTYSRV This is the terminal service module for the KL10. This module contains the TTY I/O drivers, special control character conversion routines, terminal JSYS routines and the interface to the primary and secondary protocols in DTESRV. Its device dispatch table is contained in FILMSC.

TTYSSM This is the terminal service module for the KS10. See TTYSRV.

VERSIO Version information for the monitor.

CHAPTER 7

WATCH

7.1 WATCH

WATCH is a TOPS-20 data collection tool that can be used to gather the information necessary to analyze both system and job performance. WATCH periodically samples many system variables, writing them in a format that is usable for analysis. Collecting WATCH statistics is beneficial whenever the system is running. Such statistics are often useful when usage trends are being analyzed in order to plan system growth. Any user can run WATCH and obtain most of the system information and some of the job information. These statistics are normally sufficient for determining overall system performance and for spotting short- and long-term usage trends. Expanded system and job information is available for users who are running with WHEEL or OPERATOR privileges enabled. This expanded set of statistics provides the much more detailed information that is often required to observe and tune the workload of an individual application.

7.1.1 OUTPUT

The WATCH output consists of nine different display sections:

1. Heading -- This section contains the date, time, number of jobs logged in, and the time interval over which the data sample was collected.

2. System Statistics -- This section contains system-wide statistics that reflect the resource utilization of the CPU, disk, and memory.

3. Load Averages -- Load averages indicate the number of runnable processes over specified intervals. This section indicates the load average for the system, for the interactive and computational queues, and for each class (when class scheduling is in use).

4. Directory Cache -- A cache of the most recently used directories is kept by the monitor. This section displays statistics that indicate the usefulness of this cache.

5. Normal Per-Job Information -- Per-job statistics that relate the amount of CPU resource distributed to each job are displayed along with statistics concerning class utilization (when the class scheduler is in use).

6. Expanded Per-Job Information -- In addition to the CPU information, this display presents many statistics that show the states in which the job spent time and how large the job is. The display also provides disk and swapping information.

7. System Utilization Statistics -- The system utilization statistics include a summary of the expanded per-job section, additional system statistics, and computations of several key variables.

8. Disk I/O -- Disk I/O statistics are displayed on a per-drive basis. Included in these statistics are the number of seeks, reads, and writes performed by each drive.

9. Tune Mode Display -- This display is a single line that contains some of the more revealing system statistics and summary statistics from the system utilization section. It is a useful "quick and dirty" display for users who are monitoring changes in the system load.

In addition to the above displays, data record output can also be requested. This form permits further computer analysis of data output from WATCH. Consider the following example of system statistics:

```
SUMMARY at  8-Oct-79 10:52:21
  for an interval of   1:59.9  with  54  active jobs.

USED:  21.1    IDLE:  42.7    SWPW:   2.0    SKED:   2.9
SUSE:  20.7    TCOR:   0.1    FILW:  29.6    BGND:   1.6
NTRP:  10.2    NCOR:  2.00    AJBL: 21.03    NREM:     0
TRAP:   1.2    NRUN:   0.9    NBAL:   0.9    NWSM:  66.6
BSWT:   0.5    DSKR:  18.0    DSKW:   9.4    SWPR:   5.7
NLOD: 13.52    CTXS:  21.9    UPGS: 1966.    FPGS:  205.
DMRD:   1.1    DMWR:   0.7    DKRD:   3.0    DKWR:   2.5
TTIN:  12.8    TTOU:  242.    WAKE:  13.7    TTCC:  2.50
TDIO:   7.4    RPQS:   3.3    GCCW:   1.6    XGCW:     0
CNOB:    11
QUEUE DISTRIBUTION PERCENTAGE:
                  0.20   6.95  10.28   3.64   0.00   0.00
```

The output data record for this WATCH output would have the following form:

```
)100110/08/79 10:52:2100119,90540021,100042,700002,000002,90
)020,700000,100029,600001,600010,200002,000021,030000,000001
,200000,900000,900066,600000,500018,000009,400005,700013,520
)21,901966,200205,400001,100000,700003,000002,500012,800242,
)00013,700002,500007,400003,300001,600000,00000110000,200006
95010,280003,640000,000000,000000,000000,000000,000000,000
```

All of the variables and the format of the data records are described in the "WATCH VARIABLES" section that follows.

.1.2 RUNNING WATCH

WATCH can be run by all users, though users with OPERATOR or WHEEL privileges enabled can obtain more information than others. WATCH is run by typing either "WATCH" or "R WATCH". When WATCH starts, it identifies itself with a message like:

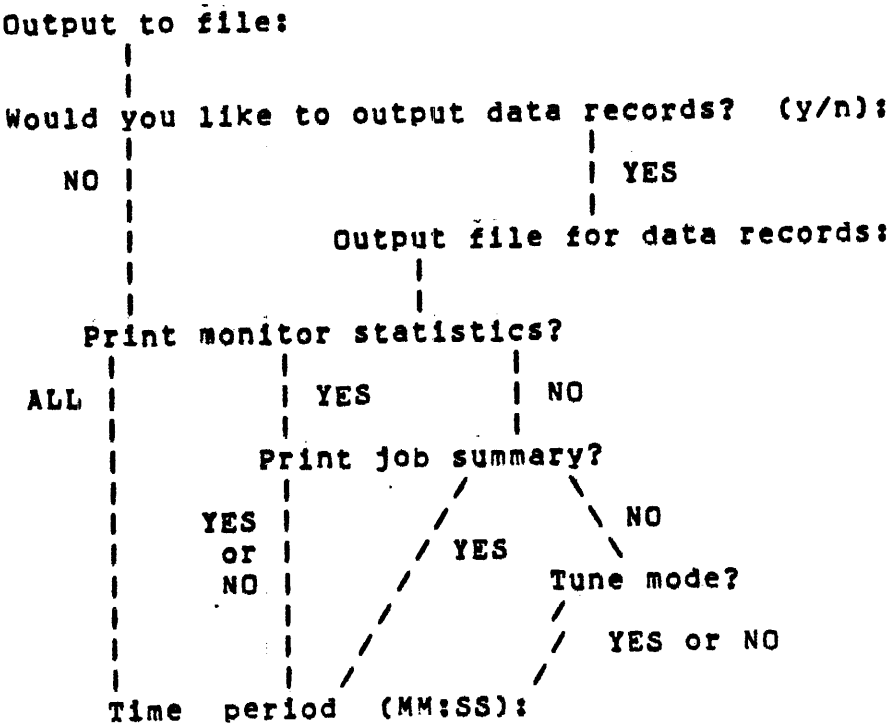
```
WATCH 4(3), /H for help.
```

Information is then requested in the following order:

```
Output to file:
Would you like to output data records? (Y/N)
Output file for data records:
Print monitor statistics?
Print job summary ?
Tune mode?
Time period (MM:SS):
```

Some of these requests are not made if previous answers indicate

that the information is unnecessary. The following diagram outlines the order of the questions:



The output of data records is indicated with the second and third questions. The output display received is as follows:

1. Headings and load average displays are always received except in tune mode.

2. "Yes" to the monitor statistics question causes the system statistics display to be output.

3. "Yes" to the job summary question causes the normal per-job statistics display to be output.

4. "All" to the monitor statistics question causes all displays but the tune mode display to be output.

5. "Yes" to the tune mode question (which occurs only when "no" is given for both the monitor and job summary questions) causes the tune mode display to be output.

After the user enters the necessary information, WATCH displays the message:

WATCH IN OPERATION --

and then takes its first sample. This message is seen immediately after entering the interval time unless the user is

requesting displays that require privileges (extended and tune mode). Then, a 10-30 second pause occurs (depending on system load) while the SNOOP breakpoints necessary to collect the information are inserted into the TOPS-20 monitor.

When the user desires to stop collecting statistics, the following procedure should be followed:

```
^C
^C
CLOSE
RESET
```

At this point, the output can be printed.

7.1.3 SOME INDICATORS

When variables have the values indicated in the following list, the system is usually in "balance." Since it is quite possible for one variable to appear in balance, while others are not, this information is only a guideline.

1. NCOR = 30 PER MINUTE -- Expensive if higher.

2. NRUN/NBAL = 1 -- All processes wanting to run can fit in memory.

3. SWPR less than 20% -- Swap reads/writes are overhead and thus should be (ideally) a small component of disk usage.

4. SWPW close to 0 -- Since this variable represents processes waiting for memory when no others can run, utilization of the system can normally be increased by adding memory until this is a small value. However, if the load has a large I/O component, additional memory may merely shift the CPU idle time from SWPW to FILW.

5. SKED = 15% -- Scheduler overhead detracts from cycles going to user programs. Programs that do very little work each time they are scheduled generally drive this value up. If this value is high, it is important to determine if there is a set of applications which could be reprogrammed in order to do more work between interactions. Programs that become active as each character is typed (like some screen formatting software) should be viewed with suspicion.

6. FILW close to 0 -- This is CPU idle time caused by processes waiting for disk I/O to complete. More memory permits larger numbers of programs to be resident. At other times, reconfiguring the disk access patterns to spread the disk I/O more evenly across the disks and channels lowers this value.

7. BSWT/NBAL small -- If a large proportion of processes in memory are waiting on the disk, the CPU is not being utilized.

8. NREM = 0 -- Since this counts the times runnable processes are removed from the balance set, performance is best when it is zero. Performance degrades rapidly as the value increases.

9. FPGS large -- If the number of free pages is low, the system needs to expend resources to garbage collect more often. This statistic, along with NREM, can be used to indicate a system overload.

10. DMRD+DMWR less than 20 per second -- Because drum reads/writes utilize a percentage of the disk system's bandwidth, higher throughput is possible when swapping is low. Normally, swapping of less than 30 pages per second does not cause any visible effect. If the normal load contains a large amount of user disk I/O, swapping at rates higher than 20 will decrease the system throughput. If the normal load is mostly interactive or computational, higher swapping rates can be sustained.

7.1.4 WATCH VARIABLES

The following is a list of the displays produced by WATCH. Beside each display type is the output data record type used for that display's information when data records are written.

1. Heading (All record types)
2. System Statistics (01)
3. Load Average (02)
4. Directory Cache (05)
5. Normal Per-Job Information (04)
6. Expanded Per-Job Information (03)

7. System Utilization Statistics (05)

8. Disk I/O Statistics (05)

9. Tune Mode Statistics (06)

In the following sections, each display is described and the format of the output data records is given. In the formats of the data records, the length of the fields is measured in characters (all fields in the records are ASCII characters). The form of each field is shown as a COBOL PICTURE (9 represents a numeric character, X an alphanumeric character). The first field of each record is the record type; the second field is the record sequence number. For record types 01, 02, and 05, the record sequence number is always "001".

.1.4.1 Heading -

The following is an example of the heading display. It shows the date and time, the length of the interval, and the number of users logged in. The interval shown is nearly equal to that specified by the user.

```
  SUMMARY at  6-Aug-79 09:28:23
    for an interval of   0:11.3    with 52  active jobs.
```

The heading information appears in fields 3, 4, and 5 of all output data records.

.1.4.2 System Statistics -

The following example of the system statistics display may be referenced while reading the descriptions of the variables.

```
USED:   87.6    IDLE:    0.0    SWPW:    0.0    SKED:   10.0
SUSE:   80.0    TCOR:    0.1    FILW:    0.8    BGND:    1.8
NTRP:   22.3    NCOR:   2.02    AJBL:  58.44    NREM:      0
TRAP:    3.2    NRUN:    5.7    NBAL:    5.7    NWSM:   61.0
BSWT:    2.8    DSKR:   23.0    DSKW:    4.8    SWPR:    6.4
NLOD:  16.12    CTXS:   64.1    UPGS:  1992.    FPGS:   147.
DMRD:    4.1    DMWR:    4.7    DKRD:   10.7    DKWR:    4.3
IT  :   11.0    TTOU:   401.    WAKE:   18.3    TTCC:   1.51
TDIO:   23.8    RPQS:    4.4    GCCW:    6.4    XGCW:      0
KNOB:     11
QUEUE DISTRIBUTION PERCENTAGE:
```

```
0.19  20.48  21.28  11.04  2.69  31.74
```

The statistics in this display are expressed either as percentages (%), as averages (AV), or as rates (Px). The rates are in units per minute (PM) or in units per second (PS).

The display includes "CPU usage statistics" from which the distribution of the CPU resource can be determined. The usage statistics are all percentages; their sum should be 100% (+/- roundoff error). The CPU Usage Statistics are USED, IDLE, SWPW, SKED, TCOR, FILW, BGND, and possibly TRAP.

Each of the variables in the example are described below; they are taken line by line in left-to-right order.

USED: (%) -- Percentage of the interval during which the CPU was executing instructions on behalf of some user. This includes user, JSYS, page fault, and interrupt processing.

IDLE: (%) -- Percentage of the interval during which the CPU was idle because there were no active processes on the system. If this number is nonzero, the system can accommodate some additional usage. If the CPU is idle when there are active processes, its idle time is accounted for in SWPW or FILW, not under IDLE time.

SWPW: (%) -- Percentage of the interval during which the CPU was idle and while one or more processes was waiting on the completion of a memory management service (normally a swap in). For time to be accounted in this variable, all active processes must be in wait states.

SKED: (%) -- Percentage of the interval during which the system was scheduling users for memory and the CPU. Other system overhead functions are measured by the variables TCOR and BGND.

SUSE: (%) -- Sum of the runtime percentages accumulated for each job running at the time of the report. This value differs from USED only by the skew that builds up during the time that it takes WATCH to collect all of the data about each job and by the loss of data from jobs that logged out during the interval.

TCOR: (%) -- Percentage of the interval spent garbage collecting memory. This represents part of the memory management overhead. The garbage collection process requires the monitor to look at the age of each page in memory to determine the ones that have not been referenced in a while. The least recently used pages become the prime candidates for being swapped out.

FILW: (%) -- Percentage of the interval during which the CPU was

idle, no processes were waiting for memory management services, and at least one active process was waiting for a short-term, user-initiated event (disk I/O) to complete.

BGND: (%) -- Percentage of the interval during which the monitor was performing background tasks. The primary background task is moving terminal input characters from a system-wide buffer to the individual terminal input buffers. This variable also includes the CPU overhead to echo terminal input characters.

PTRP: (PS) -- Number of page fault traps per second. Not all page faults require disk input to be resolved. Some page faults are resolved with pages that are currently in memory but not assigned to the process generating the page fault. Some of these pages are found in the replaceable queue (see RPQS below) while others may be shared by other processes.

MCOR: (PM) -- The average number of memory garbage collections per minute performed by the monitor during the last interval.

MJBL: (PM) -- The average number of times per minute the system was forced to adjust the balance set during the last interval. A rate of 60 times per minute is normal on a system with several users and no IDLE time.

MREM: (PM) -- The average number of times per minute the monitor had to remove a process from the balance set before the process came to a natural wait state (such as terminal input wait). This number becomes nonzero whenever there are more jobs to be run than can fit simultaneously in memory. Whenever this situation occurs, the monitor removes processes from the balance set, swapping them out to make room for other runnable processes. In general, whenever this number goes nonzero, response time gets longer.

TRAP: (%) -- Percentage of the interval during which the CPU was responding to page faults. This time is normally charged to the user, and is therefore also part of "USED" time. If the monitor was built to remove this "TRAP" time from "USED" time, it will become part of the system overhead (like SKED) and cannot be billed. In such a case, the TRAP time must be added to the "CPU Usage Statistics" in order to account for 100% of the CPU time.

NRUN: (AV) -- The average number of processes that were simultaneously active during the interval. This number represents the CPU load on the system during the interval. When NRUN is greater than 1.0, the user programs experience an average execution time at least "NRUN" times longer than if the system were stand-alone.

NBAL: (AV) -- The average number of processes in the balance set

during the interval. If this number is less than NRUN by more than .5, the implication is usually that there is not enough memory to hold all active processes.

NWSM: (AV) -- The average number of working sets in memory during the interval. If this number is significantly larger than NRUN or NBAL, working sets are not being forced out of memory when processes go into a wait state (like terminal input wait) and, consequently, response times should not be greatly affected by paging.

BSWT: (AV) -- The average number of processes in the balance set that are waiting for the completion of some event. Normally this number reflects the number of processes waiting for a page to be read in from the disk. If NBAL - BSWT is less than one, there are not enough runnable processes in memory to keep the CPU busy 100% of the time. The idle time is included in SWPW or FILW.

DSKR: (%) -- Percentage of the processes in balance set wait (BSWT) that are waiting for a file page to be read into memory.

DSKW: (%) -- Percentage of the processes in balance set wait (BSWT) that are waiting for file pages to be written back to the disk.

SWPR: (%) -- Percentage of the processes in balance set wait (BSWT) that are waiting for a page to be swapped into memory from the swapping area of the disk.

NLOD: (PM) -- The average number of working sets loaded per minute into memory.

CTXS: (PS) -- The average number of context switches performed per second by the scheduler. A context switch happens when the running process voluntarily blocks, or faults on a page that is not in memory, or when a higher priority process is ready to run. Since it takes CPU time to perform a context switch, CTXS directly affects SKED.

UPGS: (AV) -- The average number of pages assigned to processes with loaded working sets. These processes may or may not be in the balance set, but they are allocated memory.

FPGS: (AV) -- The average number of physical memory pages that are currently available for swapping in user processes. The monitor normally keeps between 20 and 100 free pages. The monitor uses these pages (and the rest of memory not in use by balance set processes) as a page cache. For example, if a process reenters the balance set after waking up from a blocked state and it still has some of its pages in memory in the free page pool, those pages are used directly without requiring any disk I/O. It has been demonstrated that this

cache plays an important part in overall system performance. Therefore, if FPGS is very small, the system performance has most likely been degraded.

)MRD: (PS) -- The number of reads per second made to the swapping area.

)MWR: (PS) -- The number of writes per second made to the swapping area.

)KRD: (PS) -- The number of reads per second made to the file system.

)KWR: (PS) -- The number of writes per second made to the file system.

'TIN: (PS) -- The number of terminal input characters received per second from all terminals (real and pseudo) on the system.

'TOU: (PS) -- The number of terminal characters output per second by all jobs on the system. This includes real and pseudo-terminals.

AK": (PS) -- The number of process wake-ups per second. Some of the types of wake-ups that fall into this category are

IPCF	Process Termination
ENQ	DISMS
Terminal Input	TIMER
Terminal Output	IIC

TCC: (PS) -- The number of terminal interrupt characters (e.g., CTRL/C) typed per second.

DIO: (PS) -- The aggregate number of disk pages read or written per second to both the file system area and the swapping area. Normally, 60 pages per second for a one-channel and 100 pages per second for a two-channel system are saturation levels. This variable is the summation of DMRD, DMWR, DKRD and DKWR.

PQS: (PS) -- The average number of pages per second that were retrieved from the replaceable queue in order to satisfy page faults. These page faults do not require disk I/O.

CCW: (PS) -- The average number of pages per second that were freed by global garbage collections.

GCW: (PS) -- The average number of pages per second that were freed by local garbage collections on specific processes. These garbage collections remove those pages from a process' working set that have not been used in a long time.

KNOB: (value) -- This is the setting of the bias control knob. The twenty possible bias control knob settings can only represent six switch settings in Release 4. Low settings favor interactive jobs; high settings favor compute-bound jobs. The default setting is 11.

The QUEUE DISTRIBUTION PERCENTAGE represents the portion of USED time allocated to processes in the various scheduling queues. The first queue is only used by Job 0 and jobs in the special high priority category. Normally the percentage of runtime accumulated in this queue is small. The second and third queues are the interactive queues. If the sum of these two values is high, there is a high interactive load on the system. The last three queues are the computational queues. Processes move onto these queues only if they have entered a compute-bound phase. If the sum of these three values is high, the system load is primarily computational. When the class scheduler is turned on, interactive users are scheduled in queue order while processes in the lower three (computational) queues are given priority on the basis of their class' distance from its target share.

The system statistics data record (record type 01) has the following form:

Item	Length	Picture
1. Record Type	2	99
2. Record Sequence Number	3	999
3. Date & Time (MM/DD/YY HH:MM:SS)	17	X(17)
4. Interval	7	9(5).9
5. Number of Jobs	3	999
6. USED	7	9(4).99
7. IDLE	7	9(4).99
8. SWPW	7	9(4).99
9. SKED	7	9(4).99
10. SUSE	7	9(4).99
11. TCOR	7	9(4).99
12. FILW	7	9(4).99
13. BGND	7	9(4).99
14. NTRP	7	9(4).99
15. NCOR	7	9(4).99
16. AJBL	7	9(4).99
17. NREM	7	9(4).99
18. TRAP	7	9(4).99
19. NRUN	7	9(4).99
20. NBAL	7	9(4).99
21. NWSM	7	9(4).99
22. BSWT	7	9(4).99
23. DSKR	7	9(4).99
24. DSKW	7	9(4).99
25. SWPR	7	9(4).99
26. NLOD	7	9(4).99
27. CTXS	7	9(4).99

28.	UPGS	7	9(4),99
29.	FPGS	7	9(4),99
30.	DMRD	7	9(4),99
31.	DMWR	7	9(4),99
32.	DKRD	7	9(4),99
33.	DKWR	7	9(4),99
34.	TTIN	7	9(4),99
35.	TTOU	7	9(4),99
36.	WAKE	7	9(4),99
37.	TTCC	7	9(4),99
38.	TDIO	7	9(4),99
39.	RPQS	7	9(4),99
40.	GCCW	7	9(4),99
41.	XGCW	7	9(4),99
42.	KNOB	5	9(5)
43.	Queue Distribution %'s	70	
	Max 10 entries; each entry		9(4),99
44.	Number of Queue Dist Entries	2	99

	Total	361 Characters	

7.4.3 Load Averages -

The term "Load Average" refers to the average number of processes simultaneously demanding service over some interval of time. The following is an example of the load average display:

```
LOAD AVERAGES:            5.29    4.06    3.39
HIGH QUEUE AVERAGES:      3.76    2.86    2.25
LOW QUEUE AVERAGES:       1.54    1.20    1.14


CLASS LOAD AVERAGES
CLA    SHR     UTIL

0    80.00   96.20     4.70     3.08     2.94
1    15.00    3.80     0.23     0.19     0.15
2     5.00    0.00     0.00     0.00     0.00
```

LOAD AVERAGES--The system keeps three exponential load averages. These values represent the average load over the last 1 minute, the last 5 minutes, and the last 15 minutes. These numbers can be used to estimate the expected elongation of the elapsed time required to run a program. If the system load average equals X, the approximate elapsed time required to run an additional program on the system is at least (1+X)*Y, where Y is the stand-alone elapsed time required to run this program.

HIGH QUEUE AVERAGES--These values represent the load of interactive jobs (queues 1 and 2).

LOW QUEUE AVERAGES--These values represent the load of compute-bound jobs (queues 3, 4, and 5). The sum of the high queue average and the low queue average equals the load average.

CLASS LOAD AVERAGES--This display is of interest when class scheduling is being utilized. The following information is presented for all classes defined for the system.

CLA -- The class number.

SHR -- The class share of the processor. This share corresponds to the percentage of the CPU that the monitor will try to distribute among the jobs in this class.

UTIL -- The actual utilization achieved by each class. The value here should be less than the share unless the class has received some windfall.

The 1-minute, 5-minute, and 15-minute load averages are displayed for each of the classes. These load averages may appear to be very large because they are computed as follows:

$$\text{Class Load Average} = \frac{\text{\# Processes in class making demands}}{\text{Maximum (Share, Utilization)}}$$

Thus, if there are 5 processes making demands in a class with a share of .20 and a utilization of .15, that class' load average is 25 (5/.20).

If the class scheduler is not running, the utilization and load averages are all zero.

The following is the form of the load averages data record (record type 02):

Item	Length	Picture
1. Record Type	2	99
2. Record Sequence Number	3	999
3. Date & Time (MM/DD/YY HH:MM:SS)	17	X(17)
4. Interval	7	9(5).9
5. Number of Jobs	3	999
6. Load Averages (3 values)	21	
Each Value		9(4).99
7. High Queue Avgs (3 values)	21	
Each Value		9(4).99

8. Low Queue Avgs (3 values)		21	
Each Value			9(4).99
9. Class Load Averages			
Max 32 entries;			
each entry as follows:			
a. Class Number		5	9(5)
b. Share		7	9(4).99
c. Utilization		7	9(4).99
d. Load Avgs (3 values)		21	
Each Value			9(4).99
10. Number of Class Load Avg.		2	99
	Total	1377	Characters

'.1.4.4 Directory Cache Statistics -

'he directory cache statistics are only available when an enabled ıser responds "ALL" to the question "Print monitor statistics ?". 'he following is a sample display:

```
   Directory Cache hits: 175
   )irectory Cache Misses - Cache Full: 0
   )irectory Cache Misses - New Entry Added: 321
```

irectory Cache hits: -- The number of times an accessed directory was found in the cache.

irectory Cache Misses - Cache Full: -- The number of times an accessed directory was not found in the cache while all the cache slots were filled with active directories. In this case the most recently accessed directory cannot be put into the cache.

irectory Cache Misses - New Entry Added: -- The number of times the accessed directory was not found in the cache, though room was available to add it (possibly as a replacement for an inactive entry).

he "hit ratio" is computed by HITS/(HITS+MISSES) and provides a ood indication of the cache's effectiveness. For instance, the it ratio in the example is 175/(0+321) or 35% (not very good).

he directory cache portion of data record type 05 is shown in he "Disk I/O Statistics" section below.

7.1.4.5 Normal Per-job Information -

The normal per-job information which is available to all users running WATCH and consists of a line for each job that had an active process during the interval. The following example shows the statistics reported for each job:

JOB	TTY	USER	PROGRAM	DELTA RT	%RT	JU	CSH
0	DET	OPERATOR	SYSJOB	7.81	4.3	23.94	3.64
1	43	OPERATOR	PTYCON	1.71	0.9	1.86	3.64
5	47	OPERATOR	OPR	0.35	0.2	0.28	3.64
10	3	KELLEY	BASIC	5.31	3.0	9.85	3.64
13	12	DENNING	EXEC	8.46	4.7	7.17	3.64
14	24	KOVALCIN	WATCH	3.80	2.1	3.92	10.00
16	2	GRAVES	BASIC	3.72	2.1	9.18	3.64
18	21	BLIZARD	SYSTAT	4.46	2.5	13.55	3.64
19	14	BOYACK	EDIT	10.74	6.0	20.71	3.64
20	20	WOLFE	EDIT	0.27	0.1	0.34	3.64

JOB -- The job number assigned by the system when the user logged in.

TTY -- The number of the terminal that is being used by the user running this job. "DET" means detached.

USER -- The name of the directory that the user logged into.

