A 13

Xerox Universal Time-Sharing System (UTS)

Sigma 6/7/9 Computers

System and Memory Management
Technical Manual

90 19 86A -1

Xerox Corporation
701 South Aviation Boulevard
El Segundo, California 90245
213 679-4511

XEROX

Xerox Universal Time-Sharing System (UTS)

Sigma 6/7/9 Computers

System and Memory Management

Technical Manual

FIRST EDITION

90 19 86A

February 1973

Price: $4.25

Printed in U.S.A.

NOTICE

This publication documents the system and memory management functions of the Universal Time-Sharing System (UTS) for Sigma 6/7/9 computers. All material in this manual reflects the C01 version of UTS.

RELATED PUBLICATIONS

Title	Publication No.
UTS Overview and Index Technical Manual†	90 19 84
UTS Basic Control and Basic I/O Technical Manual	90 19 85
UTS Symbiont and Job Management Technical Manual	90 19 87
UTS Operator Communication and Monitor Services Technical Manual	90 19 88
UTS File Management Technical Manual†	90 19 89
UTS Reliability and Maintainability Technical Manual	90 19 90
UTS Interrupt Driven Tasks Technical Manual†	90 19 91
UTS Initialization and Recovery Technical Manual	90 19 92
UTS Command Processors Technical Manual	90 19 93
UTS System Processors Technical Manual	90 19 94
UTS Data Bases Technical Manual	90 19 95

†Not published as of the publication date given on the title page of this manual. Refer to the PAL Manual for current availability.

CONTENTS

ID

SSS - Scheduler

PURPOSE

The Scheduler selects users for execution, those for swapping in and those for swapping out.

OVERVIEW

The scheduling and swapping algorithms in UTS depend upon user states. Each user in the system is in one and only one state at a time. The transition from state to state is driven by events reported to the scheduler. I/O activity, time quanta, break signals and other events are signaled to the UTM scheduler by calls on the report event routines (T:REG, T:RCE, T:RE and T:RUE) with an appropriate event code describing what has happened, and the number of the user with whom the event is associated.

Each state has a corresponding state number and a (possible empty) queue of users in that state. Each state queue is doubly linked, and is ordered by time of user arrival in that state.

The state queue headers are contained in SB:HQ, a byte table, which is indexed by state number and contains the user number of the first user in the queue for that state. SB:TQ is the corresponding table of queue tails. All queues are linked (forward and backward) through two user tables, UB:FL and UB:BL. UB:FL contains the user number of the next (chronologically) user in the given state. UB:BL contains the user number of the previous user in the given state. A 0 indicates the end of a forward or backward chain.

The scheduler selects a user for execution by following the queues of users from head to tail in a given order until it finds a user in core and ready to run. The order in which the queues are searched is dictated by the table SB:EXU which contains the queue numbers and is terminated by a 0. It is currently:

```
SB:EXU    DATA, 1    0, SNRRT, SON, SOFF, SERR, SEC, SBK, SIR, STOC, SC,
                     SCOM, SBAT, 0
```

SB:EXU is also searched by the swap scheduler to determine which user to swap in next (the first user encountered who is not ready to run).

SB:SWP is used in the same manner by the swap scheduler to determine the order in which the queues are searched for users to swap out only the search through each queue is from tail to head. SB:SWP currently is:

```
SB:SWP    DATA, 1    0, SSYMF, SSYMD, SW, SQEI, SQA, SDP, STI, STOB, SAB,
                     SIOW, SOCU, SBAT, SCOM, SIOC, SC, SBK, SEC, SERR,
                     SOFF, SON
```

Note that the queues SCU, SIOIP, SLS, STOBO and STIO do not appear on either list and thus users in these states are not selected for either execution or swap.

Example: Users 6, 4, 2 and 8 have all hit the break key on their teletypes in that order. Meanwhile users 7 and 9 are compute bound. The break state number is 4 and the compute state number is 8. The corresponding table structures are shown in Figure EA-1.

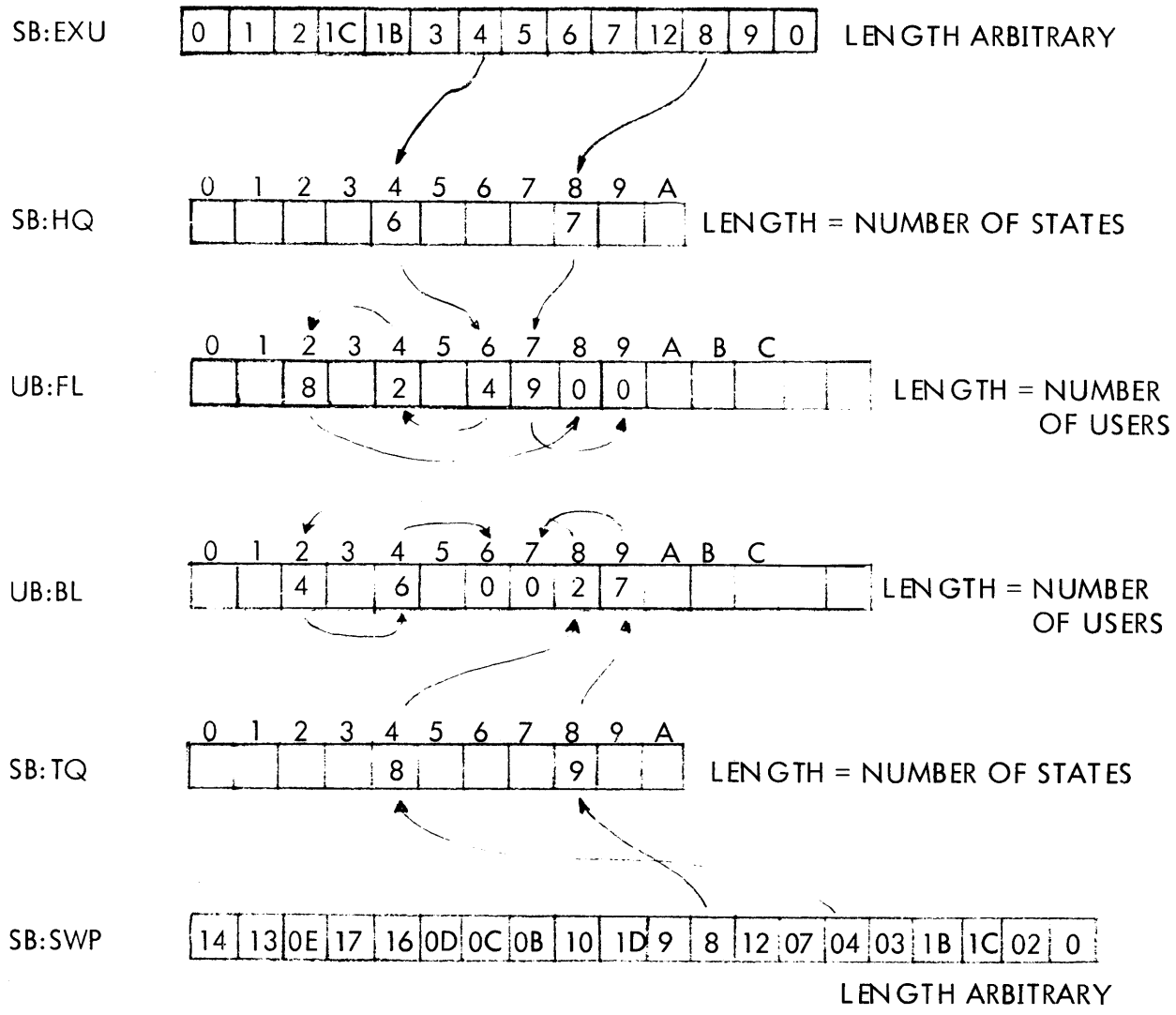


FIGURE EA-1 - Example Table Structure

The tables which control the action to be taken, given an event on a user are S:SET, SB:SET, and SB:ETT.

S:SET can be considered as a matrix with the bits in each row corresponding to states. The number of rows (1 or more) corresponds to the possible actions for a particular event given the variety of states in which a user may be when that event is reported on him. SB:SET is a byte table with one entry corresponding to each row in S:SET. SB:ETT is a byte table indexed by event number containing a pointer to the first of the one or more rows in S:SET corresponding to that event. If the first row in S:SET for the event reported (pointed to be SB:ETT) contains a 1 bit in the position corresponding to the current state of the user the event concerns, the byte in SB:SET corresponding to that word tells the event handler what to do. If that bit is zero, the next row of S:SET is tested. Theoretically there will be a 1 bit posted in one of the rows corresponding to the event in the bit position of the user's state. SB:SET controls how far the search through S:SET can continue. If the high order bit of the byte corresponding to a word in S:SET is 1, it is meaningful to look at the next word. If it is 0, the search should not continue. If no 1 bit is encountered in the correct position of S:SET, it is "impossible" to have that event reported on a user in his current state. This condition results in software check 0.

A schematic picture of the relation of the tables follows.

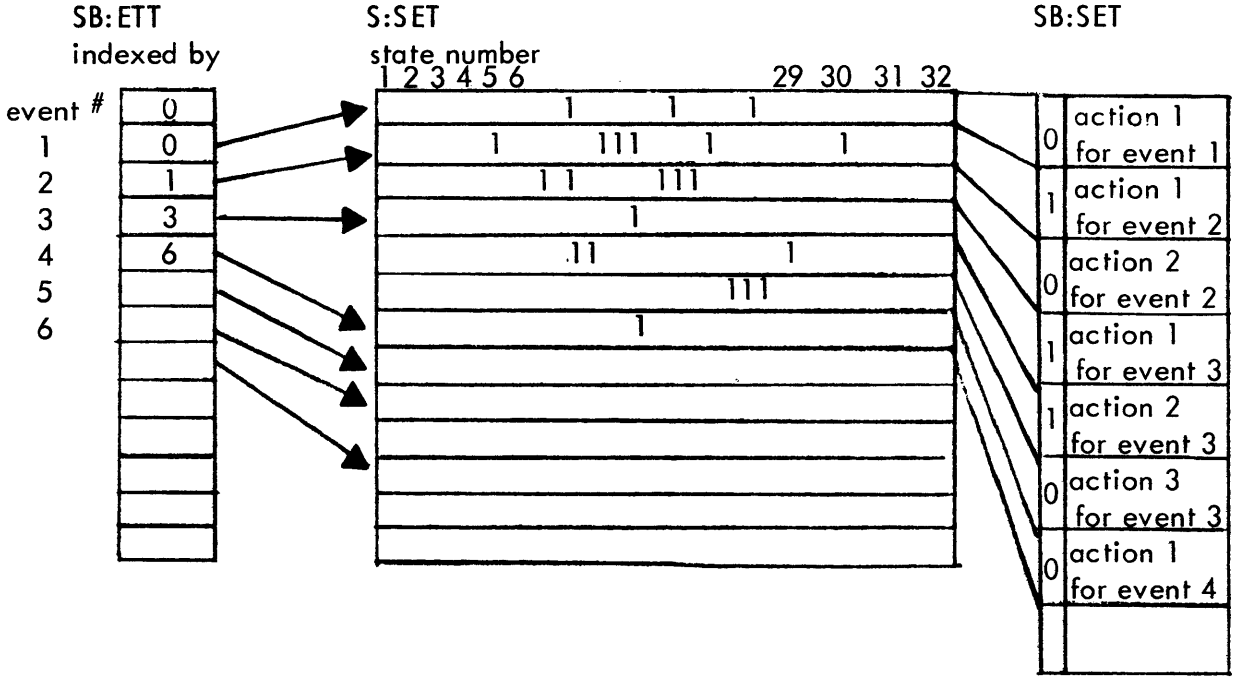


Figure EA-2 shows the details of the current S:SET and SB:SET tables. The events are listed at the left. The states are listed on top. The multiple rows possible for a given event have been compressed into one row. The contents of SB:SET is shown in those bit positions where a 1 bit is set in the row of S:SET corresponding to the SB:SET entry. If a byte of SB:SET contains a number less than 40 (high order bit ignored) it is the number of the state the user should be changed to (indicated in the figure by the state name). If the byte contains 40 or greater it is the number of a special transition event (the event requires more than a simple state change). An * is indicated instead of 40 since the special transition 40 means ignore the event. The bit positions containing nothing indicate that the event is impossible for users in the corresponding state.

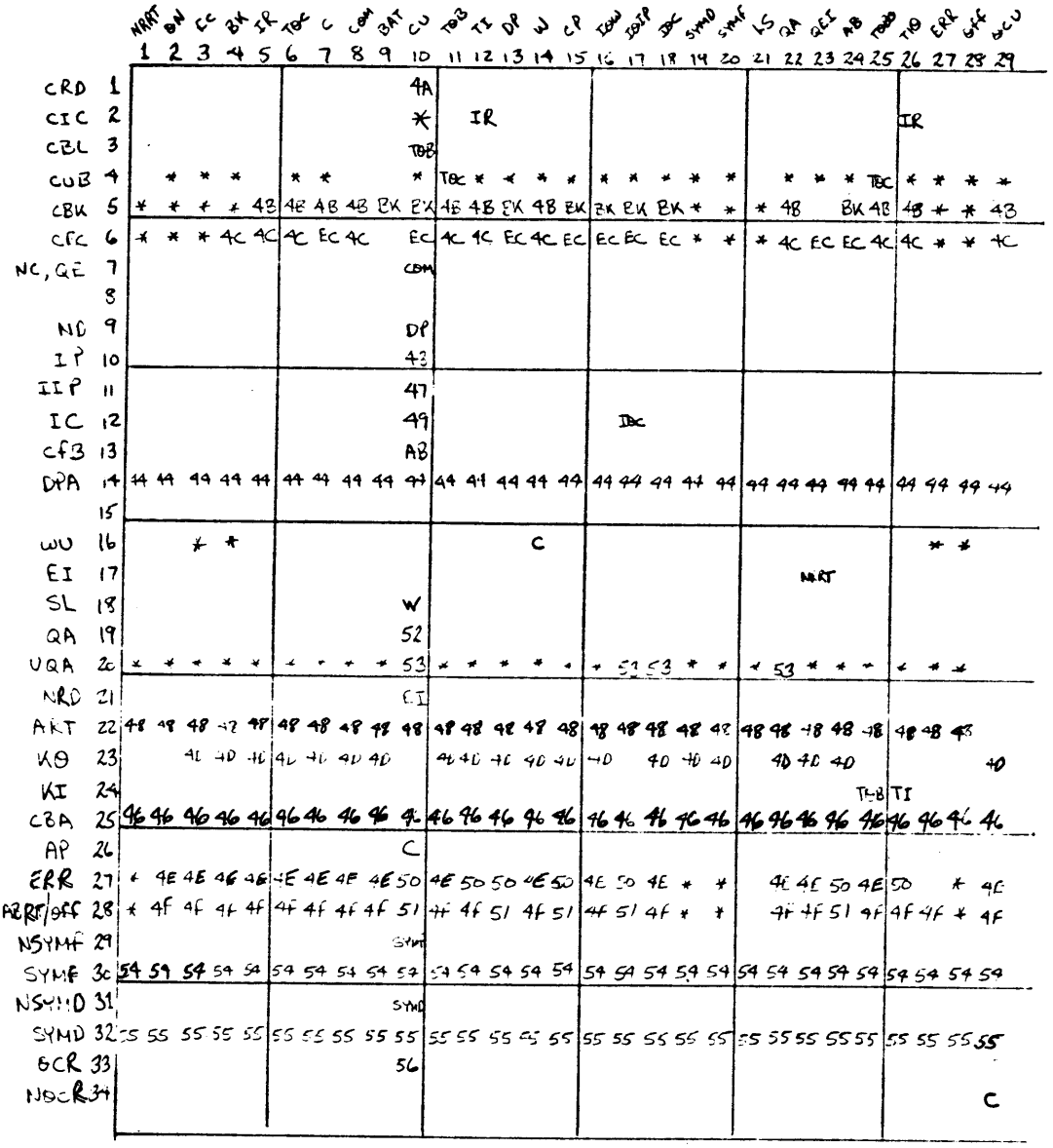

NART BN EC BK IR TBC C COM BAT CU TOB TE DP W CP IDW IDIP IBC SYMD SYMF LS QA QEE AB TOBP TD ERR GFF BCU
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
CRD 1
CIC 2
CBL 3
CUB 4
CBK 5
CFC 6
NC, QE 7
8
ND 9
IP 10
IIP 11
IC 12
CF3 13
DPA 14
15
WU 16
EI 17
SL 18
QA 19
UQA 20
NRD 21
ART 22
K0 23
KI 24
CBA 25
AP 26
ERR 27
ABRT/GFF 28
NSYMF 29
SYMF 30
NSYMD 31
SYMD 32
BCR 33
NBCR 34
4A
IR
TOB
TBC
4B
EK
BK
EC
4C
COM
DP
43
47
49
IBC
AB
44
W
52
53
C
NRT
TBC
TI
56
* = 40 = NOP

Figure EA-2

The special transition events are listed below for your convenience. Detailed explanation will be included in the description paragraph.

Mnemonic	SB:SET	Action	Special Transition Routine invoked
STNOP	40	IGNORE	none
STSBK	41	SET BRK BIT	SETBRK
STSEC	42	SET EC BIT	SETEC
STIP	43	ASK IO PERMISSION	IOPER
STDPA	44	DP→C IF ANY USER IN DP	DISCFREE
STOFF	45	SPIN	--
STBA	46	AB →TOC IF ANY USER IN AB	COCBA
STIIP	47	CU →IOIP	IOINPRG
START	48	SPIN	--
STIC	49	SET 8000 FLAG	IOCCU
STCRD	4A	CU →TI	CHCRD
STSBKC	4B	SET BRK BIT, STATE→BK	SETBRKC
STSECC	4C	SET EC BIT, STATE→EC	SETECC
STKO	4D	RESET FLAGS FOR OUT OF CORE=ADJUST STATE	KICKOUT
STSERRC	4E	SET ERR BIT, STATE→ERR	SETERC
STSABRTC	4F	STATE →OFF, ADJUST GJOB & COC TABLES	SETABRTC
	50	SET ERR BIT	SETERR
STSABRT	51	SET ABORT BIT, ADJUST GJOB & COC TABLES	SETABRT
STQA	52	IF QA, RETURN, IF NOT STATE →QA	QFORA
STUQA	53	IF QA, QA →C, OTHERWISE SET 4000	UQFORA
STSYMF	54	SYMF →C IF ANY USER IN SYMF	SFILEAV
STSYMD	55	SYMD →C IF ANY USER IN SYMD	SDISCAV
STSIOC	56	STATE → COM/BAT IOIP → C/IOC	IOCOM
STSOCU	57	IF ANY USER OPENING/CLOSING, CU →OCU	

DATA BASES (Summary)

S:SET	the state event transition table
SB:SET	a byte table of opcodes associated with S:SET
SB:ETT	a byte table of indexes into S:SET and SB:SET
SB:EXU	a byte table of queues to search, in order, for users to execute or swap in
SB:SWP	a byte table of queues to search for users to swap out
SB:HIR	a byte table of queues of users who are to be given control after the current user has had a minimum quantum rather than waiting for quantum end.
SB:HQ	a byte table, indexed by state number containing the user number of the first user in a given state; 0 implies none
SB:TQ	a byte table, indexed by state number, containing the user number of the last user in that state

The scheduler also uses certain of the user tables. They are:

UB:FL	a forward link pointing to the next user in the same state
UB:BL	a backward link to the previous user in the same state
UH:FLG	a halfword table containing user flags
U:MISC	a miscellaneous word for each user
UB:APR, APO, ASP, DB, OV -	the users associated shared processors (root, processor overlay, special shared processor, debugger and monitor overlay).
UB:US	the user's current state
UB:PCT	the user's page requirement
UH:TS	the remainder of a user's quantum

ID

Event handler - Event reporting and state changing

PURPOSE

The event handler portion of the scheduler receives notification of events from various portions of the monitor and changes the scheduling states of the users accordingly.

OVERVIEW

The event handler consists of three external entry points to serve the rest of the monitor in reporting events. These entry points diagnose the reported event and either call subroutines to perform simple state changes or they drive to the special transition event routines which eventually call the simple state changing subroutines.

USAGE

Entry Points

T:RCE	Entry to report COC event on a specified line. Registers 6 and 7 contain the event and line number, respectively. Only COCC uses this entry.
T:RUE	Entry to report event on a specified user. Registers 5 and 6 contain the user number and event number.
T:RE	Entry to report event on the current user. Register 6 contains the event number.

All routines are called by a BAL, 11

SUBROUTINES

Special Transition Event Routines, Section EA.01.01
State Change Routines, Section EA.01.02

External:

RECORD	Event recorder, Section LF
COCOFF	Initialize line for logging off, Section DC.01
RECOVER	Initiate system recovery or single user abort, Section LD

ERRORS

If an event is reported which is impossible for the user's current state, a software check 0 is reported to RECOVER.

DESCRIPTION

T:RCE, T:RE, T:RUE

T:RCE is entered by COCC to report COC events. RCE extracts the user number from LB:UN into register 4. If the user number is 0, meaning the user has not yet logged on, LOGON is called. At RCE0 the event counter is incremented because an event has occurred. An event number greater than the number of events causes a software check 0. The user's current state is extracted from UB:US. A 1 bit is shifted in register 12 to the bit position corresponding to the user's current state for comparison with S:SET. The index into S:SET is extracted from SB:ETT indexed by the event number (R6). The byte from SB:SET corresponding to S:SET is placed into register 2. Now S:SET is checked to see if it has a 1 bit in the position corresponding to the user's state. If so, register 2 contains the action code. If not, SB:SET (R2) is checked to see if the searching can continue. If not, software check 0 results. If so, the search continues.

Assuming the correct byte in SB:SET is found, the high order bit is scrubbed off and an entry is made in the event recorder. If the code in SB:SET is $\geq$ to X'40', a branch is executed through the transition vector S:TRNSVEC to the appropriate routine. Otherwise T:CHS is called to change the user's state to the state in register 2.

T:RE is entered to report events on the current user so register 4 is loaded with the current user's number and control goes to RCE0.

T:RUE is entered with the user number in register 5 so it is loaded into Register 4 and control goes to RCE0.

ID

Special Transition Event Routines

PURPOSE

When the event handler determines from the SB:SET table that more than just a simple state change is required, one of the special transition event routines is called to perform the more complex operation.

USAGE

The special event code from SB:SET, which is greater than or equal to X'40', is used to branch through a transfer vector. Each routine returns to the event reporter (caller of event handler).

DESCRIPTION

IOINPRG/IOCCU

are designed to handle the case in which the I/O interrupt occurs and reports I/O complete before IOQ has reported I/O in progress. If IOC is reported on the current user, post the 8000 flag in UH:FLG and return. When IOIP is reported the flag is checked. If it is set, IOC event has been received and the IOIP event is ignored. Otherwise the user's state is changed from CU to IOIP.

SETABRTC/SETABRT

Both these routines set the abort flag. SETABRTC also changes the user's state to OFF. If the user is an on-line user, COC is informed through COCOFF that he is in the process of being logged off. The distinction between using these two routines is there are some cases in which the user's state cannot be changed to OFF because he would get scheduled out of that state and in fact the user is blocked waiting for something to occur.

SETERR-SETERRC

These routines set the error flag for the user. If SETERRC is invoked, the user's state is changed to SERR. The discussion under SETABRT for the distinction applies here too.

SETBRK-SETBRKC
SETEC-SETECC

Same as SETERR-SETERCC, except the flags set are break or Y^C and the state change is to SBK or SEC.

DISCFREE-COCBA

these routines are invoked when swapping RAD granules or COC buffers are available. All users in the SDP or SAB queues are changed to SC or STOC, respectively, to allow them to be scheduled to acquire the resource they were waiting for.

QFORA/UQFORA

QFORA is called to block a user while waiting for a tape mount. Normally it changes the user from SCU to SQA. QFORA changes his state from SQA to SC when the mount is done. In the case that the un-queue event occurs before the queueing event, the 4000 flag is posted in UH:FLG. When the E:QA event occurs, the flag is noticed, it is reset and the user is not blocked.

SFILEAV-SDISCAV

are called when a symbiont file entry or a symbiont disc granule become available. The state of the first user in SSYMF or SSYMD is changed to SC.

KICKOUT

is called when a user is going to be swapped out. First his state is checked to insure that it hasn't changed between the swap schedulers decision to swap him out and the reporting of the event. If so, the event is reissued. If not, his ready-to-run and JIT-in-core flags are reset. His state is checked. If it is STOB or STI, his state is changed to STOBO or STIO respectively. If his state is in SB:EXU, S:SIR, the number of users who are out of memory who could execute if they were in memory, is incremented. If his state is in the SB:HIR list, S:HIR, the count of high priority users ready to run, is decremented.

OCPRGM

checks if there is a user doing an Open or Close. If not, it returns. If so, it changes the user's state to SOCU.

IOPER

always grants permission to do I/O.

IOCOM

checks the user's flags for the 4000 bit. If it is set, the user is at job step and his state is changed to SC to allow him to continue at a high priority. If the 4000 bit is reset, his state is changed to SIOC.

ID

State Change Subroutines

PURPOSE

Once it has been determined that a user's state is to be changed and what it is to be changed to, the state change routines are called. They change the user's state, remove him from his present scheduling queue and add him to his new queue.

DESCRIPTION

T:CHS changes a user from a given queue to the bottom of a new queue.

Input Reg 4 = user number
Reg 3 = current state
Reg 2 = new state
Reg 0 = BAL register
Reg 1 = is destroyed

T:CHS first checks to insure that the user's state hasn't changed during the event handling process. If so the event is re-issued to determine what to do, given the user's new state. If not, the state change proceeds with interrupts inhibited to prevent user state changes. S:SIR and S:HIR are corrected next. If he is ready to run and his current state is not in SB:HIR and his new state is in SB:HIR, S:HIR is incremented. If he is not ready to run and his current state is not in SB:EXU and his new state is in SB:EXU, S:SIR is incremented. Next Performance Measurement is called if his new state is in SB:EXU. Next T:UNQ is called to remove him from his current state and T:QT is called to add him to his new queue. T:CHS exits.

T:CHST changes the user's state to the top of a new queue. It sets the high order byte of register 2 to non-zero to cause T:QT to turn control over to T:QH to queue the user to the top of the new queue instead of the bottom. T:CHST then transfers control into T:CHS.