PROGRAM -- The name of the program being run or the name of the EXEC command being used. Please note that the program name is obtained at the time the sample is taken. It is not possible to tell if the program or command was running during the entire interval.

DELTA RT -- The incremental amount of CPU time (in seconds) that the job used during the interval.

%RT -- The percentage of the interval represented by the DELTA RT. The sum of all %RT values is used to compute the SUSE.

JU -- Job Utilization. When the class scheduler is running this will normally be a nonzero value. It represents the CPU utilization accumulated by this job and charged to the job's class share. Because the class scheduler tries to divide the class share equally among all active users in the class, computational jobs within the same class should normally receive nearly the same job utilization.

CSH -- Class Share. When the class scheduler is running, this value reflects the class' share divided by the number of active jobs in that class -- this is the target share for the job. Normally the job utilization (JU) is less than a job's class share.

Th_e is one normal per-job data record (record type 04) for each job. The sequence numbers begin with "001" and increment by '001". The form of a record is

Item	Length	Picture
1. Record Type	2	99
2. Record Sequence Number	3	999
3. Date & Time (MM/DD/YY HH:MM:SS)	17	X(17)
4. Interval	7	9(5).9
5. Number of Jobs	3	999
6. Job Number	3	999
7. TTY Number (777 = DET)	3	999
8. User Name	20	X(20)
9. Program Name	6	X(6)
0. DELTA RT	7	9(4).99
1. %RT	4	99.9
2. JU	7	9(4).99
3. CSH	7	9(4).99
Total	89	Characters

.1.4.6 Expanded Per-job Information -

ost of the information presented in this display is obtained by etting breakpoints in the monitor with the SNOOP JSYS. Thus, his information is only available to users who are running WATCH ith either WHEEL or OPERATOR privileges enabled.

he presented information occupies a full 132-character line. or purposes of explanation, the columns are broken up as ollows:

ob Identification Information:

OB TTY USER PROGRAM

hese variables are the same as those listed in the normal er-job display.

ob Utilization Information:

RT DEMD USED GRDY BRDY SWPTR DSKR DSKW RPQW OTHR

emory, Response, and Disk Information:

MF NLD NRSP RESP SP WSS UPGS SWPR DSKR TPF IFA

he job utilization information and memory, response, and disk

information are discussed in the next two sections below.

JOB UTILIZATION INFORMATION

The following is an example of the job utilization portion of the extended per-job display:

JOB	%RT	DEMD	USED	GRDY	BRDY	SWPR	DSKR	DSKW	RPQW	OTHR
0	1.0	14.3	8.6		72.9	5.7	4.0	8.8		
2	0.1	0.7	18.0		82.0					
4	0.6	3.6	19.3		34.8	38.4	5.8	1.8		
7	0.0	0.3	15.5		19.3	65.2				
9	0.5	1.2	43.0		15.7		35.6	5.7		
10	9.7	49.8	20.7		63.5	2.9	11.9	0.9		0.5
12	0.1	0.4	29.4		70.6					
15	1.0	6.0	19.4		20.2	1.1	16.0	43.3		
16	22.4	38.7	61.5		38.5					
19	1.1	5.7	27.7		72.3					

%RT (%) -- The percentage of the interval during which this job actually received CPU time.

DEMD (%) -- Summation of the percentages of the interval that each process in the job was active. If only one process in the job is active during the interval (the normal case), DEMD is less than or equal to 100%. If more than one process are simultaneously active, DEMD could exceed 100% (for a job of n processes, DEMD could be up to n*100%).

The rest of the variables in this display indicate what a job was doing during its "active" period. These statistics are all expressed as percentages of DEMD and thus their sum is 100%. When assessing the importance of the statistics for a specific job, you should multiply these percentages by DEMD to get the percentage of the interval time.

USED (%) -- The percentage of DEMD that the processes in this job spent using the CPU.

GRDY (%) -- The percentage of DEMD that processes in this job were runnable but could not fit in the balance set. The most common cause for processes to be on this list is that there is not enough memory to hold all runnable jobs.

BRDY (%) -- The percentage of DEMD that processes in this job were in the balance set but were not being run. Usually processes in this state are waiting for their turn to use the CPU.

SWPR (%) -- The percentage of DEMD that processes in this job waited for page faults from the swapping area to be satisfied.

DS (%) -- The percentage of DEMD that processes in this job waited for file pages to read in from the disk.

DSKW (%) -- The percentage of DEMD that processes in this job waited for file pages to be written to the disk.

RPOW (%) -- The percentage of DEMD that processes in this job waited for a physical memory page to become available for swapping into. Usually, when time is accumulating here, there is a shortage of memory on the system.

OTHR (%) -- The percentage of DEMD that processes in this job spent in any of the other wait states.

7.1.4.7 Memory, Response, And Disk Information -

The following is an example of the memory, response, and disk information portion of an extended per-job display:

JOB	IMEM	NLD	NRSP	RESP	SR	WSS	UPGS	SWPR	DSKR	TPF	IFA
0	917.1	1	165	0.06	6	410.0	7.7	20	17	45	40
3	99.9	0	12	0.07	6	8.0	7.0	0	0		
4	192.5	3	28	0.11	6	37.0	7.2	29	4	57	25
7	78.2	0	1	0.39	6	27.0	25.0	8	0	32	7
9	99.9	0	12	0.12	2	56.0	54.8	0	9	57	69
10	198.1	4	95	0.12	5	124.0	54.2	32	109	62	87
12	199.8	0	17	0.03	3	18.0	7.6	0	0		
15	99.9	0	12	0.61	5	27.0	17.2	3	24	45	51
16	99.9	0	357	0.13	2	22.0	20.6	0	0		
19	99.9	0	329	0.02	4	15.0	13.1	0	0		

MEM (%) -- The percentage of the time that the working sets of a job's processes are in memory. This number is the summation of the percentages for each process in the job and thus may exceed 100%.

LD (CNT) -- The number of times working sets for processes in this job were loaded into memory. If this number is zero, no working sets were loaded during the interval (i.e., the working sets were in memory for the whole interval).

RSP (CNT) -- The number of responses that a job had during the interval. A response is counted whenever a process wakes up for one of the reasons specified under WAKE:.

ESP (AV) -- The average response time in seconds during the interval. Responses that require more than two seconds of CPU time to finish are not counted in this column.

SR () -- The "stretch ratio" for each response (as represented in RESP). The stretch ratio is obtained by dividing the elapsed time of each response by the compute time required to satisfy it. (SR = elapsed time/CPU time). The only responses counted are those that require less than two seconds of CPU time to complete. Thus, the stretch ratio is the elongation perceived by the interactive user.

WSS (SUM) -- The sum of the maximum working set sizes of all active processes in the job.

UPGS (AV) -- The average number of pages actually in memory when a process from a job is in the balance set.

SWPR (CNT) -- The number of times a process in a job waited for a faulted page to be read in from the swapping area. This does not include pages that were preloaded by the working set manager.

DSKR (CNT) -- The number of times a process waited for a disk read to complete. Because many programs prefault pages, this count will be different from the actual number of pages read.

TPF (AV) -- The average number of milliseconds that it took to satisfy each page fault for this job during the interval.

IFA (AV) -- The "inter-fault average". This value represents the average compute time in milliseconds between page faults for job. A large "IFA" means that the working sets of processes in this job are very stable.

There is one expanded per-job data record (record type 03) for each job. The sequence numbers begin with "001" and increment by "001". The form of the record is

Item	Length	Picture
1. Record Type	2	99
2. Record Sequence Number	3	999
3. Date & Time (MM/DD/YY HH:MM:SS)	17	X(17)
4. Interval	7	9(5).9
5. Number of Jobs	3	999
6. Job Number	3	9(3)
7. TTY Number	3	9(3)
8. User Name	20	X(20)
9. Program Name	6	X(6)
10. %RT	4	99.9
11. DEMD	6	9(4).9
12. USED	5	999.9
13. GRDY	5	999.9
14. BRDY	5	999.9
15. SWPR	5	999.9

.6. DSKR	5	999.9
.7. DSKW	5	999.9
.8. RPQW	5	999.9
.9. OTHR	5	999.9
10. IMEM	6	9(4).9
11. NLD	4	9(4)
12. NRSP	5	9(5)
13. RESP	6	999.99
14. SR	3	999
15. WSS	7	9(5).9
16. UPGS	7	9(5).9
17. SWPR	5	9(5)
18. DSKR	5	9(5)
19. TPF	4	9999
20. IFA	4	9999

Total	170 Characters	

.1.4.8 System Utilization Statistics -

wo sections comprise the system utilization statistics. The irst consists of summaries for expanded per-job statistics, n ding summaries for the job utilization information and the emory, response and disk information. The second consists of dditional system variables and several computations.

.1.4.9 System Summary Of Per-job Variables -

hese statistics are listed under the per-job statistics on the ine that begins "System Summary". The following is an example f such a summary:

```
 DEMD USED GRDY BRDY SWPR DSKR DSKW RPQW OTHR
339.2 28.6      51.9  5.0 11.5  3.0


 IMEM NLD NRSP  RESP SR   WSS  UPGS SWPR DSKR TPF IFA
102.9  37 1772  0.13  3 2168. 231.8  370  911  52  90
```

EMD (%) -- The sum of each item in the DEMD column. This represents the total demand put on the system over the interval.

SE GRDY BRDY SWPR DSKR DSKW RPQW OTHR (%) -- These values represent the average percentage of the DEMD time that the jobs were in these states.

IMEM (SUM) -- The summation of all IMEM per-job values. This sum is significant as an indicator of how many working sets belonging to processes active during the interval were simultaneously in memory. For instance, the value 4102.9% indicates that approximately 41 working sets belonging to active processes were simultaneously in memory. This number can be compared with NWSM:.

NLD (SUM) -- This is the number of working sets loaded during the interval. It should correspond to the rate given by the variable NLOD:.

NRSP (SUM) -- The number of responses counted for all jobs during the interval.

RESP (AV) -- The average response time for those responses measured (requiring less than two seconds of CPU time) during the interval.

SR (AV) -- The average stretch ratio for all interactions requiring less than two seconds of CPU time.

WSS -- The arithmetic sum of the WSS values for all jobs. This represents the maximum amount of memory that would have been required during the interval if all active processes achieved their largest size at the same time and were all in memory.

UPGS (AV) -- The average number of pages needed by the active processes at any specified point in time.

SWPR (SUM) -- The total number of swap reads done by jobs on the system in response to page faults. This does not include pages preloaded by the working set manager.

DSKR (SUM) -- The total number of disk pages read that caused processes to wait.

TPF (AV) -- The average time required to wait for a page fault (swap or disk) to be resolved.

IFA (AV) -- The average amount of compute time a job spends between page faults.

Additional System Variables and Computations

The information in this display includes additional system statistics not available in the system statistics display and computations of various other variables. The following is an example:

```
TOTRC:  1992  LOKPGS:  104  SHR PGS:  245  AVAIL MEM:  1888
NRUN MIN,MAX:            1   11
SUMNR MIN,MAX:        1879 2073
NRPLQ MIN,MAX:          28  170
```

```
SY  MEM DMD =                                        255.2
SWAP RATIO (SUM WSS / AV MEM) =                        1.15
ACTIVE SWAP RATIO (DMD/AVMEM) =                        0.14
MEM UTILIZATION ((UPGS+SHRPGS)/AVMEM) =                0.25
AV WS SIZE =                                          28.84
AV CPU TIME (MS) PER INTERACTION =       40.57
THINK TIME (SEC) PER INTERACTION =        1.43
```

TOTRC -- The number of physical memory pages available. This is the total physical memory minus the number of pages required by the resident monitor.

LOKPGS -- The current number of pages locked down by the monitor beyond the resident monitor pages. Out of this set of pages comes the terminal buffers, magtape buffers, line printer buffers, and other pages locked down during certain file system operations.

SHR PGS -- The number of pages of physical memory being shared by more than one process at the end of the interval. This is included in the count "AVAIL MEM".

AVAIL MEM -- The difference between "TOTRC" and "LOKPGS". This is the actual number of pages available for use by user programs.

IR MIN, MAX -- The minimum and maximum number of simultaneously active processes during the interval.

SUMNR MIN, MAX -- The minimum and maximum number of pages belonging to working sets in memory during the interval.

IRPLQ MIN,MAX: -- The minimum and maximum number of pages on the replaceable queue during the interval.

SYS MEM DMD -- The system average memory demand derived by computing the integrals of the memory forecast for each process during its active period, summing over all processes and dividing by the interval time. Whereas the "system summary UPGS" is the average amount of memory actually in use at any point in time, this value is the average amount forecast at any point in time.

SWAP RATIO (SUM WSS / AV MEM) -- The swap ratio is the system WSS divided by the amount of available main memory. If this is greater than 1, it represents the amount by which main memory would have to be increased to avoid any swapping.

CTIVE SWAP RATIO (DMD/AVMEM) -- The active swap ratio is the system average memory demand divided by the amount of available main memory. If this number is greater than 1, it represents the amount by which main memory would have to be increased to hold all jobs wanting to run simultaneously.

MEM UTILIZATION ((UPGS+SHRPGS)/AVMEM) -- The memory utilization is the number of system-used pages divided by the amount of available main memory. For active swap ratios greater than 1, this indicates how well the monitor is doing in keeping memory used.

AV WS SIZE - The average working set size is calculated from the integrals computed from the working set demands over the active period of each process divided by the sum of the active periods of each process.

AV CPU TIME (MS) PER INTERACTION -- The average amount of CPU time a job spends between each response.

THINK TIME (SEC) PER INTERACTION -- The average time spent by the user between the time the system requests a response and when that response is received.

The system utilization portion of data record type 05 is shown under the "Disk I/O Statistics" section below.

7.1.4.10 Disk I/O Statistics -

The following is an example of the disk I/O statistics display:

DISK I/O

CHN,UNIT	SEEKS	READS	WRITES	
0,6	380	485	300	PS #1
0,7				REL4 #0
1,0	49	185	34	SNARK #0
1,1	2	6		LANG #0
1,2				
2,3	57	68	32	MISC #0
2,4				
2,5	602	652	449	PS #0

These statistics display the following information:

CHN,UNIT -- The channel number and the unit number on the channel to which the disk is connected.

SEEKS -- The number of times the disk heads had to be moved to get to the next request during the interval. If multiple requests can be answered on the same cylinder, no seek will take place.

READS -- The number of pages read on this unit during the

interval.

WRITES -- The number of pages written on this unit during the interval.

The last column shows the name of the structure and its relative unit number within the structure.

Data record type 05 contains the information from three displays: Directory cache, system utilization, and disk I/O statistics. The following shows the format of these records:

Item	Length	Picture
1. Record Type	2	99
2. Record Sequence Number	3	999
3. Date & Time (MM/DD/YY HH:MM:SS)	17	X(17)
4. Interval	7	9(5).9
5. Number of Jobs	3	999
6. Dir Cache Hits	6	9(6)
7. Dir Cache Misses - Full	6	9(6)
8. Dir Cache Misses - New Entry	6	9(6)
9. TOTRC	5	9(5)
0. LOKPGS	5	9(5)
1. SHR PGS	5	9(5)
2. AVAIL MEM	5	9(5)
3. NRUN MIN	5	9(5)
4. NRUN MAX	5	9(5)
5. SUMNR MIN	5	9(5)
6. SUMNR MAX	5	9(5)
7. NRPLQ MIN	5	9(5)
8. NRPLQ MAX	5	9(5)
9. SYS MEM DMD	7	9(5).9
0. SWAP RATIO	8	9(5).99
1. ACTIVE SWAP RATIO	8	9(5).99
2. MEM UTILIZATION	8	9(5).99
3. AV WS SIZE	8	9(5).99
4. AV CPU TIME	8	9(5).99
5. THINK TIME	8	9(5).99
5. Disk I/O		
Max 15 entries; each entry as follows		
a. Channel	2	99
b. Unit	2	99
c. Seeks	6	9(6)
d. Reads	6	9(6)
e. Writes	6	9(6)
f. Name	10	X(10)
g. Number	2	99
'. Number of Disk I/O entries	2	99
Total	667 Characters	

7.1.4.11 Tune Mode Statistics -

Tune Mode is designed to display on one line some of the more interesting statistics so that a system programmer can easily monitor the changes in load during a test period. This mode is useful when a very short interval is desired (around 10 seconds). The information is abstracted from the system statistics and system utilization statistics displays and includes the following variables:

USED	SWPW	SKED	CTXS	WAKE	TDIO	NRUN	NWSM	NLOD	USED
52.8	2.4	6.8	41.2	16.6	13.1	2.2	62.5	6.87	34.8

GRDY	BRDY	SWPR	DSKR	DSKW	RPQW	OTHR
	41.7	2.3	19.5	1.8		

IMEM	NLD	NRSP	RESP	SR
2207.9	0	198	0.05	2

The definition of the variables on the first row can be obtained from the system statistics display and the definitions on the second and third rows from the system utilization statistics display.

The form of the tune mode data record (record type 06) is as follows:

Item	Length	Picture
1. Record Type	2	99
2. Record Sequence Number	3	999
3. Date & Time (MM/DD/YY HH:MM:SS)	17	X(17)
4. Interval	7	9(5).9
5. Number of Jobs	3	999
6. USED	7	9(4).99
7. SWPW	7	9(4).99
8. SKED	7	9(4).99
9. CTXS	7	9(4).99
10. WAKE	7	9(4).99
11. TDIO	7	9(4).99
12. NRUN	7	9(4).99
13. NWSM	7	9(4).99
14. NLOD	7	9(4).99
15. USED	5	9(3).9
16. GRDY	5	9(3).9
17. BRDY	5	9(3).9
18. SWPR	5	9(3).9
19. DSKR	5	9(3).9
20. DSKW	5	9(3).9
21. RPQW	5	9(3).9

22. OTHR	5	9(3),9
23. IMEM	6	9(4),9
24. NLD	4	9(4)
25. NRSP	5	9(5)
26. RESP	6	9(3),99
27. SR	3	999
Total	159 Characters	

CHAPTER 8

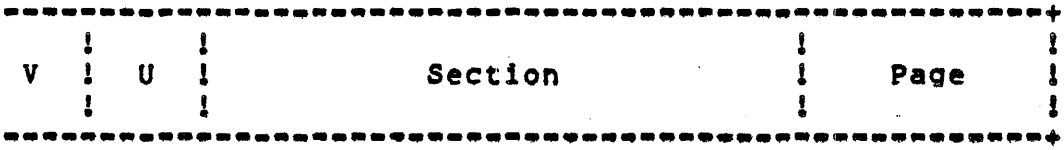
WORKING SET SWAPPING

.1 WORKING SET SWAPPING

ith TOPS-20, the working set of a process can be brought in with he overhead pages (preloading) and is removed upon being elected to be removed (postpurging). Postpurging of process orking sets always occurs. Preloading is not done by default, u' is a parameter that can be set by the administrator. If r oading is not done, the process page faults its pages in much he same way as it did previous to Release 4.

.1.1 WORKING SET DATA BASE

he information about the fork's working set is kept in a 28-word table in the PSB. This table is the working set cache. ach word contains four, nine-bit entries for a total of 512 ntries. An entry appears as follows:



here:

V — indicates whether the entry is valid.

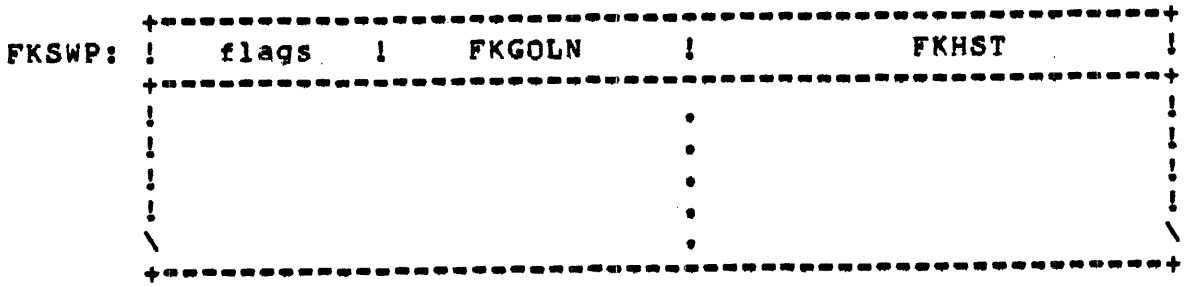
U — indicates whether the entry is a user page or a monitor page.

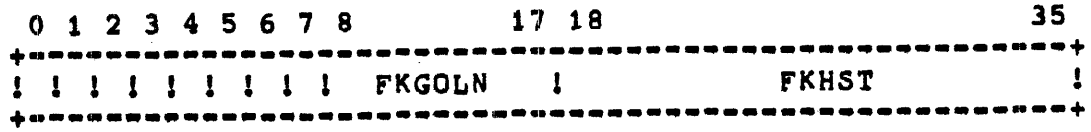
Section — indicates which section this page is in.

Page — is the higher-order two bits of the address of this page.

The lower-order seven bits of the page number are used as an index into the table. There can be conflict since pages with the same lower-order seven bits will have the same index. For this reason there are four entries at each index. If the number of conflicts exceeds four, the word is shifted left one entry. The leftmost entry is lost and the new entry is added on the right. This way the four most recent entries are "remembered."

Much of the information pertinent to the working set swapping of a particular fork is contained in the table FKSWP. This table, indexed by fork, contains a flag that indicates whether the fork's working set is in memory as well as whether other types of information are used to determine a fork's eligibility to be swapped in or out.





Symbol	Bits	Contents
FKWSL	0	Working set loaded -- on if the working set of the fork is loaded
FKBLK	1	Fork blocked -- on if fork is blocked and on the wait list; off if the fork is on the GOLST
FKIBS	2	Fork in balance set -- on if fork is in balance set (note that it is not necessarily in memory)
BSWTB	3	Balance set wait -- on if fork is in balance set (short-term, i.e. page fault) wait
BSNSK	4	NOSKED -- on if fork is NOSKED and not actually running
BSCRSK	5	Critical section -- on if fork is CRSKED and not actually running
FKIBH	6	In balance set hold -- on if fork entered balance set since last update to history
FKBSHF	7	In balance set hold -- local to AJBALS

FKGOLN	8-17	GOLST position
FKHST	18-35	Fork history -- value based on the recent history of the fork, particularly whether fork has been in the balance set or not

8.1.2 IMPLEMENTATION

Two areas of interest in the implementation of working set swapping are 1) the process of deciding which processes should be swapped in or out, and 2) the mechanics of maintaining a working set.

8.1.2.1 Swapping In/Out -

The working set manager must maintain enough available memory for processes to run, but it is desirable to keep in memory poorly behaved processes that block and unblock frequently to minimize swapping. To accomplish these goals, a fork's history is maintained.

The history (FKHST) of a fork is calculated periodically (currently twice a second) in the routine STEPFH using the following algorithm:

$$\text{FKHST (new)} = \frac{\text{FKHST (old)} * (\text{STEPC}-1) \; [+400000 \text{ (octal) if FKIBH}]}{\text{STEPC}}$$

where:

FKHST (new) is the new history value of the process

FKHST (old) is the previous history value of the process

STEPC is a time constant used to control the rate of change; currently set to 8

400000 (octal) is added if FKIBH is set, which indicates that the process was in the balance set during the previous interval

The larger the history value, the more recently and to a greater extent a process was in the balance set. This favors keeping the process' working set in memory.

To determine which working sets to keep in memory, which working sets to remove from memory, and which working sets to bring into memory, the working set manager calculates a value for each process. This value is calculated by tallying points of a process based on its characteristics. If a process has a specific characteristic, it gets the points indicated by that characteristic. The value's calculation is based on the information contained in the following table:

Item Checked	Characteristic	Value Added	Symbol For Value
PIBMP	Priority interrupt bump	40,,0	FHMBIP
BSNSK	NOSKED	4000,,0	FHMNSK
BSCRSK	Critical section	100,,0	FHMCSK
FKQN	On priority queue	400,,0	FHMPQ
FKIBS	In balance set	200,,0	FHMBS

If not runnable (based on the value FKBLK) add FKHST (history).
If runnable (FKBLK again) add FKGOLN (GOLST position).

If memory is overloaded, remove the working set of the process in memory with the smallest value.

If the largest value for a process not in memory is greater than the smallest value for a process in memory, try to bring the largest process not in memory into memory.

8.1.2.2 Mechanics Of Maintaining The Working Set -

The following operations must be able to occur:

1. Working set swapped out:

   The swapping-out is handled in a straightforward manner; the working set is selected and most of the mechanics is handled using the same code as XGC, the local garbage collection.

2. Working set swapped in:

   Swapping-in is handled by first getting the overhead pages for the process (PSP, UPT, and JSB), if necessary. Then, the working set is swapped in based on the working

set table (that was being used the last time the process was running).

3. Working set table updated as pages are used:

Maintenance of the working set table occurs in two major areas. When a process page faults, the page is added to the working set table by the page fault code. XGC (local garbage collection) now scans the working set cache to decide which pages to remove.

CHAPTER 9

SYSTEM DEBUGGING

9.1 INTRODUCTION

This document contains information pertinent to debugging. Some of it is relevant mostly to crash analysis. The last section contains information about MDDT, EDDT, and FILDDT.

Analyzing a crash is different than debugging a crash. Analysis tells you what happened and debugging tells you why (and thus, how to fix it). The information here can help you analyze what happened.

Successful crash analysis depends heavily on how well you know the data base and whether you can discover inconsistencies that give you clues about what happened. There are some basic things that help you look at any crash. The information in this document shows you how to use available tools to look at a crash and how to find basic information about the state of the machine at the time of the crash. Further analysis of a crash requires that, based on the state of the data base, you are able to propose a reason for what happened that matches the state of the data base.

9.2 CTY OUTPUT

Collect any CTY output that is relevant to the crash. This should include the KLERR printout and the BUGHLT (as well as recent BUGCHKs and BUGINFs), if any. If the machine got a KEEP-ALIVE CEASED, then the KLERR output is the only reliable information you get (in release 3A). See the section on the BUGHLT location for an explanation of why this is true.

9.2.1 Explanation Of KLERR Output

KLERR includes the PC, the last memory fetch and information on the PI system. The PI information includes the following:

PI STATE: ON or OFF	--indicates whether the PI system is on or not.
PI ON: n	-- n indicates which of the 7 channels are enabled.
PI HLD: n	-- n indicates which of the 7 channels have an interrupt in progress.
PI GEN: n	-- n indicates which of the 7 channels have a pending interrupt.