HT is the routine called to determine whether to change a batch user's state to SCOM or SBAT, and to determine whether to change an on-line user to SC or SCOM. It is invoked at quantum end. If the user's flags have the 4000 bit set, the user is on-line and at job step time and is changed to SC. If not, he is changed to SCOM unless he is a batch user and SL:BB, the batch bias, is 0, in which case he is changed to SBAT.

T:UNQ removes a user from a queue.

Reg 4 = user number
Reg 3 = state number
Reg 1 = BAL register
Reg 6-7 = are destroyed

T:QT adds a user to the tail of a queue unless the high order byte of Reg 2 is non-zero in which case it calls T:QH.

Reg 4 = user number
Reg 2 = new state number
Reg 1 = BAL register
Reg 0 = is destroyed

T:QH is called by T:QT to add a user to the head of a queue.

Registers same as T:QT.

SHIRE checks to see if the user's state is in a given byte list (SB:HIR, SB:EXU, SB:SWP). It returns to +1 if so, otherwise +2.

Reg 3 = state
Reg 5 = pointer to byte table
Reg 2 = BAL register
0-1 destroyed

LOGON has five entry points and is called to add a user to the system. (Note this LOGON is not to be confused with the LOGON processor.)

ENTRY POINTS

1. LOGON is called by T:RCE when it has an event with 0 in the byte of LB:UN corresponding to the line number it was given. It logs on a user with an on-line JIT.
2. ADD1 logs on a user whose number is in register 4.

At LOGON the event is checked to see if it is a break. If not, the event is ignored (on automatic dial up COC reports break). Next the zeroth byte of SB:HQ, i.e., the queue of available user slots is tested. If 0, the event is ignored. Next the number of on-line users in the system is compared to the number of on-line users allowed and the number of users in the system is compared with the number of users allowed. If either test indicates no new room for new users, the event is ignored. If the user is to be allowed, register 4 is loaded with a user number and control passes to ADD1.

ADD1 is called by MBS, T:GJOBSTART and LOGON to allocate a JIT granule on the swapper for the user whose number is in register 4. The current value of M:JITPAGE for this swapper is used as a target position for this user's JIT. The M:JITPAGE entry is then updated by a value obtained from MB:SPACEJIT - a number chosen to be relatively prime compared with the number of granules per track for this swapper. Next the number of users in the system is incremented. The granule acquired is placed in UH:AJIT. The swapper recognizes that UH:JIT, the disc address of the users JIT, is 0 only at logon time. It will swap the JIT in from the disc address in UH:TS and move the disc address from UH:AJIT to UH:JIT for use in the next swap-out. The user flags, special JIT access, and pure procedure swap are set. Special JIT access is posted so that the exit CAL pointed to by the environment assembled into the virgin JIT will be interpretively executed, i.e., so that the processor required (LOGON, CCI) will be associated. PPSWP is posted as a flag to the swapper. The user's page requirement, UB:PCT, is set to 1. The event flag S:EVF is incremented as an event has occurred and the user's state is changed from 0 to SON. LOGON exits.

ID

Execution Scheduler

OVERVIEW

The scheduler has three major entry points: T:REG, T:SSE, and T:SSEM. T:REG is called when it is determined that a user no longer requires the CPU. T:REG "blocks" the user by saving his environment, reporting the appropriate event, causing the user's state to be changed from current user to some other state, and causing some other user to be scheduled for execution (T:SE).

T:SSE and T:SSEM are called whenever the Monitor has been in execution, executing either synchronous or asynchronous processes. Control must come here to allow re-scheduling. If the process was asynchronous, it might have caused a change in the system requiring rescheduling (COC, Clock, or I/O interrupts). Synchronous processes (CAL's primarily) must allow rescheduling since events such as break, Y^C or quantum end would only use flags to be set if the Monitor is performing some service for the user. These events are remembered until the Monitor completes its processing. They are checked for at T:SSEM to allow the appropriate rescheduling.

USAGE

T:REG is called with register 11 containing the address at which the user is to begin execution when he is rescheduled and register 6 containing the "blocking" event.

T:SSEM is merely branched to.

T:SSE is also branched to.

INTERACTION

T:ACCTOV - ACCT, Section IC
T:ACCTEX - ACCT, Section IC

DESCRIPTION - T:REG

T:REG unloads the current user's PSD, puts register 11 (the address of the next instruction to be executed for that user) into it and saves the environment in the TSTACK in the user's JIT. It calls T:RE to report the blocking event. If the event is not I/O in progress, it calls T:SS to cause a swap schedule if S:SIR, the number of users on the swapping RAD who could execute if they were in memory, is non-zero. It then branches to T:SE to give some other user the CPU. If the event was I/O in progress, the possible swap-set has not changed so the swap schedule is avoided.

DESCRIPTION - T:SSEM/T:SSE

T:SSEM is entered at the end of every monitor service for a user. It calls T:MASTER. T:MASTER causes the interrupted environment to be pulled if it was master mode. This software disable prevents the monitor from being interrupted out of a series of CAL's. It takes care of overhead accounting by calling T:ACCTOV. Control goes to SSE1.

T:SSE is called by asynchronous routines like COC when it has reported an event, the clock 3 routine or the I/O routine handler. First, it sets the idle flag in case the system is currently idle. Next, if S:SIR is non-zero, it calls the swap scheduler. It then calls T:MASTER. This software disable results in the monitor never getting a process interrupted. It always exits cleanly. If the interrupted environment was slave, a test is made to find out if the monitor is currently mapped or unmapped by looking at the user ID in JIT. If it is 0, it is an unmapped JIT. If it is unmapped, the user environment in the unmapped stack is transferred to the user's mapped stack and the map is turned on.

T:SSE and T:SSEM come together at SSE1. Clock 4's effective address is switched to J:DELTAT, the execution time counter. (If exiting the monitor, it was pointing to the overhead counter J:OVHTIM.) Next, tests are made in order for an operator X key-in, and !E Key-in, bad run status, a user Y^C, and a user break. If the user was executing in the Monitor at the time one of these events occurred, the system merely poisted a flag to be honored on the way back to the user, namely, here.

Next, quantum tests are made. If the user has quantum ended, his execution time is accounted for and his state is changed to SBAT or SCOM, depending upon the batch scheduling philosophy and whether he is batch or on-line. This happens at SSE6 and control goes to the execution scheduler. If the user has some time left, he is allowed to continue by going to T:PULLE unless S:HIR, the count of high priority users ready to run, is non-zero. If S:HIR is non-zero, the current user is checked for having had a minimum quantum. If not, he is allowed to continue by going to T:PULLE. If he has had his minimum, how much time he has remaining is remembered in UH:TS and he is moved to the top of SCOM or SBAT unless he has less than 40 mils. left in which case accounting is done and his state is changed to bottom of SCOM or SBAT.

T:SE is the execution scheduler. It turns off the map since there is no longer a current user and causes a swap schedule if appropriate. It goes to SACT in case a symbiont output device needs starting. It then runs through the queues looking for a user ready to execute. It searches the queues in the order listed in SB:EXU, searching each queue from head to tail until it finds some user with his ready-to-run flag set. If it finds someone, control goes to SE1. If there is no one ready to execute, the system goes idle. The idle routine goes to CHECK for a security check. It then tests to see if there are any users in the system (S:CUIS). If not, it checks to see if there are any symbionts active. If not, it tests to see if the system is already quiescent. If not, it writes the HGP tables to the system RAD, sets the QUIESCENT flag and types the quiescent message on the operator's console. Finally, it tests LOSTUSER to see if any users were deleted

without closing their DCB's. If there were, files may still be open, so a software check 1E is forced to allow recovery to test the CFU's for open files and close them properly.

The idle loop does performance measurements according to whether idleness is a result of swapping in a user in the SCOM, SIOC, or SBAT state (idle swap in progress) or not and toggles the audio flip-flop at varying frequencies to keep the operator entertained.

At SE1 the user to put into execution has been selected. T:XMMC loads the map from the user's CMAP. At T:SE, the event counter was saved in register 0. Here it is checked to see if it has changed. If so, a reschedule takes place since an event has occurred which may have changed the queue structure while it was being run. If it hasn't changed, the user's number is put into S:CUN, and this previous state is remembered. S:HIR is decremented if the user's state is in SB:HIR and his state is changed to current user. The map is turned on.

The fact that he is in execution is logged (RECORD). His total time in this shot is calculated and if greater than or equal to SL:SQUAN, the don't-swap flag is reset. His quantum is calculated (batch or on-line UH:TS). If he has just swapped in or is going into execution from one of the interactive user states (SIR, STOC), his previous execution time is accounted and he is given a new quantum. CLOCK4 is changed to overhead. If the environment in the user's JIT is master, control returns to the Monitor (T:MASTER). If the environment is slave, the accounting is switched to execution and his previous state is checked to see if he's been aborted or errored by the operator or has hit Y^C or break. If not, T:PULLE is called to put him into execution, unless he has done a STIMER CAL in which case control is transferred to his specified address.

If the operator aborted the user, his flags are reset. Then his account is checked to see if he has logged on yet. If not, blanks are put into his name and account as a signal to Logoff. J:RNST is set to X'10' and an extra 19 words are pushed into his stack. Control transfers to T:RUNDOWN in STEP.

If he has been errored, his flags are cleared, J:RNST is set to X'20', the error code is set to B403, and control transfers to T:TELDELCCI in STEP.

If he has hit Y^C, his flags are cleared and control transfers to T:ECB described later.

If he has hit break, the break flag is reset. If TEL is in control, breaks are treated like Y^C. If DELTA is in control, control transfers to it (DELTAGO). If DELTA is associated DELTA in control is set. If there is no public library associated (UB:ASP), control transfers to DELTA (it's in his map). If there is a public library associated, it is disassociated and DELTA is associated, then control transfers to it. If DELTA was not associated and the user has asked for break control (J:INTENT), control goes to him (ALTENT); otherwise, TEL is called (T:ECB).

SUBROUTINES

ALTENT copies the user's environment from his TSTACK into his TCB if he has one with sufficient room (otherwise, error A300).

Register 3 contains the entry address. It is called for STIMER CAL entries and break control entries.

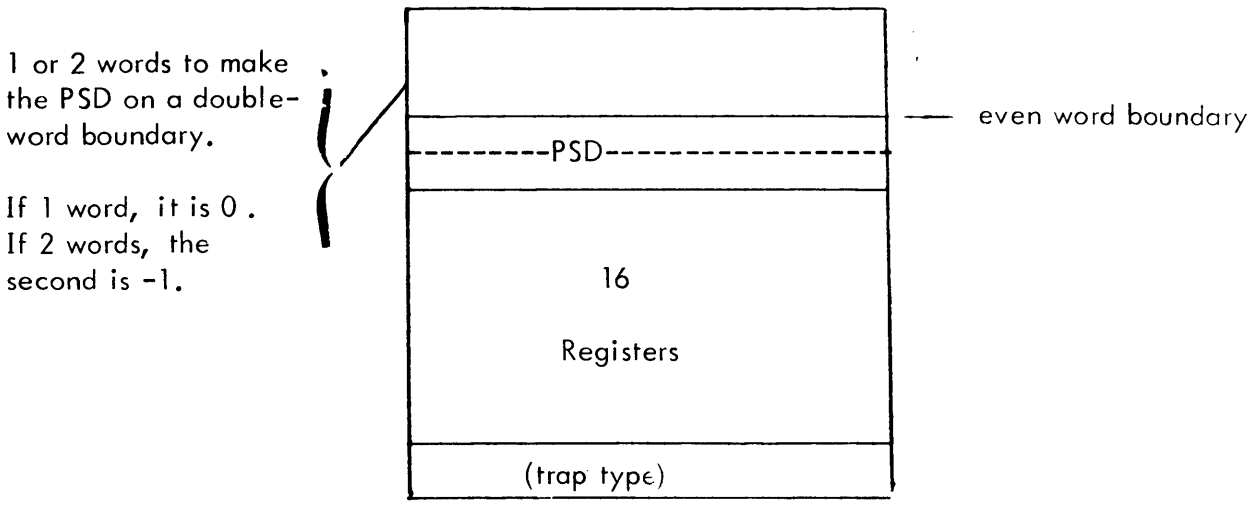
ALTENT first checks to see if the user has a TCB. If so, T:UTSXTS is called to push the 19 word environment into it. It puts the address of the PSD into register 1 and branches to T:PULLA to enter the user at the address contained in register 3 when ALTENT was called. If the TCB is not present or adequate, control transfers to ALTERR to give the user an A300 error message.

DELTAGO enters DELTA at the entry point specified by register 10. DELTA has three entry points which are in a transfer vector at words X'C', X'D', and X'E' of its context page. C is the normal entry, D is for break control, and E for traps.

DELTAGO sets the DELTA-in-control flag and calls T:PAC to get the appropriate access image into the user's special processor area. It forces the user's TSTACK to 19 words. It calls T:UTSXTS to copy the user's environment into DELTA's stack located at SPDBASE, the first word of DELTA context. It puts the contents of J:RNST into register 8, zeroes J:RNST and enters DELTA at the address originally contained in register 10.

T:UTSXTS copies the environment from the user's TSTACK into the stack pointed to by register 1. It pushes either 20 or 21 words depending upon whether the last word in the stack is even or odd.

The format is:



Register 1 points to the stack into which the environment is to be pushed.

Register 4 is the BAL register. Return is to
1, 4 if the push won't work.
2, 4 if the push was O. K.

Registers 0, 2, 3, 7, 8, 9, 11, and 15 are used.

T:UTSXTS first calls CHKPROT three times to verify that the stack pointer double-word and the first and last words to be pushed into belong to the user and are in 00 protection type. Then, it makes sure there is sufficient room in the stack to push 20 or 21 words. It then pushes in the one or two words and makes the last one 0 or -1, indicating how many more to pull out. Then, the PSD is pushed. The 16 registers are transferred and one more word is pushed in afterward. If the reason for the T:UBXTS call is a trap, the trap type will be put there. T:UTSXTS exits.

CHKPROT checks a given page to see if it has 00 protection.

Register 7 contains the word address of any word in the page.
Register 8 contains the error return address.
Register 9 is the BAL register.

CHKPROT converts the contents of register 7 to a page number and calls T:IACU in MM. T:IACU returns the protection type in the condition codes. CHKPROT returns *8 if 1 or 2 are set; otherwise, *9.

T:CHGPSD changes the instruction address of the PSD in the last environment in TSTACK.

Register 0 is the BAL register.
Register 1 contains the new instruction address.
Registers 2, 3, and 4 are destroyed.

T:PULLE pulls a 19 word environment out of TSTACK. If loads the registers and then loads the PSD. It is merely branched to.

T:PULLA changes the address in the environment to the address in register 1, then goes to T:PULLE. It is branched to.

T:MASTER checks the Master/Slave bit of the PSD in the last environment in TSTACK. If the environment is master, it pulls it by going to T:PULLE. If it is slave, it returns.

Register 2 is the BAL register.
Register 0-1 are destroyed.

T:ECB is the routine which goes to TEL on a break or y^c. If the user has the TEL-in-control flag set, it ignores the event and returns to TEL. If not, it pushes 19 words into TSTACK so that when an environment is pulled to go to TEL, the user's environment at the time of the event will remain in the stack in case of a CONTINUE command. It then posts a flag for TEL in J:TELFLGS so that TEL can know it is in control because of a break or y^c. Control goes to T:EC.

T:EC is the routine which invokes TEL, CCI or LOGOFF if J:RNST is X'10' (operator abort or LOGOFF). If the user is a ghost job, SB:GJOBFLG is cleared (the ghost is not running since this must be a LOGOFF). If the user is batch, T:ASP is called to get CCI. Otherwise, the byte displacement in M:UC is cleared for TEL. If J:RNST is not X'10', T:ASP is called for TEL. Otherwise, it is called for LOGOFF (that is, LOGON since they are the same processor).

CLOCK4 - The clock4 interrupt routine is entered at the counter 0 interrupt of clock 4. It rearms the clock. If the interrupt environment is master, it returns to the point of interrupt since quantum end will be noticed at T:SSEM when the Monitor exits back to the user. If the interrupted environment is slave CLOCK4 goes mapped and the interrupted environment is pushed into the user's TSTACK. Control goes to SSE1 to accomplish the quantum end and accounting. This is not

to be confused with the module CLOCK4, Section EE, which is the time of day clock and actually uses clock 3.

ID

Swap Scheduler

OVERVIEW

The swap scheduler is called to select a user to swap in and, if necessary, a user or number of users to swap out to make room. It choses the user to swap in by running through the queues listed in SB:EXU until it finds the highest priority user who is not ready to run.

It then acquires enough pages to swap him into. First it gets all the available pages from MB:PPUT. If this is not enough, it checks for any shared processors which are not in use. If it has enough pages now, it calls SWAPIN. If it still doesn't have enough pages, it starts running through the queues listed in SB:SWP, running the queues from tail to head to find a single user who is in memory who is big enough to make room. While it is trying to find one user who is big enough it makes a list of other users it finds who are in memory. Users are not considered for outswap if they are in an executable state, are ready to run and have not had SL:SQUAN mils since they were last swapped in. If it finds a single user, it causes an outswap. If not and it has found multiple users with enough pages it swaps them out. If not it has one last chance. It decrements the shared processor usage counts for the user in the outswap list and checks if any are now free so it can take their pages. If this is enough, the swap takes place. If not, the users in memory must be the best set of users so no swap takes place.

USAGE

T:SS is the only entry point. It is called whenever an event has occurred which might result in a swap. Register 11 is the BAL register.

OUTPUT

S:ISUN	the in swap user #
SB:OSN	the # of our swap users
SB:OSUL	the out swap user numbers
S:FPPH S:FPPT S:FPPC	the head tail and count of pages acquired by the swap scheduler

DESCRIPTION

T:SS first checks S:SIP, the swap-in-progress flag, to make sure there is not a swap in progress. The swapper and swap scheduler are not re-entrant since they both use resident monitor tables. If there is a swap in progress, T:SS exits. Otherwise, it zeroes the out swap user list (SB:OSUL), and the swappers and action flag. It zeroes a performance measurement flag which is used to count types of swaps. It does an XPSD to go

unmapped and puts register 11 into the old PSD so that return will be to the caller in his mode (mapped, unmapped, etc.). RECORD is notified that a swap schedule is starting.

T:SS next picks the user with highest priority who is not ready to run. It searches the state queues in the order specified in SB:EXU, running the queues from head to tail until it finds one with the ready-to-run flag reset. If there is none, it returns by loading the PSD it saved (SSPSD). If it finds one, it checks the event flag to make sure no events have occurred during the search which might have altered the queue structure (the search loop is not disabled). When a user is successfully selected, control goes to SSIN.

S:ISUN is established as the user to be swapped in. His required page count is picked up from UB:PCT into register 14. PRCAV is called to insure that the overlay he requires, if any, is in memory. If not, it is added to the list of shared processors required (SB:PNL) and its size is added to register 14. If the users JIT is in core, the current number of pages he has (from JBPPC in his JIT) is subtracted from register 14. If his JIT was not in core, the processor usage counts (PB:UC) of all his required processors are incremented. Getting the counts right at this point will also hold in memory any processors which he requires which are in memory. This is not done if his JIT is in memory since the counts are already correct. (They were counted up when he came in or when a processor was associated.) Then PRCAV is called for all the shared processors required in order to add them to SB:PNL if they are not in and to add their page requirements to register 14, the number of pages required for the inswap user. Control transfer to SWIPEPGS.

SWIPEPGS is the routine which acquires enough memory for the inswap user. First, it picks up all the free pages from MB:PPUT. If there are exactly enough, it goes to GOTEXAC. If there are too many, it goes to GOTNUF which relinks the extra ones to MB:PPUT and falls through to GOTEXAC. GOTEXAC calls OUTIN and control goes to the swapper. If there are not enough, it calls PRCFREE which looks for a free shared processor in memory. If it finds none, control goes to USERSOUT to find some out-swap users. If it finds one, GOTPRCPG is called to add the processor's pages to the swappers page chain and control returns to check if there are enough pages. If not, it looks for another free shared processor and continues until there are enough pages or there are no more free processors in which case control goes to USERSOUT. Any shared processors in memory which the inswap user requires are **not** free since their usage counts were incremented earlier so that they would stay in memory.

USERSOUT determines the out-swap set of users. First it zeroes SB:FPN, the list of shared processors which became free because users are being swapped out, SB:OSN the number of out-swap users, and S:OSS the number of pages gained from swapping users out. USERSOUT runs through the queues listed in SB:SWP, from tail to head looking for users with either the JIT in core or the ready-to-run flag set (users who are at least partially in memory). When a user is found, control transfers to USOUT5. If the user is the current open or close user, he is not swapped. If the user's don't-swap

flag is set (he has had less than SL:SQUAN) and he's in an executable state and he's ready to run, he is not swapped. If not, he is checked to see if he is large enough. The user's number of pages is determined from his JIT (JBPPC). His shared processors usage counts are decrement and if any are free, the number of pages they are using are added to the number of pages he, the user, had. This number is compared with the number required and if there are now sufficient pages, this user is not large enough, his shared processor's usage counts are incremented and the user number is added to SB:OSUL, the list of out swap users, unless SB:OSUL already contains enough users to make room for the inswap user (S:OSS > register 14). If the user number is added to SB:OSUL, the number of pages is added to S:OSS. In either event, the search for one user who is large enough continues.

If one user is found, a check is made at USOUT6 to ensure no event occurred during the running of the state queues which could have changed them (S:EVF must be same). If an event has been reported, the user's shared processor counts are reincremented and the swap schedule starts over. A similar test is made at the end of running all the queues if no single user is found.

At USOUT11 the user number is put into SB:OSUL and SB:OSN, the number of out swap users is set to 1. Control transfers PROUT2 to get the pages from any shared processors he freed up.

If there is a list of out swap users, any one of which was not large enough, control goes to PROCSOUT. PROCSOUT decrements the usage counts of all the shared processors of the users in the outswap user list. If there are enough pages available, control goes to PROUT2. Otherwise, an attempt is made to find any processors which are free as a result of swapping out the users in SB:OSUL. If there are still not enough pages, the swap scheduler goes to GIVEUP. A swap is not possible. If some free processors are found, their numbers are put in SB:FPN, the numbers of shared processors whose pages are to be swiped. If there are enough pages, the page chains of all these shared processors are added to the swapper page chain at PROUT2 by calling GETPRCPG. At PROUT3, the kick out event E:KO is reported on all users in the out-swap list. OUTIN is called and control goes to the out swapper.

GIVEUP returns the swapper page chain to MB:PPUT, and reincrements the processor usage counts of all users in the out-swap list. If the inswap user's JIT is not in core, his processor's usage counts were incremented so they are here decremented. The user's total size is calculated to make sure his is not larger than available memory. If he is, this is an impossibility and a software check 62 is reported to SCREECH. Otherwise, the swap-in-progress flag is cleared, the swapper's page chain is zeroed and the swap scheduler exits.

SUBROUTINES

T:TOTESZ adds up the size of the user and his shared processors (excluding special shared processors and overlays).

Register 0 returns the size.
Register 11 is the BAL register.
Register 4 is the user number.
Registers 15, 2, 10 and 1 are used.

PRCFREE looks for the "next" shared processor (starting with the one pointed to by register 3 and decrementing) which has zero usage count (PB:UC). If it finds one, it returns to BAL+2, otherwise BAL+1.

Register 6 is the BAL register.
Register 3 is the starting shared processor number and returns the number of the next one found.
Register 2 and 8 are used.

PRCFREE checks PB:UC, 3 to see if it is, it checks to see if the processor is in memory (PB:HPP = 0). If it is, it checks the corresponding bit of PBT:LOCK to see if the processor is locked in memory (PBT:LOCK is currently not set by anybody). If not, return is to 1, 6 since it has found a free processor pointed to by register 3. If any of the above conditions is not met it continues searching. If it finds no free processors in memory, it returns to 0, 6.

GETPRCPG takes the pages from a free shared processor and adds them to the swapper page chain. It zeroes the head of the shared processors page chain head (PB:HPP) to mark it not in core.

Register 6 is the BAL register.
Register 3 contains the processor number.
Register 0 contains 0.
Registers 7 and 8 are used.

MISCELLANEOUS ROUTINES

DRTEL, DTEL, ISTEL, ITEL, DPROCS, DASP, DDB, DOV, RPROCS, RASP, ROV, DRPROCS, DROV, DRASP, IPROCS, IASP, IDB, IOV, DTORP, ITORP.

These routines increment or decrement processor usage counts and/or set or reset the user's associated processor bytes (UB:OV, UB:APR, UB:APO, UB:ASP, UB:DB) or his TEL-in-control flag. I stands for increment, D for decrement, S for set and R for reset. They affect TEL, all PROCS, associated special processors (ASP), debuggers (DB) and monitor overlays (OV). Two affect TEL or all processors (TORP) according to whether TEL is in control or not. They use registers 0 through 3. Register 4 contains the user number and register 15 may be used to load the user's flags.

DRPROCS, DROV, DRASP, IPROCS, IASP, IDB, IOV, DTORP, ITORP

These routines increment or decrement processor usage counts and/or set or reset the user's associated processor bytes (UB:OV, UB:APR, UB:APO, UB:ASP, UB:DB) or his TEL-in-control flag. I stands for increment, D for decrement, S for set and R for reset. They affect TEL, all PROCS, associated special processors (ASP), debuggers (DB) and monitor overlays (OV). Two affect TEL or all processors (TORP) according to whether TEL is in control or not. They use registers 0 through 3. Register 4 contains the user number and register 15 may be used to load the user's flags.

ID

STEP - Job step control

PURPOSE

The STEP routines perform all monitor operations required to allow a user to pass from one job step to the next. A job step is defined here to be from the transfer of control to the starting address of a program to its exit, error, or abort.

Command processors such as TEL and CCI provide the means by which a user specifies job steps, and their operation is considered part of the inter-job step process.

OVERVIEW

STEP recognizes two types of program exits: an exit CAL from a command processor (TEL, CCI, LOGON) and all other exit CALs. An exit CAL from a command processor is called an interpretive exit, and connotes a request for a STEP service. Registers provided at the exit are examined to determine the action to be taken. If a program or shared processor name is provided, it is associated with the user and control transferred to it. If none is specified, the exit is a request to delete the user from the system.