Here is an example of the KLERR output on a KEEP-ALIVE CEASED error:

```
%DECSYSTEM-20 NOT RUNNING


KEEP ALIVE CEASED
KLERR -- VERSION V02-02 RUNNING
KLERR -- KL NOT IN HALT LOOP
KLERR -- KL ERROR OTHER THAN CLOCK ERROR STOP
KLERR -- KL VMA: 000000 035717    PC: 000000 035717
KLERR -- PI STATE: ON, PI ON: 177, PI HLD: 004, PI GEN: 001
KLERR -- EXIT FROM KLERR
```

9.3 GETTING A DUMP

DUMP.EXE is a pre-allocated file into which BOOT writes the dump. When the system comes up, SETSPD copies DUMP.EXE to DUMP.CPY.

9.3.1 How To Get A Dump

If the system does an auto-reload, the console front end will give BOOT the commands to get a dump. If the auto-reload doesn't work for some reason, you can force a dump as follows:

```
KLI -- VERSION VB06-07 RUNNING
```

```
KLI -- ALL CACHES ENABLED
KLI -- BOOTSTRAP LOADED AND STARTED
?DUPL STR UNI?DUPL STR UNI;problem because two PS:
                          ;structures on line.
BOOT>/d                   ;request a dump (after the
                          ;problem is corrected).

BOOT>                     ;type CR for default monitor
```

9.3.2 Where BOOT Lands

The console front end loads BOOT into KL memory. Of course, this overwrites whatever used to be in that part of memory. Therefore, BOOT is always loaded into a part of the monitor that contains pure code (i.e., so no data is destroyed); currently BOOT is brought in on top of part of APRSRV. If you need to look at code that is loaded where BOOT lands, you have to go to the listings. BOOT also uses some of high core to build the EXE directory for the file DUMP.EXE.

9.4 SYSERR

9.4.1 Overview Of SYSERR Functions And Data Base

SYSERR is a program that reads PS:<SYSTEM>ERROR.SYS and generates reports on hardware errors, system crashes, front end reloads, etc. Entries in the file ERROR.SYS are written via the SYERR JSYS. For a description of each of the types of entries, see the monitor tables.

When an ERROR.SYS entry is desired, the caller (which is one of the system programs such as QUASAR or the monitor itself) builds a SYSERR block as described in the monitor tables and does the SYERR JSYS. The SYERR JSYS adds the block to a queue in the monitor's address space; the queue header is SEBQOU (location 4). Then it wakes up the Job 0 task which processes the queue and writes the queued entries to ERROR.SYS.

9.4.2 Queued SYSERR Blocks In A Crash

A BUGHLT entry is generated when the system BUGHLTs; this entry is queued but not written to ERROR.SYS. The system is considered to be in an unsafe state at the time of the crash. When the system comes back up, code in SETSPD is called to move any queued SYSERR blocks in the dump to ERROR.SYS.

The queue header is location 24 (called SEBQOU); each SYSERR block consists of the standard SYSERR header followed by the information in the specific block type.

Sometimes it is useful to look at the queued SYSERR blocks in a crash, particularly the BUGHLT block. The BUGHLT block contains certain status information at the time of the crash. The status words are described below:

1. CONI APR,

   Read the status of the processor error and sweep flags. This information is stored in offset BG%APS of the BUGHLT block. The flags and status information returned by a CONI APR, are described on page 3-59 of the Hardware Reference Manual.

2. CONI PAG,

   This information is stored in offset BG%PGS of the BUGHLT block. A CONI PAG reads the system status of the pager. If TOPS-20 pagins is on, there is a 1 in bit 21. Bits 23-35 contain the contents of the EBR (the address of the EPT). For a description of all the fields, see page 3-41 of the Hardware Reference Manual.

3. DATAI PAG,

   This information is stored in offset BG%PGD of the BUGHLT block. A DATAI PAG returns the process status of the pager. DATAI PAG, returns the current and previous contex AC blocks, and the address of the UPT. For a complete description of the fields returned by a DATAI PAG, see page 3-42 of the Hardware Reference Manual.

4. CONI PI,

   This information is stored in offset BG%PIS of the BUGHLT block. A CONI PI returns the status of the priority interrupt system; it indicates which levels are on, whether the PI system is on, and on which levels interrupts are currently being held. For a complete description, see page 3-7 of the Hardware Reference Manual.

9.4.3 Moving SYSERR Blocks From A Crash To ERROR.SYS

As stated before, SETSPD moves queued SYSERR blocks from the crash to ERROR.SYS. A Job 0 task starts the SETSPD program at START3; this code copies DUMP.EXE to DUMP.CPY and then issues a SYERR JSYS for each queued SYSERR block in the crash.

.5 BUGHLT

ocation BUGHLT contains the location the BUGHLT came from. That ocation contains an XCT BUGHLT-name. All BUGHLT code is enerated by the BUG macro which is defined in PROLOG.

.5.1 BUG Macro

```
EFINE BUG(TYP,TAG,STR,REGS,%NAM,%STR)<
        XCT [TAG::      JSR BUG'TYP
                IRP REGS,<
        Z REGS>
        SIXBIT /TAG/]
        .PSECT BGSTR
STR:    ASCIZ \STR\
        .ENDPS BGSTR
        .PSECT BGPTR
        XWD TAG,%STR
        .ENDPS BGPTR
```

his is an example of a call to the BUG macro:

```
UGHLT,JONRUN,<JOB 0 NOT RUN FOR TOO LONG, PROBABLE SWAPPING HANGUP
```

f a JONRUN BUGHLT occurred, then the data base would look like his:

```
        BUGHLT/ CAIA CLK2+6      ;address the BUGHLT came from

        CLK2+6/ XCT JONRUN       ;generated by BUG macro

        JONRUN/ JSR BUGHLT
              / SIXBIT \JONRUN\
```

.5.2 BUGHLT Contents

ocation BUGHLT is set up with the location the BUGHLT came from f the machine BUGHLTed. That location contains an XCT UGHLT-name. All the BUGHLT's are listed in the OPERATOR's GUIDE ith a short descriptive phrase.

f there is a zero in location BUGHLT the machine probably got a EEP ALIVE CEASED. This happens if you get either a "clock error to-" or "deposit/examine failure". Both of these errors are al-are failures and Field Service should be called. Although hese two errors are the most likely cause of keep alive ceased, ou cannot rule out the possibility of a software bug. Get the

KLERR output from the CTY. It will have the PC and PI state.

Note that because the console front end just does a reload for a KEEP ALIVE CEASED, the information in the dump is not dependable. This is because the cache has not been written out, the AC's have not been saved, etc.; these things are normally done by the BUGHLT code. The only valid information is the CTY output from KLERR.

For more information on release 4 changes, refer to the chapter "Bug'Typ Macro Changes for Version 4 of TOPS-20".

9.5.3 Creating Your Own BUGHLT

If you should desire the machine to stop in a certain state and save the information in a dump to allow later examination, you may want to create your own BUGHLT or BUGCHK. To patch a BUGHLT into the monitor, follow these steps:

In the patch are, create the BUGHLT information that is generated by the BUGHLT macro. Its format is:

```
        BUGHLT name/ JSR BUGHLT
                / .             ;addresses of any locations
                                ;whose contents are to be
                                ;dumped (left half must be
                                ;zero)
               /bughlt name in Sixbit
```

Redefine the patch area to begin after the BUGHLT information you just created.

Use the DDT patch facility to insert the code to check for the condition you wish to cause a BUGHLT.

9.6 PUSH DOWN LISTS AND RELATED DATA BASES

The push down list that is currently in use implies generally what was going on. For example, if the scheduler was running, SKDPDL is the push down list. IF a page fault was in progress, TRAPSK is the push down list. Also, the former P is saved.

9.6.1 How To Look At A Stack

1. P contains the current stack pointer.

2. If an entry was made on the stack by a PUSHJ, then it will look like a PC. This isn't a hard and fast rule, but it can help. A user mode PC ususally has bits 1,2,

and 3 on and a monitor PC has bits 1 and 2 on.

3. If a return address is still on the stack (i.e., has an address less than the stack pointer), then you haven't returned from the routine.

4. The monitor uses the stack for temporary storage. The macros STKVAR, TRVAR, etc. leave recognizable things on the stack. Knowing these conventions helps you to recognize which stack loactions are being used as temporary storage.

.6.2 Push Down List/ Machine State

he monitor uses different stacks to do different things. egister P is the current stack pointer and indicates which stack as in use at the time of the stack. The stacks and their uses re listed below:

1. UPDL -- Used when running in exec mode for the user, i.e, doing a JSYS. Also used by the Job 0 tasks that run in exec mode.

2. TRAPSK -- Used for page fault handling.

3. PIPDB -- Used for software interrupt handling.

4. SKDPDL -- Used by the scheduler for the overhead cycle.

5. DTESTK -- DTE interrupt level stack (PI level 6).

6. PHYPDL -- Used by PHYSIO when queueing and IORB.

7. PHYIPD -- Used when PHYSIO is handling an interrupt.

8. MEMPP -- Used when handling APR interrupts.

.6.3 Stack Usage For Local Storage

here are several macros that provide local storage; they use he stack. What they put on the stack is usually recognizable. ee MACSYM,MEM for further information.

1. STKVAR

   STKVAR uses the stack as temporary storage; the local variables have names that are really stack locations. STKVAR uses n stack locations for local variables where

n is the number of local variables requested, a count of local variables, and the return addres .STKRT. On the stack you will see:

```
/ local variable
/ local variable
/ .
/ .
/ local variable n
/ n,,n          ;count of local variables
                ;(used to adjust the stack
/ .STKRT        ;routine to clean up the
                ;stack and return
```

Therefore, when you find .STKRT on the stack, the word before it is the count of local variables and tells you how many locations on the stack are in use by STKVAR.

2. TRVAR

TRVAR uses the stack much the same way as STKVAR but it also uses AC15; this is pushed on the stack first. The stack locations it uses look like this:

```
/ AC15
/ local variable
/ local variable
/ .
/ .
/ local variable n
/ n,,n
/ .TRRET
```

Therefore, when you find .TRRET on the stack, the word before it is the count of local variables and register 16 is stored on the stack in front of the local variables.

3. ASUBR

ASUBR saves AC15, AC's 1-4, followed by the return address .ASRET, which is a routine to clean up the stack. When you see the address .ASRET on the stack you can expect the following in this part of the stack:

```
/ AC15
/ AC1
/ AC2
/ AC3
/ AC4
/ .ASRET
```

4. ACVAR

ACVAR can save AC5, AC5 and AC6, AC5-AC7, AC5-AC10, or AC5-AC14, depending on the arguments given. In each case, the return address to clean up the stack is the last item pushed on the stack by the ACVAR macro; the return address stored on the stack is the clue to what else was pushed on the stack. Each of the possible cases is listed below:

1. AC5 saved

```
/ AC5
/ .SAV1+2          ;return address
```

2. AC5 and AC6 saved

```
/ AC5
/ AC6
/ .SAV2+3          ;return address
```

3. AC5, AC6, and AC7 saved

```
/ AC5
/ AC6
/ AC7
/ .SAV3+4          ;return address
```

4. AC5, AC6, and AC7 saved

```
/ AC5
/ AC6
/ AC7
/ AC10
/ .SAV4+5          ;return address
```

5. AC5 through AC14 saved

```
/ AC5
/ AC6
/ AC7
/ AC10
/ AC11
/ AC12
/ AC13
/ AC14
/ .SAV8+7          ;return address
```

5. SAVEAC

SAVEAC takes a list of AC's to be saved as an argument. It pushes the list of AC's on the stack, followed by the address of a literal which is the routine that restores the stack. One of the instructions in the literal does a SUB P,[.NAC,,.NAC]. This macro does not leave such easily recognizable data on the stack, but if you find a return address on the stack that is a literal that does the following, SAVEAC was used. If you look at the code in the literal, you will be able to tell which AC's were pushed on the stack and how many there are (.NAC is the count of AC's pushed). The stack is left like this:

```
/ AC
/ AC
/ .
/ .
/ last AC saved
/ address of literal to restore stack
```

The literal to restore the stack looks (approximately) like this:

```
LIT= address of literal to restore stack
for this example.


LIT-1/ 3,,3      ;count of AC's saved =3
LIT/ CAIA 0
   / AOS -N(P)
   / MOVE 1,-2(17) ;restore AC1
   / MOVE 5,-1(17) ;restore AC5
   / MOVE 10,0(17) ;restore AC10
   / SUB 17,LIT-1  ;reclaim stack locations
   / POPJ P,       ;return to calle@
```

6. SAVEP

This macro calls the routine SAVP (in APRSRV) to save the AC's P1-P6 on the stack followed by the address RESTP, which is the routine to restore the AC's.

```
/ P1
/ P2
/ P3
/ P4
/ P5
/ P6
/ RESTP
```

7. SAVEQ

This macro calls the routine SAVQ (in APRSRV) to save the AC's Q1-Q3 on the stack, followed by the address RESTQ, which is the routine to restore the AC's.

```
/ Q1
/ Q2
/ Q3
/ RESTQ
```

8. SAVEPQ

This macro calls the routine SAVPQ (in APRSRV) to save the AC's Q1-Q3 and P1-P6 on the stack, followed by the address RESTPQ, which is the routine to restore the AC's.

```
/ Q1
/ Q2
/ Q3
/ P1
/ P2
/ P3
/ P4
/ P5
/ P6
/ RESTPQ
```

9. SAVET

This macro calls the routine SAVT (in APRSRV) to save the AC's T1-T4 on the stack, followed by the address RESTT, which is the routine to restore the AC's.

```
/ T1
/ T2
/ T3
/ T4
/ RESTT
```

.6.4 Stack Adjustment

any times the stack pointer is adjusted. Table BHC, indexed by , contains n,,n which may be added to or subtracted from the t pointer.

9.7 MACHINE STATES AND RELEVANT DATA BASES

9.7.1 PC Storage

1. PC at the time of the crash

   Location BUGHLT contains the PC at the time of the crash.

2. PC when JSYS began

   Two copies of the PC are saved on the stack.

3. PFL/PPC

   Current PC of process when the process was last context switched. Can be exec or user mode PC.

4. PIFL/PIPC

   Exec mode PC saved here while the software interrupt code is in progress.

5. Temporary PC storage

   When the system is changing state, it must always be prepared for a context switch. This is a concern when a JSYS is starting, when a process blocks, and when a software interrupt begins. In each case, the PC is temporarily stored in case of a context switch while the state change is in progress.

   1. SKDFL/SKDPC - PC saved here while process is blocking.

   2. MONFL/MONPC - PC saved here while nested JSYS is starting.

   3. ENSKR/ENSKR+1 - PC saved here while entering the scheduler via the ENTSKD macro. This is the PC the ENTSKD macro was called with.

9.7.2 AC Storage

.'_?.1 AC Storage In The PSB -

ach process's PSB contains several storage areas for saving C's. AC's are saved in the PSB in these cases:

1. Nested JSYS (JSYS called by a JSYS)

   When a user called JSYS is in progress, AC block 0 contains the monitor's AC's (the current JSYS code AC's) and AC block 1 contains the user mode AC's. If the JSYS code does a JSYS, AC block 1 (user mode AC's) are saved in the UACB area and the AC block 0 AC's are moved to AC block 1. For each level of JSYS, the AC block 1 AC's are pushed onto the UACB stack and the AC block 0 AC's are moved to AC block 1. Therefore, the AC block 1 AC's are always the previous context AC's; i.e., the AC's when the JSYS was called.

   If a nested JSYS is in progress, then the user mode AC's are the first stacked AC's in UACB. If the nesting is more than one level deep, then each subsequent JSYS's calling AC's are also saved in UACB; the current JSYS AC's are saved in UAC if the process is not currently running or in BUGACU if the process was running at the time of the crash. The maximum nesting level for JSYS's is 5; this limit is dependent on how much storage is reserved for AC stacking in UACB.

   ACBAS is the "pointer" for the AC stack UACB, but is not stored as an address. The contents of ACBAS must be shifted left 4 places to make it an address. The resulting address is the first saved AC for the last pushed AC block; i.e. the saved AC's for the next higher level of nesting. If there are no saved AC's pushed on the stack, ACBAS contains its initial value of <UACB>B39-1=37677; if ACBAS contains anything else, there are pushed AC blocks saved in UACB.

2. Process is context switched while running in user mode.

   The current AC's (i.e., the user mode AC's) are saved in block UAC.

3. Process is context switched while running in exec mode.

   The current AC's (i.e., the exec mode AC's) are saved in block PAC. The previous context AC's are saved in block UAC. The AC's saved in UAC are the user mode AC's unless a nested JSYS is in progress; in this case, the AC's saved in UAC are the AC's the nested JSYS was called with. The user mode AC's for this case are saved on the AC stack called UACB; the user mode AC's are the first saved AC's on UACB.

4. Software interrupt processing

   The exec mode AC's are saved in block PIAC while a software interrupt is in progress.

9.7.2.2 AC Storage At The Time Of The Crash -

1. BUGACS

   Exec mode AC's at the time of the crash. Copied to current AC's when using FILDDT.

2. BUGACU

   Previous context AC's at the time of the crash. These are the user mode AC's unless a nested JSYS was in progress, i.e., a JSYS called from a JSYS. If a nested JSYS was in progress at the time of the crash, BUGACU contains the AC's the current JSYS was called with. In this case, the user mode AC's are saved in the AC stack called UACB.

9.7.2.3 Summary Of AC Storage -

1. UAC

   Previous context AC's are saved here when the user is context switched. For the currently scheduled process, UAC contains the AC's the last time the process was dismissed. Once again, if a nested JSYS was in progress, UAC's contains the AC's that JSYS was called with. In this case, the user mode AC's are saved in the AC stack called UACB.

2. UACB and ACBAS

   Pushed AC blocks when nested JSYS in progress.

3. PAC

   Exec mode AC's saved there for process when it is dismissed.

4. PIAC

   Exec mode AC's saved there when software interrupt is in progress.

5. BUGACS

   Exec mode AC's at time of crash.

6. BUGACU

   Previous context AC's at time of crash.

9.7.3 Fork Scheduled Or Not

If a fork is scheduled, location FORKX contains the fork's system fork number. The scheduled fork's PSB and per process pages, JSB and per job storage, and page table are all mapped into the monitor's address space. If no fork is currently scheduled, location FORKX contains a -1.

9.7.4 Fork NOSKED

If a fork is NOSKED, its fork number is stored in SSKED; if there is not a NOSKED fork, SSKED contains -1. The FKSWP table contains status bits for both NOSKED (no other process may be scheduled) and CRSKED. CRSKED is a new process state in release 6 for processes manipulating the monitor resident free pool.

9.7.5 Extended Vs. Non-extended Addressing

If the machine supports extended addressing, flag EXADDR contains a 1; if the machine does not support extended addressing, EXADDR contains 0.

9.7.6 Sizes (resident, Non-resident, Total)

MONCOR/ number of pages in resident monitor

TOTRC/ total number of swappable core pages

NHIPG/ highest physical core page number

9.7.7 MDDT Page (release 3A)

When MDDT is in use for a process, DDTPPG (currently page 774) exists. If the running process's page 774 exists, then that process has been using MDDT; you might suspect that the crash was caused by an accidental deposit in MDDT or some such thing.

9.7.8 Interrupt Vs. Non-interrupt Level

9.7.9 Relevant Data Base For Each Machine State

9.7.9.1 JSYS -

1. Stack

   UPDL

2. Initial stack setup

   Initial UPDL set up for JSYS if from user mode:

```
/ PC at time of JSYS
/ PC flags at time of JSYS
/ PC at time of JSYS
/ PC flags at time of JSYS
```

   Initial UPDL set up for JSYS if from exec mode (nested)

```
/ INTDF
/ MPP (for higher level JSYS)
/ PC at time of JSYS (return PC)
/ PC flags at time of JSYS
```

3. Previous PC

   The return PC is pushed on stack; MPP is stack pointer for return PC.

4. Saved AC's

   AC's saved in UACB if nested JSYS. See section on AC STORAGE for a description of UACB.

5. Saved stack pointer

   If the JSYS was from user mode, this is not relevant. If the JSYS is nested, the previous JSYS also used this stack and the MPP pointer can be used to determine where the stack pointer was when this JSYS began:

```
Previous
stack ptr-> /
            / INTDF
            / MPP (for higher level JSYS)
            / MONPC
    MPP ->  / PC at time of JSYS (return PC)
```

6. AC usage

   There is no standard for AC usage that all JSYS's conform to.

7. Related Storage

   1. MPP -- points to:

      --return PC for JSYS

      --last location of initial setup for this JSYS

   2. FPC = KIMUPC -- dispatch address for JSYS

   3. KIMUU1 - last UUO from user in format:

```
KIMUU1/ flags,,opcode
      / JSYS number
```

   4. INTDF

      Indicates whether the process is NOINT, and to how many levels. Set to -1 if the process is not NOINT; greater than or equal to zero if the process is NOINT.

.7.9.2 Page Fault -

1. Stack

   TRAPSK

2. Initial stack setup

   The initial stack setup differs for each of three cases:

   --page fault from user mode

   --page fault from exec mode

   --recursive page fault

1. Stack setup on page fault from user mode

```
/runtime
/return PC
/return PC flags
```

2. Stack setup on page fault from exec mode

```
/ AC1
/ AC2
/ AC3
/ AC4
/ AC7
/ AC16
/ TRAPSW
/ runtime
/ PC
/ PC flags
```

3. Stack setup on recursive page fault

```
/ AC1
/ AC2
/ AC3
/ AC4
/ AC7
/ AC16
/ TRAPSW
/ PC
/ PC flags
```

3. Previous PC

   Saved on stack; see initial stack setup for where.

4. Saved AC's

   Those AC's that are saved are kept on the stack. See the initial stack setup for where each AC is saved.

5. Saved Stack Pointer

   TRAPAP

6. AC usage

   Differs for each type of page fault.

7. Related Storage

1. TRPID --identity of page causing trap in form PTN,,PN or PTN

   This is the identity of the page the page fault handler is working on. TRPID contains the page's page table identity while the page's page table is brought into core (if the page table was not in core).

2. TRPPTR

   Storage address of the page the page fault handler is working on.

3. TRAPSW (copy of TRAPS0)

4. TRAPC

   0 if first level page fault; if greater than 0, it indicates the level of recursion.

5. TRAPFL/TRAPPC = UPTPFL/UPTPF0

   Flags and PC at time of page fault.

6. TRAPS0 = UPTPFW

   Page fail word; contains the address that page faulted.

.7.9.3 Scheduler -

1. Stack

   SKDPDL

2. Initial stack setup

   None

3. Previous PC

   Saved in PSB for a process when context switch to the scheduler; the PC is saved in PFL/PPC of the process's PSB. If FORKX contains a fork number, it is the number of the fork running when the scheduler was invoked. If FORKX is not setup, you cannot determine which fork was running last.

4. Saved AC's

   The process's AC's are saved in block PAC (exec mode AC's) and block UAC (previous context AC's) of the PSB for the process. If FORKX is not setup, you cannot determine which process was running last.

5. Saved Stack Pointer

   In the saved AC's.

6. AC usage

   FX/ -1 if no fork chosen or system fork number of chosen fork

7. Related Storage

   1. FORKX

      FORKX contains a -1 if no fork is chosen or the fork number of the chosen fork.

   2. Temporary storage while entering scheduler

9.7.9.4 Physio Queueing Level -

1. Stack

   PHYPDL

2. Initial Stack Setup

   The P and Q AC's are saved on the stack by the macro SAVEPQ; the AC's are saved in order of Q1 through Q3 followed by P1 through P6. See the section on Stack Usage for Local Storage for the format.

3. Previous PC

   Since PHYSIO is called with a PUSHJ, the previous PC is the top of the saved stack.

4. Saved AC's

   . AC's Q1-Q3 and P1 through P6 are saved on the stack. See the section in Stack Usage for Local Storage entitled SAVEPQ.

5. Saved Stack Pointer

   The previous stack pointer is saved in PHYSVP.

6. AC usage

   P4/ address of IORB being queued.

   P1/ address of CDB

   P3/ address of UDB

   P2/ address of KDB or 0 if no KDB

7. Related Storage

.7.9.5 Physio Interrupt Level -

1. Stack

   PHYIPD

2. Initial Stack Setup

   None.

3. Previous PC

   The previous PC is saved by the XPCW instruction in a two word block, beginning at the CDB-6.

4. Saved AC's

   PHYACS -- block where AC's saved

5. Saved Stack Pointer

   The saved stack pointer is in PHYACS+17.

6. AC usage

   P1/ address of CDB

   P2/ address of KDB or 0 if none

   P3/ address of UDB

   P4/ IORB address or argument
   indicating action code:

P4<0 schedule a channel cycle
(P4) = -1
P4=0 dismiss interrupt
P4> housekeep current request
(contains IORB address)

7. Related Storage

Home block check funtion. In STG, starts at CHBUDB.

9.7.9.6 APR Interrupt Level -

1. Stack

MEMPP

2. Initial stack setup

/UPTPFO= TRAPPC
/UPTPFL= TRAPFL
/UPTPFN
/UPTPFW= TRAPS0

3. Previous PC

Saved as double word PC by XPCW in location PIAPRX and PIAPRX1

4. Saved AC's

MEMPA -- block where AC's 0-10 are saved.

NOTE

Beginning in 3A, uses a different AC block while at APR interrupt level; therefore, no AC's are saved.

5. Saved Stack Pointer

MEMAP -- previous stack pointer saved there.

6. AC usage

7. Related Storage

1. Sets "local" page fail routine to MEMPTP

.7.9.7 DTE Interrupt Level -

1. Stack

DTESTK

2. Initial Stack Setup

None.

3. Previous PC

Saved in DTETRA.

4. Saved AC's

DTEACB -- block where AC's saved.

5. Saved Stack Pointer

Previous stack pointer is saved in DTEACB+17.

6. AC usage

F/ result of CONI DTEn,

A/ DTE number of DTE that caused interrupt

3/ count (if RSX20F protocol)

4/ Byte pointer (if RSX20F protocol)

7. Related Storage

.7.9.8 PSI Handling -

1. Stack

. PIPDB

2. Initial Stack Setup

3. Previous PC.

   PIDL/PIPC

4. Saved AC's

   PIAC -- block where AC's saved.

5. Saved Stack Pointer

   Previous stack pointer is saved in PIAC+17.

6. AC usage

   FX/ interrupt flags from FKINT

7. Related Storage

   1. How to tell if interrupt in progress

9.7.9.9 Job 0 Exec Mode Tasks -

1. Stack

   UPDL

2. Initial Stack Setup

3. Previous PC

   Since these are scheduled processes, this is not relevant.

4. Saved AC's

   Since these are scheduled processes, this is not relevant.

5. Saved Stack Pointer

   Since these are scheduled processes, this is not relevant.

6. AC usage

7. Related Storage

1. How can you tell this use of UPDL from a JSYS?

If the FKJOB entry for the running fork is Job 0, then this is probably a Job 0 task as opposed to a JSYS in progress. Also, if the PC is in a Job 0 routine, this indicates a Job 0 task.