An exit CAL from any program other than a command processor is a request to return to the appropriate command processor, to the associated debugger, or a request to transfer control to the user's program loaded into memory by LINK.

Action taken on error and abort CALs, and aborts of a user within the monitor depends on the associated debugger. If DELTA is associated with the user, control is given to DELTA. Otherwise, the user's current operation is terminated, appropriate error/abort messages given, all programs disassociated, and the appropriate command processor associated and given control.

The overview flow chart illustrates the operation of the STEP routines.

USAGE

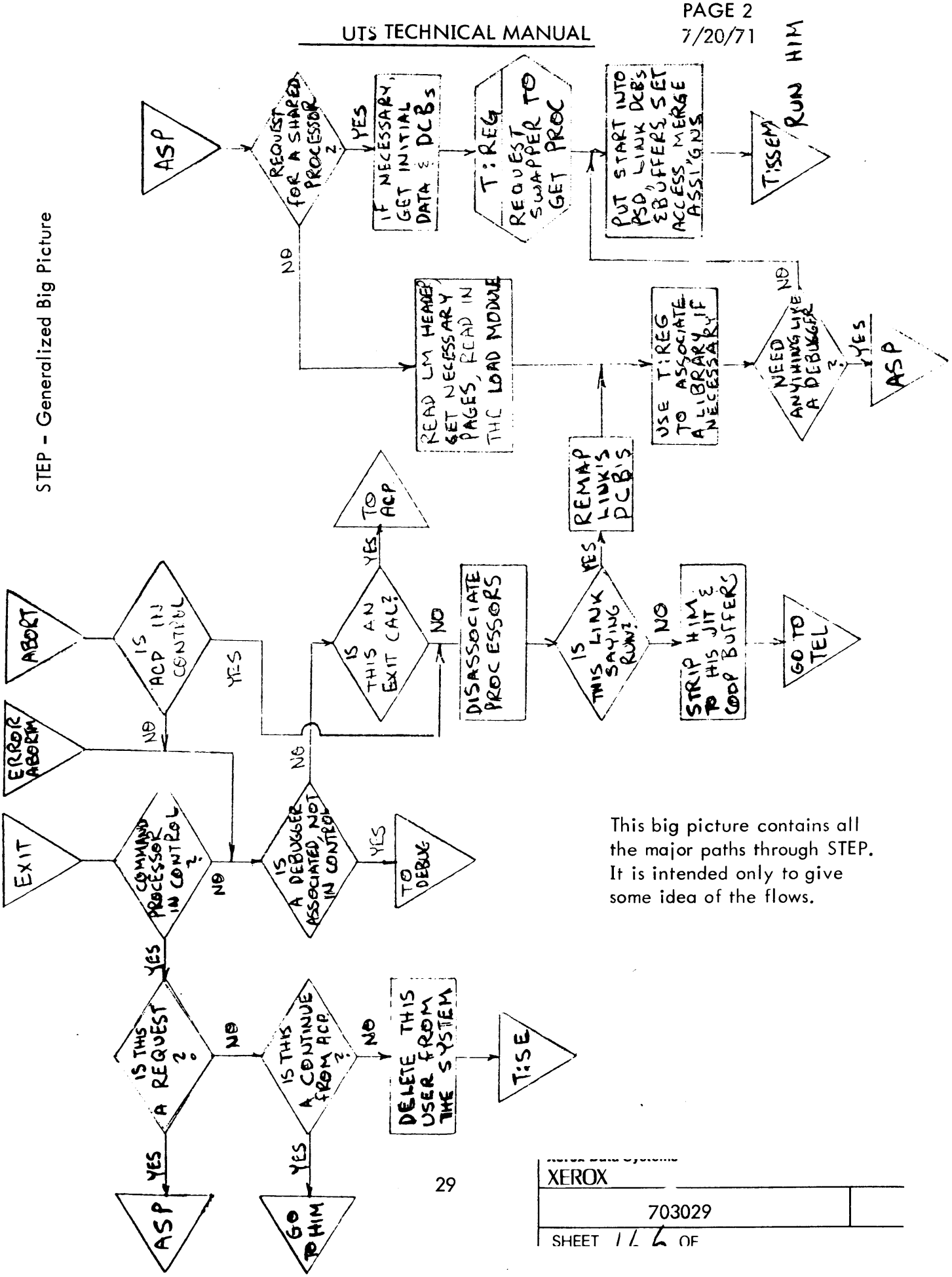
STEP usage varies from routine to routine. See the description of each logical segment to determine its usage.

INPUT

STEP uses numerous tables described in Section V. Input unique to a logical routine is detailed in the description of that routine.

The most important input to STEP is the user's exit environment containing the PSD and the 16 registers at the time of the exit. This is detailed in Part F of the description.

STEP - Generalized Big Picture



This big picture contains all the major paths through STEP. It is intended only to give some idea of the flows.

OUTPUT

STEP supplies a user who is ready to run to the scheduler. Specific output is detailed in the routine descriptions.

INTERACTION

MM Routines:

T:SGR	Release a granule on the swapping rad
T:RVPI	Release a page - physical and virtual
T:GNVNPI	Get n virtual no physical for initial data and DCBs
T:GNVPI	Get n virtual and physical pages
T:XMMC	Load memory map from JB:CMAP
T:RVSPI	Release virtual save physical for remapping LINK's DCBs
T:GVGPI	Get virtual given physical, for remapping LINK's DCBs
T:SNAC	Set access on n specified pages
T:FPP	Free a physical page
T:PAC	Load the access image
T:SAC	Set access on a page
T:GAJP	Get an additional JIT page if necessary

SSS Routines:

T:OFF	Remove the user from the system
T:BTSCHED	Schedule a batch job
T:REG	Report an event-associate processor E:AP - and give up control
T:SSEM	Schedule the current user
T:SE	Schedule with no current user
T:CHS	Change the user's state
T:EC	Associate TEL with the user
IPROCS, ISTELI, DRASP, DRTELI, DRPROCS, RPROCS, DDB, IDB, DROV, DASP	Keep track of associated processors
DELTAGO	Transfer control to DELTA

OTHER ROUTINES

Entry point		Module	Function
T:DSMT	-	T:DSMNT	Dismount all associated tapes
T:NAMECHK	-	UCAL	Compare specified name with GJOB name table
T:OV	-	T:OV	Associate an overlay
T:GHOST	-	UCAL	Inform the operator of an aborting ghost job
ABORT1	-	ENTRY	EQU to T:ABORTM

SUBROUTINES

ABN: Routine to handle I/O errors occuring while reading FETCH load module. Lost data for all operations is ignored, otherwise the error code in R10 is set into ERO in the JIT, M:XX is closed, if open after setting PMD flag in J:ASSIGN, and control passed to FETCH3, if LDOTRC was not active. Otherwise an error code of X'B5' os reported via TELLTEL.

IN: R10 contains error code
R8 contains CAL+1

BUFCHAIN: Routine to chain the user's buffers. The buffers size in words is subtracted from the top of context area, its value stored in the pool location in JIT. Size again is subtracted from the first buffer address and the result saved in first word of previous buffer for all buffers specified in the JIT. This operation is performed for FPOOL and IPOOL then returns to the calling routine.

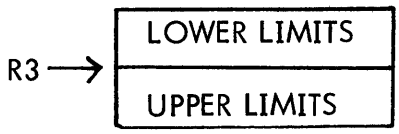
CHKDB: Routine to check P:SA given the processor and abort the user via TELLTEL unless:

a) Processor is a monitor overlay
b) The processor is a debugger.
IN: R4 contains users number
R5 contains processors number

LINKLIMS: Routine to set access on pages within a specified protection type.

IN: R1 = Relative word in head record for limits
R2 = Address of head
R3 = J:DLL or J:PLL
R13 = Protection type

LINKLIMS obtains size in R6 and starting address in R7. If size = 0, control is returned. Otherwise size and location are converted to page count and number, combined, and the first page number and last page number stored using R3:



Access is set on pages between the limits via the MM routine T:SNAC and control returned.

OUTOFPGS: Error routine to supply on X'A5' error code to TELLTEL using the logic described in ABN. OUTOFPGS loads R10 with 0 and R8 with OUTOFPGS, then falls into the ABN logic.

SPCON: Routine to obtain the address of the special processor context page in R2.

Out: R2 = J:EUP+1

TELLTEL: Routine to set the specified abort code in J:ABC, increment all associated processors via the SSS Routine IPROCS, and transfer control to T:EC in SSS which associates TEL to deliver the error message if the user is on-line. Otherwise R14 is loaded with the abort code and control passed to T:ABORTM

In: R1 contains ABC

T:RSTLMS: Routine to set all JIT memory limits to their initial values:
J:CUL = J:PUL = J:DUL
J:CUL+1 = J:PLL = J:DU = J:DDLL
J:EUP = J:DDUL

All memory is thus set to dynamic data. The upper limit lower than the lower limit indicates no pages have been obtained within that particular protection type.

XITCTRL: Routine which supplys DELTA's exit control start location to the SSS routine DELTAGO which transfers control to DELTA.

ERRORS

STEP error codes are listed in the UTS Reference Manual, table B-5. These error codes are:

A0	Invalid debugger
A1	Attempt to debug a shared processor
A5	Load module (and context) exceed user limit
A6	Load module bad or does not exist

A6	30	Bad DCBs or DCB Table
	31	Bad Head Record
	32	Load Module Bias Not on Page Boundary
	33	Pure Procedure Not on Page Boundary
	34	DCBs Not on Page Boundary
	35	Head Record in Incomplete
	36	Tree Record is Incomplete
	37	No Debugs Allowed with Link-Built LMNs
	38	Program Too Big for User Area
	39	File Not Keyed, Not a LMN
	3A	DCB Links Bad or circular
	XX	I/O Error Code on Load Module Open or Read
AA		Invalid core library name
B5		I/O error reading load module for LNKTRC

Restrictions: Details in the routine descriptions

Step entry points:

ENTRY POINT	REQUEST ENVIRONMENT	FROM (ACP)	RESULTANT ACTION
T:EXIT	Program request	Command Processor	interpretive exit
	Logging on	Logon	interpretive exit
	Logging off	Logon	delete user
	Exit with DELTA associated	user program	return to Delta
	Exit without DELTA associated	user program/ shared processor	return to ACP after re-initializing user.
	Exit from Ghost job	Ghost job	delete user
T:ERROR	Error with DELTA associated	user program/ shared processor	return to ACP after re-initializing user
	Error from Ghost job	Ghost job	delete user
T:ABORT	Abort with DELTA	User program	return control to DELTA
	Abort without DELTA	User program/ shared processor	return to ACP after re-initializing user

ENTRY POINT	REQUEST ENVIRONMENT	FROM	RESULTANT ACTION
T:ABORT	Abort from ghost job	Ghost job	delete user
	Abort from TEL/CCI	ACP	Re-initialize user and re-enter ACP
T:ABORTM	Abort for bad CAL	ALTCP	Return to DELTA or return to ACP
	Abort for bad TRAP/CAL	ENTRY	
	Single user abort	INITRCVR	
	Bad segment abort	SEGLD	
	Bad TCB specified for traps	SSS	
	Bad buffer specified	TIM	
	Bad stack address specified	TRAPC	
	Bad buffer or size	TYPR	
T:DELUS	System deleting a user	GHOST1D	delete GHOST1
	System deleting a user	INITRCVR	Single user abort
	System deleting a user	KEYN	DELETE Keyin
T:RUNDOWN	System re-initializing a user	SSS	
T:ASP	Associate a processor	SSS LNKTRC	Associate requested processor

DESCRIPTION

The following descriptions of the logical blocks comprising the UTS routine STEP are best read with the listing close at hand. STEP is a complicated routine which interfaces with a large number of other monitor routines and processors. The avid reader of this documentation should have a thorough knowledge of the scheduler, swapper, and memory management routines and tables before undertaking the task of understanding STEP. As many paths are explained as practical, implications of action to other routines pointed out, and processor interfaces discussed. The table documentation in Section V of the Technical Manual is relied upon heavily, particularly the section on the Job Information Table. For convenience, STEP has been broken into the following logical portions.

A. STEP Exit Logic
B. Debugger Exit Control Logic
C. User Re-initialization Logic
D. Load and Go Logic
E. Assemble Unshared Program Logic
F. Interpretive Exit Logic
G. Logoff/Continue Logic
H. Associate Shared Processor Logic
I. Associate Unshared Program Logic
J. DCB Validity Checker
K. Merge DCB Information Logic

This arrangement is diagrammed in Figure EB-1. Good luck.

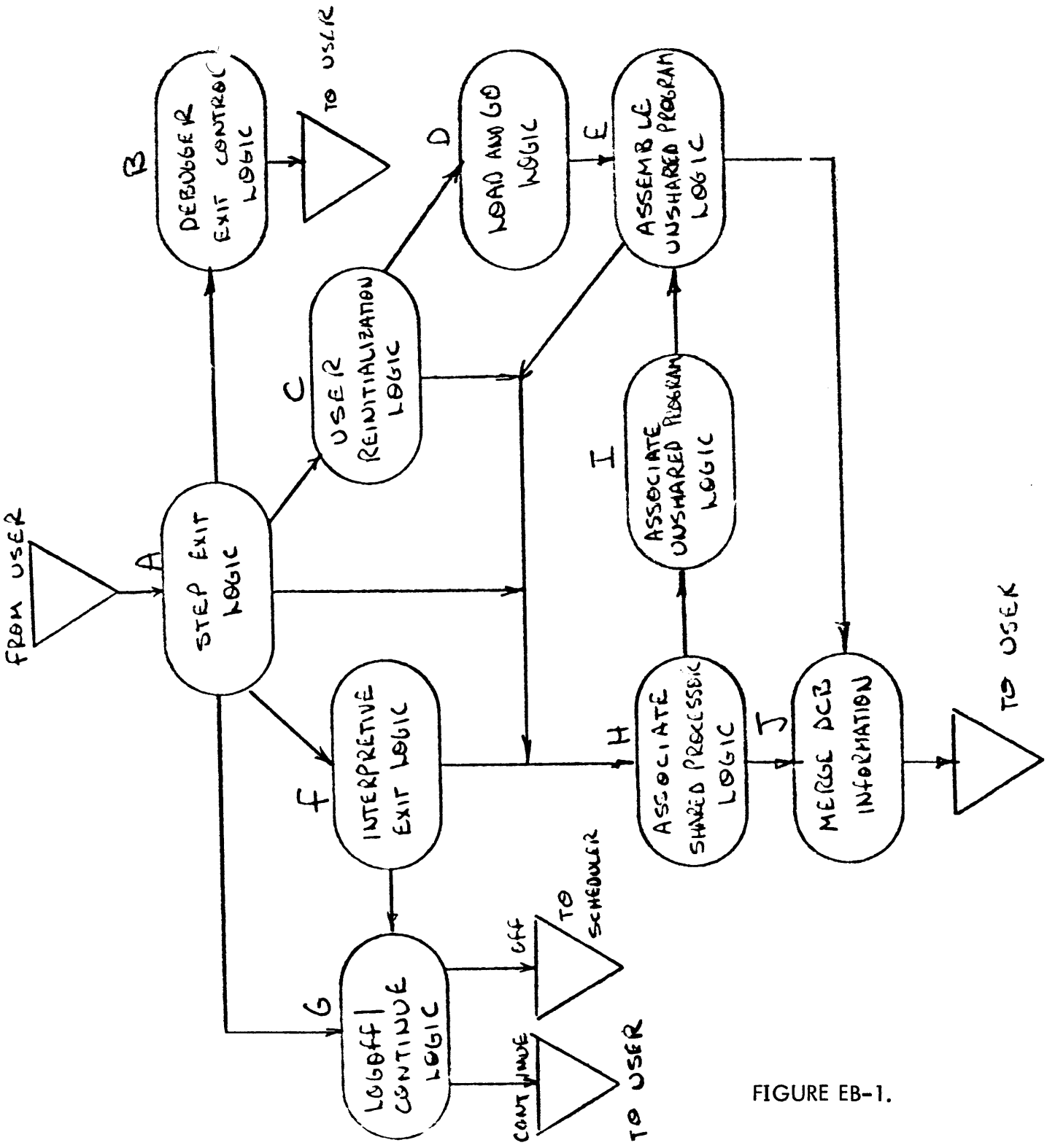

FROM USER
A
STEP EXIT LOGIC
B
DEBUGGER EXIT CONTROL LOGIC
TO USER
C
USER REINITIALIZATION LOGIC
D
LOAD AND GO LOGIC
E
ASSEMBLE UNSHARED PROGRAM LOGIC
F
INTERPRETIVE EXIT LOGIC
G
LOGOFF / CONTINUE LOGIC
LOFF
TO SCHEDULER
CONTINUE
TO USER
H
ASSOCIATE SHARED PROCESSOR LOGIC
I
ASSOCIATE UNSHARED PROGRAM LOGIC
J
MERGE DCB INFORMATION
TO USER

FIGURE EB-1.

A. STEP Exit Logic: T:EXIT, T:ERROR, T:ABORT, T:ABORTM, T:TELDELCCI

The routines comprising the STEP exit logic are entered at T:EXIT, T:ERROR, and T:ABORT from ALTCP, and T:ABORTM from a number of places within the monitor. T:ABORTM expects R14 to contain the abort code and its subcode ERO in the form [ERO | 00 | 00 | ABC]. The other entry points load R14 with the appropriate code (see Reference Manual on error codes).

T:EXIT reads the user's flags and determines what special shared processor the user is associated with. If RUNNER is exiting, control transfers to STEP003 to handle this special case. If the TIC (TEL in control) bit is set in VH:FLG, a command processor is exiting and control passes to the interpretive exit logic at STEP00.

A normal exit and all other entry points load R1 with an appropriate run status (0 = Normal Exit, 2 = Error CAL, 3 = Abort CAL, 4 = Monitor Abort) and enter common logic at SETRNST which sets run status in the high order byte of J:RNST. Common logic continues at T:TELDELCCI which stores the abort code J:ABC and its subcode ERO into the user's JIT, forces TSTACK to two environments as a precaution. Any ghost user, as determined by bit1 set in the first JIT word, is removed from the system by transferring to the logoff logic T:OFF, after first issuing an appropriate error message on all but a normal exit via the same mechanism as batch users - discussed in users reininitialization. A user with DELTA assocaited (DELA set in UH:FLG) is tranferred to DELTA via the debugger exit control logic T:DEL. On-line users without a command processor in control with non-zero run status are tranferred to the associated command processor (ACP) to be given an error message. Control is transferred to T:ECCP in SSS which uses the STEP associate shared processor logic to bring in TEL. This same SSS routine is called when a user types Y^{C}.* On-line users with TIC set in UH:FLG (TEL-in-control) or with J:RNST = 0 are transferred to the user reinitialization logic T:RUNDOWN along with all batch users, indicated by BAT set in UH:FLG.

* after issuing the error message, TEL will execute an Abort CAL to force user re-initialization.

B. Debugger Exit Control Logic: T:DEL

When a debugger is discussed in conjunction with the UTS Monitor, DELTA is understood to be the subject, and is the only debugger special-cased in numerous Monitor routines. In T:DEL, if the exit, error, or abort was from DELTA (DIC set in UH:FLG) control is transferred to the user reinitialization logic. Otherwise, T:DEL guarantees the user to be associated with DELTA, as a core library may also be assocaited. Core libraries and DELTA occupy the same virtual space - the special shared processor area.

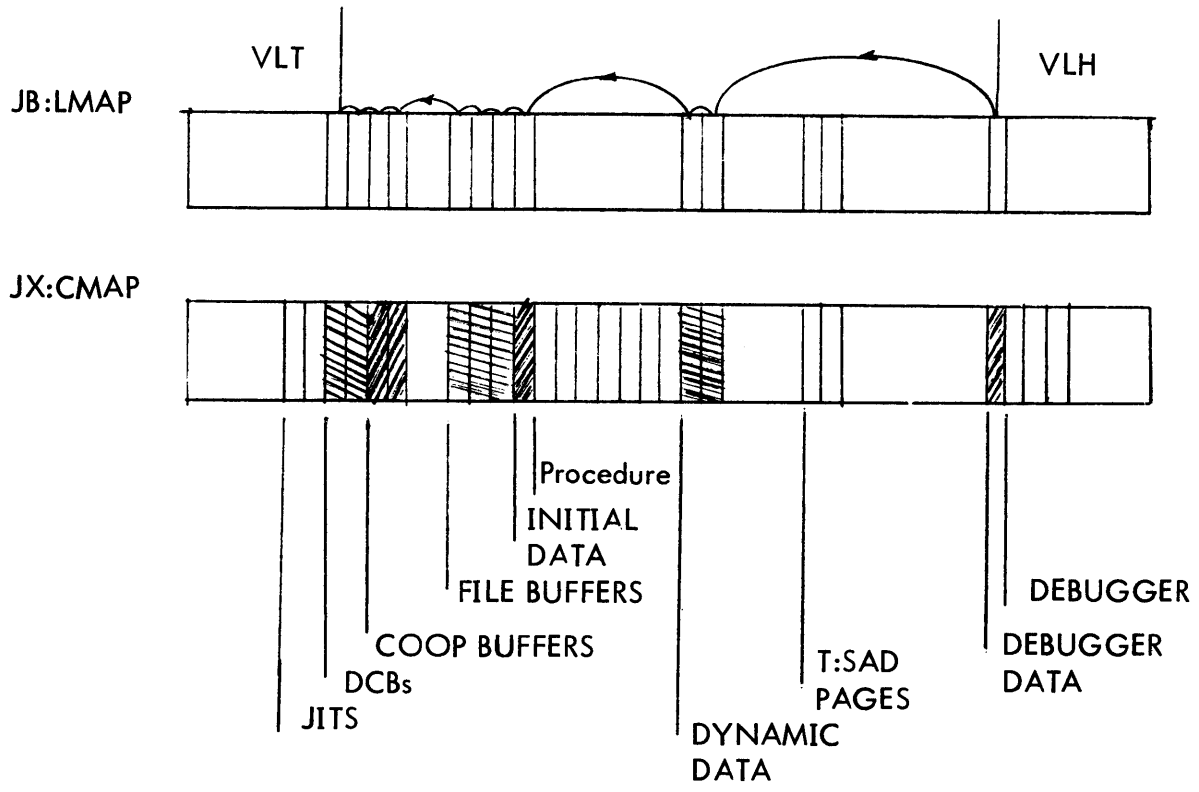
as a core library may also be associated. Core librarys and DELTA occupy the same virtual space - the special shared processor area.

If a core library is associated, recognized by a non-zero UB:ASP entry for this user, the library is removed and DELTA associated. The SSS routine DASP is used to decrement the core library use count. IDB in SSS is used to increment DELTA's use count, DELTA is set in control, and ready-to-run is reset in UH:FLG (DIC = 1, RTR = 0) as flags to the swap scheduler to indicate a request for association with Delta. An associate processor event E:AP is reported via the SSS entry point T:REG to associate DELTA.

If a core library was not associated or when control returns from the "REG", control is transferred to the routine XITCTRL which sets up registers 1, 10, and 11 for the SSS routine DELTAGO which will give control to DELTA.

C. User Reinitialization Logic T:RUNDOWN

Before entering user reinitialization, a typical user appears as shown below with an associated shared processor, data, DCB's buffers, pages obtained by the Change Virtual Map CAL, and a debugger.



T:RUNDOWN first checks if the user being reinitialized is also going through the single-user abort process. If the current user S:CUN is not equal to S:CRASHUN, his user number replaces its current value. If equal, the current user has been aborted again while being reinitialized and is deleted from the system by transferring control to T:DELUS. This drastic action is deemed necessary as the user is too damaged to be gracefully returned to the command processor, and may result in files being left open and the attached CFU's busy. These files are closed and CFU's returned to the system at quiescence, when a software check 1E is forced if any user has been deleted in this manner.

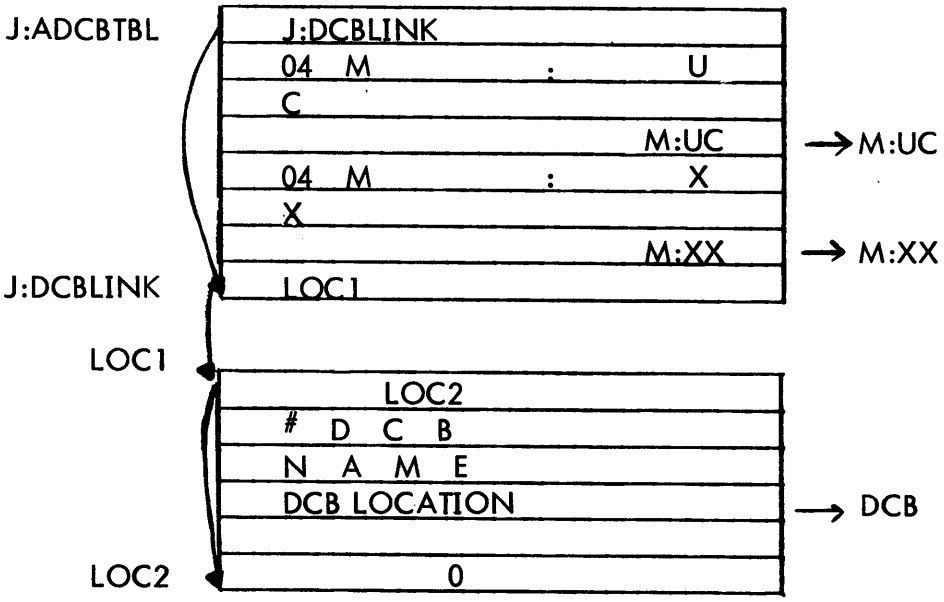
Assuming the user has not been deleted, T:RUNDOWN sets the X'4000' flag in UH:FLG to signal the scheduler to speed the I/O process during STEP by placing the user in the special compute queue, rather than I/O complete or compute bound, upon completion of every physical I/O operation or quantum end. This action lowers the probability of the user being swapped out during the STEP process.

TSTACK is forced to the one exit environment to facilitate possible error messages, then any associated processors and overlays are disassociated by the SSS routines for that purpose-DROV, RPROCS, etc. A special check is made for an associated command processor (TIC in UH:FLG) as separate routines are required. Consult the SSS documentation for further details.

If the user is on-line, his prompt character is reset in M:UC, and the monitor set running in J:RNST. If his symbiont DCB is still open, the right halfword of MUPO in his JIT points to its location, and control transfers to CHKPMD along with all batch users. Otherwise, CHKPMD for the on-line user is bypassed. The open symbiont DCB is a very rare occurrance, possible only if the user were aborted from the middle of a T:JOBENT CAL.

CHKPMD transfers control to the DEBUG overlay segment to give an error message to the batch user if the first byte of J:RNST is non-zero (see TELLUSR documentation, section LB), to perform a PMDI if requested (indicated by bit 14 set in J:ASSIGN), or to close an open symbiont DCB. DEBUG is requested via the OVERLAY proc, which loads the overlay number into R2, the entry point into R0, and branches to T:OV. (See T:OV, section EC). Upon return from the DEBUG segment, errored or aborted Ghost Jobs are removed from the system by transferring control to the logoff logic T:OFF. Otherwise, the monitor is set running in J:RNST and normal reinitialization continues at CLOSEDCBS.

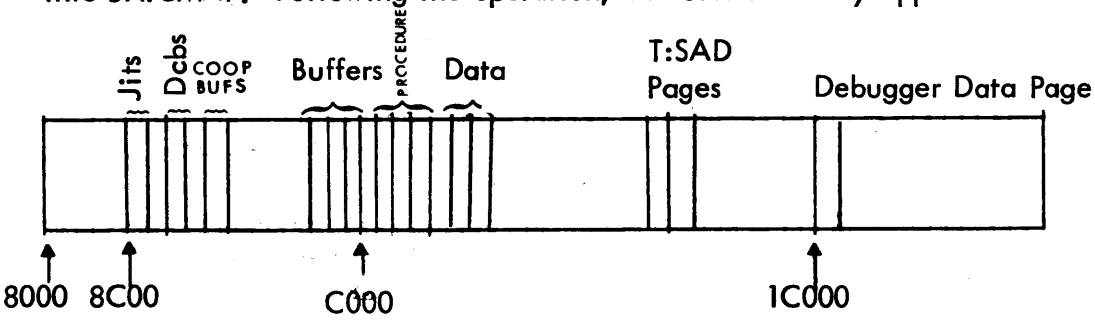
As the exit environment is no longer required, TSTACK is emptied, and the first type of run status zeroed to avoid re-entering STEP at T:ABORTM from IOSPRTN in CALPROC. The DCB chain is used to locate and close all open DCB's except those assigned to a non-tape device. The DCB chain begins in the JIT by J:ADCBTBL and is of the following format:



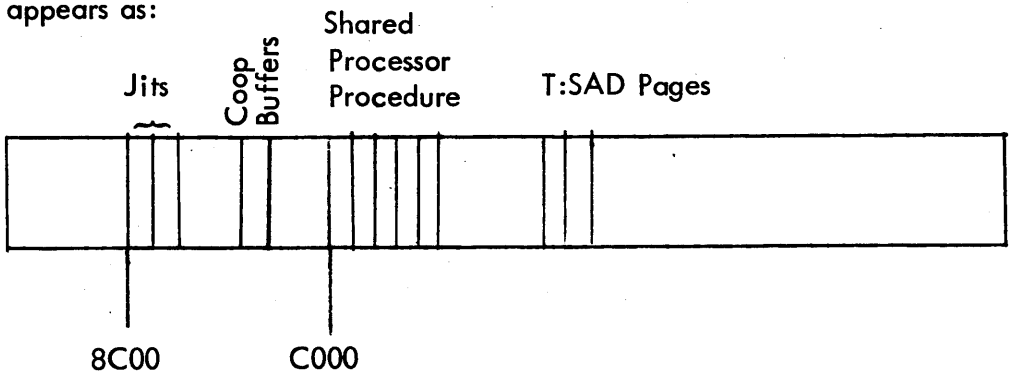
CLOSEDCB picks up the count of characters in the name, displaces past the name to get the address of the DCB, and tests the open bit (Y002) to determine the action to take. M:DO is closed with SAVE specified if open, others are closed with their default. Devices associated with open non-tape device DCB's are removed from diagnostic use at this time by resetting the diagnostic use flag in DCT3. These DCB's are not closed in order to avoid the unnecessary overhead required to close them.

When the end of the DCB chain is encountered, the exit logic continues by resetting the activation character set for on-line users in the COC line table MOD2, then checks the name of the associated special processor in P:NAME to see if its name was LINK. The index R7 into the processor tables was established during the processor removal logic in T:RUNDOWN. If LINK was associated, control is transferred to the Load and Go Logic (D).

Otherwise all of the previously associated special processor or debugger pages are removed from the user's memory area by resetting the access on those pages to 11 (no access) and storing the free page map constant (FPMC) into the user's JX:CMAP at those page locations. Access is reset by submitting access in R4, first virtual page of the processor procedure from PB:PVA in R7, and number of pages from PB:PSZ in R6 to the MM rountine T:SNAC. Upon returning from MM, a simple BDR loop on R6 with the same information stores the FPMC into JX:CMAP. Following the operation, the user's memory appears as :



User reinitialization continues at LDLNK, which checks for the existence of any temporary files built by the overlay LDTRC during a load-and-link operation. If the low order byte of J:RNST is non-zero (the counter for LDTRC files), LDTRC is called by transferring control to T:OV with R1 containing the overlay entry and R2-R3 containing its name in TEXTC format. If the counter is zero and such files do not exist, or upon return from the overlay, the virtual link chain JB:LMAP is used to remove all corresponding pages in JX:CMAP except the symbiont buffers. Each virtual page to release is determined by linking through JB:LMAP starting with JB:VLH. If the page obtained is not between JCOVP and JCOVP+1, the COOP buffer limits, it is released via the MM rountine T:RVPI, and the next page obtained. Upon completion of this process, the user's memory area appears as:



All JIT memory limits are reset to their initial values via a call to the subroutine T:RSTLMS. Other JIT cells referring to locations in the released memory area are zeroed. The results in the JIT appear as:

from T:RSTLMS	J:CUL = J:PUL = J:DUL
(All user memory set	J:CUL+1 = J:PLL = J:DLL = JBTDP
to Dynamic Data)	J:EUP = J:DDUL

Other results:

J:TCB = 0
J:DCBLINK = 0
J:INTENT = 0
J:TIMENT = 0
J:USENT = 0
J:CLMN = 0
J:IPOOL = 0
J:FPOOL = 0

All of CMAP and LMAP cells between the beginning and the end of the user area, BUP and EUP, are initialized to FPMC and zero respecitvely, to reset shared processor procedure and release any pages obtained by the Change Virtual Map routine T:SAD in MM. The MM routine T:SAC is used on each page to insure access is set to 11. The user now consists only of his JIT, his additional JIT (if any), and his COOP buffers.

Upon conpletion of the reinitialization of user memory, Tel-in-Control is reset in UH:FLG.

An environment is bumped into TSTACK for T:ECCP, and S:CRASHUN and RCVUSER zeroed. Control is then transferred to T:ECCP in SSS to associate the appropriate command processor with the user.

D. Load and Go Logic: XITLINK

LINK specifies action to be taken by STEP in its data page, the first page of the special processor area. The interesting locations (the first 26) of that page are illustrated below:

Head	0	84 0 0 ff 30		
	1	8	Start	Start address
	2	TCB	BIAS	
	3	00 Size	00 Loc	Data Size and Loc (Doubleword)
	4	01 Size	01 Loc	Procedure size and Loc (Doubleword)
	5	Max REFDEF	Tree Size	
	6	DCB Size	DCB Loc	Original DCB Loc (DW)
	7	GST Size	GST Loc	Global Symbol Table (DW)
	8	IST Size	IST Loc	Internal symbol table (DW)
	9	TEXTC		
	A	CORE LIBRARY		
	B	NAME		
	C	TEXTC		
	D	Debug Processor		
	E	NAME		
	F	X 'C'		Tree Size/Load and Go Flag
Tree size	10	TEXTC		
Tree	11	Load Module		
	12	NAME		
	13			
	14			
	15	00 Size	00 Loc	
	16	REFDEF Size	0	
	17	01 Size	01 Loc	
	18	0	0	
	19	0	DCB Origin	
	1A	0	0	

If LINK was associated with the exiting user (after being disassociated by the Reinitialization Logic) then control is given to the Load and Go Logic at XITLINK. This routine locates LINK's data area via the subroutine SPCON, saves the load module name from the first three words of the TREE in J:LMN, then checks the LINK run flag - the TREE size. If non-zero, control returns to the Reinitialization Logic at XITNRUN. A zero signals go; the first operation is to map the LINK-manufactured DCBs into the context area. The LINK-manufactured DCBs are those built by LINK to be associated with the user's program. LINK, on the other hand, had been using the context DCBs to perform it's own I/O. With LINK disassociated and "GO" indicated, the manufactured DCBs are moved to the context area to be linked to the user's program. This operation begins by locating the first virtual page containing LINK-built DCBs, pointed to by the DCB location in the HEAD. The corresponding physical page is obtained, and the virtual page released via the MM routine T:RVSPI (Release virtual save physical). The first page of the context DCB area is released using T:RVPI (Release virtual and physical), then T:GVGPI (get virtual given physical) is supplied with the physical page of link-built DCBs and the first virtual page of the context DCB area requested. If the request is successful, the physical page has been mapped into the context DCB virtual page. If not, the FPMC remaining in JX:CMAP indicates the user has exceeded his maximum page count and control given to the routine OUTOFPGS to abort the user. If there is a second page of LINK-built DCBs, the re-mapping is repeated. After successfully completing this process, control is given to the assemble unshared program logic.

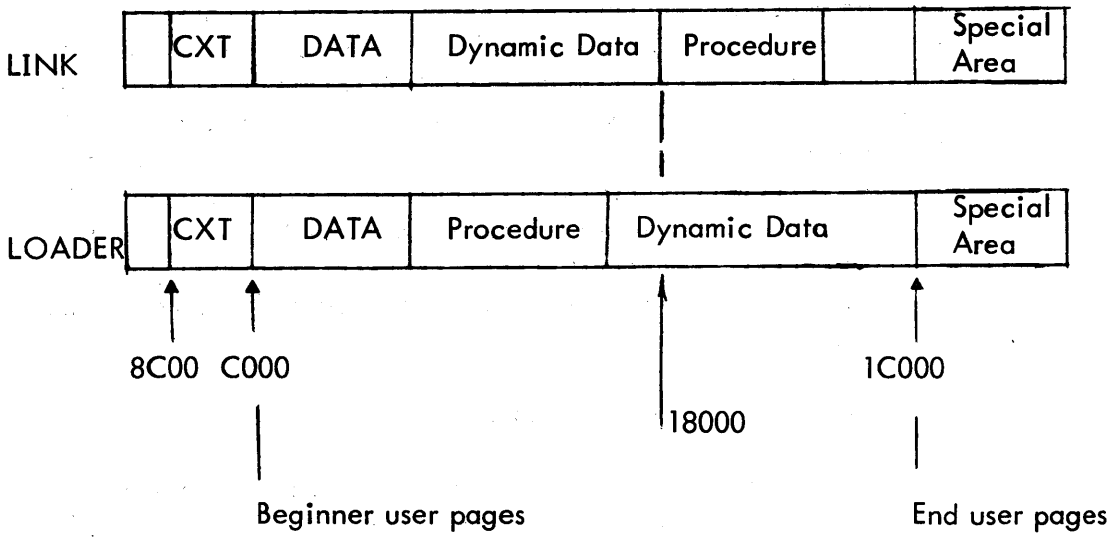
E. Assemble Unshared Program Logic: XITI0

The logic produces a user program arranged correctly in memory complete with any debugger or library requested. Beginning at XITI0 TSTACK is forced to a single environment, J:ASSIGN reset, the user set running in J:RNST. The difference between the load module bias and the start of its data area is calculated; this value is usually zero, but FORTRAN leaves this space for blank common. If a difference exists, those pages are obtained via T:GNVPI (get m virtual and physical) by specifying the number of pages desired in R6.

If a TCB exists, the location for DCB chain is obtained from its tenth word, otherwise the first word of the DCB area is assumed; one or the other is stored into J:DCBLINK unless the DCB size is zero. This link is checked to assure it is within the DCB area, and the DCB chain itself checked to assure it is complete. A failure results in the ABC/ER0 code A63A. The load module name from the tree is saved in J:CLMN for DELTA to use (if necessary) when reading symbol tables. The symbol table locations are also obtained and stored in the JIT:

GST Size and Location	J:GST
IST Size and Location	J:IST

Access and JIT memory limits for the program are set for the user by specifying the JIT limit in R3, the relative location in the head in R1, and the access (00, 01) in R13 to the routine LINKLIMS. The dynamic data count is obtained at the start of this process, and the count of pages in each protection type from LINKLIMS subtracted. The remainder is common storage, and set back into the count of dynamic data pages. The dynamic data lower limit is set to either one page past data upper limit for a program built by LINK, or one page past procedure upper limit for a program built by LOADER. Because the loaders allocate user memory as in the following illustration they must be special-cased here.



After storing the starting address and the TBC address from the HEAD into J:START and J:TCB, and the TCB location placed into R0 of the environment, the core library and debugger names are saved in registers and the first page of the special area containing the head and tree records released via T:RVPI in MM.

If the load module was built by the LOADER, the address of the first word of the procedure is set into JITREE in the JIT.

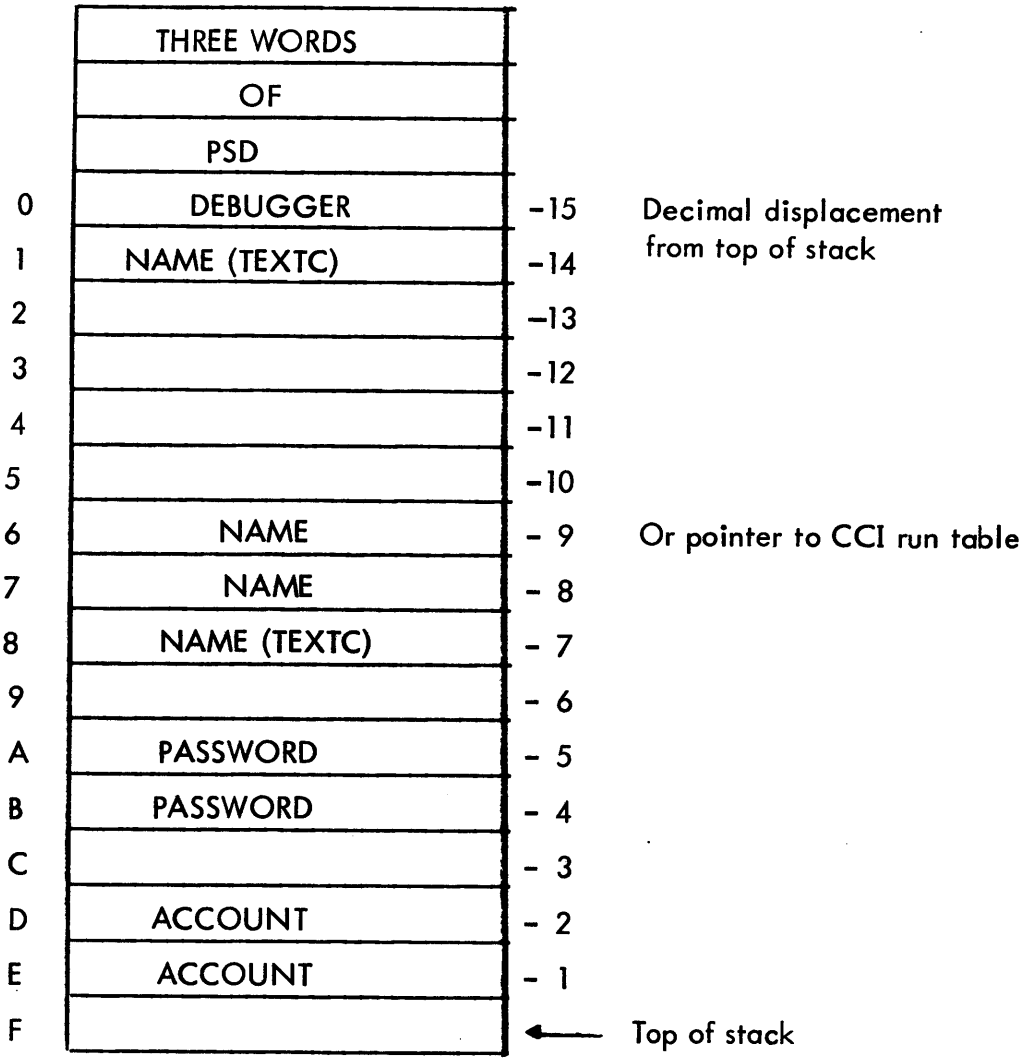
A series of tests to determine if the user specified a valid core library are made; if a library was specified (name saved in R14-15), the name is found in the P:NAME table, and the bit set in its corresponding entry in P:SA indicating "yes indeed, I'm a library, " then its number is stored into UB:ASP. R6 at this point may contain zero or the name of a requested debugger. If zero, RTR is reset in UH:FLG, and an associate processor event reported via T:REG in SSS as a request to the swapper to bring in the library. If an invalid core library was specified, control is given to TELLTEL to abort the user. Upon return from the REG or if no core library was specified, the hardware memory map is loaded via the T:XMMC routine in MM to reflect the user's current memory layout, and control given to the merge DCB information logic.

If R6 contained the name of a debugger, which is found in P:NAME, control is passed directly to the associate shared processor logic to bring it in. If not in P:NAME, control is passed directly to the associate shared processor logic to bring it in. If not in P:NAME, ACP is told.

An odd path through the above logic results when R6 is loaded with zero if either of the two high-order bits of J:CFLGS was set: This occurs when the overlay LDTRC is associated, and load-and-link/transfer control is taking place. The result is to bypass the tests for a debugger and library, to rejoin the common logic at XIT31 to load the map.

F. The Interpretive Exit Logic: STEP00

The function of the Interpretive Exit Logic is to process the request of a command processor by interpreting its registers in the environment produced by the EXIT CAL. The environment is of the following form:



Special JIT access is reset immediately at STEP00, then the processor issuing the interpretive exit is identified and removed from JX:CMAP by obtaining its size from PB:PSZ and starting virtual page from PB:PVA, then running a simple loop to store the FPMC into the appropriate CMAP location.

If the exiting processor was RUNNER, a CCI overlay which builds the debug and modify clobber tables, its DCB page is released via T:RVPI. Control is then given back to the Associate Unshared Program logic which requested RUNNER.

If not RUNNER, then R6 of the exit environment is checked for a request. If zero, control is passed to the Logoff/Continue logic. Otherwise, the rest of the command processor - the context, DCB, and special context pages - is removed from the user. All pages between JCOVP+2 and JBUPVP, the third page of the buffer area (above the COOP buffers) to the beginning of the user area, are released using the MM routine T:RVPI. JIT pointers into this area, J:FPOOL, J:IPOOL, J:ABUF, are zeroed. If the command processor lived in the special area, as determined by bit1 of P:SA, the first page of that area - the context - is released. DCB pages indicated by PB:DCBSZ are released via T:RVPI, and J:DCBLINK zeroed. T:RSTLMS is called to reset all memory limits to their initial values, as in the User Reinitialization Logic. At this point, the user is reinitialized except for common pages.

R6-8 are loaded from the exit environment into R6-8. If a command processor performed the exit, as indicated by TIC set in UH:FLG, the processor is decremented and reset by the SSS routine DRTEL1. If the exiting command processor was LOGON, the P:NAME table is searched for TEL; if found, TEL's index is put into UB:ACP and registers 4 and 5 in the user's stack are loaded with TEXTC TEL. This allows autocalled programs to return to TEL when they abort or hit Y^C. Control then passes to T:ASP.

R6 may contain one of two things: The first word of the TEXTC name of the load module requested, if TEL performed the exit, or the address of the run table in the common area, built by CCI. If the first byte of R6, the count, is non-zero control is passed directly to the Associate Shared Program logic. Otherwise, words 3-5 from the run table, containing the requested load module, are obtained and stored into R6-8 in the exit environment. Similarly, the account and password are moved to the appropriate environment registers. After thus simulating a TEL exit for CCI, R6-8 are re-loaded with the TEXTC name of the request and control passed to the Associate Shared Processor Logic.

G. Logoff/Continue Logic STEP10

If a special shared command processor, recognized by TIC set in an on-line user's UH:FL **G**, and bit1 of P:SA, issued an interpretive exit with no request specified in R6, a continue operation is indicated. This logic at STEP10 sets TSTACK to one environment, resets ready to run in UH:FLG, increments all the user's associated processors and sets their access into the JIT access image J:AC via the SSS routine IPROCS and the MM routine T:PAC. An associate processor event is reported to T:REG as a request to re-associate all processors. Upon returning from the REG, control is passed to the scheduler at T:SSEM to transfer control to the user's program.

If LOGON or CCI issued the interpretive exit with zero in R6, the interpretation is a logoff request, and the user is removed from the system. T:DELUS first insures all associated tapes are dismounted with the routine T:DSMT which removes them. All pages represented in JB:LMAP are released by repeatedly releasing the head of the chain JB:VLH until it becomes zero, using T:RVPI. The user's additional JIT page is released via the MM routine T:FPP and the corresponding swapper granule from UH:JIT released by T:SGR in MM.

If the user is a ghost with a name in the ghost job name table (verified by T:NAMECHK) all flags in SB:GJOBFLG are reset and the user number in SB:GJOBUN zeroed. The swapper granule for the user's JIT is released via T:SGR, the count of batch users in system S:BUIS decremented if the user was batch, and the total number of users S:CUIS is decremented. If this user was opening or closing a file when deleted, OPNCLSUS is zeroed. All associated processors are decremented and reset via the SSS routine DRPROCS, any associated processor overlay decremented, and the following tables zeroed:

UB:OV
UX:JIT
UH:ID
UH:FLG

If the user was on-line, the line tables.

LB:UN
PROMPT

are zeroed.

The user's state is changed to zero via T:CHS in SSS, a batch job scheduled if the user was batch by T:BTSCHED, and control is transferred to the scheduler to schedule with no current user at T:SE.

H. Associate Shared Processor Logic T:ASP

Control comes to T:ASP with the user reinitialized and a program request in R6-8. If the program is identified as a shared processor, the processor tables are used to allocate user memory, otherwise control is passed to the Associate Unshared Program Logic.

T:ASP first resets the INIT flag in UH:FLAG, possibly set during reinitialization by MM, and also sets the STEP I/O flag X'4000' as in the Reinitialize User Logic to speed the I/O.

The P:NAME table is searched for the request and the processor number, if found, preserve in R5.

The processor flags, P:SA, are checked to see if the request is for a command processor; if so, the user must be proper for the requested C.P., ie. batch if requesting a batch C.P., on-line for an on-line C.P., etc. If the user does not match the requested C.P., he is aborted with error code 'A2'.

The run flags in J:RNST are checked, and if non-zero, the user is not at job step and may be calling for a debugger or library. CHKDB is used either to abort the user via TELLTEL if the request is not a debugger, or library, or to return in line at ASP23. Buffer pages are obtained for the user if he has none by consulting the specified number for each in the JIT, calculating the required pages, and requesting those pages via T:GNVPI in MM if the total requested is less than the buffer area available - 2 pages at the front are reserved for COOP buffers. The calculations are -

from J:NIPOOL and J:NFPOOL

(Number of IPOOLS) * (IPOOL size) +X'IFF' shift→ Pages required for IPOOL request

(Pages of IPOOL's) + (Number of FPOOL's) = Total pages required

(Total pages required) < JBUPVP - (JCOVP+2)

If (Total pages required> JBUPVP - (JCOVP+2), the number of each buffer type in the JIT is set to two, and three pages requested. These buffers are sufficient to allow TEL or CCI to give the user an error message and to run him down.

If the account specified in the exit environment is not :SYS, or the processor number in R5 is less than the greatest overlay number (to avoid an on-line user requesting OPEN, for example), or the processor size in PB:PSZ is zero, control is passed to the Associate Unshared Program Logic at FETCH.

If this request is from LNKTRC, the extra environment is removed from TSTACK. If the requested processor has the special JIT access bit set in PSA, SJAC is set into UH:FLG.

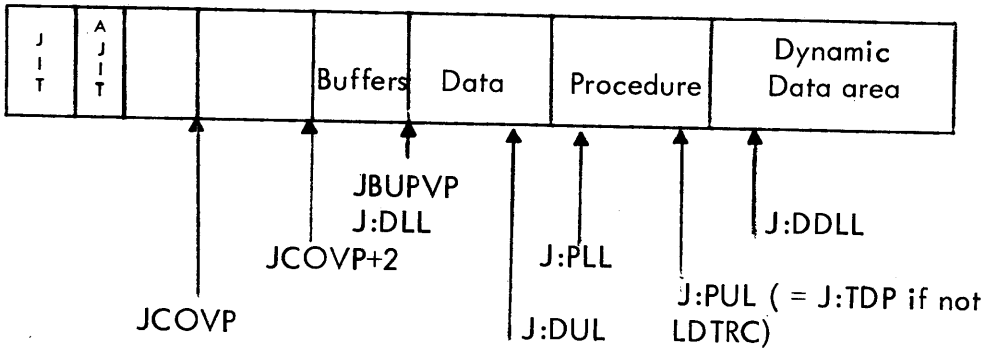
If the processor lives in the special area, user memory limits are not changed, otherwise the access on the data and procedure pages is set via T:SNAC in MM as follows:

J:DLL to J:DLL+PB:DSZ set to 0 for data
PB:PVA to PB:HVA set to 1 for procedure

and the following memory limits are established:

J:DDLL = PB:HVA
J:PUL = PB:HVA-1
J:PLL = PB:PVA
J:DUL = PB:PVA-1

so the user's memory area now appears as:



If no LDTRC action is taking place, J:TDP = J:DPUL+1.

If the requested processor is a debugger or a core library and does not have special JIT access in P:SA, the user is set running in J:RNST. Otherwise processor running is set unless it is a command processor in which case all run flags are reset to indicate command processor running.

If the processor being associated requires DCB's as indicated by PB:DCBSZ, the pages of them are acquired via the MM routine T:GNVNPI (get n virtual no physical) and PPSWP and DCB's set in UH:FLG.

T:GNVNPI places the no-page map constant NPMC into the requested page locations of CMAP, indicating to the swapper to place the initial DCB pages there during swap in. Also in the above logic, the total size of the user area is submitted to the MM routine T:GAJP, to get an additional JIT if the user's total swap size exceeds the command list in the JIT.