.7.9.10 User Mode -

he system never BUGHLTs in user mode; but it could KEEP ALIVE EASE. The PC is from user mode if the flag UMODF is set in the C.

.8 DDT'S

.8.1 FILDDT

he latest version of DUMP.CPY is the last crash. The program ILDDT is used to analyze a crash.

.8.1.1 How To Use FILDDT On A Crash -

o look at a crash with FILDDT you need the dump and the monitor ile it came from (for symbols). For example:

```
@ENABLE
$FILDDT
FILDDT>LOAD <SYSTEM>MONITR.EXE  ;load symbols
FILDDT>GET <SYSTEM>DUMP.CPY     ;load dump
```

he AC's contain their contents at the time of the dump. By efault you look at physical (not virtual) addresses.

.8.1.2 $U Command -

ILDDT can simulate KL paging. You can set paging using the ILDDT $U command. This command allows you to specify the ocation of the page table different ways, depending on how much nformation you have on where the page table is. For more nformation on the $U command, refer to the section on DDT ersion 41. In the part of the monitor that BOOT loads, there is one-to-one correspondence between physical and virtual

addresses; MMSPTN is in this part of the monitor's address space.

If you wish to look at some fork's address space, find its page table's SPT slot in the left half of FKPGS, indexed by fork number.

If you wish to return to physical addressing (i.e., no KL paging simulation), type $$U.

9.8.2 Relevant DDT/FILDDT Commands

These are standard DDT commands; however, you may not be familiar with them. They are included here along with examples of their use.

9.8.2.1 Question Mark (?) -

If you type a symbol followed by a question mark, DDT tells you which module(s) that symbol appears in; the module name is followed by a G if the symbol is global. A local symbol may be defined in more than one module.

This facility can be used to locate symbols, like GLOB, but faster.

```
BUGSTO?
APRSRV          ;symbol is local and defined in  APRSRV

SPT?
STG G           ;symbol is global and defined in STG
```

9.8.2.2 Underscore (_) -

A value, followed by underscore, is a request to DDT to find a symbol with that value.

This facility can be used to locate the symbolic address of a value.

```
14156_LSCHED+5

101400_SPT
```

.8.2.3 Effective Address Search ($E) -

he $E command is used to search for all locations where the ffective address, following all indirect and index-register hains to a maximum length of 64, base 10, equals the address eing searched for.

he format of the command is a<b>c$E; a<b> is the range and is ptional. If no range is specified, the whole address space is ssumed. The c argument is the address to search for.

```
MMSPTN$E
PGRI10+3/ MOVEM T1,MMSPTN
FPTA4/ SKIPA T1,MMSPTN
MLKPGM+2/ CAMN T2,MMSPTN
SWPER3+2/ CAMN T2,MMSPTN
GSMLER+11/ HRL T1,MMSPTN
BSMGP1+2/ HRL T1,MMSPTN
212777/ HRL T1,MMSPTN
SNPF0A+15/ HRL T1,MMSPTN
SNPF5B+10/ HRL T1,MMSPTN
UT1LL+1/ HRL T1,MMSPTN

JSB<JSB+5>0$E
JOBMAP+2/ 0
JOBMAP+3/ 0
JOBMAP+5/ 0
```

.8.2.4 Word Search ($W) -

ord search compares each storage word with the word being earched for in those bit postitions where the mask, located at M, has ones. The mask word contains all ones unless set by the ser. If the comparison shows equality, the word search types ut the address and the contents of the location; if the omparison results in inequality, the word search types out othing.

he format of the command is a<b>c$W. a<b> is the range and c is he quantity searched for. To set the mask type n$M where n is he quantity to be placed in the mask word.

uppose we wish to find all share pointers in the current user's age map between pages 0 and 10. We want to store a 7 (for ointer type) in bits 0-2 of the mask. Our command is PTA<UPTA+10>200000,,0$W and works as follows:

```
7000000,,0$M

UPTA<UPTA+10>200000,,0$W
UPTA+2/ 206000,,1244
```

```
UPTA+4/ 206000,,1242
```

9.8.2.5 Not Word Search ($N) -

Not word search works like word search, the only difference is that it types out the contents of the register when the comparison is an inequality and types nothing when an equality is reached.

Not word search is commonly used to type out all non-zero locations in some range. Suppose we wish to find all existint (non-zero) entries in the JSB map.

```
-1$M
JOBMAP<JOBMAP+66>0$N
JOBMAP/ 224000,,635
JOBMAP+1/ 124003,,7044
JOBMAP+4/ 124003,,2764
JOBMAP+6/ 124003,,7050
```

9.8.3 MDDT

MDDT is a part of the monitor that allows you to look at the running monitor with the standard DDT commands; your process is always the running process when you use MDDT. You can also call monitor routines to map pages, etc.; however, extreme caution should be taken when using MDDT. If you change any locations, you can crash the monitor. It is a good practice to type carriage return immediately after you open any location to prevent accidental deposits into memory.

You can enter MDDT in either of two ways. In the first example, the running fork will be the top fork of your job; i.e., the EXEC. In the second example, the running fork will be the fork running user level DDT.

```
@ENABLE
$^EQUIT
MX>/
MDDT


@ENABLE
$SDDT
MDDT%$X          (Jsys 777)
MDDT
```

ou can use either method to enter MDDT. Return from MDDT by alling the routine MRETN. Do this by typing:

```
MRETN$G
```

.8.3.1 MAPPING MONITOR PAGES -

here is oftentimes a need to examine pages of the Monitor which re not currently part of its virtual address space. Examples ight be the JSB or PSB of a hung job or fork or the Index Block f a file. The safest and easiest way to do this is to use the ETMPG routine (or MSETMP for a set of contiguous pages) to map he page(s) into the User's address space. Optionally read and rite or just read access can be requested. The only valid eason for write access would be if you intended to change a ocation in the mapped page in order to un-hang the job.

The following is the setup for calling SETMPG or MSETMP:

```
AC1/    Internal Identity of page
        (in form   0,,SPT index
                or
           SPT index,,Page number)

AC2/    Access,,Address to map to
        ( 500000,,XXX000  for read access
                mapping to page XXX
                        or
          540000,,XXX000 if read and write
                access desired)

AC3/    Number of Pages
         (if calling MSETMP)
```

The following is an example of mapping the JSB area pages f a job and the PSB for a fork:

```
ena
get <system>monitr.exe          ;get monitor's symbols
st 140                          ;DDT
DT

idt%$x                          ;go to MDDT
DDT

Mapping the PSB

cpgs 40[   1672,,1704           ;SPT index of fork 40's PSB
```

```
T1!      1704                    ;internal ID
T2!      500000,,psbpga          ;access,,destination

call setmpg$x                    ;only one page so call
<>                                 SETMPG

mretn$g                          ;return to user
<>


uptpf1/   CAIA 0                          ;flags,,pc of fork's
UPTPFO/   T1,,.SFRKV+12                   ;  last page fault
UPTPFN/   PGRTRP

fkrt[   4160                              ;fork's runtime

mddt%$x                          ;return to MDDT
MDDT

;Mapping the JSB pages for a job

fkjob 40[   22,,2152             ;get fork 40's JSB index

T1!      2152,,0                 ;SPT index,,page 0 of JSB
T2!      500000,,jsbpga          ;access,,destination
                                   of mapping (JSB area)
T3!      jslsta'1000-jsbpga'1000;number of pages to map

               ;JSBPGA is the address of the first page
               ; in the JSB area (the JSB itself)
               ;JSBLSTA is the addresss of the last page
               ; in the JSB area
               ;the first N locations in the JSB (where
               ; N is the number of pages in the JSB
               ; area) contain pointers to the JSB
               ; area pages

call msetmp$x                    ;call the routine
<>

mretn$g                          ;return to user mode
<>                                 to see mapped pages


jobmap[   224000,,2152           ;look at first few locations
JSBPGA+1[   124003,,7170           in JSB
   JSBPGA+2[   0
```

```
BPGA+3[   0


ilnen mljfn[   331501,,331503  ;pointer to extension
31503[   777777,,2   $t          string for first JFN
31504/   EXE


SRNAM[   3                          ;user name
SRNAM+1[   422371,,640620   st;DONAH
SRNAM+2/   UE
```

y GET'ing MONITR.EXE you can map the pages to their own ddresses and then be able to use the defined symbols rather than etermining the offsets within a different page address. Since he pages have been mapped to the User's Address space rather han to the Monitor's it is not necessary to unmap them when one. A RESET will get rid of them.

.8.4 EDDT

h you are in EDDT, timesharing ceases. Only the EDDT process an run.

DDT must be locked in memory. This is accomplished by one of he following:

- Bringing the monitor up with EDDT and setting a flag to request that EDDT be kept locked in memory.
- Calling the routine LCKINI from MDDT; this routine locks EDDT in memory and must be called BEFORE entering EDDT.

.8.4.1 Debugging Switches -

he locations described in this section can contain flags that re useful for debugging. The flags must be set before normal onitor startup; i.e. from EDDT startup. The default setting or each of the locations is zero.

EDDTF	1	Keep EDDT in core when system comes up
DBUGSW	0 or 1	Stop on BUGHLTs
	2	Write-enable swappable monitor, start up Sysjob using the file SYSJOB.DEBUG for the Sysjob commands and stop on BUGHLTs.
DCHKSW	0	Do not stop on BUGCHKs

DINFSW 0 Do not stop on BUGINFs

9.8.4.2 Loading The Monitor With EDDT -

Observe the following steps to load the monitor with EDDT locked in memory:

Load the monitor and start it at location 141 (EDDT startup).

Set the EDDTF flag to 1; this keeps EDDT locked in memory. The DBUGSW switch can also be set at this time if desired.

Set any desired breakpoints in the resident monitor. Only the resident monitor is loaded; therefore you cannot access any part of the swappable monitor. The breakpoint which is usually set at this point is GOTSWM. When the montiro reaches GOTSWM, the swappable monitor is already loaded.

Start the monitor at 147. This is the normal startup address. Since EDDTF is set, EDDT remains in memory.

When it reaches the breakpoint at GOTSWM, you may wish to do one or more of the following: remove the breakpoint, write-enable the swappable monitor (Call SWPMWE), lock the swappable monitor (Call SWPMLK), set breakpoints, continue the monitor ($P).

Whenever you hit an EDDT breakpoint, timesharing ceases until you continue from the breakpoint.

9.8.4.3 Page Faulting In EDDT -

If you reference a page that is swapped out while you are in EDDT, the page is NOT fauled in; EDDT types ? in this case. EDDT does not change the state of the system. You can cause a page to be brought in by typing SKIP LOC$X where LOC is some location in the page you wish to have swapped in.

9.8.4.4 How To Re-enter EDDT -

Once EDDT has been locked in memory, you may re-enter by doing the following: enter MDDT; type EDDT$G. If the system hangs, and you wish to enter EDDT: type control backslash to get into the command parser; examine the PC to see where the system is hung; start the monitor at location 140. If this fails, ask to

start at 141 to force a Reset and then enter EDDT.

9.8.4.5 To Leave EDDT -

To exit from EDDT (unless in a breakpoint), type MDDTSG to reenter MDDT; type MRETNSG to return from the Jsys.

9.8.5 Sending TOPS-20 Crash Dumps In To Be Analyzed

When sending your crashes in for analysis, here are some hints which will make the analysis easier:

> Put a copy of the dump AND a copy of the monitor that was running at the time of the crash on a magtape. The tape should NOT be in DUMPER interchange format. Use a density of 1600bpi. Put a second copy of the dump and monitor on the tape to minimize the possibility of read problems.
>
> Use the DUMPER Print command to make a hard copy of the savesets and file names on the tpae, and send it with the tape.
>
> Include the CTY output at the time of the crash.

CRASH DUMPS

ach time there is a BUGHLT there is an automatic dumping of the ystem core image into PS:<SYSTEM>DUMP.EXE. If there is sufficient oom on the DSK tne data that was previously in DUMP.EXE will be opied into DUMP.CPY by SETSPD after the system is reloaded. DUMP.CPY oes not get deleted and you may find several generations of DUMP.CPY.

In the case you have set no auto reload you can dump the crash by and by typing /D to the system BOOT> prompt. You can get into BOOT f you are reloading the system by bringing the system up from the witch registers rather than hitting <ENABLE> <DISK> on the console. ee the Operators Guide for a discussion of the meaning of the various witcnes on the DEC-20.

9-Sep-80

CRASH ANALYSIS

First when analyzing software or software/hardware problems be sure you have the proper tools:

1. Internals reference materials (tables and flowcharts).
2. A full copy of the current release microfiche MONITOR and EXEC.
3. A MONITOR CALLS REFERENCE MANUAL.
4. A SYSERR manual.
5. A listing of the SYSERR log, especially if hardware is suspected.
6. A CTY output for BUGHLTs and BUGINFs or other problem indications, or an accurate reproduction of this information.
7. Any other manuals you may need for reference such as the proper version Installation Guide, Operators Guide, System Managers Guide, etc.
8. A TOPS-20.BWR file.

You will need listings of the latest versions of monitor modules in case the microfiche are not up to date. FILDDT is on the customers distribution tape.

Be sure you have analysed the SYSERR log. Be sure, also, that you have looked up the BUGHLT and/or BUGCHKs in question in the listings (microfiche) and have at least read the comments around them. Probably tracing down how it got called is a good idea. If you happen to be without a GLOB (provided on microfiche) you can find the BUGHLT tag of interest in the monitor as follows:

```
$GET <SYSTEM>MONITR.EXE
$ST 140
DDT
ILPP3?                    ; BUGHLT of interest followed by "?"
PAGEM G                   ; it is defined in PAGEM and is global
```

Some other useful bits of information. There is a GLOB listing provided in the microfiche which contains a list of all the global symbols in the monitor. Most of the symbols are defined in the module STG.MAC. If you don't know a tag name but want to look at the storage for DTEs, say, look through STG. STG also contains some small portion

f code mostly to do with restart, start, auto reload, dispatches for I channels and A few scheduler tests. STG stands for storage. Note hat some stuff may be defined in PROLOG, and of course lots of stuff s defined throughout the monitor. You may also want to get a listing f MACSYM to be able to understand the macros you see while reading he monitor listings; MONSYM is also useful at times. Be sure you now how PARAMS has been changed in case it has. See BUILD.MEM on the istribution tapes for the currently distributed information on what o do to change various system parameters in PARAM0.MAC. Be sure that ou know about any variables that the site may have changed in STG as ell.

EXAMINING THE MONITOR

Debugging a complex, multi-process software system is largely a matter of absorbing sufficient knowledge, experience and folklore about the particular system with a considerable element of personal preference, or 'taste' also involved. This document is a cursory description of features built into the system to aid debugging, and such folklore as can be described in written English.

There are four different versions of DDT that may be used to examine the monitor. Each is used for a different purpose and has special capabilites. The versions of DDT are:

1. UDDT (user DDT) used to examine or modify the MONITR.EXE file.

2. MDDT (monitor DDT) used to examine or modify the running monitor under timesharing.

3. EDDT (exec DDT) used to examine or modify the running monitor from the CTY in a stand-alone mode.

4. FILDDT used to examine dumps.

All the DDT's are versions of TOPS-20 DDT documented in the TOPS-20 DDT manual, and have all of the features described in the manual. See also the document DDT41.MEM.

The use of all four versions of the DDT's is the same and will be described latter, however, each version is started differently.

D_;

To use UDDT to modify your MONITR.EXE file on system, you must ive the following EXEC commands:

```
@GET <SYSTEM>MONITR.EXE
@START 140        or on Release 4 systems, @DDT
```

his causes EDDT to start in user mode. This is the same DDT that is sed when examining any program. You may now look at or change any art of the monitor. If you make changes to the monitor and want to ave it, you should get back to the EXEC by typing ^Z. Then you may ave the monitor.

You will probably have to be enabled in order to save the monitor ack in <SYSTEM>. This is the safest, best, and recommended method of utting patches into the monitor.

MDDT:

A version of DDT which runs in monitor space is available. It can examine and change the running monitor, and can breakpoint code running as a process but not at PI or scheduler level. When patching or breakpointing the swappable monitor, the normal write protection must be defeated, either by setting DBUGSW to 2 on startup, or calling SWPMWE. If you insert breakpoints with MDDT, remember monitor code is reentrant and shared so that the breakpoint could be hit by any other process in the system. In this event, the other process will most likely crash since it will be executing a JSR to a page full of zeros.

To use MDDT you must have WHEEL or OPERATOR capabilities. You first issue the EXEC command:

```
@ENABLE
$^EQUIT

        ; You are now in the mini-exec and receive a prompt
        ; of MX>. Now you give the "/" command:
MX>/
        ; You are now put into MDDT. To return to the EXEC
        ; you can issue a ^Z or a ^C which produces a
        ; message like "INTERRUPT AT 17372" and returns you
        ; to the mini-exec. If you type a ^P in MDDT you
        ; will get a message, "ABORT", and be returned to
        ; the mini-exec. If you once go into the mini-exec
        ; the CONTROL-P interrupt is enabled and typing this
        ; character will return you to the mini-exec. This
        ; is a good thing to use when debugging programs
        ; that do CONTROL-C trapping. From the mini-exec
        ; you may give either:
MX>S
        ; or
MX>E

        ; The S is filled out as START and the E as EXEC.
        ; both of these commands will return you to the
        ; EXEC. See the document EXEC-DEBUGGING.MEM for more
        ; about ^P and getting out of the EXEC to MX> and
        ; returning from MX> to either your copy of the EXEC
        ; or the system EXEC.

        ; You may also give the command:

MRETNSG

        ; From MDDT to return directly to the EXEC. While
        ; in MDDT you may examine any core location in the
        ; running monitor. You may also change any location
        ; in the resident monitor (done frequently by
        ; accident). If you wish to change any of the
```

```
                ; locations in the swappable  monitor you must  give
                ; the command:

        CALL SWPMWE$X

                ; To write enable the monitor.  After you have  made
                ; your changes you must give the command:

        CALL SWPMWP$X

                ; to write protect the monitor again.
```

MDDT may also be entered from process level via JSYS:

```
    JSYS 777$X
        or
    MDDT%$X ; will enter MDDT from the context of the current process
```

If you wish to examine the system from the EXECs inferior fork
onitor context:

```
    @ENA
    $SDDT
    DDT

    JSYS 777$X
    MDDT
```

) return to user context:

```
    MRETN$G
```

;e SETMPG to map pages to this context:

page 677 has been traditionally used for this;
but any unused page may be used. To make sure that the page
is currently unused type:

```
    ADDRESS/   ?     ; the question mark from DDT indicates that the
                     ; page is nonexistent.
```

when the destination page has been found, set up AC2 as:

```
    AC2/ ACCESS,,677000
```

If the page has its own SPT slot:

```
    AC1/SPT INDEX
```

: the source page does not have its own SPT slot, it will belong to
.ther a file or process page table. It will be represented as an
ıdex into this page table:

```
    AC1/ SPT INDEX OF PAGE TABLE,,INDEX INTO PAGE TABLE
```

```
Access = read or/and or write access
Read/Write access = 140000 in LH
```

Therefore, to map a page, call with either:

```
AC1/SPT INDEX OF PAGE
AC2/140000,,677000

        or

AC1/SPT INDEX OF PAGETABLE,,INDEX INTO PAGE TABLE
AC2/140000,,677000

AND SAY:

CALL SETMPG$X
```

The page will then be mapped to page 677. In examining locations 677000-677777, you will be looking at the contents of the page.

If you desire to map another page into this slot, merely call SETMPG again with arguments for the new page. You need not first un-map the old page. However, when you are finished, page 677 should be un-mapped in the following manner:

```
AC1/0
AC2/ACCESS,,677000
CALL SETMPG$X
```

WARNING:

Calling SETMPG incorrectly can crash the system. Be CAREFUL! Do not use SETMPG on a time sharing system if a crash will cause bad feelings.

NOTE

Not to be confused with ^EEDDT command to get into UDDT used with the command processor.

To get into EDDT you must bring the system up using the witch-register. See the DECSYSTEM-20 Operators Guide for a iscussion of switches. Go through the KLINIT dialog and when you get he prompt BOOT>, respond with:

```
BOOT>/L
BOOT>/G141
```

he "/L" command causes the monitor to be loaded, but not started. he "/G141" starts the monitor at location 141, which is a jump to DDT. You can use EDDT like UDDT under timesharing on the MONITR.EXE ile by giving the following commands:

```
$GET <SYSTEM>MONITR.EXE
$START 140
```

DDT is linked into the monitor and is always there. You may also get o EDDT from MDDT by issuing the following:

```
EDDT$G
```

rom MDDT. This stops timesharing. To resume timesharing and /or get ack to MDDT give the command:

```
MDDT$G                  ; back to MDDT
MRETN$G                 ; back to normal timesharing
```

Breakpoints may be inserted in the resident monitor with EDDT, ut not in the swappable monitor in general, because its pages may be wapped out and be unavailable to EDDT. You can bring them in by yping:

```
SKIP LOC$X              ; where LOC is some address not in core
```

There are some locations in the monitor that are very useful when using EDDT for debugging. They must be set before going on to start the monitor.

They are:

EDDTF	1	keep EDDT in core when system comes up
	0	delete DDT when system comes up (default)
DBUGSW	0	do not stop on BUGHLTs, crash and reload
	1	stop on BUGHLTs (hit EDDT breakpoint)
	2	write enable swappable monitor, do not start up SYSJOB, and stop on BUGHLTs, Also it dosn't run CHECKD automatically on startup.
DCHKSW	0	do not stop on BUGCHKs (default)
	1	stop on BUGCHKs (hit EDDT breakpoint)
DINFSW	0	do not stop on BUGINFs (default)
	1	stop on BUGINFs (hit EDDT breakpoint)

In addition the symbol GOTSWM appears in the code just after the swappable monitor is loaded. So, if you want to debug the swappable part of the monitor you must put a breakpoint at GOTSWM (to get swappable part in core) by,

```
GOTSWM$B
```

Then start the MONITOR by,

```
147$G

CALL SWPMLK$X
```

CALL SWPMLK is used to lock swappable monitor in core for debugging. You must have more than 96k of core to give this command since the resident and swappable monitor are larger than 96k. To start up the monitor after you have gone into EDDT and set up your breakpoints (remember the last two are used for BUGHLT and BUGCHK) give the command:

```
147$G
```

or

```
SYSGO1$G
```

If you are in EDDT and DBUGSW is not 2, that is the monitor is write protected, you can use the routines SWPMWE and SWPMWP to write enable and write protect the monitor. CALL SWPMWE$X in DDT.

I[illegible]DT:

ILDDT is distributed on the customer software tape.

ne following is an chewed-up FILDDT.HLP file.

ET(FILE) FILE-SPEC

oads a file for DDT to examine. If you are looking at a monitor dump ou must load DUMP.CPY explicitly. FILDDT looks for MUMBLE.EXE not JMBLE.CPY that is DUMP<ESC> will tell you that there is no such file will load DUMP.EXE. When looking at a dump and you wish to load ne symbols you must first issue the load command followed by the get ommand. Be sure that the file from which you get the symbols is the ime version as the dump. Be sure, also that the monitor that was imped is the same monitor you use for symbols. That is don't get ONMED symbols to use with MONBCH etc.

OAD (SYMBOLS FROM) FILE SPEC

oads specified file and builds internal symbol table. This must be ne first command to FILDDT before "GET" when looking at a dump. You ill most probably use <SYSTEM>MONITR.EXE which would have been the onitor running at the time of the dump.

EXIT (FROM FILDDT)

returns to command level. You then may type a save command if a load ommand was just done to preload symbols. You will get a version of LDDT that has the symbols you just loaded in it so you no longer ed to "LOAD" symbols. You now have a monitor specific FILDDT, which s common practice for TOPS-10, but is not generally done for PS-20.

LP

pes something like this text.

ABLE PATCHING

lows writing on an existing file specified by a GET.

ABLE DATA-FILE

sumes file is raw binary (i.e. no ACs, and not an EXE file).

T[illegible]EATURES:

EP$OU Sets monitor context for FILDDT mapping. EP is a symbol which is equal to the page number of the EPT. (Rel 4)

<CTRL/E> Returns to FILDDT command level.

TRACKING DOWN UNMAPPED ADDRESSES:

The resident monitor may be looked at without any difficulties, but the swappable monitor may not be in core at the time of the dump. If the value of the symbol is in the swappable monitor you must sometimes go through the monitor map to find where the location really is. The location MONCOR contains the number of pages of resident monitor and the location SWPCP0 contains the first page of real core for swapping. So if the value of the symbol is greater than contents of MONCOR times 1000 then it is in swappable monitor.

If the page of the swappable monitor you want to look at is in core it will probably not be in core in the location that it's address refer to since the dump is of core and relocation of pages does not happen. To find where a symbol really is in the dump, first type the symbol followed by an "=". DDT will respond with the value of this symbol. The value of the symbol can be divided into two, three octal digit, fields. The high order three digits are the page number and the low order three digits are the offset into the page.

If the value of the symbol is 324621 the high order three digits, 324, are the page number and the low order three digits, 621, are the offset into the page. To find the location of the page in question in the dump you must look at the monitor map indexed by the page number. For example:

MMAP+324/

would give you the monitor map word for page 324. This word contains some protection bits for the page and the address of the page when the dump was taken.

The page may have been in core, on the swapping area or on the disk at the time of the dump.

If bits 14-17 in the monitor map word are non-zero the page was on the swapping area or disk and is no longer available.

If bits 14-17 are zero then the page was in core, and the right half of the word contains the page number in the dump of the page you are looking for (the dump program overwrites the last several pages of memory, the dump therefore does not contain these last pages.)