Initial data, indicated by PB:DSZ being non-zero, is obtained in a similar manner to the DCB pages: INIT and PPSWP are set in UH:FLG, the number of pages from PB:DSZ and the first page from J:DLL (or J:EUP if a special processor) are presented to T:GNVNPI. If the pages are not obtained, as with the DCB pages, control is passed to OUTOFPGS to abort the user.

Depending on the processor being associated, the following action is taken:

a) If a command processor, the processor number is stored in UB:ACP, ISTEL1 in SSS is used to increment and set the command processor.
b) If a special processor, its number is set into UB:ASP.
c) If DELTA, DIC and DELA are set into UH:FLG; the processor number set into UB:DB.
d) If a normal processor, its number is set into UB:APR, and its link if overlaid from PB:LNK set into UB:APO. If restoring a processor for LDTRC byte1 of J:CFLGS contains the overlay link and it is set into UB:APO.

In cases b-e, the SSS routine IPROCS is used to count up the processors, then common logic continues at ASP12 which resets RTR in UH:FLG and reports an associate processor event via T:REG as a request to the swapper to associate all the newly incremented processor. Upon return from the REG, the user's buffers are linked via BUFCHAIN, registers set up for the logic to merge DCB Information, and the DCB's linked to the JIT at J:DCBLINK if its previous value was zero.

If the processor being associated is GHOST1, R2-3 are loaded with a master/mapped PSD - only the MAP bit set. Otherwise both SLAVE and MAP are set. The registers are now set up as follows for the merge DCB information logic.

R2-3	Master/Slave mapped PSD
R4	User number
R8	Starting address
R9	First procedure page
R10	TCB address

I. Associate Unshared Program Logic: FETCH

The logic to associate a Unshared Program begins at FETCH. If debug commands were encountered, RUNNER is requested to process them.

If not, or upon returning from RUNNER, the requested load module is read into memory as specified in its head and tree records. Any debug or modify tables built by RUNNER are merged into the program, and control transferred to the link unshared program logic, part E.

FETCH begins by setting up M:XX to read the requested load module. The STEP error/ abnormal return is set into M:XX and the DCB closed if open. The password, name, and account are taken from the exit environment and placed into M:XX in the appropriate locations in the variable parameter list. The DCB is set in the input mode and FILE set into the ASN field. If the user is on-line or LDTRC action taking place, control is given to FCH4. If CCI performed the exit, however, recognized by BAT and TIC set into UH:FLG, the following action is taken depending on specifications in the run table.

a) If DEBUG commands were encountered (Byte 1 of run table $\neq 0$), associate RUNNER.
b) If MODIFY commands were encountered (Byte 2 of run table $\neq 0$), associate RUNNER.
c) If the first byte of word X'A' in the run table is non-zero, a TEXTC start address symbol is specified, associate RUNNER.
d) If any PMD records exist, recognized by bit X'20000' set in J:ASSIGN, associate RUNNER.

RUNNER is associated by releasing the special area context page via T:RVPI, loading R13-14 with :SYS, loading R6-8 with "RUNNER" in TEXTC, forming an environment to look like an interpretive exit, and branching to the Associate Shared Processor Logic. Upon returning from RUNNER, the run table page (last page of the user's data area) is released.

Common logic continues at FCH4 which first obtains the first page of the special area is a buffer into which the HEAD and TREE records of the requested load module are read. The page, J:EUP+1, is obtained via T:GNVPI. The debugger name specified by the exit from TEL in R0-1 is saved in words 12-13 of the buffer page, as only 12 words of head record will be read, then TSTACK is emptied to obtain as much space as possible for the registers pushed by file management.

The HEAD record is read via a CAL1, 1 first resetting user running in J:RNST and setting Y4 in J:ASSIGN to avoid buffer limit checks in IORT. On return from the CAL, security checks on the HEAD and TREE verify the following:

a) The first word of the head is of the proper format.
b) Bias, procedure, and DCBs all start on a page boundary.
c) The proper number of bytes (X'30') was read for the head record
d) The proper number of bytes (X'30') was read for the Tree record
e) The file is keyed
f) RUNNER is not associated with a LMN built by LINK.

If one of the above tests is failed, control is given to FETCH3 to provide the specified ABC/ERO to TELLTEL. If the tests pass, the file is assumed to be a valid load module, and the tree read into the buffer at word 15, avoiding the specified debugger name. Upon returning from the CAL, the size is again checked in ARS of M:XX to verify the TREE record is the proper size - 8 words.

With the HEAD and TREE records in the buffer (see Load and Go logic for illustration) the dc a, procedure, and DCB's are read into their proper locations, this operation consists of specifying the following registers to FETCH1 to read in the requested record:

R0 = Pointer to key - LMN name in TREE
R7 = Word from TREE containing size and location
R8 = R0
R10 = Key number (3 = Data, 5 = procedure, 7 = DCB's)

FETCH1 calculates the pages required from the size and starting location in R7. If zero, control is returned. Otherwise the location specified in R7 is checked. If valid (within user limits or DCB's), the necessary pages are obtained via T:GNVNPI. A key is formed by concatenating the load module name in the tree with the specified key number in R10, and used to read the appropriate record, after setting Y4 in J:ASSIGN to avoid limit checks. Upon returning from the CAL, control is passed back to the FETCH logic.

This operation is performed for both data and procedure; any remaining DCB pages are released by T:RVPI, and the same routine called for the load module's DCB's. M:XX is then closed.

Unless this is a load and link operation, an environment is bumped into TSTACK, to be removed by the Assemble Unshared Program Logic at XIT10.

If the load module was built by LINK or is not overlaid, recognized by X'C' in the first word of the tree, no M:SEGLD DCB to be used by the SEGLD routine exists; otherwise the first 10 words of the variable parameter list from M:XX are set into the corresponding locations of M:SEGLD.

If RUNNER was not associated, control is given to the Assemble Unshared Program Logic at XIT10. Otherwise, RUNNER is decremented via DRASP in SSS. The clobber table location is determined from the last word in the special data area where RUNNER left it. The clobber table values are set into the procedure, preserving the replaced instruction in the debug FPT if a debug table, or merely replacing it with the table value if a modify. Consult the RUNNER documentation for details of debug table formats. Upon completion of this operation, control is passed to the Assemble Unshared Program Logic at XIT10.

J. DCB Validity Checker: DCBCHK

This subroutine will maintain a minimum standard for all DCBs used in UTS.

When the user's DCBs have been read into core, DCBCHK will be called and the DCBs will be checked to see that they meet the minimum specifications outlined here. If they do not, the error conditions are set and return is to the caller.

DCBCHK is called by the following instruction:

```
BAL, 11    DCBCHK
```

DCBCHK assumes that the DCBs to be checked are in core and that the caller has supplied a page-sized working buffer. Input registers are as follows:

R0 = beginning address of the DCBs

R1 = total size of the DCBs in words

R2 = beginning of the DCB name chain

R4 = address of the working buffer (one page)

R5 = 0 if the DCBs are to be initialized

If the DCBs meet minimum standards, the condition codes are set to 0 and byte 0 of R11 is set to 0. If the DCBs are not acceptable, the error byte is stored in R11.

The DCB buffer is checked for the following:

a) The DCB-name table must lie within the buffer, either at the beginning or the end.

b) The DCB-name table must not be linked, i.e., only one set of DCB names is allowed.

c) The pages of the DCB buffer are checked for type 10 protection.

d) All pages of the DCB buffer must be contiguous.

The DCB addresses are then located and stored into the working buffer; they are sorted from lowest to highest as they are stored. If the name table is improperly formed, the error return is taken. If the M:SEGLD DCB is encountered, its address will be stored in R3 for return to caller.

The individual DCBs are now checked; if any of the following conditions is found, the error return is taken.

a) DCB is less than 22 words long. (DCB length is defined as the difference between the beginning of the DCB and the beginning of the next DCB.)

b) The DCB crosses a page boundary.

c) KBUF does not lie within the DCB's variable parameter area (the space following the required 22 words, and continuing up to the beginning of the next DCB).

d) KBUF overlaps the next DCB (i.e., there are not 8 words between KBUF and the beginning of the next DCB).

e) FLP does not lie between the 22nd word and the beginning of KBUF.

f) FLP's overlap KBUF.

When all DCBs have been checked, return is to caller.

Following is a list of DOs and DON'Ts which should be observed in the construction of DCBs.

DO	put the DCB-name table in the DCB record, either as the first thing or the last.
DO	build DCBs on contiguous pages (virtual).
DO	build DCBs with protection type 10.
DON'T	build linked name tables.
DON'T	cross a page boundary with a DCB.
DON'T	have more than 509 DCBs.
DON'T	point KBUF or FLP out of the DCB or into the first 22 words of the DCB.
DO	make KBUF the last thing in the DCB.
DON'T	allow less than 8 words for KBUF.
DON'T	overlap FLP into KBUF or KBUF into the next DCB.

K. Merge DCB Information Logic: ASP14

The logic to merge DCB information completes the process of making a user ready to run for both the associate shared and unshared program logic.

At ASP14 the start address from R8 is stored into the PSD in R2-3 unless the processor is a core library in which case the start address is obtained from J:START. The PSD is then stored into the PSD of the TSTACK environment unless LDTRC initiated the request.

The user's access is loaded via the MM routine T:PAC and the following action taken depending on the processor:

a) If TEL, bypass all merge DCB information logic by transferring control to ASM167.
b) If CCI, merge assigns without setting TCB and TREE addresses into the JIT by transferring control to ASP17.
c) If a normal processor, recognized by P:SA, store the TCB address from R10 into R0 of the environment and J:TCB, unless restoring a processor for LDTRC.

The tree address - the first page of the procedure - is stored into JITREE. Consult the register information provided by the Associate Shared Processor Logic. If not performing a transfer control for LDTRC, R8 of the environment is zeroed, the register for I/O errors is passed to the user.

The assign merge begins at ASP17, which runs the DCB table, matching its entrys with entrys n the assign merge record built by TEL or CCI, and issues the Adjust DCB CAL on each match. Certain defaults are set into M:GO, and M:OC and M:UC are avoided entirely.

This operation begins by avoiding the assign merge logic entirely if a processor is being restored for LDTRC. The close bit is loaded in R9 as a flag to the OPEN logic to perform security checks at each Adjust DCB CAL. Starting with the J:ADCBTBL, the first byte of each entry is obtained and if zero, the link is made to the next portion of the DCB table (illustrated in the user reinitialization logic). Each name in the table is checked, and the following action taken by name:

a) M:OC, M:UC: Bypass and get next DCB in the chain.
b) M:GO: Set the following defaults into the DCB:

1)	File type	ASN
2)	Out	FUN
3)	0	Open bit in 1st word

4)	1 word name	First word of name entry
5)	"id G"	Name entry (id from the 1st word of the JIT)
6)	0	Account entry

c) All other DCB's: Reset the open bit in the word 0.

When the link to the next portion of the chain is zero, control is transferred to the second half of the assign merge logic at ASM3.

If J:AMR is zero, no assign merge record exists, and control is passed to the exit assign merge logic at ASM167. Otherwise, the "Assigns merged" flag is set in the high order bit of J:ASSIGN, and J:ABUF accessed, which contains the buffer address of the assign merge record. If zero, a buffer is unlinked from J:FPOOL, and its address stored into J:ABUF. If M:XX is open, it is closed, then the "Bypass limit check" flag (Y4 in J:ASSIGN) is set so the buffer limit check routines in IORT will not check the next CAL. Then the read assign merge record (RAMR) CAL is issued specifying the buffer address in R7. Upon return from the CAL, common logic continues at ASM0.

The assign merge record is fully documented in the UTS reference manual in the discussion of the Adjust DCB CAL. It contains a series of FPTS built by TEL or CCI in response to ASSIGN or SET commands. The first name in the record is compared with the DCB names in the DCB table until a match is found. When matched, the DCB address from the table is set into the PLIST of the Adjust DCB FPT for that DCB in the assign merge record. If the DCB matched is M:UC or M:OC in the JIT, however, the search for a match continues, allowing a user to have an M:UC DCB of his own.

After the DCB location is set into the FPT, the Adjust DCB CAL is issued. Upon return from the CAL, the search begins again for a match in the DCB table with the next entry in the assign merge record.

After all assigns have been merged, ASM167 resets the X'4000' flag in UH:FLG terminating STEP I/O operations, re-links the buffer used for the assign merge record (if any) back into J:FPOOL, moves sense switch settings from user's JIT to his TCB (if batch), zeros J:ABUF, then transfers control to T:SSEM with the user completely ready to run.

ID

Overlays - Monitor, Shared processor, user program

OVERVIEW

Monitor and shared processor overlays are treated exactly as shared processors. There are two user tables UB:OV and UB:APO which contain the shared processor number of the associated monitor overlay and/ or the shared processor overlay, respectively. The process of associating an overlay is simply a matter of establishing the number of the overlay in the appropriate table, increasing its usage count (PB:UC), resetting the ready-to-run flag in UH:FLG to force an in-swap, and doing an associate processor (E:AP) REG. The swap scheduler will set up the swap to include any overlays and the swapper will bring them in and fill them into the user's map.

User overlays are loaded by calling M:SGLD to open the user's load module file and read the segment into the place indicated by the tree table as well as its backward path.

ID

T:OV - Monitor and Shared Processor Overlays

PURPOSE

T:OV associates all monitor and shared processor overlays.

USAGE

T:PROCOV is called to overlay shared processors. Register 4 contains the overlay number in the shared processor table, Register 1 contains the entry address.

T:OVER is called to associate a monitor overlay. Register 2 contains the segment number. Register 0 contains the overlay segment entry point number (all monitor overlays begin with a transfer vector to the possible entry points).

T:OVERLAY is called like T:OVER. It performs the same function except that it remembers the contents of register 11 and the current overlay segment number to allow return to the caller.

T:REMEMBER is BAL'd to on register 14. It saves the current overlay number and the contents of register 11 in OSTACK in JIT for return from a future segment.

DATA BASES

T:OV uses the user tables and shared processor tables to locate segments and update user associated shared processor tables.

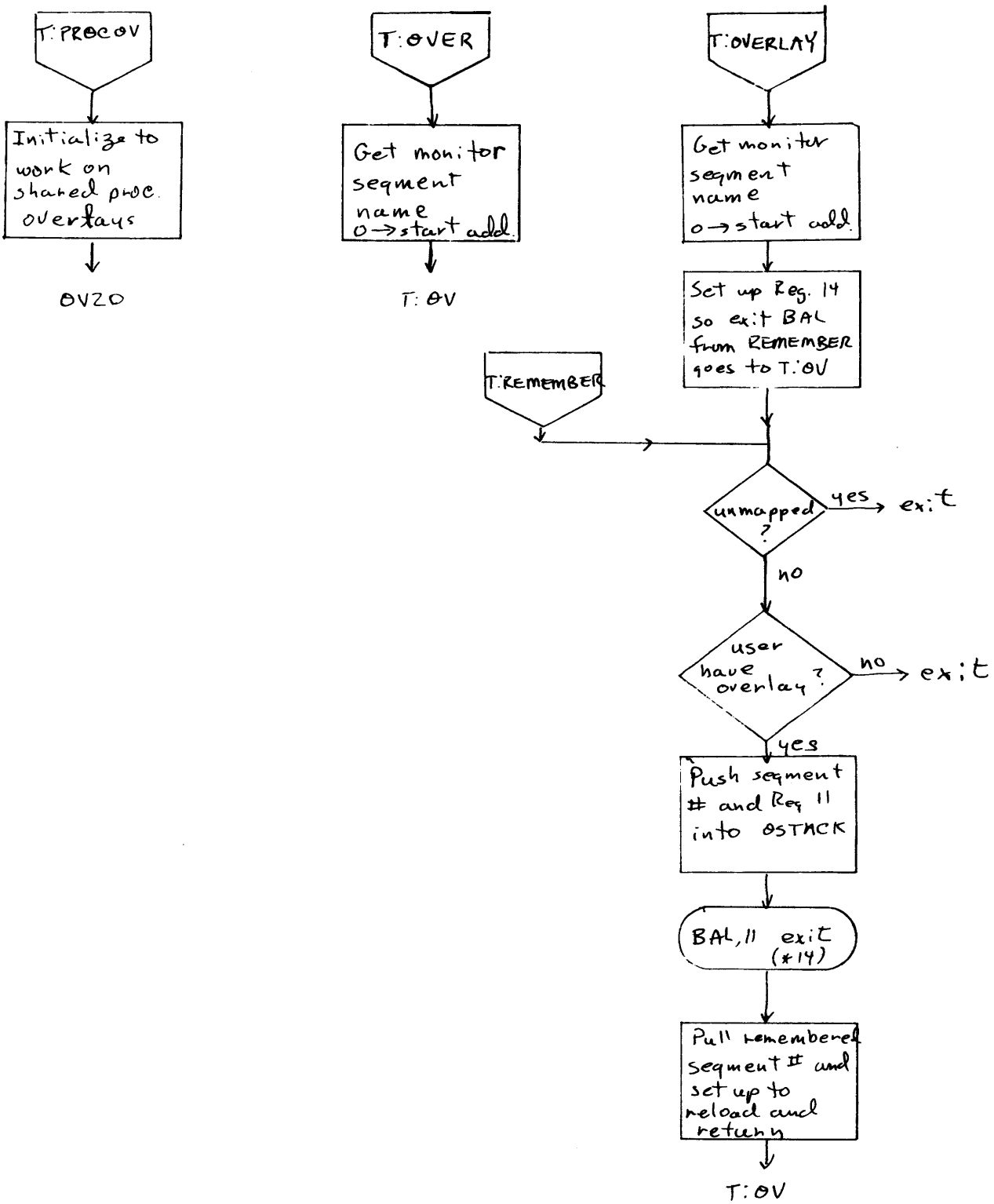
ERRORS

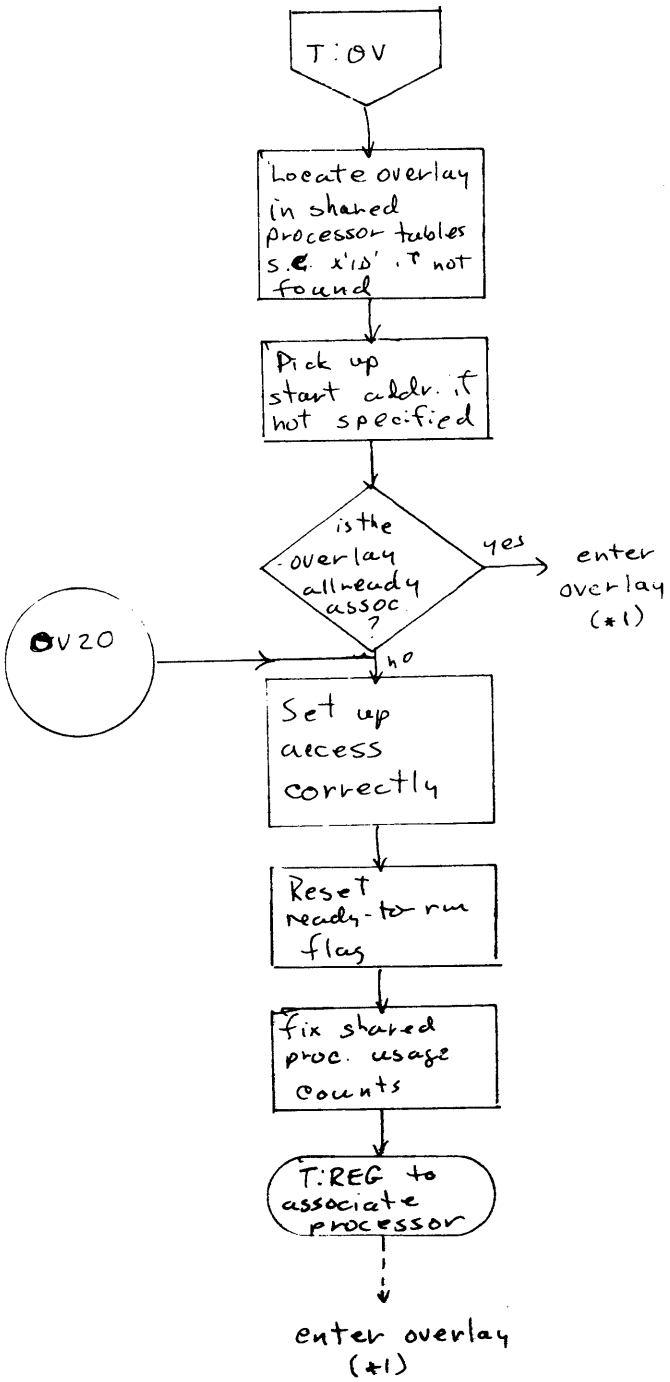
Software check X'1D' occurs if the specified segment cannot be located.

DESCRIPTION

See flow chart.

Figure EC-1


T:PROCOV
Initialize to work on shared proc. overlays
OVZO
T:OVER
Get monitor segment name 0→start add.
T:OV
T:OVERLAY
Get monitor segment name 0→start add.
Set up Reg. 14 so exit BAL from REMEMBER goes to T:OV
T:REMEMBER
unmapped ?
yes
exit
no
user have overlay ?
no
exit
yes
Push segment # and Reg 11 into OSTACK
BAL,11 exit (*14)
Pull remembered segment # and set up to reload and return
T:OV


T:OV
Locate overlay in shared processor tables s.e. x'1D' , T not found
Pick up start addr. if not specified
is the overlay allready assoc ?
yes
enter overlay (*1)
no
OV20
Set up access correctly
Reset ready-to-run flag
fix shared proc. usage counts
T:REG to associate processor
enter overlay (*1)

ID

MSEGLD - User program overlays

PURPOSE

To load a specified overlay segment into core storage.

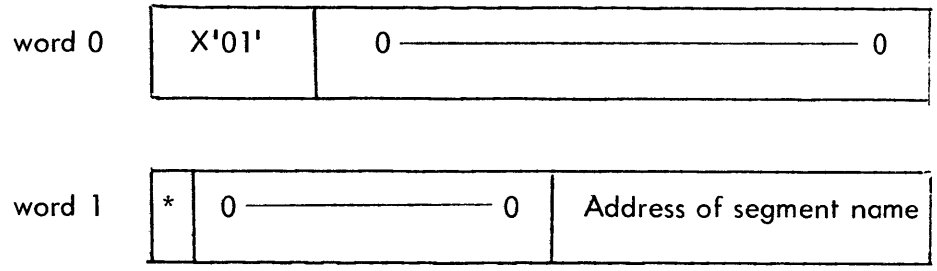
USAGE

B MSEGLD

(R5)= address of JIT.
(R6)=word 0 of FPT.
(R7)=address of word 1 of FPT.

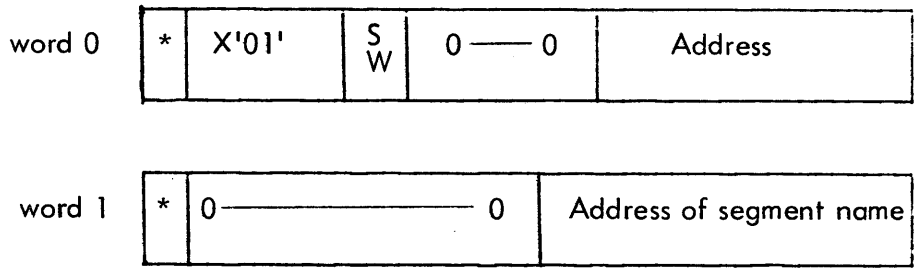
One of two FPT's may be associated with this CAL, a user FPT and a Monitor FPT.

USER FPT



Segment name must be in TEXTC format.

MONITOR FPT (Currently not used)



The segment name must be in TEXTC format.

where

SW = 1 if the exit is to the debug overlay
Address is the location whose first bit is to be set to zero.

EXIT

B TRAP EXIT of M:ENTRY if SW = 0.
Overlay debug segment if SW = 1.

SUBROUTINES

MEMSET: MEMSET gets all pages between the lower limit specified (either J:DLL or J:PLL) and the highest page required to contain the highest segment, releasing all pages between the highest page + 1 and the upper limit (either J:DUL or J:PUL) in: R2 = number of the highest page of data or procedure required to contain the highest segment
R3 = pointer to lower and upper limits of memory area, J:PLL or J:DLL.

MEM1: used by MEMSET to perform actual logic to get or free pages specified via T:GNVPI and T:RVPI.

ERROR CODES

B100	cannot find requested segment
B101	tree record is bad
B102	tree is circular
B103	data will not fit
B104	procedure will not fit
B105	bad bias on segment
B106	cannot obtain page
B107	SAD page encountered in unallocated memory area
B1XX	I/O error reading segment, where XX is the error code

Errors are reported via a branch to T:ABORTM in STEP after loading R14 with

DESCRIPTION

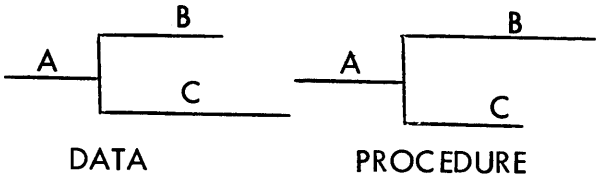
MSEGLD first checks to see if the seg-load CAL was issued by a shared processor. If it was, the name is verified in P:NAME and, if found, control goes to T:PROCOV in T:OV. If it is not found, the user is aborted with code X'B1'. If a user requested the seg-load, his DCB is verified. After checking whether or not the SEGLOAD DCB

is open and, if it is not, opening it, the routine verifies that the opening was normal, aborting the job (or ABORT1 of ENTRY) if it was not. Then it searches the Tree tables for segments previously in core, as indicated by the A bit (bit 0 of word 3 of each segment in the user's TREE) bein g set to 1. The routine resets this bit to 0 but stores the indication in the B bit (bit 1 of word 3). As a result all segments in the Tree table are marked as being out of core in bit A, while a record of segments actually in core is kept in bit B. This prevents having to reload these segments in case they are needed. This search also checks for a circular tree table, allowing only as many segments to be marked as there are in the table. If circular, a B101 ABC/ERO is returned.

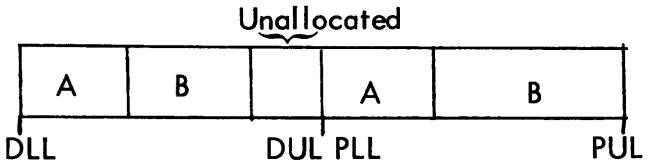
Next, the Tree table is searched for the requested segment. If the segment is found, and not in core, memory in both data and procedure is allocated via MEMSET for the segment and its backward path, which is then read into core. If the segment is already in core, the highest segment in core on the forward path is marked in, and MEMSET accessed to allocate memory accordingly. If the requested segment is not found, the job is aborted with a B100 ABC/ERO.

MSEGLD normally exits by branching to TRAPEXIT of ENTRY.

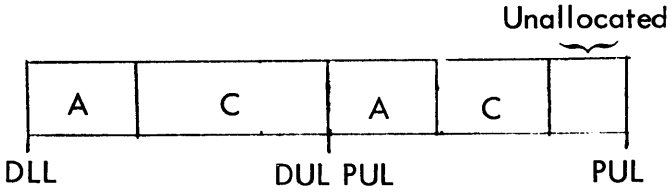
EXAMPLE: Assume segment B calls Segment C as shown below:



Memory allocation before SEGLD is called is as follows:



Memory allocation after SEGLD is called is:



ID

RDSET

PURPOSE

To read in the appropriate segment.

ENTRY

(SR4)= return address.
(SR2)= address of segment under consideration in Tree tables.
(R1)= 3 to read segment into data area (00 protection type).
= 5 to read segment in procedure area (01 protection type).

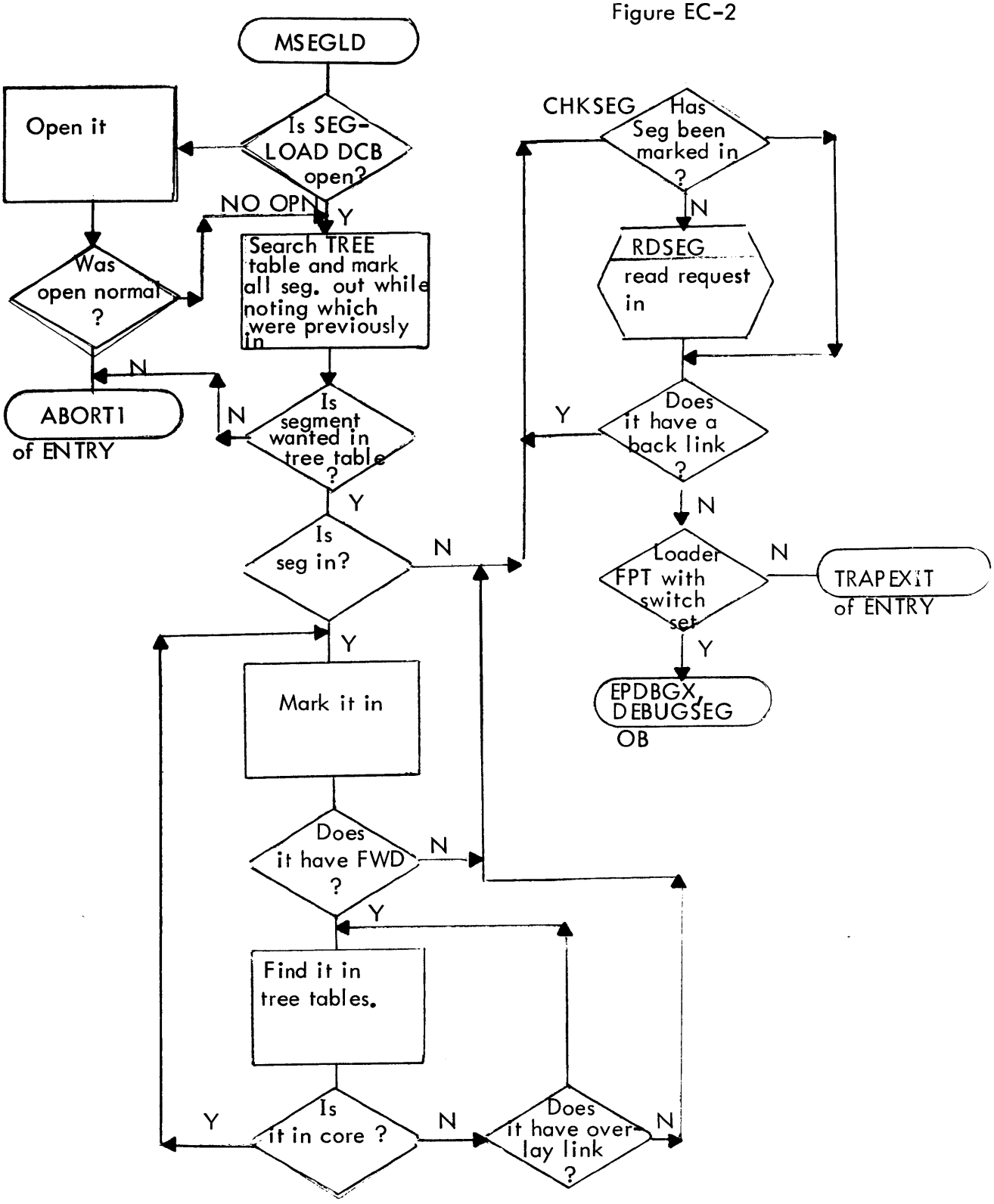
EXIT

*SR4 or EREXIT (which aborts to ABORT1 of ENTRY) if the segment was not read in normally.

DESCRIPTION

RDSEG searches the Tree table for the protection type indicated by R1. If the segment size is zero, RDSEG returns without reading a segment; otherwise, it prepares a PLIST in TSTACK and calls EPRDWT, ROOTSEG to read in the segment. RDSEG then checks the debug size of the segment and, if it is zero, returns to the caller. If the debug size is not zero, RDSEG replaces all the instructions indicated by CLTAB (Section LB.01, Clobber Table) with the appropriate debug CAL indicated by CLTAB. In case of an abnormal read-in or an error, RDSEG branches to EREXIT which eventually will abort the job at ABORT1 of ENTRY with an abort code of X'B1'.

Figure EC-2


MSEGLD
Is SEG-LOAD DCB open?
Open it
NO OPN
Y
Search TREE table and mark all seg. out while noting which were previously in
Was open normal ?
N
ABORT1 of ENTRY
Is segment wanted in tree table ?
N
Y
Is seg in?
N
Y
Mark it in
Does it have FWD ?
N
Y
Find it in tree tables.
Is it in core ?
Y
N
Does it have over-lay link ?
N
CHKSEG
Has Seg been marked in ?
N
RDSEG
read request in
Does it have a back link ?
Y
N
Loader FPT with switch set
N
TRAPEXIT of ENTRY
Y
EPDBGX, DEBUGSEG OB

ID

Swapper

PURPOSE

The purpose of the Swapper is to insure that a user is in memory in a form ready for execution. However, the Swap Scheduler rather than the Swapper determines what is needed to accomplish this. The Swapper and the Swap Scheduler try to keep memory filled with the users who are most likely to be chosen for execution.

The Swap Scheduler decides who should be Swapped in, whether the user needs any processors brought in from the Swapping RAD, and how to satisfy the physical page requirements of the user. The Swap Scheduler then passes this information on as input to the Swapper. The Swapper does all of the actual work involved. This may be as extensive as swapping several users out and bringing one user in from the Swapping RAD or as simple as associating another physical page, previously unavailable, with a user already in core.

The section on the Swap Scheduler indicates how the major decisions are made regarding what work is needed. This section describes what the work is that the Swapper is requested to do and why it is requested to do it. How this work is accomplished is detailed in the next two sections on SWAPOUT and SWAPIN.

OVERVIEW

Some memory management concepts are necessary for an understanding of this section. For more detail, see section F on Memory Management.

A user's mapping image, contained in JIT and called JB:CMAP, is the byte table containing physical page numbers that are loaded into the mapping registers prior to user execution.

A user is allocated a RAD granule whenever he is allocated a virtual page. Although these RAD granules are not necessarily sequential, they are ordered with the pure procedure always last, i.e., last encountered as the RAD rotates. The command chain or list (J:CL) and the table containing these granule addresses (JH:DA), used for Swapping a user, are contained in his JIT. Each four word entry of the command list contains a seek IOCD pointing to a disc address table entry and a read/write IOCD containing the physical page byte address.

A user is always swapped in and out in his entirety except when it can be determined by a bit in a user flag table, that a good copy of his pure procedure is already on the RAD, in which case it is not swapped out.

Following are the situations in which the Swapper is executed.

1) There is space in memory for a user presently on the RAD. The Swapper brings him into memory and further prepares him for execution. This Swapper preparation consists of putting his physical page numbers and those of the processors he shares into his mapping image, contained in JIT, and linking his page numbers together to construct his physical page chain.

2) There are one or more users in memory whose space is needed in order to bring in a user on the RAD who is more likely to be chosen for execution. In this situation, those users are Swapped out and the user on the RAD, more likely to be chosen for execution, is brought into the newly available pages and prepared for execution.

3) The system has associated a shared processor with a user by putting the processor's number into a user table. The Swapper must put the physical page numbers of those pages containing the processor into the user's mapping image. This can only be done if the processor is in memory. If it isn't, the Swapper must first bring it in from the Swapping RAD. These same actions for processors may also be required in conjunction with the first two situations mentioned above. Setting the processor into the user's mapping image, after swapping the processor in when necessary, must be done whenever a processor is initially associated with a user and also anytime a user is brought into core.

4) Some shared processors also require initial DCBs or data or both to be allocated to a user. Once allocated they become part of the user and are swapped with him. However, they must be initially brought in from a home area on the swapping RAD at the same time the shared processor is initially being set up by the Swapper.

5) A user has been allocated one or more virtual pages and at the same time there were no physical pages available. If a user requests virtual pages and no physical pages are available, Memory Management sets up the virtual pages, using for physical pages a monitor pure procedure page for identification, and allocates RAD granules for them. However, since Memory Management can't put good physical page numbers into the user's mapping image, it temporarily gives up control to the system. The only way that physical pages can now be put into the mapping image is by the Swapper. When the Swap Scheduler determines that pages are available, the Swapper sets up the mapping image. Some time later, now that the user is again ready to run, the system returns control to Memory Management following the point of give up. Memory Management can continue execution knowing that the Swapper has replaced the No Page Map Constants (NPMC, the monitor pure procedure page) with physical pages.

USAGE

The Swap Scheduler transfers control to one of two Swapper entries. Input and output are contained in core and not in the registers upon entering and exiting from the Swapper.

If users must be swapped out the Swap Scheduler branches to SWAPOUT which in turn goes to SWAPIN. Otherwise, the entry point is SWAPIN.

INTERACTION

T:SIO is used only by the Swapper and its function is to drive the I/O for the Swapping RAD.

Two other routines are used in this module. T:SENSE returns to the Swapper the present position of the RAD head so that when several sorted command lists are chained together, the chain can be broken such that the one closest to the present head position can be initiated first. T:SEXIT is used by both T:SIO and the Swapper to give up control either to wait for I/O completion or to exit.

ERRORS

Two methods of detecting RAD errors are used by the Swapper besides the normal hardware error detection using I/O error status bits.

The first is write checking. If a software switch (DOWTCK) is set, and it normally is, then the swapper's I/O module, T:SIO, performs the RAD hardware write checking function after all writes. If errors occur in write checking, T:SIO retires N times a sequence of write then write check.

The second method is read checking which is also controlled by a software switch (DORDCK) which is always turned off (reset) because of problems recovering HGPs that have been read checked. If read checking were performed it would operate as follows. When a user is Swapped out, unique consecutive identifiers are placed in the next to the last halfword of each page. The two halfword identifiers for 1) the user's first page and 2) his first page of pure procedure (needed since pure procedure is not swapped out every time), are saved in the user's JIT. When the user is swapped in, a software check is made by the Swapper of the identifiers in the user's pages to see that they start with the value saved in JIT and are consecutive. The halfword destroyed by an identifier is saved in the command chain for each page before it is swapped out and is restored during the read check.

RESTRICTIONS

The Swapper executes master, unmapped.

ID

SWAPOUT

PURPOSE

The purpose of SWAPOUT is to move one or more users from memory to the swapping RAD to provide core needed by SWAPIN.

INPUT

SB:OSN contains the number of users that are to be swapped out.

SB:OSUL is a byte table containing the user numbers of the users to swap out. The first byte of this table indicates the number of entries presently in the table and is called SB:OSN. SB:OSUL and SB:OSN are equated to the same location.

OUTPUT

S:FPPH, S:FPPT and S:FPPC are the Head, Tail and Count of the swapper's free page pool. They define the chain, linked in MB:PPUT, of the physical pages the Swap Scheduler is passing to the Swapper to be used for swapping in the user. SWAPOUT adds the page chains of each user being swapped out to the chain the Swap Scheduler has already created.

S:SCL is a Shell or empty Command List in resident core. It is used to built the JIT CLs for the users being swapped out and to build the processor CLs for swapping in processors pure procedure in SWAPIN.

SH:SDA is the area in resident core for the Disc Addresses needed by S:SCL.

#SWAP$DEV indicates to TSIO the number of interrupts it must receive before I/O is complete since that many requests were made of TSIO.

DATA BASES

JHSWPID is the displacement in JIT of the location containing the two half word identifiers used in software read checking. (Software read checking is explained in the SWAPPER section under errors.)

S:SWPCNT is used in connection with software read checking. It contains the next unique identifier and must be updated by SWAPOUT to be one greater than the value used in the last page to be swapped out.

S:BCL is a table containing the addresses of the beginning of the command lists for each user being Swapped out and each processor being Swapped in. It is used for linking the CLs together after sorting for RAD optimization.

S:ECL is a table containing the addresses of the locations following the CLs where a Transfer In Channel (TIC) is placed when linking the users' or processors' CLs together.

S:BDA is a table containing the first Disc Address for each user being swapped out or processor being swapped in. The table is used to sort and order the CLs for RAD optimization.

SH:EDA is a table containing the last DA of each user being swapped out or processor being swapped in.

J:CLP is set to contain a pointer to the word in the user's CL that is destroyed by a TIC in swapping the user out.

J:CLS contains the word following the user's CL that is destroyed by a TIC during SWAPOUT.

SB:OSULT is a temporary working table created identical to SB:OSUL by SWAPOUT.

SUBROUTINES

T:PGCHK checks that a physical page chain, linked through MB:PPUT, is valid and consistent with its head, tail and count. Input is the address of the head in register 7. If an error is detected, control is transferred to RECOVER with a screech code of 01.

T:RECORD creates a record of information in a wrap around recording buffer for use by ANLZ in case of a crash. Input is an identification code in register 1 as an indication of what information to collect.

The following three routines are used when multiple command lists must be ordered and chained together. The 4 tables S:BCL, S:ECL, S:BDA, and SH:EDA are created for use by these routines.

OSAC puts an index to the other 3 tables into byte 0 of each entry of S:BDA. It then sorts this table of beginning disc addresses in ascending order with respect to sector position. Input is the number of table entries in register 8. Output is the sorted S:BDA table. Control is transferred directly to CCL0.

CCL0 orders the command lists. It determines which command list follows most closely after another command list by finding which BDA is closest but larger in sector position to another CL's EDA. It starts with the first BDA, as sorted in OSAC, and places its index (i.e. backward link) into byte 1 of the BDA entry which most closely follows in sector position the EDA of the first. It then places a TIC from the first CL to the CL it has just determined is second. It continues by finding which CL follows the second and so on until the last is chained to the first. Control is then transferred back to the routine which called OSAC.

ULCLC is called after a sense of the RAD. Input is the RADs head's present sector position in register 10. ULCLC adds a delay (SDLAY) to this position and determines which BDA most nearly follows this. The corresponding BCL is set in register 6 and points to the beginning of the chain. The backward link, in byte 1 of S:BDA, points to the entries which more define the last CL. The ECL address of this last CL is set in register 5 and the routine exists. ULCLC provides the necessary information and the calling routine does the actual breaking of the chain.

DESCRIPTION

SWAPOUT sets up the command lists (chains of IOCDs) for all the users being swapped to one RAD, links these lists together and calls upon module TSIO to perform I/O to this RAD. If there is more than one swapping RAD, this process is repeated for each swapping RAD until all of the users in the out swap user list (SB:OSUL) have been swapped out.

At the beginning of SWAPOUT, the out swap user list (SB:OSUL) is copied into a similar table called the out swap user list temporary (SB:OSULT). The number of different RADs for these users is recorded in #SWAP$DEV to indicate the number of times TSIO must receive interrupts before I/O is complete. As SWAPOUT chooses the users from SB:OSULT that are on one RAD, it zeros their numbers in this list, sets up their chains and calls upon TSIO. When the list contains only zero, it exits.

The following then describes the functions done for each RAD before calling upon TSIO to perform all I/O to a RAD. When more than one user is being swapped out, (to a particular RAD), the command list for each user is built and then the command lists are ordered and chained together to optimize output. This ordering for optimization is done between users but not within a user since optimization within a user is done during dynamic granule allocation.

The following actions are executed for each user that must be swapped out. The user's physical page chain is added on the swapper's free page pool (chain). The two unique half word identifiers used in software read checking are set in JIT, unless pure procedure is not to be swapped out, in which case only the first is set up.

The command list contained in the user's JIT or AJIT is filled in with write orders. At the same time, the next to the last half word of each user page is saved in an unused part of the command list and the software read check identifier, incremented once per page, is placed in each page. The word following the user's CL (soon to be destroyed by a TIC) and the word's location are saved in JIT in J:CLS and J:CLP respectively so that the word may be restored when the user is swapped back in. The command lists for AJIT and JIT are built in that order in a resident area of core with a TIC to the user's CL. Setting up and connecting the CLs for the user and for his JIT is repeated for each user going to the same RAD in the out swap list.

If there is more than one user, the CLs are now ordered and chained together to optimize the RAD I/O. This ordering is accomplished using four tables created as the command list was set up for each user. The four values entered into the tables for each user are the first and last DA, used for ordering, and the beginning and ending locations of the user's command list, used for chaining the users together. After the lists have been chained together, in the multiple user case, a sense of the RAD is made and the circular list broken so that the I/O can be started on that user whose area on the RAD is coming up next.

Now SWAPOUT calls on TSIO to swap the one or more users out to their RAD. When TSIO returns, SWAPOUT sets up the users for the next swapping RAD, if more than one, and drives again to TSIO. When TSIO has completed the I/O for all swapping RADs, as indicated by a counter (#SWAP$DEV), SWAPOUT passes control to SWAPIN.

ID

Swapping RAD I/O - T:SIO

PURPOSE

When the swapper has set up a command chain, for which swapping RAD I/O must be performed, it calls upon T:SIO. TSIO calls upon the I/O system (IOQ) to do the actual I/O and interrupt processing. The I/O system returns to TSIO for end action.

OVERVIEW

TSIO performs error checks on the CL chain, sets up information in registers and calls upon NEWQ to queue up the request. When the interrupt occurs and processing is complete, the I/O system transfers control to the end action routine in TSIO. If an error occurred, the I/O system entered a record in the error log file, output a message to the operator's console and passed information about the error to the end action routine. The end action routine will retry the call N times, and if that fails it will set a user flag indicating the error and continue. If the I/O was successful, TSIO returns to the SWAPPER still on end action. However, if the function performed was a write, the I/O system is called upon to do a check write. If the function was reading a user, then TSIO performs a software read check before returning to the SWAPPER.

USAGE

```
T:SIO    BAL, 11    T:SIO
```

R6 = Address of beginning of command list.

R5 = Address of end of command list.

R7 = Function code; 2 for read and 1 for write

ERRORS

The screech codes reported by T:SIO are as follows:

0A Read or write orders in command list are not consistently one or the other but a mixture or in analyzing N read errors order is invalid.

0B Didn't find seek or sense order in command list when one or the other was expected.

OC Physical page number from byte address in IOCD with read or write order is not between values contained in LOW and HIGH.

OD Termination of command list doesn't agree with command list ending address input T:SIO or termination IOCD doesn't have flags of X'1E'.

OE No I/O is needed as indicated by input beginning address of command list address being equal to input ending address.

OF The function input parameter is not read or write.

93 N write errors occurred and the offending command list can't be found.

94 Discovered invalid order trying to continue write checking the rest of command list after N errors occurred.

95 N read errors occurred and there is an invalid address pointing to the offending command list.

96 N errors occurred trying to read a processor.

If a hardware error occurs, IOQ types a message, logs the error and returns to TSIO. After N errors occur, one of three flags is set in a user flag table (UH:FLG2) and TSIO continues, i.e., returns to the SWAPPER. Prior to execution of the user if one of these three flags is set, the error is logged and appropriate action taken. If the flag (bit 13) indicates that a write or write check failed on any page of the user or a read or read check failed and it wasn't in the user's context area (JIT, DCBs, etc.) then the message "SYSTEM SWAPPING ERROR" is output to the user and execution continues as usual. If however the error was in reading or read checking the user's context (bit 14) or user's JIT (bit 15), then the user is deleted.

INTERACTION

T:SSE Control is returned to the system following an interrupt.

RECOVER Is called as a result of failing consistency checks and unrecoverable I/O errors.

T:SEXIT Control is returned to the system to wait for I/O completion.

DOWTCK Is a software switch, normally set, requesting write checking.

DORDCK Is a software switch, normally set, requesting read checking.

SUBROUTINES

SET$REG sets the arguments into registers that are required for the call to NEWQ. Input to SET$REG is the doubleword command list address in register 0 and the DCT index in register 14.

DESCRIPTION

If software checking is required as indicated by sense switch 4 being set, T:SIO ripples through the complete chain of command lists checking for errors. Each command list entry, consisting of 4 words i.e. 2 IOCDs, must have an IOCD with a seek order followed by an IOCD with a read or write order. In one command list there must be only reads or writes but not both. Each 4 word entry must have termination flags of X'4C' in the second IOCD, or be followed by another 4 word entry with a seek order in the 1st IOCD, or be followed by a transfer in Channel IOCD. Each TIC IOCD must be followed by an IOCD containing a seek or a sense order. The command list must be terminated by an IOCD with a sense order or with X'4C' flags and this termination point must agree with the address of the end of the command list specified as input to T:SIO. All physical page numbers contained in the byte addresses of IOCDs with read or write orders must be within the range of physical pages, not containing the monitor, used by the system as defined (the range) by the values contained in locations LOW and HIGH. If any errors are found, T:SIO transfers to RECOVER with a screech code indicating the error.

If there are no errors, the number of retries is initialized and SET$REG is called to set up the arguments in registers for NEWQ. NEWQ is called upon to queue up the request. When it returns, T:SEXIT is executed.

T:SEXIT pulls a return address from the stack and transfers control to that location. When the swap scheduler was entered, the address of the caller was pushed into the stack. The first time T:SEXIT is called, it will return to that caller. When the I/O system has finished processing a swapper interrupt, it transfers control to end action in TSIO. This end action routine pushes into the stack the return location of the I/O system. End action transfers to the swapper, the swapper calls TSIO again and finally T:SEXIT gets executed again, which finally pulls and returns to the I/O system which returns to the point of interrupt. (See diagram DB-1.)

When the I/O system finishes the I/O and processes the interrupt, it transfers to T:SIOEA, the end action routine in TSIO, with information about any errors. T:SIOEA pushes the return address into the stack.

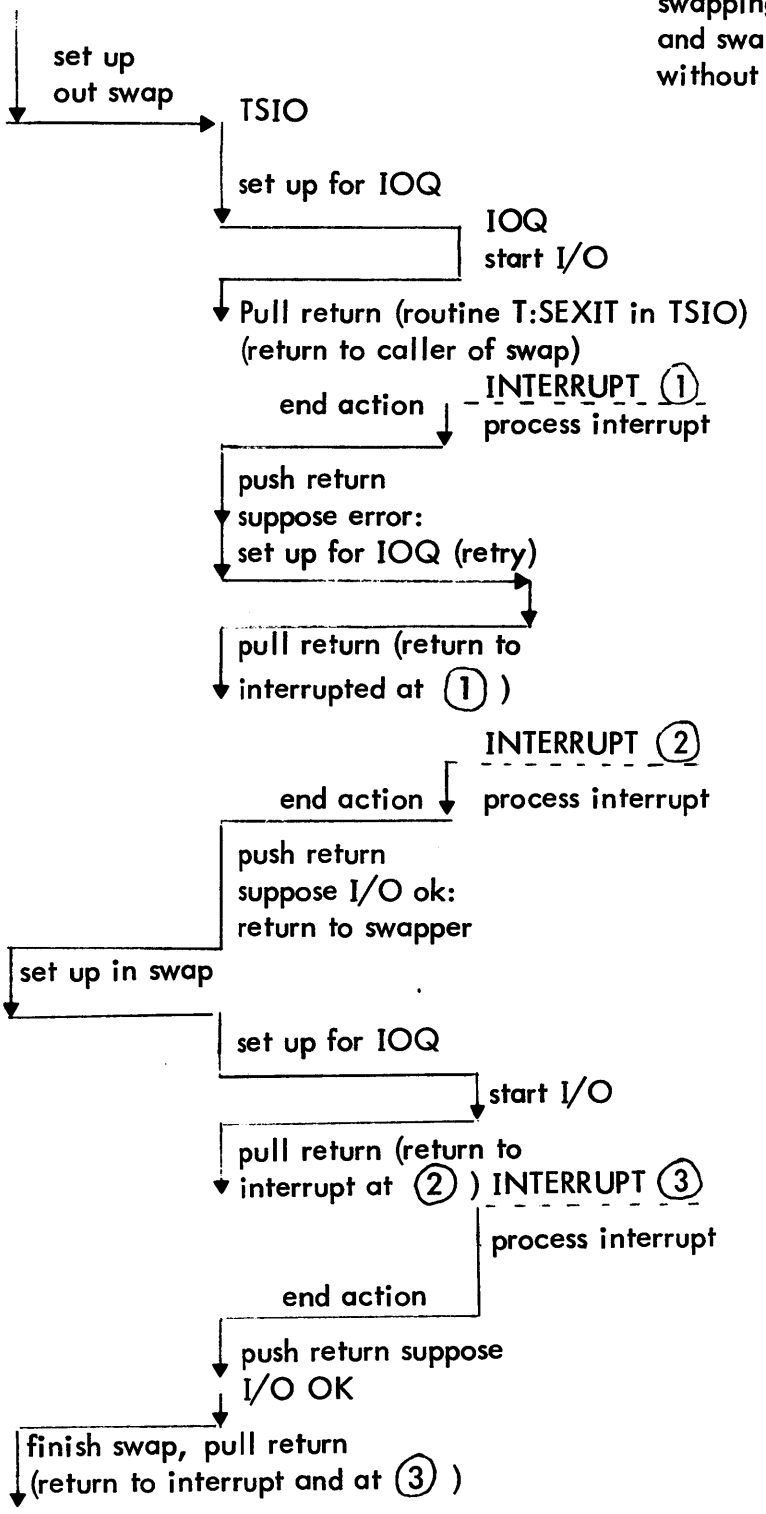
If the I/O system detected any errors, TSIO retries (by calling NEWQ) N times. If these retries are all unsuccessful, a user flag is set as indicated in the error section and TSIO returns to the swapper. If the function was a write check, retry consists of re-writing and then retrying the write check. If software read checking fails, retry consists of rereading.