If the page was in core the new address of the symbol you are looking for can be found by using the page number from the monitor map word and appending the offset into the page to it. For example if MMAP+324 contains 104000,,256; then the new address of our symbol would be 256621.

l ddress in the swappable monitor must be resolved in this manner. n addition address of 600000 and above are in the JSB or PSB (PSB is age 777) and must be resolved by finding the page containing the JSB r PSB of the process that was running when the dump occured. There re some locations and tables in the monitor that make this easy:

NAME	INDEX	DESCRIPTION
FORKX	none	Number of the fork that was running at the time of the dump, -1 if in the scheduler.
JOBNO	In PSB	Job number to which current fork belongs.
FKJOB	Fork #	Job number,,SPT index of JSB
JOBDIR	Job #	logged in directory number
JOBPT	Job #	controlling TTY number,,top fork number
FKSTAT	Fork #	test data,,address of fork wait routine
FKPGS	Fork #	SPT index of page table,,SPT index of PSB

PT indexes are indexes into a share pointer table starting at SPT. o find the PSB of fork 20, you first look at FKPGS+20, If this ocation contains 425,,426, the word at SPT+426 is the pointer to the SB. This pointer can point to disk, swap area, or a page in the ump. If bits 14-17 are zero it is a pointer to a page in the dump nd the right half of the SPT word is the page number of the PSB in ne dump.

n you look at a dump, you should first try to find why the dump ccured by looking at the location BUGHLT. If BUGHLT is zero then you ould check the CTY log to find out why the dump was taken and for nformation like the PC at the time of the dump and the status of the l system. If BUGHLT is non-zero it is the address of where the JGHLT was issued. You should look up the BUGHLT in BUGSTRINGS.TXT or JGS.MAC to find additional information about the BUGHLT. If at this oint you are not sure as to why the BUGHLT occured, you will have to ook at the listings for more information. A copy of BUGSTRINGS.TXT s in Appendix A of the Operators manual. You can find the location f the call to the BUGHLT by typing the BUGHLT tag to DDT followed by "?". DDT will tell which monitor module the BUGHLT is in and you an go to your microfiche and read all about the conditions recipitating the BUGHLT.

ext if necessary look at FORKX. If it contains a -1 the scheduler as running; otherwise it is the number of the fork that was running nen the crash occurred. The registers are saved at BUGACS on a JGHLT, but if BUGACS+17 contains something,,BUGPDL+n, then the egisters are invalid and you must go to the SYSERR buffer to get the ood registers. This is done by adding to the right half of the fSERR buffer pointer, SEBQOU, the offset into the buffer for the eading and ACs, SEBDAT+BG%ACS. This value points to a 16 block of ords containing the users ACs. You may have to chain down more than ne queued-up SYSERR entry to get to the BUGHLT block.

> Do not forget to get a print out of the SYSERR log which will give you and the field service representative much of the information you can get out of the dump. The SYSERR output is much easier to examine, however, clearly you cannot get as much info as you can from a dump.

Some other locations in the PSB of interest are:

LOCATION	DESCRIPTION
UAC	User's ACs when he did his last JSYS.
PAC	monitors ACs
PPC	processors PC
UPDL	users pushdown stack while in a JSYS
NSKED	0 = ok to run scheduler
	>0 = cannot run scheduler
INTDF	-1 = ok to receive software interrupts
	>= 0 , cannot receive software interrupts

It may be useful to know the status of a fork when it is hung or you are unsure of its status. This can be determined by looking at FKSTAT indexed by the fork number. The right half of this location is the address of a test routine and the left half is data to be tested. For example if FKSTAT+12 contains 23,,FKWAT, then fork 12 is waiting for fork 23 to complete. FKWAT is a routine that waits for another fork to complete and its data (the left half of the word) is the number of the fork it is waiting for. There are many different wait routines and you will have to look at the code to see what individual ones are waiting for. There is a memo on scheduler tests which details most all of the scheduler tests in the monitor.

You can easily determine all of the forks associated with a job by giving the commands:

```
-1,,0$M
FKJOB<FKJOB+NFKS>N,,0$W
```

Where N is the job you are looking for. A fork structure can usually be determined by looking at the FKSTAT of the forks and seeing which forks are waiting on which forks. A FKSTAT of FKSKP indicates a fork is inactive.

You should refer to STG.MAC for other fork and job tables and other locations in the PSB and JSB of interest. All of the above locations can be examined with MDDT or EDDT while the monitor is running. Of course at these times you do not have to go through MMAP and the PSB and JSB that are in core are your own.

There are two separate patch areas in the monitor (FFF and SWPF). FFF is the resident patch area and SWPF is the swapable patch area. These two symbols should be updated to point to the next free location in

h patch area when a patch is inserted. PAT.. is defined to be qual to SWPF. By convention, all distributed patches are applied at FF. This serves the purposes of reducing confusion, always working ntil the patch area is exhausted, and leaving patches always present n a dump for the cases where that is important.

here are several general purpose routines that can be used to look at he the monitor while it is running. These routines should be used ith caution since it is certainly possible to crash the monitor by sing them incorrectly. Two of the more general routines are MAPDIR, or mapping a directory into core, and SETMPG for mapping pages someone elses PSB or JSB) into core. You will have to look at the isting for the exact use of these and other general routines. Beware f the precautions that should be taken when using them. You can find he module they are located in by looking in the GLOB listing which is cross reference listing of all the global symbols in the monitor. ou get a GLOB listing in your microfiche.

BUGHLT, BUGCHK, BUGINF

The monitor contains a considerable number of internal redundancy checks which generally serve to prevent unexpected hardware or software failures from cascading into severely destructive reactions. Also, by detecting failures early, they tend to expedite the correction of errors.

There are two failure routines, BUGCHK and BUGHLT for lesser and greater severity of failures. Calls to them with JSR are included in code by use of a macro which records the locations and a text string describing the failure. The general form is:

```
BUG (TYPE,NAME,<STRING>)
```

Where type is HLT or CHK, and string describes the cause.

For example,

```
BUG(HLT,SKDPFL,<PAGE FAULT FROM SCHEDULER CONTEXT>)
```

The strings are constructed during loading and are dumped into a file. The BUGSTRINGS.TXT file will produce an ordered listing of the bug messages for operator or programmer use.

BUGCHK is used where the inconsistency detected is probably not fatal to the system or to the job being run, or which can probably be corrected automatically.

Typical is the sequence in MRETN in the SCHED module.

```
AOSGE INTDF
BUG(HLT,IDFOD2,<AT MRETN - INTDF OVERLY DECREMENTED>)
```

This BUGCHK is included strictly as a debugging aid. Detection of a failure takes no corrective action. This situation usually results from executing one or more excessive OKINT operations (not balanced by a preceding NOINT). It is considered a problem because a NOINT executed when INTDF has been overly decremented will not inhibit interrupts and will not protect code changing sensitive data.

BUGHLT is used where the failure detected is likely to preclude further proper operation of the system or file storage might be jeopardized by attempted further operation. For example, the following appears in the SCHED module:

```
MOVE 1,TODCLK    ;CURRENT TIME
CAML 1,CHKTIM    ;TIME AT WHICH JOB0 OVERDUE
BUG(HLT,JONRUN,<JOB 0 NOT RUN FOR TOO LONG>)
```

This check accomplishes two things:

1. A function of JOB0 is to periodically update the disk version of bittables, file directories and other files. Absence of this function would make the system vulnerable to considerable loss of information on a crash which loses core and swapping storage. JOB 0 protects itself against various types of malfunction, this BUGHLT detects any failure resulting in a hangup.

2. Detects if the entire system has become hung due to failure of the swapping device or some such event, on the basis that if JOB 0 isn't running, nobody's running.

NOTE

For Release 4, the program form the BUGxxx calls takes has been modified, and the new file BUGS.MAC contains hopefully useful information on each of the BUGxxx calls in one place. This should be considered a required debugging file.

3UGSW:

monitor cell, DBUGSW, controls the behavior of BUGHLT and BUGCHK ıen they are called. DBUGSW is set according to whether the system ; attended by system programmers.

: C(DBUGSW)=0, the system is not attended by system programmers, so ll automatic crash handling is invoked. BUGCHK will return +1 ımediately, appearing effectively as NOP. BUGHLT will, if called :om the scheduler or at PI level, invoke a total reload from the disk ıd a restart of the system. The BUGCHK/INF output will appear on the TY and in the SYSERR log when JOB0 gets around to them.

: the system continues to run or is restarted properly, the location : the bug (saved over a reload) and its message will be reported on ıe CTY.

: C(DBUGSW).NEQ.0, the system is attended, and one of the EDDT 'eakpoints will be hit. This allows the programmer to look for the ıg and/or possibly correct the difficulty and proceed. There are two :fined non-zero settings of DBUGSW, 1 and 2, which have the following .stinction.

C(DBUGSW) = 1

Operation is the same as with 0 except for breakpoint action. In particular the swappable monitor is write protected and SYSJOB is started at startup as described.

C(DBUGSW) = 2

Is used for actual system debugging, the swappable monitor is not write protected so it may conveniently be patched or breakpointed, and the SYSJOB operation is not started to save time.

BUGCHK and BUGHLT procedures are the same as for 1.

The following is a summary of DBUGSW settings:

MEANING	0 Unattended	1 Attended	2 Debugging
BUGCHK action	NOP	Hit Breakpoint	Hit Breakpoint
BUGHLT action	Crash System	Hit Breakpoint	Hit Breakpoint
SWPMON write protect?	Yes	Yes	No
CHECKD on startup	Yes	Yes	No

Other console functions:

In addition to EDDT, several other entry points are defined as absolute addresses. The machine may be started at these as appropriate.

```
140     JRST EDDT               ; go to EDDT
141     JRST SYSDDT             ; reset and go to EDDT
142     JRST EDDT               ; copy of EDDT address
143     JRST SYSLOD             ; initialize file system
144       0
145     JRST SYSRST             ; restart
146     JRST SYSGOX             ; reload and start
147     JRST SYSGO1             ; start
```

The soft restart (address 145, EVRST) restarts all I/O devices, but leaves the system tables intact. If it is successful, all jobs and all (or all but 1) process will continue in their previous state without interruption. This may be used if an I/O device has malfunctioned and not recovered properly. The total restart initializes core, swapping storage and all monitor tables.

A very limited set of control functions for debugging purposes has been built into the scheduler. To invoke a function, the appropriate bit or bits are set into location 20 via MDDT. The word is scanned from left to right (JFFO). The first 1 bit found will select the function.

BIT 0:

Causes scheduler to dismiss current process if any and stall (execute a JRST ,), with -1 in AC0. Useful to effect a clean manual transfer to EDDT. System may be resumed at SCHED0.

BIT 1:

Causes the job specified by data switch bits 18-35 to be run exclusively. Temporarily defeats JOB 0 not run BUGHLT.

BIT 2:

Forces running of JOB 0 backup function before halting the system.

BUG'TYP MACRO CHANGES FOR VERSION 4 OF TOPS-20

Version 4 of TOPS-20 will include some changes in the BUG code generation. The purpose of these changes is to generate a document describing the TOPS-20 BUGCHKs, BUGHLTs, and BUGINFs that are more descriptive than the previous BUGSTRINGS.TXT file.

The logistics of this change include moving the BUG definitions out of the monitor source listings and into a central source file. This source file will serve both as the definition file for the bugs and as documentation for the BUGS. This file is called BUGS.MAC and will be distributed to all sites on the distribution tape. These BUGS are still referenced in the source module where the bug is invoked but they are defined in BUGS.MAC.

This involves a modification to the old BUG macro and a new macro called DEFBUG. The BUG macro appears in the source modules and the DEFBUG macro appears in BUGS.MAC.

The format of the new BUG macro is as follows:

```
BUG (BUGNAM,<<x1,des1>,<x2,des2>...>)
```

This is placed in the monitor code where the BUG called BUGNAM is to occur. This macro executes a macro with name 'BUGNAM' which generates a XCT BUGNAM where the contents of BUGNAM is a JSR BUG'TYP. Following the location BUGNAM are the Accumulators to be printed (one AC per word) followed by SIXBIT/BUGNAM/. The Accumulators to be printed are defined with the DEFBUG macro while the locations specified in the BUG macro are for documentation only.

Accompanying this BUG macro is a DEFBUG macro which is placed in the file BUGS.MAC. This entry completely defines the BUG, including its type (BUGHLT, BUGCHK, or BUGINF) and documentation.

The format of the DEFBUG macro is:

```
DEFBUG (TYP,TAG,MOD,WORD,STR,LOCS,HELP)
```

For a description of the arguments to this macro see the SWSKIT article called BUGS.MEM.

In order to make listings (output from MACRO or CREF) more informative than before, the BUG macro will cause the statement of the short description displayed in the listing where the BUG macro is called. Also, the flavor of bug (INF, CHK, or HLT) and whether it's hardware or software related will be displayed in the listing. Hence the OVRDTA bug would appear in the listing as

```
BUG(OVRDTA)
;BUG Type:                hardware-related BUGINF
;BUG description:         PHYSIO - OVERDUE TRANSFER ABORTED
```

When fully documented, the BUGS.MAC file will be extremely useful or specialists. It will describe, in one convenient place, what the dditional data printed on the console is, what caused the bug, and hat the site or specialist should do if that particular bug occurs.

ere is a section of the current BUG definition/documentation for the UG GIVTMR from BUGS.MAC:

```
EFBUG(INF,GIVTMR,JSYSA,SOFT,<GIVOK TIMEOUT>,<<T2,FUNC>>,<

ause:  The access control job has not responded with a GIVOK within
       the designated time period.

ction: If this consistently happens with the same function code, you
       should see if the processing of the function can be made
       faster.

       If there is no obvious function code pattern, you may need to
       increase the timeout period or rework the way in which the
       access control program operates.

a      FUNC - the GETOK function code


       >)
```

NF specifies the bug is a BUGINF. GIVTMR is the name of the bug. SYSA is the module that the bug would occur in. SOFT specifies that t is likely the bug is caused by a software bug. <GIVOK TIMEOUT> is ne bug string. <T2,FUNC> specifies the data that will be printed on ne operator's console. The initial spec called for the descriptor JNC to be included in the operator's message but at this time, this escriptor is just for source documentation.

ne blurbs following the initial line of the BUG definition attempt to escribe to the specialist, in a more detailed manner than the escription printed on the console, what it means when this bug occurs id what should be done first in order to resolve the situation. In is case the ACTION is to examine the GETOK routine which is executed or the additional data FUNC. This routine is getting hung up. ometimes, the ACTION will state to call the hot line or to submit an R. These descriptions will help the specialist be more informed out the bugs which may occur at one of their sites and save them the me of calling the hot line or searching through the source module r an idea of the problem.

TOPS-20 SCHEDULER TEST ROUTINES

The following is a tabulation of (hopefully) all of the scheduler tests used by the TOPS-20 monitor, time-frame approximately Release 3A. This includes ARPA and DECNET tests. This is the data one finds in the monitor table FKSTAT indexed by fork number for forks which have blocked and left the GOLST (i.e. LH(FKPT) contains WTLST). The format of the FKSTAT table words is TEST DATA,,TEST ROUTINE ADDRESS. The scheduler test routines are called periodically to determine if a process can be unblocked. This is indicated by a skip return from the scheduler test. A nonskip return is taken if the process cannot yet be unblocked.

When examining the monitor because of a hung job or fork, the FKSTAT table can often reveal the reason the fork is hung, and this sometimes even allows corrective action to be taken.

The table below gives routine name, what you should expect to see in the FKSTAT table, and the module in which the scheduler test is defined, followed finally by a short description of what the particular condition is which is being tested.

SCHEDULER TESTS

TEST	CONTENTS OF T1 AT TIME OF SCHEDULER CALL	DEFINED
BALTST	[CONNECTION #,,BALTST] Wait for network bit allocation.	[NETWRK]
BATTST	[UNIT #,,BATTST] Wait for US.BLK, the lock bit for the BAT blocks on the unit, in the UDB to be zero.	[DSKALC]
BLOCKM	[TIME,,BLOCKM] Wait for TIME in BLOCKM format which is the low order 17 bits of the desired future time to be compared against a suitably masked TODCLK.	[SCHED]
BLOCKT	[TIME,,BLOCKT] Wait for TIME in BLOCKT format which is a value that is shifted left 10 bits and compared against a suitably masked TODCLK, providing a longer delay than BLOCKM, but less precision.	[SCHED]
BLOCKW	[TIME,,BLOCKW] Wait for TIME in BLOCKW format (same as BLOCKM).	[SCHED]
CDRBLK	[UNIT NUMBER,,CDRBLK] Wait for card-reader offline, or not waiting for a card.	[CDRSRV]

:H_DK [ADDRESS,,CHKLOK] [NSPSRV]
Wait for NSP block lock at address to free.

:OFTST [TIME,,COFTST] [MEXEC]
Wait for job in FKJOBN to be attached or time in BLOCKT form to elapse.

)BWAIT [DTE #,,DBWAIT] [DTESRV]
Wait for the TO-10 doorbell from the given DTE.

)GLTST [0,,DGLTST] [DIAG]
Wait for DIAGLK lock to be free.

)GUIDL [UDB ADDRESS,,DGUIDL] [DIAG]
Wait for the unit to show as idle in the UDB.

)GUTST [UDB ADDRESS,,DGUTST] [DIAG]
Wait for the maintenance bit to set in the UDB.

)ISET [ADDRESS,,DISET] [SCHED]
Wait for contents of ADDRESS to be zero.

)ISGET [ADDRESS ,,DISGET] [SCHED]
Wait for contents of ADDRESS to be positive.

IS_T [ADDRESS,,DISGT] [SCHED]
Wait for contents of ADDRESS to be greater than zero.

ISLT [ADDRESS,,DISLT] [SCHED]
Wait for contents of address to be less than zero.

ISNT [ADDRESS,,DISNT] [SCHED]
Wait for contents of ADDRESS to be non-zero.

MPTST [COUNT,,DMPTST] [IO]
Wait for COUNT to be less than DMPCNT to indicate dump mode buffers freed.

SKRT [PAGE #,,DSKRT] [PAGEM]
Wait for CSTAGE for PAGE # to not be PSRIP, meaning disk read completed.

WRTST [PAGE #,,DWRTST] [PAGEM]
Wait for DRWBIT to clear in CST3(PAGE #), meaning write completed.

)QTST [FORK #,,ENQTST] [ENQ]
Wait for the lock on ENFKTB+FORK #.

:B_ [ADDRESS OF FE UDB,,FEBWT] [FESRV]
Wait for EOF or input bytes available from FE. Wake also on invalid assignment.

FEDOBE	[ADDRESS OF FE UDB,,FEDOBE] Wait for output buffer empty and all bytes are acknowledged by the FE. Wake also if not a valid assignment.	[FESRV]
FEFULL	[ADDRESS OF FE UDB,,FEFULL] Wait for the current count of output bytes to be less than the count of bytes in the interrupt buffer. Wake also on invalid assignment.	[FESRV]
FORCTM	[SUPERIOR FORK INDEX,,FORCTM] Identifiable wait forever, forced termination.	[SCHED]
FRZWT	[PREVIOUS TEST,,FRZWT] Identifiable wait forever, frozen fork.	[FORK]
HALTT	[SUPERIOR FORK INDEX,,HALTT] Identifiable wait forever for halted fork.	[SCHED]
HIBERT	[TIME,,HIBERT] Wait for TIME in BLOCKT format.	[SCHED]
HUPTST	[<0:9>TIME<10:17>HOST #,,HUPTST] Wait for IMPHRT bit set for host or time out in BLOCKW form.	[NETWRK]
IDVTST	[0,,IDVTST] Wait for the lock on IDVLCK to free, lock it.	[IMPDV]
IMPBPT	[0,,IMPBPT] Wait for IMPFLG nonzero, or IBPTIM timer to run out, or IDVLCK lock free and output scan needed for the IMP.	[IMPDV]
JB0TST	[TIME,,JB0TST] Wait for JB0FLG set nonzero for explicit request or time in BLOCKT form to elapse.	[MEXEC]
JRET	[0,,JRET] Wait forever, interruptible.	[SCHED]
JSKP	[0,,JSKP] Unconditional skip used to schedule immediately.	[SCHED]
JTQWT	[0,,JTQWT] Wait for JSYS trap queue.	[SCHED]
LCKTSS	[ADDRESS,,LCKTSS] Wait for lock at ADDRESS to unlock, lock it.	[IO]
LKDSPT	[0,,LKDSPT] Wait for room in LDTAB table of directories currently locked.	[STG]

,K_ST	[INDEX INTO LDTAB,,LKDTST] Wait for bit in LCKDBT to clear, indicating directory unlocked.	[STG]
,ODWAT	[ADDRESS OF STATUS WORD,,LODWAT] Wait for flag LP%LHC to set in the addressed word, indicating loading has completed of the VFU or RAM file.	[LINEPR]
PTDIS	[UNIT ADDRESS,,LPTDIS] Wait for an error condition on the addressed unit, or for all buffers cleared and no bytes still in the front-end, before finishing close operation on the device.	[LINEPR]
TARWT	[IORB ADDRESS,,MTARWT] Wait for IRBFA in the IORB to indicate that this IORB is no longer active.	[MAGTAP]
TAWAT	[UNIT #,,MTAWAT] Wait for all outstanding IORBs for unit to be finished.	[MAGTAP]
TDWT1	[UNIT #,,MTDWT1] Wait for the count of outstanding requests on the unit to go to one.	[MAGTAP]
CPLKT	[0,,NCPLKT] Wait for lock NCPLCK to free, lock it.	[NETWRK]
[CTST	[0,,NICTST] Wait for SUMNR less than or equal to MAXNR or only one fork in BALSET.	[PAGEM]
,TTST	[<0:8>CONNECTION #<9:17>STATE,,NOTTST] Wait for connection to leave state.	[NETWRK]
;PTST	[0,,NSPTST] Wait for KDPFLG nonzero, indicating KMC11 wants service, or MSGQ nonzero, indicating messages to process.	[NSPSRV]
/TNTT	[<0:8>OPTION #,<9:17>LINE #,,NVTNTT] Wait for completed NVT negotiation.	[TTNTDV]
'NLKT	[OFN,,OFNLKT] Wait for OFN unlocked--SPTLKB zero in SPTH(OFN).	[PAGEM]
:DWAT	[FORK #,,PIDWAT] Wait for bit for fork in PDFKTB to set.	[IPCF]
:B_ST	[0,,SEBTST] Wait for SECHKF to go nonzero before starting Job 0 task to write queued SYSERR entries.	[SYSERR]

SEEALL	[0,,SEEALL] Waits for SNDALL to go to zero, indicating the send-all buffer available.	[TTYSRV]
SPCTST	[0,,SPCTST] Wait for a node.	[DTESRV]
TST	[0,,SPMTST] Wait for page in SPMTPG to be on SPMQ or the time SPMTIM to expire.	[PAGEM]
ST	[0,,SQLTST] Wait for the special queues lock SQLCK and lock it.	[IMPDV]
ST	[SDB ADDRESS OF STRUCTURE,,STRTST] Wait for the structure lock to be free.	[MSTR]
AT	[ADDRESS OF STATUS WORD,,STSWAT] Wait for flag CD%SHA to come on in the addressed word, indicating that cardreader status has arrived.	[CDRSRV]
AT	[ADDRESS OF STATUS WORD,,STSWAT] Wait for flag LP%SHA to set in the addressed word, indicating that printer status has arrived.	[LINEPR]
KT	[FORK #,,SUSFKT] Wait for fork to be on WTLST in either SUSWT OR FRZWT.	[FORK]
T	[PAGE #,,SWPRT] Wait for CSTAGE for PAGE # to not be PSRIP, meaning swap read completed.	[PAGEM]
T	[0,,SWPWTT] Wait for NRPLQ nonzero. Increment CGFLG each time test is unsuccessful.	[PAGEM]
T	[FORK #,,TCIPIT] Waits for no interrupts pending for FORK #.	[TTYSRV]
T	[LINE #,,TCITST] Wait for line inactive, no fork in input wait, or input buffer non-empty.	[TTYSRV]
TCOTST	[LINE #,,TCOTST] Wait for line inactive, or output buffer not too full to add a character to it.	[TTYSRV]
TRMTS1	[0,,TRMTS1] Identifiable wait forever for inferior fork termination.	[FORK]

:R_ST	[FORK #,,TRMTST] Wait for FORK # to be on WTLST for either HALTT or FORCTM.	[FORK]
:RPOCT	[MINIMUM NRPLQ,,TRPOCT] Wait for NRLPQ to be above stated minimum or normal minimum. Increment CGFLG each time test is unsuccessful.	[PAGEM]
'SACT1	[LINE #,,TSACT1] Wait until line inactive, becoming active, or has a full length dynamic block assigned.	[TTYSRV]
'SACT2	[LINE #,,TSACT2] Wait for line available--inactive or fully active.	[TTYSRV]
'SACT3	[LINE #,,TSACT3] Wait for line inactive--dynamic data unlocked.	[TTYSRV]
'STSAL	[0,,TSTSAL] Wait for SALCNT to go to zero, indicating the send-all is finished for this buffer.	[TTYSRV]
TBUFW	[NUMBER,,TTBUFW] Wait for NUMBER of buffers.	[TTYSRV]
TIBET	[LINE #,,TTIBET] Wait for line inactive or input buffer empty.	[TTYSRV]
TOAV	[LINE #,,TTOAV] Wait for line inactive and output buffer not empty.	[TTYSRV]
TOBET	[LINE #,,TTOBET] Wait for line inactive or output buffer empty.	[TTYSRV]
DITST	[0,,UDITST] Wait for at least two free IORBs on UIOLST.	[PHYSIO]
DWDON	[IORB ADDRESS,,UDWDON] Wait for IS.DON to set in IRBSTS for this IORB.	[PHYSIO]
PBGT	[CONNECTION INDEX,,UPBGT] Wait for LTDF connection done flag to set, or output buffers to appear.	[IMPDV]
SGWAT	[0,,USGWAT] Wait for lock on queued USAGE blocks to free.	[JSYSA]
VBWAT	[UNIT #,,VVBWAT] Wait for the MDA to reset TPVV handling EOV.	[TAPE]
ATTST	[<0:8>CONNECTION #<9:17>STATE,,WATTST]	[NETWRK]

Wait for connection to be in state.

WTFKT	[FORK #,,WTFKT] Wait for fork to be on WTLST.	[FORK]
WTSPTT	[PAGE #,,WTSPTT] Wait for share count on PAGE # to go to 1.	[SCHED]

MAPPING DIRECTORIES IN MDDT

Release 3 of TOPS-20 can take advantage of the extended ddressing features of the model B processor. Some of its data has een reorganized and moved into non-zero sections of the addressing pace. One of the things moved was directories. Directories are now apped into section 2, starting at the beginning of the section. Thus he old procedure of reading a user's directory in MDDT is no longer alid. This will describe how to map a directory correctly, for elease 2 and for releases 3, 3A, and 4.

The procedure for release 2 was the following. You first have to ind out the structure number and directory number for the directory o be mapped. You can use the TRANSL program to get the directory umber, or use the ^EPRINT command to list the directory information. s an example, suppose you want to find the directory and structure nformation for the directory SNARK:<DBELL>. You run TRANSL and btain the results:

```
TRANSL SNARK:<CURDS>
N  :<CURDS> (IS) SNARK:[4,117]
```

he "programmer number" obtained is the directory number, in octal. n this example, the directory number is 117. If the directory is in ad shape, and you can't run TRANSL or use ^EPRINT, you will have to ind out the directory number by looking at the output from a DLUSER r ULIST run, or from BUGCHK output.