All successful writes are write checked if DOWTCK is set. No matter how many CLs are in chain, it is executed at one time if the function is read or write. Write checking requires the chain to be partitioned and I/O initiated separately for each part. The AJIT and JIT are write checked first. When this is completed, the JIT can be altered by setting write check orders in the user's CL. If there is another user's JIT CL following, it can be done at the same time. So the routine ripples through the chain, changing write orders to write checks, until it finds a TIC from a JIT CL to a user CL, at which point it resets the chaining flag and sets the interrupt flag. After this I/O is completed, it continues where it left off until it finds the next user CL, and so on, until everything written has been checked. An unsuccessful write check results in only that section just checked, being rewritten and the rechecked.

When the function was reading a user (not processors, JIT or initial data and DCBs), a software read check is performed if DORDCK is set. Comparison is made to insure that halfword identifiers in the user's page start with the value saved in JIT and are consecutive. The halfword destroyed by an identifier is saved in the command chain for each page before it is swapped out and restored during this read check.

When all requested I/O has been completed, TSIO returns to the swapper.

Diagram DB-1: Relationship of SWAPPER, TSIO and IOQ. Illustration shows swapping out 1 user with one error and swapping in a user (JIT in core) without error.



ID

Swapping Disk Pack I/O - DPSIO

PURPOSE

When the swapper has set up a command chain, for which swapping Disk Pack I/O must be performed, it calls upon DPSIO. DPSIO calls upon the I/O system (IOQ) to do the actual I/O and interrupt processing. The I/O system returns to DPSIO for end action.

OVERVIEW

DPSIO performs error checks on the CL chain, converts the CL for use on a Disk Pack, sets up information in registers and calls upon NEWQ to queue up the request. When the interrupt occurs and processing is complete, the I/O system transfers control to the end action routine in DPSIO. IOSERCK is called to check for and log errors, MSGOUT is called to type error messages on the OC. If an error has occurred DPSIO will retry the call N times, and if that fails it will set a user flag indicating the error and continue. If the I/O was successful, DPSIO returns to the SWAPPER still on end action. However, if the function performed was a write, the I/O system is called upon to do a check write.

USAGE

```
DPSIO   BAL,11    T:SIO
```

R6 = Address of beginning of command list.

R7 = Function code; 2 for reading processors and JITs and 1 for write, 4 for reading user data.

ERRORS

The screech codes reported by DPSIO are as follows:

0A Read or write orders in command list are not consistently one or the other but a mixture or in analyzing N read errors order is invalid.

0B Didn't find seek or sense order in command list when one or the other was expected.

0C Physical page number from byte address in IOCD with read or write order is not between values contained in LOW and HIGH.

0D Termination of command list doesn't agree with command list ending address input T:SIO or termination IOCD doesn't have flags of X'1E'.

0E No I/O is needed as indicated by input beginning address of command list address being equal to input ending address.

0F The function input parameter is not read or write.

93 N write errors occurred and the offending command list can't be found.

94 Discovered invalid order trying to continue write checking the rest of command list after N errors occurred.

95 N read errors occurred and there is an invalid address pointing to the offending command list.

96 N errors occurred trying to read a processor.

If a hardware error occurs, DPSIO types a message, and logs the error. After N errors occur, one of three flags is set in a user flag table (UH:FLG2) and DPSIO continues, i.e., returns to the SWAPPER. Prior to execution of the user if one of these flags is set, the error is logged and appropriate action taken. If the flag indicates that a write or write check failed on any page of the user or a read failed and it wasn't in the user's context area (JIT, DCBs, etc.) then the message "SYSTEM SWAPPING ERROR" is output to the user and execution continues as usual. If however the error was in reading or read checking the user's context, then the user is deleted.

INTERACTION

T:SSE Control is returned to the system following an interrupt.

RECOVER Is called as a result of failing consistency checks and unrecoverable I/O errors.

T:SEXIT Control is returned to the system to wait for I/O completion.

DOWTCK Is a software switch, normally set, requesting write checking.

SUBROUTINES

SET$REG sets the arguments into registers that are required for the call to NEWQ. Input to SET$REG is the doubleword command list address in register 0 and the DCT index in register 14.

SSN - Set Sense Orders.
SSN replaces all SEEK within the limits contained in R2 and R3 (R2 = lower limit, R3 = upper limit) with SENSE orders. The second word of each SENSW CPW inserted by SSN is set to have a byte count of 0 and the skip flag on. These CDW's when executed are effectively NOP's.

SET$SCL - Insert SEEK order (in S:SCC) and disc address.
Inputs to SET$SCL are in R0 and R1. R1 contains an index into XTBL and R0 contains a disc address. SET$SCL gets a beginning and ending CL address from S:BCL and S:ECL and inserts a halt TIC in the latter and SEEK order in the former. The DA in R0 is placed in the two half words pointed to by the byte address in the aforementioned SEEK order. The byte count of the SEEK order is set to 4.

SET$JCL - Insert SEEK order (in S:JCL) and disc address.
Input to SET$JCL is a DA in R0. SET$JCL inserts a SEEK order in the first CDW of S:JCL and puts the DA(R0) in SH:JAJDA. S:JCL is the address of the command list used for swapping in JIT's/AJIT's.

SET$CL - Insert SEEK order and disc address.
Inputs to SET$CL is a DA in R0 and the address of the first word of a CL. SET$CL puts a SEEK CDW at the address specified in R4 and inserts the DA(R0) in the two half words pointed to by the SEEK order.

OU - Outswap User.
When DPSIO is entered with R7 = 1 a call on OU is made. OU first determines how much of S:SCL (commands for swapping out JIT's and AJIT's for each user are located in S:SCL) has been used for this swap. All seek commands in the list are then changed to SENSE (See SSN) with the byte count set to 0 (this has the effect of NOP when the CDW is executed). Next, items in SB:OSUL are reversed in order and placed in SB:OSULT so that entries in SB:OSULT will be parallel to entries in S:BCL and S:ECL. If SB:OSN>1, REORDR is called to form tables #TBL and XTBL (see Tables 1 and 2).

OU1 is the starting point for preparing individual user command lists for execution. User numbers are selected from #TBL in order of decreasing magnitude and a call is made to NEWQ for first writing then write-checking each user. When a number is selected from #TBL the corresponding command list in the user's JIT (or AJIT) is modified so that all SEEK's are replaced by SENSE commands with byte count of 0 (results in NOP). Then a SEEK command is inserted in the first CDW for this user in S:SCL. The disc address pointed to by the above SEEK is computed as follows: cylinder number comes from entry in UB:C# (set at initialization); track number = 0; sector number = 0 if AJIT present, 2 if no AJIT. A command (M:HLT1C) which results in a halt is placed at the end of the users command list. Thus each user is written to the disc pack by the execution of one SEEK command, and N (N = number of pages for the user) write orders.

Write checking is accomplished by first write-checking JIT and AJIT then placing a SEEK command (same as above except that sector number = 4) in the first CDW in JIT (AJIT). Therefore each user outswapped requires 3 calls on NEWQ; write user; write-check user's JIT/AJIT; write check user. At the completion of the write checking sequence and if the number of users left to swap out is >0, OU1 is called to prepare the next user for swapping.

IP - Inswap Processor.
If DPSIO is entered with R7 = 2 and S:SCL≤R6<SLC$END, and PB:HPP>0 for one of the procs in SB:PNL then IP is called to swap in one or more processors. The largest value in S:ECL is found and is used as the upper limit for calling SSN (S:SCL is the lower limit). If SB:NP>1 REORDR is called. Then all the processor CL's are chained together, ordered by increasing magnitude of Processor number. If the JIT for user in S:ISUN is in core a halt is placed at the end of the CL for the last processor in the chain. If the inswap user's JIT is out of core then a TIC to S:JCL is placed at the end of the processor CL and IJ is called to setup S:JCL.

IJ - Inswap JIT.
IJ is called if DPSIO is entered with R6 = S:JCL. If the JIT for the inswap user has never been in core then the location of the skeleton JIT is used in a call to SET$JCL. If the user has been in core then the user disc address is used for the call to SET$JCL. (See SET$JCL.)

IU - Inswap User
There are two sets of conditions that will result in a call on IU when DPSIO is entered. They are:

1. R7 = 4
2. a. R7 = 2
   b. S:SCL<R6<SCL$END.
   c. no processors to swap in.

The second case implies that only initial data and/or DCB's are to be inswapped. If R7 = 4 there is read checking to be done and therefore some user data is to be swapped in. There may also be initial data and/or DCB's to be swapped in the first case. If there is no initial/DCB data then IU does nothing. Otherwise IU calls SSN (R2 = (TSC3)+2 and R3 = SCL$END), computes the disc address of the initial/DCB data, and then calls SET$SCL. The swapper has already placed a TIC at the end of the user CL (if any) to the start of the initial/DCB CL.

REORDR - Reorder user or processor number tables.
Input to reorder is the address of table SB:OSULT or SB:PNL. REORDR uses the entries in the argument table to form tables #TBL and XTBL.

ACATDA - Determines disk address of ALLYCAT's JIT or data.
Input is a condition code setting resulting from a test of table UB:C# and, in R0, the normal sector address of AJITS, JITS, or data (0, 2, or 4). If the condition code is nonzero, ACATDA exits. Otherwise, ALLYCAT's bias of four sectors is added to R0 and physical disk address is computed.

If there are no errors in the construction of the command list[†], the appropriate CL conversion routine is called, the number of retries is initialized and SET$REG is called to set up the arguments in registers for NEWQ. NEWQ is called upon to queue up the request. When it returns, T:SEXIT is executed.

[†]For a description of how the command list is checked, see Section DB (TSIO).

T:SEXIT pulls a return address from the stack and transfers control to that location. When the swap scheduler was entered, the address of the caller was pushed into the stack. The first time T:SEXIT is called, it will return to that caller. When the I/O system has finished processing a swapper interrupt, it transfers control to end action in DPSIO. This end action routine pushes into the stack the return location of the I/O system. End action transfers to the swapper, the swapper calls DPSIO again and finally T:SEXIT gets executed again, which finally pulls and returns to the I/O system which returns to the point of interrupt. (See diagram DB-1.)

When the I/O system finishes the I/O and processes the interrupt, it transfers to T:SIOEA, the end action routine in DPSIO with information about the errors. T:SIOEA pushes the return address into the stack.

If the I/O system detected any errors, DPSIO retries (by calling NEWQ) N times. If these retries are all unsuccessful, a user flag is set as indicated in the error section and DPSIO returns to the swapper. If the function was a write check, retry consists of re-writing and then retrying the write check.

All successful writes are write checked if DOWTCK is set. The AJIT and JIT are write checked first. When this is completed, the JIT can be altered by setting write check orders and a SEEK order in the user's CL. The routine changes write orders to write checks, until it finds a TIC from a JIT CL to a user CL, or to a HALT, at which point it resets the chaining flag and sets the interrupt flag. An unsuccessful write check results in only that section just checked, being rewritten and then rechecked.

When all requested I/O has been completed, DPSIO returns to the swapper.

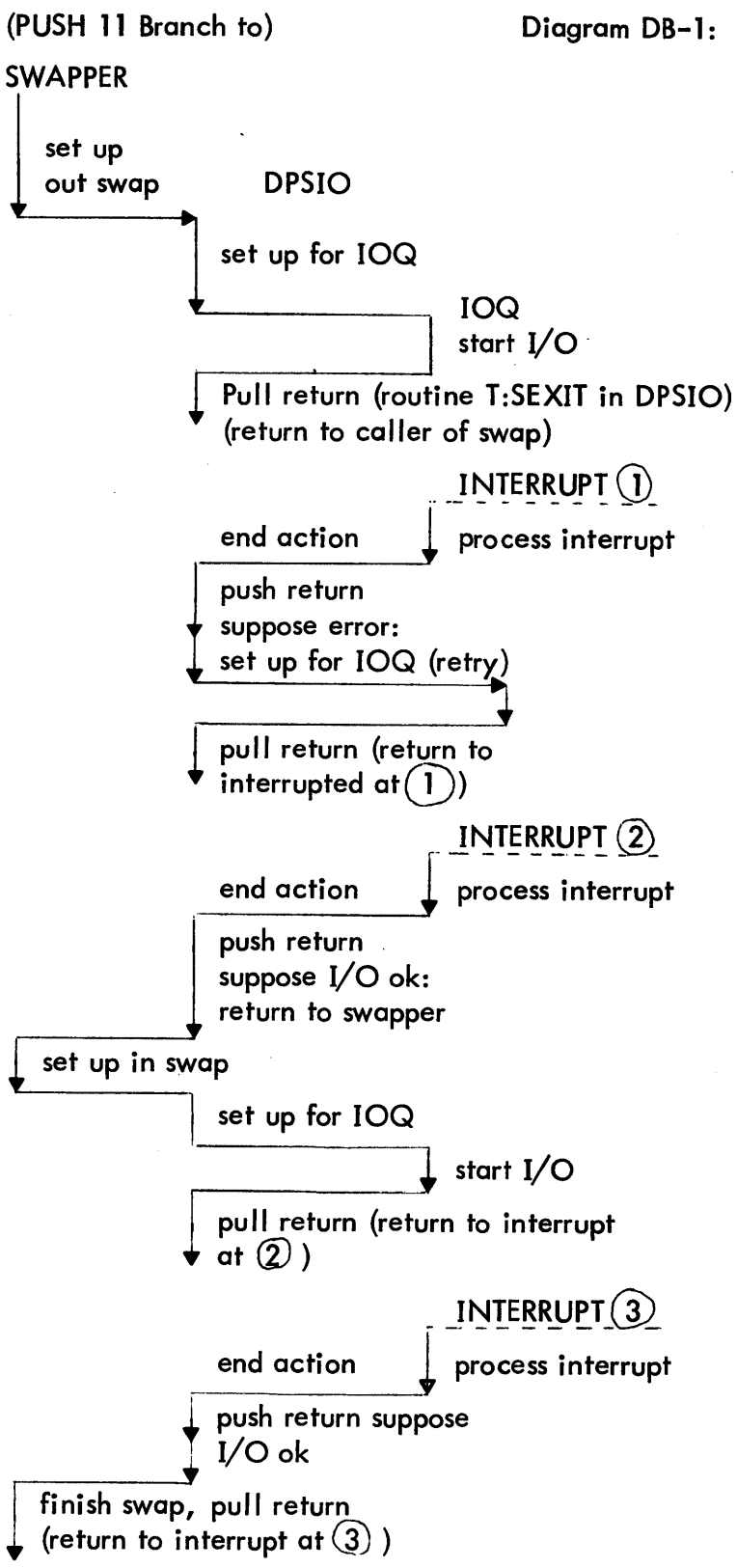


Diagram DB-1: Relationship of SWAPPER, DPSIO and IOQ. Illustration shows swapping out 1 user with one error and swapping in a user (JIT in core) without error.

TABLES

1. #TBL

   #TBL is a table of user or processor numbers ordered by increasing magnitude. #TBL is built by REORDR.

2. XTBL

   XTBL is a table of indices, built parallel to #TBL. Each index in XTBL is a byte offset in either SB:PNL (for processor numbers) or SB:OSULT (user numbers) where the parallel entry in #TBL is located. XTBL is built by REORDR.

3. S:SCL, S:BCL, S:ECL, S:JCL, SB:PNL, SB:OSUL, SB:OSULT

   See Section ED of UTS Technical Manual.

ID

SWAPIN

PURPOSE

SWAPIN sets 1) the numbers of any physical pages a user needs and 2) the numbers of the physical pages of any shared processors associated with him into JB:CMAP, (his mapping image) and adds the page numbers of the physical pages he needs to the user's physical page chain.

SWAPIN may also be required to bring the user and the shared processors associated with him into core from the swapping RAD. However this is not always necessary since the user and his shared processors may already be in core. This point is made in more details in the swapper section ED under overview.

INPUT

SB:PNL is a list of the processors that must be swapped in for the "in swap user." The maximum size of this list is four--a shared processor, one of its overlays (for our purposes considered a separate processor), a special shared processor such as TEL or DELTA, and a monitor overlay.

SB:NP is the first byte of word SB:PNL and contains the number of processors that must be swapped in.

S:PCT is the total page count required to swap in the user and processors.

S:ISUN is the number of the user that SWAPIN is to set up in a form ready for execution.

OUTPUT

UB:JIT is a resident user table containing the JIT's physical page number set up by SWAPIN.

S:SIP is a flag which is set by the swap scheduler whenever a Swap is In Progress so the swapper will not be reentered. When the swap has been completed SWAPIN resets it.

JAJ is the displacement in JIT of the location set to the physical page number of AJIT by SWAPIN.

DATA BASE

S:JCL is a resident area used to build the CLs for swapping in AJIT and JIT.

SH:JAJDA is a resident table used for disc addresses tor S:JCL.

SDLAY is a value equated to the sector delay required on the RAD between the granule for the user's JIT and the first user granule.

S:AJP is a location used to save AJIT's physical page number between setting up to swap in JIT and AJIT and setting AJIT's physical page number in JIT.

S:JSP is a location used the first time a JIT is brought into save JIT's Sector Position until the JIT has been swapped in. After a delay has been added to it, the value is set into JIT so that memory management will know at which position to attempt to obtain the user's first granule.

UH:JIT is a resident user table containing JIT's disc address.

UH:AJIT is a resident user table containing AJIT's disc address. The first time JIT is swapped in it must be brought from a home or initial area on the first swapping RAD. On this occasion UH:TS has been set up to temporarily contain the home disc address. UH:JIT has been set to zero to indicate that this is the first time and UH:AJIT contains the granule allocated for JIT. When the I/O is set up, the contents of UH:AJIT are moved to UH:JIT and UH:AJIT is zeroed.

SUBROUTINES

SPMAP puts the processor, whose number is specified in register 2, into the user's mapping image.

Other routines are described in the SWAPOUT section ED.01 under subroutines.

S:JITERR sets up the physical page head for any processors swapped in. It releases any unused physical pages back to the monitor free page pool and releases the user's AJIT page if he has one. The JIT page is left allocated to him until STEP deletes him. Then this routine goes to the end of the Swapper to exit.

DESCRIPTION

SWAPIN releases any extra unneeded pages, obtained as a result of swapping out users, to the monitor's free page pool. However, this is unnecessary if the swap scheduler transfers directly to SWAPIN since the scheduler provides the exact number when it can, i.e., when it doesn't have to swap users out to acquire them.

If processors must be brought in from the swapping RAD, a set of functions are executed for each processor. The processor's command list is set up in a resident core area containing a shell or empty command list reserved for building processor CLs for SWAPIN and JIT CLs for SWAPOUT. As the physical page addresses are set in the CL, their numbers are entered into the processor's physical page chain. The head of the processor's page chain is saved and set up after the processor is actually in core since the system determines whether a processor is out of core by testing the head for zero. During the building of the command list for each processor, the beginning and ending addresses of the command list and the first and last disc addresses are entered into four tables for use later in ordering and chaining the lists together.

Next, if the user is not in core, as indicated by a flag in a resident user table (UH:FLG), the command list for the user's JIT and AJIT are built in a resident area dedicated to that sole purpose. The AJIT, if allocated, is built first, followed by the JIT command list, since this is their ordering on the RAD.

If there are more processors than one to bring in, their command lists are ordered and linked together for I/O optimization using the information collected in the 4 tables as mentioned above. In order to start I/O on the processor which resides closest to where the RAD head presently is (plus a few sectors delay), the RAD is sensed and the circular command list appropriately broken. Then if both at least one processor must be swapped in and the user must be swapped in and is on the same RAD as the processors, a TIC is placed from the processors' command lists to the AJIT and the JIT command lists. The commands for AJIT and JIT are always executed last so that upon I/O completion the user command list contained in his JIT may be set up and the I/O started during the delay between reading the JIT granule and approaching the first user granule. The I/O is initiated for the processors first, if both the user and at least one processor must be read and are contained on different swapping RADs. As soon as TSIO has initiated the processor I/O, it returns control to the swapper who calls it again to initiate the I/O for the user's JITs. When all I/O is for one swapping RAD, there is only one call to TSIO, since all of the lists are linked together. In either case, once all I/O has been initiated, TSIO waits until all I/O is completed, as indicated by the counter, #SWAP$DEV, before it returns to the swapper.

If the JIT was not swapped in successfully, as indicated by a flag set by TSIO in UH:FLG2 or by the id in the 1st word of JIT not comparing with the user's id from UH:ID, then S:JITERR is called to handle this situation and exit from the swapper.

Whenever the shell command list (where the processor lists were built) was destroyed by TICs, it is reinitialized for use next time. If the JIT has come in for the first time, i.e., if it is a clean JIT for a new user, it is initialized and the necessary user defaults set up.

The heads of the physical page chains are set up for any processors swapped in. The physical page numbers for JIT and AJIT are set into the user's mapping image and into the user's physical page address, if any, and the user's physical command list address are set into JIT. If a TIC destroyed a word of the user's CL when the user was swapped out, this word was saved in JIT and is now restored to the CL. The physical addresses in the command list pointing to the disc addresses are initialized to reflect the physical page now containing JIT.

It is now possible to prepare the user's CL, and to set physical pages in his CMAP and his physical page chain. LMAP, the user's virtual page chain, provides the means to ripple backwards through the user's virtual pages in order to set up the CL, CMAP and the physical page chain. The process is completed either at the end of the chain or when the Virtual Link Chain Stop (J:VLCS) is reached, an indication that everything following is correct. If the user is in core, physical pages are entered into the tables only when there is a No Page Map Constant (NPMC) in CMAP, indicating that when the virtual page was allocated there was no physical page available. Now the swapper is supplying it.

If the user is not in core, physical pages are set in all entries of each table. The presence of a NPMC in CMAP is ignored unless a processor's initial data or DCBs or both must be associated with the user. In this case, when a NPMC is observed, a separate command list is built in the shell command list area using the homing area disc addresses for the initial data and DCBs disc addresses. If the user also must be brought in (because he was swapped out) then the read commands (not seeks) in the user's list for the initial and DCB pages will be shipped as a result of SWAPOUT setting the skip bit in the command list flags when it observed NPMCs.

If both the user is out of core and he requires initial data and DCBs, then there is a command list in his JIT (or AJIT) with skip bits set and a list in the shell command list area. If they are for different RADs, a halt is placed at the end of each and TSIO is called twice to start up each. When both are complete TSIO will return. If they are on the same RAD, a transfer in channel (TIC) will be placed from the user list to the shell list. In this case and when there is only one list required, TSIO is called once and returns when I/O is complete.

The pages of the processors associated with the user are set into his CMAP. The just-swapped-in and don't-swap flags are posted in UH:FLG2 to guarantee the user a new quantum and SL:SQUAN mils before outswapping him again. The final swapper function is to reset the Swap In Progress (S:SIP) flag so that another swap may be started.

UTS TECHNICAL MANUAL

ID

CLOCK4

PURPOSE

To process clock 3 counter zero interrupts including; update of date and time cells, report activation for sleeping users whose time has elapsed, periodic polling of COC lines for dial or hang-up, polling of busy I/O for time out, triggering the multi-batch scheduler (MBS) if a job can be started, triggering the remote batch scheduler if it is present, and computation of performance values. The code that performs these functions is located in the several appropriate modules (e.g. COC, IOQ). The clock routine simply adjusts counters and enters the special function code when the time comes.

USAGE

The clock routine is entered directly upon execution of the XPSD at clock 3 counter zero interrupt location X'5A'.

INTERACTIONS

ENTRY - A procedure defined in module ENTRY (Section CA) which pushes a 19 word environment into the unmapped monitor temp stack. The PSD pushed is indicated in the argument field of the ENTRY statement.

T:WAKEUP - a routine in module UCAL (Section IA) which decrements a counter for each user in the "sleep" queue. When the count goes to zero a wakeup event is reported on the user.

T:COCHC - A subroutine in COC which polls all the COC lines to see if an active user has hung up his phone or if someone has dialed into an inactive line.

RBSS - A subroutine in the remote batch modules used to test remote batch lines for dialup or hangup and generally maintain control over remote batch operation.

SGCQ - A subroutine in the module SACT used to trigger the RBBAT ghost to cause batch scheduling. The subroutine T:BTSCHED (used by KEYIN and others) is imbedded in-line in CLOCKP to allow quickest access in this the most frequent usage. If no partition modification is occuring (PL:LK, 0) the

INTERACTIONS (cont'd.)

following calculation is done:

(S:BUAIS - S:BUIS)*S:BFIS

If the result of this calculation is positive then there exists a job in the system not yet started and the number of batch jobs executing is less than the number allowed so that if proper resources are available RBBAT should start another job.

CTRIG - A subroutine in IOQ which results in busy devices being examined for device time out.

T:SYSTEMLOAD - A subroutine in PM which computes a running average 90% response point and the Execution Time Multiplication Factor (ETMF).

T:SSEC - an entry point in the execution scheduler (SSS) which results in a transfer of control back to the point of interrupt after seeing if anything else needs scheduling.

NEWQ - the non-DCB entry to IOQ. Used to output the time of day on the operator's console.

SUBROUTINES

DATESTIME - updates the date and time and outputs the time on the operators console. This routine is invoked every minute.