To find the structure number, you have to work harder. If the tructure is mounted as PS:, its structure number is always 0. For tructures mounted other than PS:, you do the following. You get into DDT, and look at the table STRTAB. This table contains all of the ddresses of the structure data blocks in the system. The first word f each structure data block is the structure name in SIXBIT. So you earch the tables looking for the desired structure. The offset into he table STRTAB is then the structure number. For our example:

```
ENABLE
SDDT
DT
SYS 777$X
DDT
$6T
TRTAB/    ,8[    /    PS
TRTAB+1/        M^I    /    REL3
TRTAB+2/        M_%    /    SNARK
```

n the example above, you see that PS: is the first structure, ollowed by the structures REL3: and SNARK:. Since the offset into

STRTAB was 2 for SNARK:, the structure number you want is 2.

Knowing the structure number and the directory number, you can now map the directory and look at it. When the directory is mapped, location DIRORA will point to the area in the monitor you can find it at. This is currently the address 740000. To save typing, you can use the symbol DA, which has the value 740000 (none of the examples here uses this symbol however). To map the directory, you call the routine MAPDIR which is in the module DIRECT. It takes two arguments. The directory number goes in AC1, and the structure number goes in AC2. For our example, the output looks like:

```
DIRORA[   740000
740000/   ?

1!   117
2!   2
CALL MAPDIR$X
$$

740000[   400300,,100
```

The skip return from MAPDIR means you have successfully mapped the directory. You can now look at the whole directory by examining the proper locations. The number of pages that are mapped by MAPDIR is 30, which is the length of a directory, so the whole thing is available to look at. By examining or changing location 740000+N in core, you are examining or changing location N of the directory. When you are finished, you can just leave MDDT by jumping to MRETN or by typing ^C.

In release 3, however, when you examine location DIRORA after calling MAPDIR, it doesn't have to contain 740000. If it does, then your machine cannot support extended addressing and the monitor is running the same as release 2 did. In this case you can ignore the rest of this document. If your machine does have extended addressing, when you examine location DIRORA you will see the number 2,,0. This address is now in section 2 of the monitor, and MDDT cannot read the data there directly. If you look at the location 740000 after calling MAPDIR, it will still be unreadable, since the directory is no longer read in there. Those pages are now unused.

To be able to read the directory now, you have to tell the monitor to map in the pages where you can see them with MDDT. The first step is to examine the location DRMAP. This location is the section pointer for section 2, where the directories are mapped. This is a share-type pointer, which contains the OFN for the desired directory in the right half. This number is one of the arguments for the MSETMP routine. MSETMP takes the following arguments. AC1 contains the OFN in the left half, and the first page number to be mapped in the right half. AC2 contains flag bits in the left half, and the address where you want to map the pages in the right half.

AC2 contains the number of pages to be mapped. For mapping directories, you can use 740000 as the address, and you want to map 30 pages. You also want to set flag bits so that the directory can be changed. For the example, you do the following:

```
)RMAP[   224000,,147

!!   147,,0
!!   140000,,740000
!!   30
:ALL MSETMP$X
!
```

After the call to MSETMP, the directory is now mapped in 740000, and you can proceed as you used to in release 2. When you are finished with the directory, you should call MSETMP again to unmap the directory. This is done by supplying the same arguments as before, except that ac 1 contains zero. As an example:

```
.!   0
!!   140000,,740000
!!   30
:ALL MSETMP$X
!
```

Now you can simply ^C out of MDDT or jump to MRETN.

For Release 4 of TOPS-20, the various flavors of DDT have been trained to understand extended addresses, so the mapping contortions used for 3 and 3A are once again unnecessary. On extended machines one can reference section two directly as below:

```
IRORA[   2,,0

,,0[   400300,,100
```

Then done, you can still just ^C out or jump to MRETN.

An Easy Way to Examine the PSB and JSB of Another Job

There is an occasional need to look at the state in detail of another job on the system. A common reason for doing this is to find the cause and cure of a "hung job" which cannot be logged out. To find out what the job is doing you usually start by looking at the JSYS stack in the PSB. But you cannot examine such data easily because the fork data in the PSB and the job data in the JSB are not in the monitor's address space until the fork is run. If you try to look at the PSB or JSB using MDDT you will see the data for your own fork. To look at the data for another fork you must do what the monitor does, and that is to map it.

A procedure for doing the mapping of a PSB or JSB was given in the release 3 and 3A SWSKITs. You first find the SPT index of the PSB or JSB you want to map, then you call SETMPG or MSETMP to set up pointers to the data, and then you can examine it. But there are several problems in using that method, which are:

1. You have to find an empty set of pages in the monitor's address space which can be used for mapping.

2. There is not enough room to map all of the PSB and JSB. So if you want to examine many different things you have to do the mapping many times.

3. The routines SETMPG and MSETMP do no validity checking of their arguments. Thus if you feed them bad data the system will probably crash. So if you need to map things many times your chances are you will make a mistake once too often.

4. The addresses of the data are not correct. To look at PPC for example, you can't just examine location PPC (which would be for your own fork). You have to look in the page you are using for mapping. So every reference has to be offset by some constant.

5. When you are done looking at the fork, you can't simply leave MDDT. You have to call SETMPG or MSETMP again to unmap the data.

Since that documentation was written I have found a procedure hich is much easier. It eliminates almost all of the above problems. he procedure is this:

1. Do a "GET" of the file the monitor was loaded from, usually SYSTEM:MONITR.EXE.

2. Enter user mode DDT in the file you got, and then do a JSYS 777 to get into MDDT.

3. Find out the SPT indexes as before, and call MSETMP to map the PSB or JSB to the USER address space, in the correct place!!

4. Return from MDDT, and examine PSB and JSB locations directly, and see the correct data in the right place.

5. When you are done, just ^C and do a RESET.

The rest of this document will document step by step how the rocedure above is done, by using an example. Assume that we wish to xamine the state of fork 105, which belongs to job 21. We then e:

```
@ENABLE                                  !Get a copy of the monitor
$GET PS:<SYSTEM>MONITR.EXE
$START 140                               !Get into user DDT
DDT

JSYS 777$X                               !Enter MDDT
MDDT
```

!Following is an example of the procedure to map the JSB of a job:

```
FKJOB+105[   25,,2035                    !Get the SPT index of the JSB
                                         !of fork 105

T1!      2035,,0                         !Put SPT index in left half
T2!      540000,,JSBPGA                  !* Flags and where to map to
T3!      JSLSTA'1000-JSBPGA'1000         !Number of pages to map

CALL MSETMP$X                            !Do the mapping
$
```

!Following is an example of the procedure to map the PSB of a fork:

```
FKPGS+105[   2657,,2332                  !Get the SPT index of the PSB
                                         !of fork 105

T1!      2332,,PSBMAP-PSBPGA             !Put SPT index in left half,
                                         !and offset in right half
T2!      540000,,PSSPSA                  !* Flags and where to map to
T3!      PSBMSZ                          !Number of pages to map

CALL MSETMP$X                            !Do the mapping
$
```

E_mple of returning to user mode and looking at data from both the PSB and the JSB of the fork:

```
RETN$G                                  !Return to user mode

SRNAM[   3                              !Examine job's user name
SRNAM+1[   422050,,546230   $T;DBELL

TRLTT[   777777,,777777                 !Controlling terminal

ILBYT+MLJFN[   4400,,334010             !Start of data block for JFN 1

PC/   T1,,DISXE#+2                      !Current PC of the fork

AC+17/   -215,,UPDL+62                  !Current stack pointer

PDL/   CHKHO5#                          !First few stack locations
PDL+1/   CAM CHKAE0#+12
PDL+2/   CHKHO5#
PDL+3/   CAM CHKAE0#+12
PDL+4/   T1,,.COMND+1
PDL+5/   -273,,UPDL+4
```

E_mple of terminating the mapping we have done:

```
C
RESET                                   !To finish, just quit and reset
```

The procedure as given above maps the JSB and PSB write-enabled. o if you find something you want to change, you can simply deposit ie new value into the location. If you want the data to be rite-protected, then change the 540000 to 500000 in the two steps arked with an asterisk.

Warning: The procedure of mapping things into your user address pace has its limitations. Mapping the JSB and PSB works because the ser core used for mapping was previously empty. In general, you can nly map things into your user core if your core pages are either nexistant or are private. If you call MSETMP or SETMPG and map mething over a shared page, the old file page is unmapped without ie share counts being updated, which prevents your job from logging it later. To get around this problem you can BLT your core image to or all of the pages to be private.

HOW TO USE BREAKPOINTS IN CODE THAT MANY USERS EXECUTE

When inserting a breakpoint into the running monitor, you have to be careful that no other users will execute the code containing the breakpoint. If some other user hits the breakpoint, they will blow up with an illegal instruction since MDDT will not be there to handle the breakpoint. This normally limits the places you can set breakpoints, since most of the monitor can be gotten to by any user. Even if you run the system stand-alone, it is possible that the routine you are debugging will be called by job 0. However, it is still possible to do such debugging, even on a system which is not stand-alone, and this document will describe how this is done.

The essential element of this technique is to put in the patch in such a way that only your own fork can ever reach the breakpoint. First you write a simple routine which will skip if it is not being run by your particular fork. This can be done easily if you remember that the location FORKX contains the currently running fork number. An example of such a routine is the following:

```
@ENABLE
$SDDT
DDT
JSYS 777$X
MDDT

FORKX[   23                          ; check our fork number

FFF/   0    NOTME:   PUSH P,T1       ; save an AC
NOTME+1/    0    MOVE T1,FORKX       ; get currently running fork number
NOTME+2/    0    CAIE T1,23          ; is it us=23?
NOTME+3/    0    AOS -1(P)           ; no, setup skip return
NOTME+4/    0    POP P,T1            ; restore the saved AC
NOTME+5/    0    POPJ P,             ; and return to caller
NOTME+6/    0    FFF:                ; reset the position of FFF
```

The routine above simply saves AC T1, gets the currently running fork number, compares it with your own fork number which you obtained by looking at location FORKX, and skips if they differ.

Now assume that you want to set a breakpoint into the following ode, which is in the routine BLKSCN in the module DIRECT.

```
LKSC2/   HLRZ C,BLKTAB(B)
LKSC2+1/   CAME A,C
LKSC2+2/   AOBJN B,BLKSC2
LKSC2+3/   JUMPGE B,BLKSCE
LKSC2+4/   HRRZ B,BLKTAB(B)
```

ssume you want the breakpoint at location BLKSC2+3. You do the ollowing:

```
LKSC2+3/   JUMPGE B,BLKSCE   FFF$<      ; patch this location
FF/   0   PUSHJ P,NOTME                 ; call the NOTME routine
FF+1/   0   .$B   JFCL$>                ; me if it gets here, set breakpoint
FF+2/   JUMPGE B,BLKSCE
FF+3/   JUMPA A,BLKSC2+4
FF+4/   JUMPA B,BLKSC2+5
LKSC2+3/   JUMPA NOTME+6
```

otice that the breakpoint has been set in the JFCL instruction ollowing the call to NOTME. Only your fork will execute it, so you an now debug the section of code while other users are executing it t e same time. Remember to remove the breakpoint when you are one.

To run a particular program while having breakpoints set, you ist remember that the breakpoint has to be set by the same process ich you expect to hit it. So for example, typing ^EQUIT, setting a eakpoint, returning to the EXEC and running your program will not rk. You must enter MDDT and set the breakpoints from your program u want to debug. As an example:

```
ENABLE
ET PROGRAM      ; get the program to be used
DT              ; enter DDT
T
YS 777$X        ; and enter MDDT from there
DT

UT IN "NOTME" ROUTINE AND SET BREAKPOINTS HERE)

ETN$G           ; return to the context of the test program

;               ; start the test program
```

Using Address Break to Debug the Monitor

Sometimes when examining a set of dumps, you will notice the crashes are caused by some location being destroyed. If you have no idea where the destruction is done from, finding the problem could be very difficult. One useful procedure in such cases is to use the address break feature of the hardware to track down the problem (except for 2020's!). The only problem is that the use of address break is not obvious. This is a manual describing how to use address break in the TOPS-20 monitor.

In order to use address break, four things must be done. First, the current routines the monitor uses to set address breaks for users must be disabled. Secondly, your own address break must be set from MDDT or EDDT. Thirdly, instructions which you want to execute properly have to be modified so that they will not cause an unwanted address break. Finally, breakpoints must be placed in the monitor so that the state of the monitor can be examined when the address break occurs. The following is a step by step example of doing this.

1. Load the monitor for debugging, and enter EDDT. The procedure starting from BOOT is the following:

```
BOOT>/L                          ;Load monitor but don't start it
BOOT>/G140                       ;Start EDDT
EDDT
DBUGSW/   0   2                  ;Set debugging mode
EDDTF/    0   1                  ;Keep EDDT once system starts
GOTSWM$B                         ;Install useful breakpoint
SYSGO1$G                         ;Start the monitor


[PS MOUNTED]
$1B>>GOTSWM   0$1B               ;Remove breakpoint now
```

2. Disable the monitor's normal changing of the address break. This is currently done at two places:

```
KISSAV+4/   DATAO UNPFG1+26   JFCL      ;Disable instruction
SETBRK+12/   DATAO A   JFCL             ;Here too
```

Set your own address break at the desired location. Refer to the Hardware Reference Manual for details. The instruction to set an address break is:

```
DATAO APR,ADDR          ;Note:  APR = 0
```

where ADDR contains the following fields:

Bits	Description
9	Break at given address on instruction fetches
10	Break at given address on reads
11	Break at given address on writes
12	0=exec address space, 1=user address space
13-35	Address to break on.

So now assume you want to catch a bug which is blasting location CURDS. You want to break only for writes, and want to use exec virtual space. Therefore you type the following:

```
FFF/   0   100000000+CURDS      ;Put data in convenient place
DATAO APR,FFF$X                 ;Set the address break
```

Now you want to disable address break for all instructions which you expect to change the given location. Assume in this example that only location DIDDLE should change location CURDS. Then you do the following for a model B CPU:

```
FFF!   IT:                      ;Define location to get old flags
IT+1!                           ;Old PC
IT+2!                           ;New flags
IT+3!   IT+4                    ;New PC
IT+4!   EXCH IT                 ;Save AC and get old flags
IT+5!   TLO 1000                ;Set address break inhibit bit
IT+6!   EXCH IT                 ;Restore flags and AC
IT+7!   JRST 5,IT               ;Return to caller
IT+10!   FFF:                   ;Redefine FFF

DIDDLE/    MOVEM A,CURDS   FFF$< ;Insert patch
FFF/   0    JRST 7,IT$>         ;Call above routine
FFF+1/   0    MOVEM A,CURDS     ;Typed by DDT when finishing patch
FFF+2/   0    JUMPA A,DIDDLE+1
FFF+3/   0    JUMPA B,DIDDLE+2
DIDDLE/    MOVEM A,CURDS   JUMPA IT+10
```

The JRST 7,IT instruction is used to save the old PC at IT and IT+1, and take a new PC from IT+2 and IT+3. There the old PC is changed to include the address break inhibit bit. Then a JRST 5,IT is done which returns to the caller. The next instruction then executes without causing an address break. You have to insert the JRST 7,IT instruction at every

instruction you want to succeed.

For model A CPUs the procedure is similar, but a little easier:

```
FFF!   IT:                              ;Define location to hold PC
IT+1!   EXCH IT                         ;Get old PC and save AC
IT+2!   TLO 1000                        ;Set address break inhibit flag
IT+3!   EXCH IT                         ;Restore PC and AC
IT+4!   JRSTF @IT                       ;Return to caller
IT+5!   FFF:                            ;Redefine FFF

DIDDLE/   MOVEM A,CURDS   FFF$<  ;Insert patch
FFF/   0   JSR IT$>              ;Call above routine
FFF+1/   0   MOVEM A,CURDS       ;Typed by DDT when finishing patch
FFF+2/   0   JUMPA A,DIDDLE+1
FFF+3/   0   JUMPA B,DIDDLE+2
DIDDLE/   MOVEM A,CURDS   JUMPA IT+5
```

5. Now put the breakpoints into the monitor so that when an address break occurs, you will get into EDDT. There are two locations to patch, one for PI level and one for non-PI level. You also have to patch a monitor bug in release 3 and 3A so that the page fail dispatch code works properly.

```
ADRCMP$B                            ;Set breakpoint at non-PI routine
PFCD23$B                            ;Set breakpoint at PI routine
PIPTRP+1/   MOVE A,TRAPSW   MOVE A,TRAPS0       ;And fix a bug
$P                                  ;Now let the monitor proceed
```

6. When either of the above breakpoints is hit, the flags and PC of the instruction which caused the address break will be in locations TRAPFL and TRAPPC. If the address break was from JSYS level (breakpoint was to ADRCMP and location INSKED is zero) then an $P will proceed properly. If the address break was from the scheduler or from PI level, doing $P will be useless since the monitor will then BUGHLT because it doesn't want to see an address break under these conditions. However, this is ok if all you want to do is find the instruction causing the trashing.

If the location still gets trashed after trying to catch it this way, either your procedure is wrong; you are trying this on a 2020 (which has no address break feature); the location is being changed by some IO being done (RH20s, DTEs, etc); or else the machine is having some hardware problems.

RECOVERING FROM DIRECTORY ERRORS

Sometimes after a monitor crash due to disk problems, some of the irectories on the system will contain errors. These errors cause UGCHKs such as DIRFDB, NAMBAD, DIRPG0, and DIRPG1. It is sometimes ossible to find the error in the directory by getting into MDDT, apping the directory, finding what is wrong, and fixing it. This rocedure is described in the SWSKIT. However, this is not always asy, and may take a lot of time. It is therefore better in many ases to simply delete the bad directory and recreate it. This is asy to do for most directories. But special procedures are necessary or the directories <SYSTEM> and <SUBSYS>. The rest of this memo will escribe the methods of recovering from bad directories, handling in articular the difficult case of the <SYSTEM> directory.

You can first try to give the EXPUNGE command with the REBUILD nd PURGE subcommands. If the problem with the directory is very imple, it may fix your problem. As an example, suppose the directory S:<SICK-DIRECTORY> is incorrect. You would type:

```
$EXPUNGE (DIRECTORY) PS:<SICK-DIRECTORY>,
$$REBUILD (SYMBOL TABLE)
$$PURGE (NOT COMPLETELY CREATED FILES)
$$
 PS:<SICK-DIRECTORY> [NO PAGES FREED]
$
```

If this does not help the problem, you will have to delete the irectory and then recreate it. Before proceeding, you should make ure that any files you can reference are copied to another directory, e else are saved on tape. Now first try to delete the directory ormally, as follows:

```
$BUILD (USER) PS:<SICK-DIRECTORY>
[OLD]
$$KILL
[CONFIRM]
$$
$
```

If this is successful, then simply recreate the directory again, and restore the user's files. You should recreate the directory with the same directory number as it had before, so that DLUSER's data will still be correct.

The procedure above will fail if either the directory is mapped by another job, or if it is totally unusable. If it is mapped, and the directory is a random user, you can wait until the directory is no longer in use, or you can take the system stand-alone so that no user can reference it.

If the directory is totally unusable, you will then have to try to delete it the hard way. Before proceeding, you should try to delete and expunge all files in the directory. This will minimize the amount of lost pages that will result. Now there are two cases to consider. If the directory is not a sub-directory, you type the following:

```
$DELETE (FILE) PS:<ROOT-DIRECTORY>SICK-DIRECTORY.DIRECTORY,
$$DIRECTORY (AND "FORGET" FILE SPACE)
$$
 <ROOT-DIRECTORY>SICK-DIRECTORY.DIRECTORY.1 [OK]
$
```

If the directory is a subdirectory, you modify the above command by replacing "ROOT-DIRECTORY" by the name of the next higher directory. Thus if the directory was PS:<ANOTHER.BAD-ONE>, you type:

```
$DELETE (FILE) PS:<ANOTHER>BAD-ONE.DIRECTORY,
$$DIRECTORY (AND "FORGET" FILE SPACE)
$$
 <ANOTHER>BAD-ONE.DIRECTORY.1 [OK]
$
```

The above procedure tells the monitor to treat the directory file like a normal file, and to delete it as such. This means that any files in the directory will become "lost". The disk pages can be recovered later with CHECKD. If the above works, you simply can recreate the directory and restore the files.

The only reason the above command should fail is if the directory is still mapped. For PS:<SUBSYS>, you can bring up the system stand-alone so that no programs are run from it, and then delete it. For PS:<SYSTEM>, even taking the system stand-alone will not help, for it is always mapped by job 0. But there are two procedures you can use which do work.

The safest method can be used if the user's system has mountable tructures, If you have built another PS: structure, you can mount he pack with the bad directory as an alias, and then the directory ill not be mapped and can be deleted. As an example:

```
$SMOUNT (FILE STRUCTURE) SICK:,
$$STRUCTURE-ID (IS) PS:
$$
WAITING FOR STRUCTURE SICK: TO BE PUT ON LINE...
STRUCTURE SICK: MOUNTED
$
$DELETE (FILES) SICK:<ROOT-DIRECTORY>SYSTEM.DIRECTORY,
$$DIRECTORY (AND "FORGET" FILE SPACE)
$$
 SICK:<ROOT-DIRECTORY>SYSTEM.DIRECTORY.1 [OK]
$
```

Then you can build the new directory, restore the files to it, id then use it again for your normal PS: pack. Be sure to build the w directory with the same number. This is especially important for e special system directories.

If you do not have another disk drive or another PS: disk, or if u don't want to bother SMOUNTing the disk, you can fix the <SYSTEM> ea by using MDDT. The basic idea is to patch the monitor so that it longer thinks that the directory is in use. This is done as llows:

```
$^EQUIT

INTERRUPT AT 17117
MX>/MDDT
CHKOFN/   JSP CX,.SAVE   JRST RSKP
MRETN$G

$
```

Then you should have no problems deleting the directory. mediately after doing the delete, you should reload the system. en the system restarts, you can read the monitor and the EXEC either om the distribution magtape or from another directory where you had pt copies. Then recreate the <SYSTEM> area, making sure to give it e same directory number as it had before. Then you can restore the le- and let the users back on. Finally, you should run CHECKD me to recover the lost pages.

INDEX

1.0 Introduction and Overview

This document is designed as a users guide to DDT version 41 in so far as it has changed from previous versions of DDT. It is not a complete users guide to all the wonders of DDT, just those new features which have recently been implemented (although directed primarily at new features only in DDT version 41, some documentation is included to describe other aspects of DDT which have been around for a longer period of time, but were never fully understood or otherwise documented).

Throughout this document it is assumed that the reader is already familiar with DDT and the MACRO assembly language in general as well as the appropriate operating system(s).

This is the first revision of this document, incorporating the additional changes to DDT version 41 as of edit 260.

2.0 Configurations

DDT version 41 will run on KA-10's, KI-10's, KL-10's, and KS-10's, using no paging, KI-paging, or KL-paging, with or without extended addressing in user or executive mode (user and file DDT's run only in user mode) with no special assembly needed. DDT version 41 must be assembled to run under either the TOPS-10 or the TOPS-20 operating system.

It traditionally has been a goal to maintain one single set of source files from which all flavors of DDT are built. This goal has been maintained.

Note

> TOPS-20 UDDT (and SDDT) now use memory locations 764000 through 777777 (previously 770000 through 777777), but the starting address for DDT continues to be location 770000. This requires version 4 of the PA1050 "compatibility package".

3.0 Memory and Address Control

The single biggest change to DDT version 41 from earlier versions is in the realm of memory control and how the user addresses memory locations.

3.1 Extended addressing

All flavors of DDT except FILDDT will run in any memory section. Full extended addressing is supported, as are "large" addresses - DDT will now accept a full 36-bit expression as an address although obviously only FILDDT can actually handle an address over 30-bits wide. In all cases the actual address must be positive (i.e., effectively a 35-bit address).

.1.1 Symbol table restrictions

here are certain restrictions however which must be adhered to in rder for DDT to function correctly. The first restriction is that the ymbol table logic is essentially section-dependent, i.e., the symbol able and its pointers (.JBSYM=116 and .JBUSY=117, also .JBHSM=6 rela- ive to the start of the "high segment") must reside (i.e., be mapped) n the same section as that in which DDT itself is running. Further, he symbol table can be no longer than 128K words in length and must e RADIX-50 format.

uch thought is being given towards the implementation of a totally ew symbol table scheme which would address all of these problems, the ingle biggest one of which is simply how is extended addressing going o be used - as a single fixed address space with one or more "global" ymbol tables (like the TOPS-20 monitor currently works), or as a ollection of independent sections each of which has section-local ymbols/symbol tables (whatever that means), or what?

.1.2 Breakpoint restrictions

he second restriction of which the user must be aware concerns reakpoints. Since the hardware has no facility to unconditionally ransfer control to DDT using only 36-bits, DDT must be mapped into uch section (at the same relative address obviously) which contains ode into which the user wishes to place breakpoints.

.1.3 Location examining restrictions

ven if running on an extended addressing machine if DDT is running in ection 0, then only locations within section 0 (addresses 0 to 77777) may be manipulated. DDT will make no effort to outsmart the ombined efforts of the user and the operating system by sneaking into non-zero section even momentarily to do the memory reference.

.2 Effective address calculation

DT version 41 can calculate effective address references using either local" or IFIW (Instruction Format Indirect Word) or "global" or EFIW Extended Format Indirect Word) formats. In a normal DDT address-open- ng command ("/", "\", <TAB>, etc.) a single <ESC> delimiting the ddress expression (e.g., "MOVE 3,@200(10)$/" or just "$[") instructs DT to treat the expression as an IFIW word and calculate the effec- ive address exactly like the hardware would, were the hardware to ecute that 36-bit word as an instruction at location "." (whether or ot location "." is currently open).

wo <ESC>'s delimiting the address expression instructs DDT to treat he 36-bit expression as an EFIW word and calculate the effective ddress exactly as the hardware would, were the hardware to indirectly ddress the 36-bit expression at location "." (whether or not location ." is currently open). A strange case can come up about which the er should be cautioned - there is an ambiguity as to where (i.e., a section") to start the effective address calculation. DDT as- mes the left half of "." (i.e., the last location opened by the er). If for example having opened location 0,,1234 which contains ,,4321 the user issues the command "$$[" then DDT will calulate the ffective address as the contents of location 4321 in section 0 ndexed by the right half of register 7, and if bit 13 is on, treating

that word as an IFIW and continuing the address calculation. This, although probably not what was expected, is in fact exactly what the hardware would do since the indirect word came from section 0. Had the user opened location 1,,1234 (containing 7,,4321) then DDT would take the contents of location 7004321 and continue from there.

If no <ESC>'s delimit the address expression, then DDT simply uses the full 36-bit expression as the address (e.g., "30,,30/" says open location 30000030 and "-1/" says open location 777777777777). Again, only FILDDT can actually reference an address greater than 30-bits wide (not that anyone has that much disk space, but the hardware will not permit an address space over 30-bits wide), and in any case the address must be a non-negative 36-bit integer.

There is a special case in which DDT does something "kinky" - if a space was typed in entering the address expression, or if no explicit address was typed (i.e., the user is using the "last word typed" by simply typing only (for example) <TAB>), DDT will form the 36-bit actual address by using only the right half of the 36-bit address expression plus the left half of "." as the section number. This not-at-all-obvious behavior is so that the user can type in expressions such as "JRST PAT<TAB>" and have DDT go to location PAT in the same section as the JRST PAT instruction rather than going to address 254000000000+(PAT modulo 2**18). Another common usage of this "feature" would be in chaining down linked lists where the link pointer is an 18-bit section-local address in the left half of a word. To do this the user may type "sp$$Q/" (where "sp" means space). This is one of those cases where usefullness outweighs cleanliness of implementation and documentation.

3.3 Modifying memory

Two new commands have been added to facilitate DDT's manipulation of the user address space.

3.3.1 Automatic write-enable

The $W or $0W command instructs DDT to, if the user attempts to deposit into a write-protected memory location, automatically attempt to write-enable the memory location, do the memory deposit, then finally re-write-protect the memory location (default for TOPS-10); the $$W or $$0W command instructs DDT to simply give an error indication if the user attempts to change a write-protected memory location (default for TOPS-20). For FILDDT the use of this command is restricted to non-file usage such as "DDT'ing" the running monitor/memory space.

3.3.2 Automatic page-creation

The $1W command instructs DDT to automatically try to create the page the user is trying to deposit into if it doesn't already exist (default for TOPS-20); the $$1W command instructs DDT to simply give an error indication if the user attempts to write into a non-existant page (default for TOPS-10). EDDT and FILDDT doing super I/O or "DDT'ing" an .EXE file will NEVER attempt to create a non-existant page. For FILDDT the user must specify patching the file when he starts FILDDT in order to be able to create new pages (e.g., extend the file or fill in a gap in the middle of the file (TOPS-20 only)).

,4 Page mapping and physical addressing

n DDT version 41 all flavors of DDT support page mapping and address
elocation as well as register and physical address manipulation. All
f these functions use some variation of the $U/$$U DDT command. In
eneral these functions may be mixed together (for example address
elocation and page mapping).

*** Warning ***

The $U command syntax in DDT is totally different (and mainly incompatible) from previous versions of DDT! The user is MOST strongly urged to carefully read this section on memory mapping and addressing!

,4.1 Physical addressing

)T now has the concept of "physical" addressing in addition to its
)rmal "virtual" addressing. The $U command instructs DDT to use nor-
al virtual addressing (what it used to do); the $$U command instructs
)T to manually track down the honest physical address rather than the
irtual address space in which DDT finds itself running. Physical ad-
:essing is really applicable only to EDDT or to FILDDT looking at
inning monitor/memory (TOPS-10 only). User mode DDT (including EDDT
in ng in user mode, MDDT (TOPS-20 only), and VMDDT (TOPS-10 only))
id FILDDT looking at a disk all treat $U and $$U identically. In
)ysical addressing location 0 is not register 0 (i.e., DDT's internal
)py of user register 0) but rather physical memory location 0 page 0
ank 0 box 0 (that memory location on the hardware memory bus that
esponds to all address bits = 0).

)en the $$U DDT command is issued "physical" locations 0 to 17 become
:egisters" 0 to 17. For user mode DDT this means locations 0 to 17
ecome DDT's registers rather than the user's registers (although the
ser's registers will be properly restored on DDT-exit, $$U merely
irects DDT not to use the internal "fake" (i.e., user) registers).
)r FILDDT this means file words 0 to 17 (as mapped by the .EXE
irectory if used) become locations 0 to 17 (normal for a data file).

ibsequent issuance of the $U DDT command will redirect locations 0 to
/ to being DDT's internal "fake" registers again, except for FILDDT
)oking at an data file or doing super I/O to a disk.

)te that for executive mode EDDT to utilize physical addressing the
iging hardware must have been enabled PRIOR to DDT-entry. This re-
iirement exists because EDDT, in order to access all of physical
)mory, needs to map the desired physical address into its own (execu-
ive) virtual address space, which it does by fondling the already-
:tant page maps. For EDDT to provide physical addressing capability
.thout this restriction would require 2 (3 if KL-paging) more memory
ig be dedicated to EDDT for building temporary page maps, plus
ipport code etc.

)r FILDDT to examine/modify physical memory a 7.00 or later release
: the TOPS-10 monitor is required; no release of TOPS-20 supports
:LDDT'ing physical memory.

3.4.2 Page mapping
All flavors of DDT now support page mapping in both the KI- and the KL-tradition. EDDT in executive mode will dynamically figure out which style of paging is in effect and operate accordingly. All other flavors of DDT (including EDDT running in user mode) will assume the mode of paging used by the operating system for which DDT was assembled - KI-paging for TOPS-10 and KL-paging for TOPS-20. To select KI-paging emulation the flg$10U command is used; to select KL-paging the flg$11U command is issued; in either case if flg is zero then the paging emulation is disabled, if flg is non-zero then the appropriate paging emulation is enabled.

In executive mode EDDT or FILDDT looking at running monitor/memory space DDT will internally utilize physical addressing in order to provide the user the true mapped virtual address space desired.

3.4.2.1 KI-paging - For KI-paging (TOPS-10 default) the page mapping command for the executive virtual addressing space is [upt<]ept$[0]U where upt is the optional physical memory page number of the user process table (for setting the "per-process" addressing space - exec virtual addresses 340000 through 377777) and ept is the physical memory page number of the executive process table. The user virtual addressing space is selected by the upt$1U command. The command $U returns DDT to regular unmapped virtual addressing.

3.4.2.2 KL-paging - For KL-paging (TOPS-20 default) the page mapping command for the executive virtual addressing space is ept$[0]U where ept is the physical memory page number of the executive process table, or epx$$[0]U where epx is the index into the SPT of the executive process table pointer. To select the user virtual addressing space the command is upt$1U where upt is the physical memory page number of the user process table, or upx$$1U where upx is the index into the SPT of the user process table pointer. The command $U returns DDT to regular unmapped virtual addressing.

To map a single section (256K address space) under KL-paging the command is either sec$2U where sec is the physical memory page number of a KL-paging section map, or sex$$2U where sex is the index into the SPT of the section map.

Basically, under KL-paging, $0U selects the ept, $1U selects the upt, and $2U selects a single section. A single $ indicates the physical memory page number and two $'s indicate an SPT index.

3.4.3 Setting the SPT
FILDDT will automatically define the start of the SPT from a disk file (assumed monitor dump) from the symbol "SPT" if it exists (TOPS-20 only). The command spt$6U specifies to DDT that the SPT starts at address spt.

3.4.4 Register addressing
The command acs$5U instructs DDT to use the 20 consecutive locations starting at acs as the registers (DDT maintains an internal copy of the registers so changing "register" 3 will not affect (for example) acs+3). FILDDT, when reading an .EXE file, will automatically load its internal "fake" registers as though the user had typed CRSHAC$5U if

OPS-10 or BUGACS$5U if TOPS-20. Note that if physical addressing mode as been entered (the user has issued the $$U command) then the inter- al "fake" registers are ignored; if the user subsequently reenters irtual addressing (via some form of the $U command) then an acs$5U ommand may also have to be re-issued to get the registers back (this oes not affect the saving and restoring of the hardware registers in ser or executive DDT, only what DDT will use for typing out locations to 17).

he command flg$3U explicitly controls the usage of DDT's internal fake" registers - if flg is 0 then the "fake" registers are ignored i.e., 0 to 17 are taken from the true current addressing space), if lg is non-zero then addresses 0 to 17 are taken from DDT's internal opies of the registers.

he $U command, except for FILDDT'ing a data file or doing super I/O o a disk, will return DDT to its internal "fake" registers. The se- ection of registers is completely independent of any page mapping in ffect. Changing virtual address spaces does not change the "regis- ers".

n executive mode DDT only the command n$4U will switch DDT to use and thus display) hardware AC block n (available only for KL-10's and S-10's). The user is warned that 7$4U on a KL-10 will bring rapid and ab death (the microcode uses AC block 7). On DDT exit DDT will estore the ac block context to the state it was in at DDT entry.

.4.5 Address relocation and protection
s an aid to looking at data structures which are formed using ointers as offsets rather than pointers as absolute values DDT ver- ion 41 will allow the user to set both a base relocation address to e added to all addresses used in location examining commands and a rotection address beyond which the user "virtual" (note the use of virtual" here as meaning pre-relocated) address is illegal. This is coincidently) exactly analguous to the KA-10 hardware relocation and rotection strategy, and in fact could be used as such to "mimic" the J KI/KL/KS-10 functionality on a KA-10 in executive mode. The form of his command is bas$8U where bas is the base virtual address, and rt$9U where prt is the maximum address the user will be allowed to ype in. Note that page mapping and address relocation and protection re independent mechanisms, with address relocation and protection be- ng performed before any mapping is done. The protection address has o effect on the final "physical" address generated by any mapping urrently in effect.

.4.6 $U command summary
.1 $U/$$U commands take the following form:

$U	Unmapped virtual addressing
$$U	Unmapped physical addressing
pt$[$][0]U	Select executive virtual addressing
upt$[$]1U	Select user virtual addressing
sec$[$]2U	Select single section
flg$3U	Select (deselect) internal fake registers
acb$4U	Select hardware ac block
acs$5U	Load internal fake registers

9. spt$6U Select base of SPT
10. bas$8U Set base relocation address
11. prt$9U Set protection address
12. flg$10U Select (deselect) KI-paging
13. flg$11U Select (deselect) KL-paging

where:

1. acb := integer ac block number
2. acs := address of 20-word register block
3. bas := base relocation address
4. ept := executive process table page number
5. flg := selection flag, zero to deselect, non-zero to select
6. prt := protection (maximum allowable) address
7. sec := section map page number
8. spt := address of SPT
9. upt := user process table page number

3.4.7 Address checking (Executive EDDT only)

EDDT version 41, when running in executive mode, now is much more extensive in validity-checking memory references. In particular, EDDT will not cause a NXM (page fault) trap to the resident operating system if the user types in an illegal (non-existent or unmapped) address, but rather will simply type its ubiquitous ?<DINK><TAB> error message.

3.4.8 Address breaking

DDT will no longer cause an address break to occur when examining or depositing a location at which an address break condition has been set. This applies only to "user" examines and deposits, an address break set in DDT will still cause an address break to occur.

4.0 Specifying the Start Address

The $G command now expects a 36-bit address (obviously with bits 0 to 5 off) at which to start the user program. This means that the users of programs such as the TOPS-10 monitor which define symbols like "DEBUG=:<JRST .>" can no longer go either DEBUG$G or DEBUG$X at the users whim but must decide on one form or the other (the default obviously being to do nothing - i.e., to settle for the DEBUG$X form)

5.0 Symbolic expression typein and typeout

DDT version 41 has expanded the range of both symbolic typein and symbolic typeout.

5.1 Symbolic typein

The JSYS opcode (opcode 104) has been added to TOPS-20 DDT, as have all the TOPS-10 UUO's (but not the CALLI's etc.) for debugging programs which run under the compatibility package.

2 Multiply-defined symbol typein

the user types an ambiguous symbol (a symbol defined two or more aces outside of the current local symbol table and not in the cur- nt local symbol table) DDT will issue an "M" error message.

3 Selecting no local symbol table

e $: command issued without an explicit module name to use as the cal (or "opened") symbol table will deselect any local symbol table. is is the initial state in which DDT starts.

4 Symbol cache

T now has a symbol "cache" of symbols recently used to type out lues. This cache is used primarily for typeout; typein will check e symbol cache for a matching symbol from the currently opened or cal symbol table, if no match is found the cache is ignored and the gular symbol table is used. The symbol cache is "flushed" on the suance of any $: command.

5 Symbolic typeout

T now goes to great pains to find any possible user-defined symbol u as an OPDEF) to match the expression DDT is trying to type out. e order in which DDT searches for a symbol match in symbolic typeout de for non-I/O instructions is:

Full 36-bit match; OP, AC, I, X, and Y fields (e.g., the TOPS-20 monitor calls such as GTJFN)
OP, I, X, and Y fields (e.g., the TOPS-10 monitor calls such as FILOP.)
OP and AC fields (e.g., the TOPS-10 monitor calls such as INCHWL or "instructions" such as HALT)
OP field only (e.g., user UUO's or "OPDEF XMOVEI [SETMI]")
DDT's internal hardware opcode table

e order in which DDT searches for a symbol match in symbolic typeout de for I/O instructions is:

I/O OP and DEV fields (bits 0 to 12 - e.g., KL-10 APRID or KS-10 RDCSB)
Regular (non-I/O) OP field (e.g., KS-10 UMOVE)

0 ASCII typeout

T version 41 adds the typeout mode commands $8T and $9T to typeout 8 t SCII or 9 bit ASCII respectively (i.e., pick up 8 or 9 bit bytes d ype" them straight as is - which with current TOPS-10 and TOPS- operating systems means as 7-bit ASCII).

7.0 Command files

The $Y command (TOPS-10 DDT only) has been changed somewhat, both in input and output (logging) functions.

7.1 Command input

If the user does not type a 36-bit expression to be used as a file name (such as $""FILNAM"$Y) but just types $Y by itself then DDT will prompt with "File: ". After the prompt the user can enter a TOPS-10 file specification in the form dev:name.type[directory]/switches where [directory] can of course contain SFD's.

7.1.1 /A switch

The /A switch instructs DDT to abort the command file if a DDT-detected command error occurs (such as reference to an undefined symbol).

7.2 Command output (logging)

When reading a command file ($Y command) DDT will no longer "log" all output onto device LPT: but rather just type out onto the user terminal.

8.0 Automatic patch insertion

The automatic patch insertion facility ($< and $> commands) are basically the same as in version 40 of DDT with only minor differences.

8.1 Patch opening

The user may specify patching either by sym$< where sym is the name of a symbol (which will be automatically updated at the termination of the patch) or via exp$< where exp is any 36-bit expression representing the address of the resultant patch. If the later form of the patch command is used no symbol will be updated to the end of the patch.

8.2 Default patching symbol

The list and order of default patching symbols which DDT uses when the user does not supply an explicit patching symbol is now:

1. PAT (TOPS-10 EDDT only)
2. FFF (TOPS-20 EDDT and MDDT only)
3. PAT.. (all flavors)
4. PATCH (all flavors)
5. PAT (all flavors except TOPS-10 EDDT)

8.3 Default patching address

If the user does not supply an explicit patching symbol and DDT is unable to find one of the default patching symbols then the address specified by the right half of location .JBFF (even on TOPS-20) is used. On patch close ($> command) if the patching address was defaulted to via .JBFF, then both the right half of location .JBFF and the

e half of location .JBSA are updated.

.4 Patch closing confusion and restriction

ith DDT version 41 it no longer matters how (when) the user types the > command, either immediately after the final word expression, or fter a <CR> or <LF> to terminate the final word expression - DDT will ever generate a 0 word for free.

here is a very obscure restriction however on the use of the # ommand in conjunction with the $> command. If the user is referencing n undefined symbol in the expression for the last word of the patch hen that expression must explicitly be terminated in such a fashion s to close the location before terminating the patch. For example, MOVE T1,BLETCH#$>" is illegal but "MOVE T1,BLETCH#cr$> (where "cr" ndicates a carriage return) is ok.

,0 Breakpoints

he breakpoint logic in DDT version 41 has been extensively revamped n order to support extended addressing. The default number of break- oints is now 12 (decimal); and can be set (by defining the symbol BP=number of breakpoints) arbitrarily high (within memory space limi- tions) rather than being limited to 9 or 36 (decimal) depending on hich code restriction one choose to believe.

.1 Setting breakpoints

DT can now set a breakpoint in code running in any section with two strictions:

- If DDT is currently running in section 0 then breakpoints can only be set in section 0 (see section 3.1.3 above).
- DDT must be mapped in the section containing the code in which breakpoints are to be placed (the logic of this is that since there is no way for DDT to cause unconditional transfer of control to DDT with only 36 bits some portion of the section address space must be devoted to DDT; therefore, given this restriction, one might just as well put all of DDT in that section since it makes for a cleaner and simpler implementation). Note that this does not mean DDT must be running in that section, but merely that DDT must be mapped in that section!

does not matter into how many different sections the same code is pped as long as DDT is mapped into the same sections since DDT is ection-independent". For example (taking the TOPS-20 monitor which ps section 0 and section 1 identically) if a breakpoint is set at dress 1004567 (or 1,,4567) but the PC was 4567 (or 0,,4567, i.e., in ction 0 rather than section 1) when the breakpoint was executed DDT es not care (as long as DDT is mapped in that section, which in the ar e of the TOPS-20 monitor it is).

e syntax for setting a breakpoint is now opn<bpt$nB where n is tional and, if specified, declares the breakpoint number to be as- gned to that address; bpt is the 36-bit address at which to place a

breakpoint; and opn is an optional 36-bit address to open and display upon execution of the breakpoint. The syntax was changed because two full 30-bit addresses could not be squeezed into two halfwords.

DDT will no longer assign two different breakpoints to the same address, either accidentally or under user control - if the user attempts to set a breakpoint at a location at which a (different) breakpoint is already set, the old breakpoint is cleared first.

9.2 Breakpoint typeout

Upon execution of a breakpoint DDT will now always typeout the user instruction (in instruction format regardless of the permanent typeout mode) at that breakpoint and set "." to the breakpoint address. If, further, opn was specified as in section 9.1 above, then DDT will also display the contents of location opn in the permanently set typeout mode and "." will be updated to opn (with the breakpoint address itself becoming the previous PC sequence and so available via the $<CR> etc. commands).

9.3 Examining breakpoint locations

The $nB command continues to be the "address" of breakpoint n's database, but $nB is no longer equal to $n-1B+3. The breakpoint database of interest to the user now has the following format:

1. $nB+0/ If nonzero the address for breakpoint n
2. $nB+1/ The conditional break instruction (break if skips)
3. $nB+2/ The proceed count (break on transition to 0)
4. $nB+3/ If greater than or equal to zero then the address to be displayed

The rest of the breakpoint data base should not be of use to the user.

9.4 Unsolicited breakpoints

DDT version 41 has a new breakpoint facility - the ability to handle unsolicited breakpoints (i.e., breakpoints that DDT did not itself set). If control passes to location $0BPT+1 ($0BPT is a global DDT symbol) then DDT will act as if a breakpoint had been set at the address-1 contained in location $0BPT. The address in $0BPT must be setup as if the cpu executed a JSR $0BPT instruction - if in section 0 then flags,,PC otherwise just global 30-bit PC. After "hitting" an unsolicited breakpoint the user can proceed with program execution with the $P command (all arguments to the $P command such as proceed count or auto-proceed ($$P) are ignored).

Although this facility gives programs the ability to cause breakpoints at any time (thus getting into DDT with the program state carefully preserved) it is intended to be of most use in conjunction with an as-yet-unimplemented monitor command (such as control-D) to "force" a breakpoint on a program without having to control-C/DDT the program. Then the user could simply continue with the program by typing $P.

) Single-stepping the program

ıe $X DDT command has been significantly modernized (and sped up in
eneral) with version 41 of DDT.

).1 New opcodes

ıe ADJSP, DADD, DSUB, DMUL, and DDIV instructions have been added to
)T's $X table although double- and quad-word integers (for DADD etc.)
'e still typed out as two or four single words rather than one big
ıltiple precision integer. All of the extended JRST-class instruc-
ons are correctly simulated/traced. A user-UUO being executed in a
ın-zero section is simply XCT'ed and is not traced.

'.2 Byte-manipulation typeout

rudimentary byte-manipulation-instruction typeout facility was added
o DDT version 40 actually) to display the byte pointer and the
ntents of the effective address of the byte pointer. The EXTEND-
ass instructions are not handled.

.3 Effective address calculation

T now always calculates the effective address of the instruction
ing $X'ed rather than just blindly "doing it" in order to both pre-
n DDT from getting an illegal memory reference as well as to make
T e independent of the section in which the user PC resides (i.e.,
T does not have to be mapped into the user PC section to handle
'es although if the user PC is in a non-zero section then DDT must
in a non-zero section). Besides, it's usually faster too!

.4 KS-10 I/O instruction trace

e KS-10 specific I/O instructions which reference the UNIBUS (execu-
ve mode only) are not traced, only the contents of the register
ecified in the AC field are displayed. Since the UNIBUS device reg-
ters can be reference-volatile (i.e., merely referencing one can
ıse it to change - such as the DL-11 data registers) DDT does not
)eout the contents of the referenced UNIBUS address. Further, since
e effective address of the instruction is not calculated in a
ındard format (at least as far as DDT is concerned) the effective
ress itself is not even displayed.

.5 PC skipping

the user instruction being $X'ed skips then DDT will now typeout
KIP>" if the PC skips by one location, or "<SKIP n>" if the PC
ps by n locations, where n is less than or equal to the DDT assem-
parameter SKPMAX (by default 3). If the PC changes more drastical-
than that (e.g., goes to a smaller address) DDT will type "<JUMP>
tead.

6 ERCAL/ERJMP

(TOPS-20 only) will now handle instructions followed by either an
AL or an ERJMP instruction (which is really just a 72-bit instruc-

tion with two effective addresses). If the instruction being executed does not take the error jump then DDT will print "<ERSKP>" after the normal instruction trace to indicate to the user that an ERCAL or ERJMP was just skipped (i.e., the PC incremented by 2 rather than 1) and will not display the ERCAL or ERJMP instruction. If the instruction does take the error jump then the ERCAL or ERJMP instruction will be displayed, if an ERCAL instruction then register 17 will also be displayed, and the PC will be changed to the error address.

DDT will print "<ERSKP>" rather than showing the ERCAL or ERJMP instruction since DDT has no way of telling whether or not the instruction itself caused the skip (as in a SKIPA) or if the PC merely "fell through" the ERCAL or ERJMP instruction (as in a successful MOVE).

Users of EDDT and MDDT should be cautioned about $Xing instructions followed by an ERCAL or ERJMP in non-zero sections - the monitor has a tendency to transfer control to the error address in section 0, which will cause a BUGHLT because DDT (running in executive mode) does non-zero section things thinking it is still in a non-zero section.

10.7 $X'ing an INIT

DDT will now let the user $X an INIT (TOPS-10) monitor call. DDT will print out <SKIP 2> if the INIT fails or <SKIP 3> if the INIT succeeds.

10.8 $X speed up

By building into DDT a table of instructions which can cause the state of the known world to change, and assuming the state of the world does not change if the instruction being $X'ed is not so marked, the time required to $X an instruction is cut by roughly a factor of 10. This results in a dramatic performance increase especially for EDDT on KL-10's where waiting for the console front end to switch between secondary and primary protocol is very time-consuming.

10.9 Repetitive $X'es

The $$X command now takes an optional address range. Normally $$X will terminate when the user PC inclusively enters the range .+1 to .+SKPMAX (default value of SKPMAX is 3). The user may specify lwr<upr>$$X where lwr is the lower address boundary and upr is the upper address boundary which, if the user PC ever inclusively enters the range so specified, terminates the $$X. If only lwr is specified then upr defaults to lwr+SKPMAX. This command is very useful for recovering from having $X'ed a (for example) PUSHJ instead of having $$X'ed the (for example) PUSHJ.

10.10 $X'ing from instr$X

If the user $X'es a return from a subroutine which was entered by doing an instr$X (for example "PUSHJ P,SUBRTN$X where SUBRTN has a breakpoint in it) then DDT simply "returns" from the original instr$X rather than proceding to $X the internals of DDT itself. This is a very obscure condition so don't worry too much about it.

ingle-stepping the program

0 $$X status

DT will now respond to a ? character being typed during an $$X se-
uence by typing "Executing: " followed by the current user "pc" and
nstruction being executed. Typing any other character terminates the
$X immediately.

).12 $X PC

ie $. command now acts like the . command only $. returns the value
f the $X PC (i.e., the address of the next instruction to be $X'ed).
ie $$. command returns the previous $. value (useful for $$.<$$X as
) section 10.9 above).

:.0 Searches

)st of the differences in how DDT version 41 handles searches are
.mply bug fixes, not major changes in the logic of searching.

.1 Non-existant pages

)T version 41 now simply skips over pages which don't exist in the
dress space being searched, rather than terminating the search as
on as a hole has been found.

.2 Missed matches

e bug which caused TOPS-20 DDT to miss many valid matches is fixed
DDT version 41.

.3 Effective address searches

nce almost all address calculations start with an IFIW basis (with
e exceptions being such things as interrupt vectors and the like on
-10's or KS-10's), DDT version 41 will assume that each word it
amines is an instruction and perform an IFIW effective address cal-
lation. The final result must match in all 30 bits (actually
ternally DDT will do a full 36-bit compare so the address being
arched for had better not contain anything in bits 0 to 5).

.4 Address limit defaults

th the advent of extended addressing and physical addressing the
dress limits are defaulted somewhat differently than from previous
rsions of DDT:

EDDT, MDDT (TOPS-20 only), UDDT, and VMDDT

1. Lower Limit: <current section>,,0
2. Upper Limit: <current section>,,777777

'ILDDT looking at an .EXE file

1. Lower Limit: 0
2. Upper Limit: highest virtual address mapped

3. FILDDT looking at a data file

    1. Lower Limit: 0
    2. Upper Limit: highest word written in file

4. FILDDT looking at disk structure/unit

    1. Lower Limit: 0
    2. Upper Limit: highest word in disk structure/unit

5. FILDDT looking at runing monitor

    1. Lower limit: 0
    2. Upper limit: 777777

6. FILDDT looking at physical memory (TOPS-10 only)

    1. Lower Limit: 0
    2. Upper Limit: Highest extant memory address

As with any defaults not all cases will be properly "guessed" by DDT. In particular if the user has mapping or address relocation in effect the virtual address range so produced may have nothing whatsoever in common with the address limit defaults chosen by DDT.

11.5 Search matches

DDT will leave each address matched by its search on the "pc stack" available to $<CR> etc. commands. When the search is terminated DDT will set "." to the last address searched.

11.6 Searching status

DDT will now respond to a ? character being typed during a search by typing "Searching: " followed by the current location and value being searched. Typing any other character terminates the search immediately.

12.0 Watching

DDT version 41 allows the user to "watch" a location, waiting for it to change. Although primarily useful for FILDDT'ing the running monitor, it is present in all flavors fo DDT for completeness. The syntax of the watching command is exp$V, where exp is the address to be watched. If no explicit address is specified the last location opened by the user will be used.

Upon initial issuance of the $V command the location is displayed. Thereafter the location is continuously monitored, and will be displayed every time its contents change. In user mode DDTs (and this includes TOPS-20 MDDT as well) the location is checked once a clock tick (approximately 50 to 60 times a second), in exec mode EDDT the location is continuously being monitored - no "pause" is attempted.

DDT will respond to a ? character being typed during an $V sequence by

y g "Watching: " followed by the current location and contents
eing watched. Typing any other character terminates the $V immediate-
y.

3.0 Zeroing memory

he algorithm used by DDT previous to version 41 has only limited
sefulness in today's modern virtual world (especially on TOPS-20).
owever, to avoid "breaking" already extant control or MIC files which
ay use the $$Z command it remains unchanged. A new command has been
mplemented - lwr<upr>exp$z where lwr is the lowest (starting) ad-
ress, upr is the highest (ending) address, and exp is the 36-bit
uantity to deposit in each word inclusively bounded by lwr and upr.
oth lwr and upr must be specified. If exp is not specified then 0 is
ed as the default.

special note: The creation of zeroed pages (which formerly were non-
istent) by the $Z and $$Z commands is under the control of the
tomatic page create flag (i.e, the $1W and $$1W commands - see sec-
on 3.3.2).

T will now respond to a ? character being typed during an $Z
quence by typing "Zeroing: " followed by the current location and
lue being "zeroed". Typing any other character terminates the $Z
mediately.

.0 Special masks

T version 41 (it actually started with DDT version 40) has several
w "masks" (for lack of a better name and/or command) of interest to
e user. None of these masks are currently displayable (e.g., "$3M/")
FILDDT although they may be set normally.

.1 $0M - Search mask

e operation of the search mask continues unchanged. The search mask
y now be referenced by either the $M (old style) or the $0M com-
nds. The default value remains 777777777777.

.2 $1M - TTY control mask

is mask controls special TTY behavior (primarily TOPS-10 and exec
de EDDT).

.2.1 Tab separator display
t 17 controls whether DDT will print its usual <TAB> or three spaces
r the <TAB> separator. A 0 (the default) selects three spaces, a 1
lects a <TAB>.

.2.2 Tab simulation
t controls tab simulation. A 0 selects literal <TAB> characters
.e., the terminal handles <TAB>'s directly, a 1 selects space-fill
stead. This condition is automatically set for user mode DDT's (in
r mode <TAB>s are always output literally) - it is only useful to

manually set tab simulation in exec mode EDDT.

14.2.3 Rubout control
Bit 35 controls rubout (and ^W) operation. A 0 selects "hardcopy" operation (DDT will echo a "\" character and the character being deleted), a 1 will cause rubouts to echo as a backspace, space, backspace sequence. This condition is automatically set for user mode DDT's (if TTY DISPLAY is set then rubouts echo as <BS><SP><BS>) - it is only useful to manually set fancy rubouts in exec mode EDDT.

14.3 $2M - Offset range

The 36-bit "mask" in this case is really a value, used as the maximum offset allowable for typing addresses in the form symbol+offset. The default offset is 1000 (octal).

14.4 $3M - Byte mask

This mask is used in conjunction with the $O command for typing bytes in a word that are not necessarily evenly spaced. Whenever an $O command is issued without an explicit byte size the byte boundaries are determined by one-bits in the byte mask - each one bit in the byte mask marks the low order bit of a byte. Bit 35 is always considered on. The default value is 0 (i.e., one 36-bit byte). For example the DDT command 040100200401$3M sets the byte mask for typing right-justified 8-bit bytes (preceded by the leading 4-bit byte).

15.0 RADIX-50 symbol typein

Since prehistoric times DDT has supported RADIX-50 symbol typein, but that fact was never documented. The syntax for using a RADIX-50 symbol as an 36-bit item in an expression is sym$5" where sym is the desired RADIX-50 symbol. For example, to search for all occurences of the symbol PAT.. the DDT commands 37777,,-1$M (only look at low-order 32 bits) and PAT..$5"$W suffice.

16.0 New DDT runtime information

Several new words have been added to DDT's runtime table describing the state of the machine upon (executive mode only) DDT-entry. These words are all accessible via the DDT command $I+offset (not available in FILDDT):

1. $I-01/ APR CONI word
2. $I+00/ PI CONI word
3. $I+01/ Mask of PI channels turned off by EDDT
4. $I+02/ Executive virtual address of EPT
5. $I+03/ Executive virtual address of UPT
6. $I+04/ Executive virtual address of CST
7. $I+05/ Executive virtual address of SPT
8. $I+06/ Original AC-block word (DATAI PAG) if acb$4U

7. Obsolete commands

ie executive mode paper tape facilities (^R, $J, and $L DDT commands) re no longer supported. The code is left in the source file for eference purposes but will soon be removed.

8.0 FILDDT startup and commands

ILDDT is a special version of DDT with the facilities for "DDT'ing" idress spaces other than its own, such as disk files and in particu- ir .EXE files. FILDDT has existed for years but has always been off the background as a specialized "tool" for the exclusive use of onitor programmers looking at crash dumps. With DDT version 41 FILDDT now a general purpose utility for use by the "general public", irticularly people who have databases resident in disk files (.REL les for example).

8.1 Symbols

t of efficiency considerations FILDDT builds the symbol table(s) it ll actually use at runtime in its own address space. Virgin FILDDT s no symbols (the symbol table (if any) for FILDDT in FILDDT.EXE is mpletely independent of the address space being FILDDT'ed and does t count). There are special commands to instruct FILDDT to extract n' build internal-to-FILDDT copies of) symbol tables from .EXE files e below). Once FILDDT has setup its internal symbol table(s), it y then be SAVEd with the internal symbol table(s) for later use by iting to monitor level (with the ^Z FILDDT command) and typing the AVE" command.

8.2 TOPS-10

en FILDDT is started it will prompt "File: ". The user may at this me optionally enter a standard TOPS-10 file specification in the rm dev:name.type[directory]/switch. At least one function switch is ndatory. SFD's are of course legal in the directory specification.

8.2.1 /D command

e /D command or function switch instructs FILDDT that the file ecified is a data file - i.e., do not map the file as an .EXE file d use real file words 0 to 17 for locations 0 to 17.

8.2.2 /F command

e /F command or function switch instructs FILDDT to "DDT this file yway". It is useful only in conjunction with the /S command or func- on switch which normally re-prompts for another file specification. ed in conjunction with /S (which implies an .EXE file) FILDDT will e the file from which symbols were extracted as the file to be DT'ed".

8.2.3 /H command

e /H command or function switch instructs FILDDT to type out a brief lp text, abort the current command, and prompt the user for another mmand.

18.2.4 /J command
The /J command or function switch is applied to a job number rather than a file specification and instructs FILDDT to "DDT" the address space of the job specified. Since FILDDT uses JOBPEK monitor calls to access the specified job's address space the success or failure of any given memory reference is dependent on the job being resident in main memory - if the job is swapped out or if the memory reference is to a page which is paged out the memory reference will fail. This is a privileged command.

18.2.5 /M command
The /M command or function switch instructs FILDDT to "DDT" the currently running monitor and physical memory address space (controlled by use of the $U and $$U commands). This is a privileged command.

18.2.6 /P command
The /P command or function switch instructs FILDDT to enable for writing as well as reading the specified address space. Note that DDT's internal fake registers are always writable.

18.2.7 /S command
The /S command or function switch instructs FILDDT to only extract the symbol table from the file specified, replacing any symbol table FILDDT may already have. Unless overridden by the inclusion of a /F command FILDDT will, after having read the symbol table, again prompt the user for the next FILDDT command.

18.2.8 /U command
The /U command or function switch is applied to a file structure or disk unit only rather than a complete file specification and indicates to FILDDT that the user wants the entire physical address space represented by that file structure or disk unit name independent of any "file structure mapping" normally imposed by the monitor. This is a privileged command.

18.3 TOPS-20

With DDT version 41, FILDDT on TOPS-20 runs in native mode, and in particular, uses the PMAP monitor call for all regular file access. FILDDT will also type a brief message telling what address space is about to be "DDT'ed" before going into DDT mode.

18.3.1 DRIVE command
The format of the DRIVE command is:

DRIVE (FOR PHYSICAL I/O IS ON CHANNEL) c (UNIT) u

The DRIVE command allows examination of the disk unit u on system channel c without regard for whether it is mounted as part of a file structure, or indeed whether it even has the necessary information so that it could be so mounted (as if the HOME blocks were wiped out). If, however, the drive is part of a mounted file structure, FILDDT will type a message indicating the structure to which it belongs. This is a privileged command.

8.3.2 ENABLE DATA-FILE command
he ENABLE DATA-FORMAT command instructs FILDDT to treat the file as ure data, even if a valid .EXE directory is detected, and in particu- ar to use real file words 0 to 17 as locations 0 to 17.

3.3.3 ENABLE PATCHING command
ie ENABLE PATCHING command instructs FILDDT to enable any subsequent- / specified address space for patching (writing). This command is gnored when looking at the running monitor since there is no monitor ill to "poke" the running monitor.

3.3.4 EXIT command
ie EXIT command instructs FILDDT to return to command level. If ILDDT has an internal symbol table (due to a previous LOAD or GET ILDDT command) then a SAVE command will save FILDDT with the symbols e-loaded.

3.3.5 GET command
ie format of the GET command is:

GET (FILE) filespec (optional switches)

ie GET command instructs FILDDT to set up the disk file filespec as ie address space to be "DDT'ed", as modified by the optional switches previous ENABLE commands. The available switches are:

.3.5.1 /DATA - The /DATA switch is equivilent to a previous ENABLE TA-FILE command.

.3.5.2 /PATCH - The /PATCH switch is equivilent to a previous ENABLE TCHING command.

.3.5.3 /SYMBOL - The /SYMBOL switch instructs FILDDT to extract sym- ls from the specified .EXE file before "DDT'ing" the file, discard- g any symbols that FILDDT may already have. This switch is legal ly with .EXE files.

.3.6 HELP command
e HELP command instructs FILDDT to type out a short summary of the ailable FILDDT commands.

.3.7 LOAD command
e format of the LOAD command is:

LOAD (SYMBOLS FROM) filespec

e LOAD command instructs FILDDT to extract symbols from the disk le filespec, which must be an .EXE file, then to return to FILDDT mmand level. This command is legal only for .EXE files.

3.8 PEEK command
PEEK command instructs FILDDT to use the currently running monitor the address space to be "DDT'ed". The address space so available is rently limited to monitor executive virtual addresses 0 to 777777, ice the PEEK monitor call will only accept 18-bit address arguments executive virtual addresses. Physical memory addressing is not

available. This is a privileged command.

18.3.9 STRUCTURE command
The format of the STRUCTURE command is:

STRUCTURE (FOR PHYSICAL I/O IS) str:

The STRUCTURE command instructs FILDDT to use as the address space to be "DDT'ed" the entire disk file structure str independent of any "file structure mapping" normally imposed by the monitor. This is a privileged command.

18.4 Defaults

Following is a list of the various defaults supplied by FILDDT:

1. DSK: is the default file device unless super I/O is specified (which requires an explicit file structure or disk unit name).
2. .EXE is the default file type or extension unless either a data file or super I/O is specified, in which case there is no default file type or extension.
3. The default directory is the user's default directory.
4. The specified address space is read-only.
5. If "DDT'ing" an .EXE file and FILDDT does not already have a symbol table, extract the symbol table (if any) from the .EXE file first.
6. If "DDT'ing" an .EXE file and the symbol CRSHAC (if TOPS-10) or BUGACS (if TOPS-20) exists, give a "free" CRSHAC$5U or BUGACS$5U command. If the CRSHAC/BUGACS symbol does not exist then use file words 0 to 17 (if any) as mapped by the .EXE directory for locations 0 to 17. For TOPS-20 only, if the symbol SPT exists then also give a free SPT$6U command as well.

18.5 Other FILDDT-specific commands

Following are the commands which are unique (or different) to FILDDT.

18.5.1 ^E command
The ^E command instructs FILDDT to exit the current address space and prompt the user for a new address space. The ^E command is equivilent to a ^Z, START command sequence.

18.5.2 ^Z command
The ^Z command instructs FILDDT to exit to monitor level after having written out any changes to the current file (if any). It is most important that the user exit only via ^Z (or ^E which does an implicit ^Z) in order to guarantee the integrity of the file data (if any) - a ^C can leave a file in an indeterminate state (some changes written out to the disk and some not).

18.5.3 I/O errors
Should FILDDT incur an I/O error reading or writing a disk file, a warning message will be issued but FILDDT will otherwise ignore the error. This is to allow the user the ability to manually fix a file with bad data by rewriting the data correctly (hoping the rewriting

P____tion clears the error condition - if the physical disk surface tself is at fault, then it is probably hopeless).

SUBROUTINE CALLING - JSYS

Monitor-call JSYSes may be used in user or monitor code. All ACs are preserved over a JSYS call unless an explicit statement to the contrary appears in the JSYS description. ACs are changed over a JSYS call only when values are to be returned to the caller.

The JSYS name shall appear as the opcode in the statment which performs the call. The JSYS mnemonic includes the instruction field, so no other fields are supplied by the user.

Unimplemented JSYSes will invoke the illegal instruction sequence (with error code ILINS2). Defined and implemented JSYSes will return to caller +1 on success, or will invoke the illegal instruction sequence on failure. The illegal instruction sequence recognizes an ERJMP or ERCAL following the failing JSYS and causes the appropriate action. If the following instruction is not an ERJMP or ERCAL, an illegal instruction interrupt is requested which will be handled by the executing fork if enabled, or otherwise cause a forced fork termination. See paragraph below on JSYS returns for proper indication of JSYS failure.

All constant values, bits, and fields of JSYS arguments shall have mnemonics defined according to the rules in MONSYM. The JSYS code itself shall use these symbols for loading arguments, testing bits, etc.

When writing code to implement a JSYS, the following conventions shall be observed:

1. The entry point of the JSYS is defined by a global tag which consists of a DOT concatenated with the symbolic name of the JSYS, e.g. .GTJFN::.

2. The first statement of the JSYS code shall be MCENT (Monitor Context ENTry). This establishes the normal JSYS context for a "slow" JSYS. At this writing, MCENT is a null macro and the JSYS entry procedure is invoked automatically. The use of MCENT is required so that this implementation may be changed in the future if necessary.

3. All caller ACs are automatically preserved by the entry and exit procedures. Therefore JSYS routines are specifically required NOT to save and restore the ACs. The contents of the caller's ACs 1-4 are copied into the callee's ACs. No callee ACs are copied back to the caller's AC block on return however; one of the "previous context" instructions* must be used to return any values to the caller. E.g.,

```
UMOVEM T1,T1        ;store monitor T1 into user T1
```

A previous context instruction may also be used at any time to fetch the original contents of the caller's ACs unless they have been explicitly changed by a previous context store operation. E.g.,

```
UMOVE T2,T1         ;load user T1 into monitor T2
```

- -

* - UMOVE, UMOVEM, XCTU [instruction], etc.

4. Return from JSYS code should be effected by the statement

```
MRETNG          ;Monitor RETurn Good
```

This transfers to the JSYS exit sequence (returning caller +1) and should be used to indicate successful completion of the JSYS. If the JSYS could not be completed successfully, the following statement should be used:

```
ITERR errcod    ;causes an Instruction Trap
                ;ERRor, leaves
                ;the error code in LSTERR
```

Certain other statements are defined which effect JSYS returns according to a previous convention. They are:

```
RETERR errcod   ;RETurn ERRor, return
                ;caller +1 with error code
                ;left in AC1 and LSTERR

EMRETN errcod   ;Error Monitor RETurn, return
                ;caller +1 with error code left
                ;in LSTERR
```

These should not be used in new JSYS code but may be needed if existing JSYSes are modified.

All error returns shall include an error code (mnemonic) which shall be defined in MONSYM.MAC. If the appropriate error code has already been loaded into AC1, then the errcod field may be omitted from the above and the contents of AC1 will be taken as the error code. No JSYS shall return other than +1 or instruction trap, therefore no occurrance of AOS 0(P) should ever be required in JSYS code.

When invoking a JSYS error return, it is not necessary to pop temporary quantities from the stack. The successful return however, should be given only when the stack is properly cleared.

SUBROUTINE CALLING - Internal monitor routines

The allocation of ACs for all inter- and intra-module subroutine calls shall be:

ACs 1,2,3,4 — General temporary, may be clobbered by subroutine.

ACs 0, 5-15 — Preserved, not changed by subroutine (or saved and restored if necessary).

AC 16 — Temporary, used by JSYS call/return procedure and reserved for use by other call/return procedures.

AC 17 — Global stack pointer

Call and return shall be effected by PUSHJ P, and POPJ P, respectively. A set of assembler mnemonics has been defined for subroutine mechanics as follows:

'CALL' (= PUSHJ P,) shall be used to call subroutines, e.g. CALL SUBR.

'RET' (= POPJ P,) shall be used to return +1 from subroutines.

'RETSKP' shall be used to return +2 from subroutines. RETSKP is equivalent to:

```
JRST [  AOS 0(P)
        RET]
```

'RETBAD errcod' shall be used to return +1 with an error code from a subroutine. The error code field is optional as with JSYS error returns above. RETBAD is equivalent to:

```
JRST [  MOVEI A,ERRCOD
        RET]
```

'CALLRET' may be used to call a subroutine and return immediately thereafter. It is an abbreviation for

```
        CALL SUBR
        RET
```

or

```
        CALL SUBR
         RET
        RETSKP
```

Note that CALLRET is not guaranteed to be a single instruction; therefore it may not be skipped over. The other returns above are guaranteed to be single instructions.

These mnemonics are used to emphasize the FUNCTION being performed (calling, returning) rather than the mechanics of the function (pushing, jumping, etc.). Also, these mnemonics could continue to be used even if a more general calling standard were adopted at some time in the future.

Return may also be effected by transferring control to the global tag R or RSKP, e.g.

```
JUMPE A,R          ;equivalent to JUMPE A,[RET]

JUMPN A,RSKP       ;equivalent to JUMPN A,[RETSKP]
```

The general temporaries shall be used for passing arguments to subroutines and returning values. AC1 shall be used for a single argument routine, ACs 1 and 2 for a two-argument routine, etc.

A routine defined to return caller +2 (skip) on success and caller +1 (noskip) on failure is acceptable. Returns greater than caller +2 are not permitted.

AC DEFINITIONS

The following mnemonics have been chosen to be consistent with the AC use conventions above. The preserved ACs are divided into three groups, F (1 AC) intended for Flags, and Q1-Q3 and P1-P6 intended for general use. The ACs within each group are consecutive.

0 - F	10 - P1
1 - T1	11 - P2
2 - T2	12 - P3
3 - T3	13 - P4
4 - T4	14 - P5
5 - Q1	15 - P6
6 - Q2	16 - CX
7 - Q3	17 - P

The programmer should assume that each group (Tn, Qn, Pn,) is in ascending order, e.g. that T2=T1+1, but that the specific assignment of numbers may change. Explicit numeric offsets from AC symbols (e.g. T1+1) should NEVER be used. Instructions which use more than one AC (e.g. DIV, JFFO) must be given an AC operand such that the other AC(s) implicitly affected are in the same group. E.g. T3 (and T4) is OK for IDIV because T3+1=T4, but Q3 is not because Q3+1=??.

There are several facilities in the monitor to save and automatically restore ACs. Each of these will save all of the indicated ACs on the stack at the point of execution and will place a dummy return on the stack which causes these ACs to be restored automatically when the current routine returns. Use of these facilities eliminates the need for matching PUSH/POP pairs at the entry at exits of routines and eliminates the bugs which often arise from an unmatched PUSH or POP. The available macros are:

SAVEQ - saves ACs Q1-Q3

SAVEP - saves ACs P1-P6

SAVEPQ - saves ACs Q1-Q3 and P1-P6

SAVET - saves ACs T1-T4

Defining a different mnemonic for a preserved AC may be of value when the AC is used for a specific function by a large body of code. However, it offers the possibility of confusion because two different symbols may refer to the same AC unbeknownst to the programmer. In smaller programs, use of certain ACs can be restricted to specific functions, and a global definition is appropriate. A timesharing monitor however, is too large to accomodate all of the possible dedicated ACs.

Therefore, when a specific function-oriented AC definition is made, it shall be explicitly decided which modules shall use the definition. Within these modules, the usual name for the AC must be purged so that there is no possibility of using two different symbols for the same AC.

Only preserved ACs may be used for special definitions. Parameters to subroutines may be passed in functionally defined ACs in the following cases:

1. On an intra-module call where the contents of the AC are appropriate to its function definition.
2. On an inter-module call where the same definition exists in both modules and the AC is being used for its intended function.

A parameter may NOT be passed in a preserved AC unless both caller and callee know it by the same name, and that name must be a specific one related to the function which the AC is performing.

The procedure for declaring a functionally defined AC is:

```
DEFAC NEWAC,OLDAC
```

This must be done at the beginning of an assembly, and it defines NEWAC to be equal to OLDAC. OLDAC must be the mnemonic for one of the regular preserved ACs, and this mnemonic will be purged and therefore unavailable for use in the current assembly.

An AC with a special definition should not be used for other purposes; e.g. "JFN" should not be used to hold some quantity other than a JFN merely because it happens to be available.

SUBROUTINE DOCUMENTATION

The following is a suggested format for documenting the calling sequence of a JSYS or subroutine. A description of this sort should appear at the beginning of every subroutine, no matter how short.

```
;name of subroutine - function of subroutine, etc.
; T1/ description of first argument
; T2/ description of second argument
;   ...
;       CALL NAME or JSYSNAME
; RETURN +1: conditions giving this return,
;  T1/ value(s) returned
; RETURN +2: conditions and values as above.
```

1. The arguments, if any, should be documented as the contents of registers and/or variables as shown. MONSYM mnemonics should be used when available; e.g. at JSYS entry points.

2. The actual instruction to do the call should be shown. This will be "CALL subname" in the case of internal subroutines, and the single-word JSYS name in the case of a JSYS entry point.

3. The return(s) should be noted as shown; "ALWAYS" or "NEVER" may be used as the condition where appropriate; the +2 return need not be shown if it does not exist; values returned should be described in the same form as arguments.

Examples:

```
;SIN - COMPUTES SINE OF AN ANGLE
; T1/ ANGLE IN RADIANS, FLOATING POINT
;       CALL SIN
; RETURN +1: FAILURE, UNNORMALIZED NUMBER OR OUT OF
RANGE
; RETURN +2: SUCCESS, T1/ VALUE, FLOATING POINT

SIN::   ..

        - - - - - - - -

;GJINF - GET JOB INFORMATION JSYS
;       GJINF
; RETURN +1: ALWAYS,
;  T1/ LOGGED-IN DIRECTORY NUMBER
;  T2/ CONNECTED DIRECTORY NUMBER
;  T3/ JOB NUMBER
;  T4/ TERMINAL NUMBER OR -1 IF DETACHED

.GJINF:: ..
```

MULTI-LINE LITERALS

The use of multi-line literals is encouraged as a technique for making code more readable and easier to follow. The following additional rules apply:

1. The opening bracket for a multi-line literal should occur in the position that the first character of the address field would have appeared if the instruction had an ordinary address, e.g.

```
SKIPGE FOO
JRST [
```

2. The first and all following instructions within the literal shall begin at the second tabstop, e.g.

```
JRST [  MOVE A,MUMBLE  ;COMMENT
        JRST FIE]      ;COMMENT
```

The tab between the open bracket and the first opcode may be omitted if the line position is already at or beyond the second tab stop, e.g.

```
JUMPGE A,[MOVE A,MUMBLE
```

3. The closing bracket shall follow the last field of the last instruction (as above), and shall be before the comment on the same line.

4. Nesting of multi-line literals to a depth greater than one is discouraged because of awkward formatting problems.

5. Tags may not appear in multi-line literals.

6. No hard and fast fules can be given as to when to use or not use multi-line literals. However, a literal longer than about 10 lines becomes suspect.

7. Use of ".+1" is legal in a literal to return to the main sequence.

FLOW OF CONTROL - BRANCH CONVENTIONS

In general, jumps should be to tags forward (down the page) from the point of branch except for loops. Tops of loops should be identified by comment.

The expressions ".+1" and ".-1" are the only legal uses of "." (this location). All other potential uses should be avoided in favor of an explicitly defined tag.

"Global" jumps should be avoided altogether. Higher-level languages do not permit them and with good reason. The only exceptions are jumps to well defined and published exit sequences, e.g. .R, RSKP (see subroutine conventions, above).

NUMBERS

In general, there should be no occasion to use a literal number in in-line code. All parameters, bit definitions, CONO/CONI codes, etc. should be defined mnemonically at appropriate places. It is much easier to err in the direction of too little use of mnemonics rather than too much; therefore, when in the slightest doubt, define a mnemonic.

Appendix A
LIVING IN AN IMPERFECT WORLD

Much of the present TOPS20 code was written before the existence of this standard and therefore does not conform to it. A great deal of systematic editing has already been done to improve conformance, but obvious irregularities exist. In general, new code being added should conform exactly to this standard even if being integrated with old code. The following are some specific problems which may arise and the recommended solutions:

1. AC Mnemonics

Some code uses absolute numeric ACs. If new code is being integrated into a sequence which uses numeric ACs, it is desirable that the existing code be edited to use the standard mnemonics, particularly for the preserved ACs. If the programmer cannot take the time to do that, then the mnemonics T1-T4 should be used for ACs 1-4, and other ACs should be referenced in the same way as is done by the existing code.

Some code uses mnemonics A,B,C,D for the temporary ACs. These same mnemonics should be used for new code integrated into this existing code, or all references can be edited to use the standard mnemonics.

You may write some code using the standard mnemonics for preserved ACs and then discover that the module into which you wish to put this code has redefined some of these ACs. The solution is one or a combination of the following:

1. Move the new code to a module which does not redefine the preserved ACs.

2. Use different preserved ACs -- ones which have not been redefined. (Note it is not acceptable to use an AC with a special definition for other than its special purpose.)

Clearly, code which needs some of the special definitions must be placed in a module which has these ACs defined and must therefore use only the other preserved ACs.

Note that a value which usually resides in a special AC need not ALWAYS reside there. For example, if code in JSYSF needs to call a routine in PAGEM and pass a JFN index as an argument, the JFN should be loaded into T1-T4 for the call since PAGEM does not have JFN defined and cannot accept an argument in it.

2. Stack Handling

Use of the several stack variable facilities defined in MACSYM is recommended. Some old code uses explicit PUSH and POP and references of the form -n(P); however, and when anything more than trivial modifications must be made to such code, it is most strongly recommended that the code be edited to use STKVAR or TRVAR. Failing that, references must be consistent with the existing code.