DESCRIPTION

When the clock routine is entered a check is made to see if 1.2 seconds has elapsed. If not, the total clock count is incremented and the clock pulse counter is reinitialized. If the pulse counter went more than 60 mls. (30 tics) negative, the above test is repeated (this implies 60 mls of disabled code or the run switch in idle). If the counter was less than 59 mls. negative, exit is directly back to point of interrupt, clearing the interrupt level by the LPSD. If 1.2 seconds have elapsed, the clock 4 pulse counter address is saved, the clock 3 interrupt level is armed and disabled (allowing lower level interrupts to come in) and an ENTRY procedure is executed to push an environment. The ENTRY procedure also modifies the clock 4 pulse count address to point to the overhead time bucket (J:OVHTIM). At every 1.2 second interval T:WAKEUP is invoked to wake any "sleeping" users whose elapsed time sleep interval has

DESCRIPTION (cont'd.)

expired. Also SS2 is tested and, if reset, the COC line polling routine T:COCHC is called. Every 4.8 seconds the I/O time out clock is incremented and the I/O activity scheduler CTRIG (in IOQ) is entered to queue a device time out checking cycle. Every minute the date and time are updated and the time is output on the operator's console. Also, T:SYSTEMLOAD (in PM) is called to update the ETMF and 90% response point. The total tics and the clock pulse count are incremented. If an environment wasn't pushed (fast path above) an LPSD takes control back to point of interrupt. If one was pushed, the clock 3 counter zero interrupt is enabled, the address for the clock 4 pulse counter is restored and exit is to T:SSEC in SSS.

ID

T:BTSCHED

PURPOSE

To select job for execution and initiate job if runable.

USAGE

Entry:	BAL, SR4 T:BTSCHED
Volatile Registers:	R1-R4, R12-R15
Exit:	B *SR4

DATA BASES

PL:LK	Partition lock flag (Section VI.02)
PL:CHG	Partition change flag (Section VI.02)
S:BUAIS	Batch users allowed in system.
S:BUIS	Batch users currently in system.
S:BFIS	Batch files in system

DESCRIPTION

T:BTSCHED is coded in-line in the module CLOCK4 for most efficient usage. It is used by GHOSTID, KEYN and STEP at system startup, operator signaled startup and batch job logoff to determine if another batch job may be started. If more batch users are allowed than are present (S:BUAIS greater than S:BUIS) and a job exists to start (S:BFIS non zero) MBS is signaled in the ghost job RBBAT to schedule one or as many as are startable.

ID

MM - Memory Management

PURPOSE

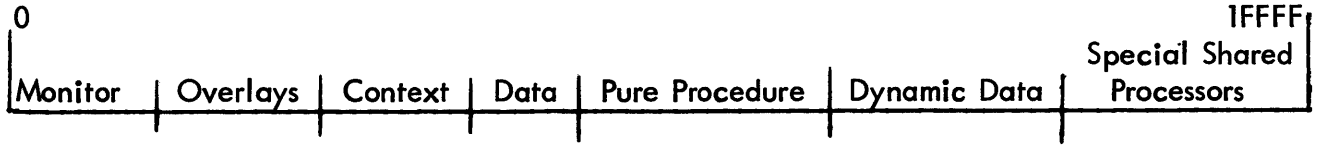
The purpose of Memory Management (MM) is to handle allocation of virtual and physical memory and allocation of the swapping RAD. This includes the setting up and loading of the map (JX:CMAP) and access (J:AC) images and loading them into their respective registers.

OVERVIEW

Besides explaining memory and swapping RAD management and details of the numerous routines involved, this section also clarifies the relationship between MM and the swapper. This is an explanation of many separate but related concepts. The first is about virtual memory. A user always executes in the mapped mode and addresses virtual memory. The hardware filters virtual addresses through the mapping registers, replacing the virtual page number portion of each address with a physical page number to obtain actual addresses. (In this section core memory pages are referred to as physical pages.) Virtual page numbers are used by the hardware as indexes to obtain the physical page numbers contained in the mapping registers. Prior to execution, these mapping registers are loaded with the user's mapping image, a byte table in JIT, JX:CMAP containing physical page numbers indexed by virtual page number.

Virtual memory looks like this:

For users loaded by the LOADER



For users loaded by LINK

The Monitor root is mapped one-to-one so that in the mapped and unmapped modes the addresses are the same.

Users context consists of JIT, AJIT, DCBs and buffers. Data is everything loaded with 00 protection type and may be altered at execution time. Dynamic data is obtained at execution time by a Get Page, Get Common Page or Get Virtual Page CAL and, when allocated, has 00 protection type. Pure procedure is everything loaded with 01 protection type so it may be looked at and executed but not stored into.

The virtual pages not allocated to the user have a free page map constant (FPMC) in

their CMAP entries and a 11 protection type (no access). FPMC is the physical page number of a monitor pure procedure page. If the user attempts to address virtual pages not allocated to him, he traps due to the 11 access code associated with each of these pages. When virtual pages are allocated to the user and there are no physical pages available, another monitor pure procedure page called no page map constant (NMPC) is set into CMAP as a flag for the swapper to put in a physical page.

All physical pages not containing the Monitor and the Monitor JIT are chained in MB:PPUT (physical page usage table), a byte table containing physical page numbers indexed by physical page number. There are four types of chains in PPUT and each chain is defined by a head, tail, and count. M:FPPH, M:FPPT, and M:FPPC define the Monitor's free page pool chain which consists of all free physical pages, i.e., pages not presently in use. S:FPPH, S:FPPT and S:FPPC define the chain containing the pages the swapper will allocate to a user about to be swapped in. PX:HPP, PX:TPP define the chains for the processors which are in core. A nonzero value for PX:HPP indicates that the processor is in core. PB:PSZ indicates the number of pages required for the processor pure procedure whether the processor is in core or not. JX:PPH, JX:PPT, and JB:PPC in JIT define the chain of physical pages allocated to a user in core.

Swapping storage is ordered on the RAD, with respect to sector position, in the same way that LINK orders virtual memory. That is, context first, data second, dynamic data next, and pure procedure last. The pure procedure is always last to avoid swapping it out, if possible. The user's swapping RAD granules are not necessarily contiguous or on the same track, or band, but they are ordered sector-wise with increasing virtual pages occupying increasing sector positions around the RAD. Swapping storage is allocated in groups of four contiguous granules, and the sector address of the first granule of a group is added to the disk address table, JH:DA.

A command list, J:CL, is maintained in the JIT for use in swapping the user's pages. If the user's size requires a larger command list than can be contained in the JIT, an additional JIT page, known as AJIT, is obtained and the command list is moved into it. Each allocated virtual page has an IOCD entry in the command list, ordered in the same way as the swapping storage on the RAD. Since swapping storage is allocated in groups of four granules, the command list contains one seek command for every four page I/O commands.

To maintain the ordered relationship between the user's command list, disk address table, and virtual pages, a chained list of allocated virtual pages is maintained in the byte table, JB:LMAP, containing and indexed by virtual page numbers. The chain is backwards, with the head, JB:VLH, corresponding to the last of the user's pages on the RAD and the last command list entry. The tail of the chain, JB:VLT, corresponds to the first user page, excluding JIT and AJIT and the first command list entry.

Command list entries are not made for the JIT or AJIT pages. Instead, a small command list is built in resident Monitor tables at the time of the swap.

As each virtual page is allocated, it must be inserted into the JB:LMAP chain and the command list in the proper order. It must also be given a granule of swapping storage at the correct sector position on the swapping RAD. The proper position in JB:LMAP and in the command list is found by simultaneously searching through both of the tables. When the proper position is found, the page address is linked into the LMAP chain. During the search, each entry in the command list is moved down one slot by moving the physical page address. When the proper position is found, the virtual page address is linked into LMAP and the physical page address of the new page is inserted into the command list entry.

Seek commands in the command list are not altered as commands are moved down, and each page in the command list, beginning at the one just inserted, is now being swapped onto the granule previously occupied by the command that follows it. The last entry in the command list has now been moved down and does not have a swapping granule assigned to it. There are two ways a granule may be obtained for this entry. If any granules remain from a previously allocated group of four, one of these is used and the number of remaining granules, JB:NGG, is decremented by one. If there are no granules remaining, a group of four is allocated. The disk address of the first of these is added to the end of the disk address table, JH:DA, and a seek command is added to the command list preceding the last page #IOCD. The number of remaining granules is set to three. To insure that swapping storage is allocated in the correct order on the RAD, a pointer is maintained containing the next sector position at which to allocate a granule. If a granule is not available at this position on any track, the allocation routines continue searching around the RAD until one is found.

DATA BASES

MX:PPUT contains a byte for each physical page in the machine. Through it are linked all the chains of physical pages in the system as follows:

M:FPPH, M:FPPT, and M:FPPC are the head, tail, and count of the Monitor's free page pool.

S:FPPH, S:FPPT, S:FPPC are the head, tail, and count of the swapper's free page pool which is the chain used to keep pages obtained by the Swap Scheduler or Swap Out until they are allocated to the user being swapped in.

JX:PPH, JX:PPT, and JB:PPC, in the user's JIT, are the head, tail, and count of the user's physical page chain for users who are in core.

PX:HPP and PX:TPP are tables which contain the heads and tails of physical page chains for each shared processor. They are indexed by processor number. If PX:HPP is non-zero, the processor is in memory.

Monitor pages, the physical unmapped JIT and Exec Delta pages do not appear in any chain.

JX:CMAP is a byte table (indexed by virtual page number) which contains the physical page number allocated to each virtual page. This image is loaded into the mapping registers when a user is in execution. Virtual pages which do not have a physical page allocated have a Monitor pure procedure page number, referred to as FPMC (currently 20, but any similar page will do), in that byte or halfword of CMAP. Access is not permitted to this Monitor page and the page is write locked. Virtual pages allocated to the user (a granule of swap storage is obtained) which do not yet have a corresponding physical page are mapped into another Monitor pure procedure page, referred to as NPMP (currently page 22). This occurs when the user requests a page and none is available; the virtual page is allocated, the tables are set up, and memory manage reports an event which causes the allocation to be made by the swapper directly if physical pages are available or through a swap if not. When control returns after the swap, the physical pages will have been assigned by the swapper.

J:JAC contains two bits for each virtual page and is the image loaded into the access registers when a user is in execution.

JB:LMAP has a byte for each virtual page through which the chain of virtual pages allocated to the user is linked. Pages in this chain are linked from high page number to low and divided into three segments: 1) pure procedure, 2) data, and 3) context, DCBs and buffers. A flag of '1' is set into LMAP to indicate pages acquired via the change virtual map CAL (T:SAD).

J:CL is the command list for controlling the user's swap. It consists of a write IOCD for each page allocated to the user, and a seek IOCD for every four pages. It is contained in JIT until so many pages are allocated that it won't fit (currently 16). At that time the virtual page above JIT, J:AJIT (for additional JIT) is assigned to J:CL. JIT and AJIT are not swapped via this command list, but rather using command pairs in resident Monitor core.

JH:DA is the table of disk addresses for the seek IOCDs in the command list.

Note that the ordering of the users on the RAD is AJIT (if any), JIT, context, data, dynamic data, and, finally, procedure. The command list is also necessary in this order, excluding AJIT and JIT. The virtual page chain in LMAP is in the reverse order. This ordering on RAD is designed to minimize swap transfer time. Each of the elements is obtained in ascending sector order. If pure procedure has not changed, it is not necessary to write it as the user is swapped out, and the swap is cut short. When a new group of four granules is allocated, the DA obtained is put at the end of JH:DA. The virtual page chain is searched, the physical page addresses are moved down one entry each until the proper sort order position for the new virtual page is found. Then the new physical page number is inserted in CMAP and the physical page address is placed in the command list entry made vacant by rippling down the physical page address.

Table KG-1: Memory Management Tables

Job (User) Associated Tables

JB:PPH	head	of physical pages associated with the user
JB:PPT	tail	
JB:PPC	count	
JB:CMAP	map image	
J:JAC	access control image	
JB:LMAP	link table through which the virtual pages of the user are chained	
J:CL	the I/O command list which controls swaps of all but JIT and AJIT	
J:DA	list of disk addresses for each page associated with the user	

Resident Tables

MB:PPUT	link table through which all physical pages are chained	
M:FPPM	head	of free physical pages
M:FPPT	tail	
M:FPPC	count	
PB:HPP	head	of physical pages assoicated with each shared processor
PB:TPP	tail	
SB:FPPH	head	of free pages accumulated between out and in phases of the swap
SB:FPPT	tail	
SB:FPPC	count	

UTS TECHNICAL MANUAL

ID

User CAL Service Routines

T:GP or T:GDP	Get Page or Get Dynamic Page
T:GCP	Get Common Page
T:GVP	Get Virtual Page
T:FP or T:FDP	Free Page or Free Dynamic Page
T:FCP	Free Common Page
T:FVP	Free Virtual Page
T:GL	Get Common Limits
T:SMP	Set Memory Protection
T:SAD	Search and Display

PURPOSE

T:GP	To acquire the next N available dynamic pages for a user from the bottom (lower addresses) of the user's dynamic data area.
T:GCP	To acquire the next N available Common pages for a user from the upper area (higher addresses) of his dynamic data area.
T:GVP	To acquire a specific virtual page.
T:FP	To free the last N dynamic pages allocated.
T:FCP	To free the last N Common pages allocated.
T:FVP	To free a specific virtual page.
T:GL	To get the limits of Common storage already allocated to the user; i.e. address of first (low) word and address of last (high) word allocated in Common.
T:SMP	To set memory protection; i.e. access, on the pages specified to the value (access code) specified.
T:SAD	To set the page number of a specified physical page into the mapping image entry of a specified virtual page so that the physical page may be looked at or stored into as determined by the privilege level of the user.

USAGE

CAL1,8 FPT

where FPT contains in

	word 1	word 2
T:GP	*X'08', N	—
T:GCP	*X'0C', N	—
T:GVP	*X'04', A	—
T:FP	*X'09', N	—
T:FCP	*X'0D', N	—
T:FVP	*X'05', A	—
T:GL	X'0B', 0	—
T:SMP	X'0A', A	AC, B (8, 24)
T:SAD	X'07', P	A

where N = Number of pages to allocate
where A = Address of virtual page to allocate or release or start setting access on
where P = Physical page address
where B = Last page address

Input at Entry Point			Output to Caller	
	Reg. 6	Reg. 7	Reg. 8	Reg. 9
T:GP	N	—	# allocated	Adr of lowest page allocated
T:GCP	N	—	# allocated	Adr of lowest page allocated
T:GVP	A	—	—	—
T:FP	N	—	# allocated	Adr of lowest page allocated
T:FCP	N	—	# allocated	Adr of lowest page allocated
T:FVP	A	—		—
T:GL	—	—	Adr of lowest common word allocated	Adr of lowest page allocated
T:SMP	A	FPT+1 adr	—	—
T:SAD	P	FPT+1 adr	—	—

If unable to accomplish what was requested, CC1 is set.

Reg. 5-11 non-volatile. Rest volatile.

SUBROUTINES

T:GP, T:GCP, and T:GVP use the general get page routine T:GVPM.
T:FP, T:FCP, and T:FVP use the general release page routine T:FVPM.
T:SMP uses T:SAC, the general set access into image routine.
T:XMMC, used to load the access register.
MAP and UNMAP used to go mapped and unmapped.
T:SETCC is used by all to exit. CC1 is set up in this routine.
T:SAD uses T:SAC, T:SXAC which loads specified AC registers and T:SXMAP which loads specified mapping registers.

ERRORS

If a routine is unable to accomplish its request, CC1 is set by the exit routine, T:SETCC.

RESTRICTIONS

These routines execute mapped, master.

DESCRIPTION

T:GP, T:GCP, and T:GVP are the interfaces between the user and the generalized get page routine, T:GVPI, which does the work, common to both the user (CALs) and the Monitor get page routines. They set up the input registers, call T:GVPI for each page required, update the pointers to the next available dynamic or Common page (JB:TDP - top of dynamic pages and JB:BCP - bottom of common pages) and set up the output registers in the stack to return to the user.

T:FP, T:FCP, and T:FVP are the interfaces to the general free page routine T:FVPI. They set up the input registers, call T:FVPI for each page to release, update appropriately the next available page pointers and set up the output registers.

T:GL gets the page number in JB:BCP, adds one to it, and makes it into a word address (by shifting left nine bits) and uses this value as the address of the lowest word allocated. It converts the contents of J:DDUL (dynamic data lower limit) to a word address and adds X'1FF' to it to obtain the address of the highest word allocated. If no common pages have been allocated yet (lowest word allocated - 1 = highest word allocated) the "address of highest word allocated" is used for both values.

T:SMP checks for pages acquired via the T:SAD operation; if so, the error return is taken. Otherwise, T:SMP sets up the input registers and calls on T:SAC, the general set access code routine, for each allocated page for which access must be set. It then calls on T:XMMC to load the access registers with that part of the image (J:AC) which was changed.

T:SAD first insures that the user has sufficient privilege (X'80') and that the virtual page to be used is not yet allocated. It then sets the physical page number into JB:CMAP (user's mapping image) and instead of linking the virtual page into LMAP, a flag of "1"

is set into JB:LMAP (a byte table containing a chain of the user's virtual pages). It sets the access to 00 for privilege X'B0' and higher and to 10 for privilege X'80' to X'B0'. It finally loads the map and access registers where affected.

ID

Routines to Set the Users Access Image

T:SNAC	Set N Access
T:IACU	Interrogate Access Code in User Image
T:ZPUP	Zero Pure Procedure Access Code
T:SAC	Set AC

PURPOSE

T:SNAC	To set a given Access Code (AC) on N pages starting at a given virtual page.
T:IACU	To interrogate user's AC image to determine AC on a given page.
T:ZPUP	To set AC to 00 on a given virtual page in pure procedure.
T:SAC	Sets the specified AC, access code, into the user's image for the specified virtual page.

USAGE

Calling Sequence		Input	Output	Volatile Registers
BAL, 11	T:SNAC	6 = No. of pages 7 = Starting virtual page no. 4 = AC to set	———	2, 3, 12, 13, 15
BAL, 11	T:IACU	7 = No. of virtual page to look at	CC3 and CC4= AC on given page	None
BAL, 11	T:ZPUP	7 = No. of virtual page whose AC is to be set = 00		None
BAL, 11	T:SAC	4 = AC 7 = VP# 13 = -1 if called by T:SMP, otherwise set to 0.	———	2-4, 12, 13, 15

SUBROUTINES

T:SAC, the routine that sets AC into the user's access image, is used by T:SNAC

ERRORS

If T:ZPUP is requested to zero access on a page which is not pure procedure, the action is not performed and the access code for that page is set into CC3 and CC4.

If T:SNAC is requested to set access on a page greater than X'FF', control is transferred to RECOVER with a X'21' screech code.

DESCRIPTION

T:SNAC is a driver consisting of a loop which calls upon T:SAC to set access on one virtual page. The loop first insures that the virtual page number is less than X'100' otherwise, transfer is made to RECOVER with a X'21' screech code. The loop then BALs to T:SAC. Upon return, one is added to the virtual page number and if the number of pages specified on input has not been completed it transfers to the beginning of the loop, otherwise, it exits.

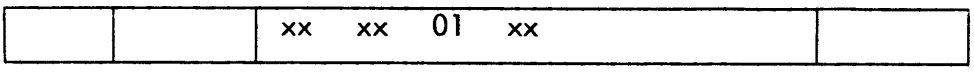
T:ZPUP sets a flag (in a register), indicating the action to be accomplished is zeroing AC on a pure procedure page, and drives to an entry in T:IACU which performs this function.

T:IACU goes immediately to IACU6, the terminating function with an AC of 3 if the virtual page is less than JOVVP, the virtual page number of the beginning of the monitor overlay area. Otherwise, it sets the flag register to indicate the requested function is to interrogate the AC and enters the common code used for both zeroing pure procedure and interrogating the access code image.

The AC for the given virtual page is obtained from the image. If the function is interrogation, control is transferred to IACU6. If the function is to zero pure procedure it also transfers there if the AC obtained was not 01; i. e. pure procedure access. Otherwise, it zeros the AC in a reproduction of the image setup in registers and loads this affected word from the image into the access registers and then falls into IACU6, the terminating function. IACU6 sets the AC to the requested virtual page into the top byte to the return address and into CC3 and CC4 and exits.

T:SAC divides the virtual page number by four to obtain 1) a displacement of the appropriate byte in the image and 2) the position of the double bit to be set within that byte. The position of the double bit is used as an index to get the appropriate mask containing ones in that double bit position. The AC to set is used as an index to get a byte containing the AC in all double bit positions. The byte is pulled from the image. The AC is set into the byte by a selective store and the updated byte stored back into the image; e.g. to set 01 AC into VP #10 MOD16:

Even reg.	01	01	01	01	Appropriate AC in all double bits
Odd reg.	00	00	11	00	Appropriate mask
			↓↓		selective store into
	xx	xx	01	xx	byte from image
			↓		store



Prior to storing the byte in the image if a register flag indicates this routine was called by T:SMP, a check is made to insure that the AC to set is not less than the access allowed on this virtual page. If it is less the store is skipped.

ID

Routine to Load the Hardware Map and Access Registers

T:SXMAP	Execute MAP
T:SXAC	Execute AC
T:SMMC	Set up MMC
T:XMMC	Execute MMC
T:PAC	Processor Access Control

PURPOSE

T:SXMAP Executes an MMC instruction to load the mapping registers with the user's image starting with the specified virtual page for a specified number of pages.

T:SXAC Does for access what T:SXMAP does for the map.

T:SMMC Does the actual setting up of registers for the MMC instruction for T:SXMAP and T:SXAC which are just drivers.

T:XMMC Sets up the entire user's mapping and access registers prior to user execution. This includes setting the access for a special shared processor when associated.

T:PAC Loads the AC registers with the access for a special shared processor.

USAGE

Calling Sequence		Input	Output	Volatile Registers
BAL, 11	T:SXMAP	12 = image adr 14 = no. of pages 15 = starting VP no.	3 = index to executing MAP MMC inst. 4 = shift code of -2 for T:SMMC	3, 4, 12-15
BAL, 11	T:SXAC	same as T:SXMAP	3 = index to execute AC MMC inst. 4 = shift code of -4 for T:SMMC routine	3, 4, 12-15
BAL, 13	T:SMMC	same as T:SXMAP and T:SXAC plus their output in reg. 3 and 4.	———	3, 4, 12-15
BAL, 11	T:XMMC	4 = user number	———	0, 2-4, 12-15
BAL, 11	T:PAC	———	———	0-4, 14, 15

SUBROUTINES

T:SXMAP and T:SXAC use T:SMMC
T:XMMC uses T:SXMAP and T:SXAC

DESCRIPTION

T:SXMAP and T:SXAC set up an index to be used to execute the appropriate MMC instruction; i.e. MAP or AC. They set up a shift code used by T:SMMC and then they BAL to T:SMMC which sets up the registers required by the MMC instruction. After returning, the MMC is executed and they exit.

T:SMMC sets up the registers required for the MMC instruction. Given A, the image address, N, the number of pages, P, the starting virtual page number, and X, the number of pages represented in an image word; i.e. 4 for MAP and 16 for AC, then the routine sets the registers to look like this:

even reg.	bits	0							15							31
		0							A + (P/X)							
odd reg.	bits	0					7	8		14	15			22	23	31
		((N-1+P/X)-(P/X)+1						0			(P/X)*X				0	

T:XMMC sets up input and calls on T:SXMAP and then does the same for T:SXAC. Then if the user has a special shared processor associated, determined by the special JIT access flag being set in the user's flag table, it loads a canned access word which makes JIT access 00.

T:PAC sets access into the user's image and loads the access registers for the appropriate processor of the current user. The appropriate processor is TEL if the TEL-in-control flag is set in the user's flag table. Or it is the associated debugger (contents of UB:ASP if the Debugger-in-Control flag is set, otherwise, it is the associated special shared processor (contents of UB:ASP). If nothing is set then default access is set up by using processor number 0. Once the number of the processor is ascertained, its two words of access are obtained from P:AC and set into the user's image, J:JAC and then that part of the image loaded into the access registers.

ID

Get and Release N Virtual and Physical Pages

T:GNVPI Get N Virtual and Physical Pages
T:GNVNPI Get N Virtual, No Physical Pages
T:GVGPI Get N Virtual, Given the Physical Page
T:RVSPI Release a Virtual Page, Save the Physical Page

PURPOSE

All of these routines are monitor requests:

T:GNVPI Gets N virtual pages (swap granules and core memory) for the user.
T:GNVNPI Gets N virtual pages (swap granule) but does not allocate physical core pages to them but rather puts the NPMC, No Page Map Constant, in the map.
T:GVGPI Gets a virtual page (swap granule) and uses the physical core page provided as input.
T:RVSPI Releases the specified virtual page (swap granule) but passes the physical core page back to the calling routine.

STEP uses T:GVGPI and T:RVSPI to move a user's DCBs into the DCB virtual pages from where they were created by Link.

USAGE

Calling Sequence		Input	Output	Volatile Registers
BAL, 11	T:GNVPI	6 = # of pages 7 = 1st virtual page #	5 = 0 = indication to get phy pg	0-4, 12-15
BAL, 11	T:GNVNPI	Same as T:GNVPI	5 = -1 indication to not get phy pg	0-4, 12-15
BAL, 11	T:GVGPI	3 = physical pg # 7 = virtual pg #	5 = -2 indication that phy pg is provided	0-4, 12-15
BAL, 11	T:RVSPI	7 = virtual pg #	5 = -1 indication to save phy pg 3 = physical pg #	0-4, 12-15

SUBROUTINES

T:GNVPI and T:GNVNPI use T:GAJP, if it is necessary to get an additional JIT page
T:GVPI, the general get virtual page routine
T:REG, if it is necessary to report No Pages and give up.

T:GVGPI also uses T:GVPI.

T:RVSPI uses T:RVPI, the general release virtual page routine.

DESCRIPTION

T:GNVPI and T:GNVNPI set a register indicating whether to get physical pages and then execute common code. If no physical pages are to be obtained, it is necessary to first get an additional JIT page (by calling on T:GAJP) now, if one will be required during this call. Since should no physical page be available for the AJIT page after some of the virtual pages had been obtained, the Swapper would give physical pages to those virtual pages and they aren't supposed to have any.

The general get virtual page routine, T:GVPI, is called N times. If the error return indicates the requested page was already obtained, the routine continues. However, if it indicates that the limit on the number of virtual pages allowed has been reached, it returns an error indication in the CC's to the calling routine. Whenever T:GVPI indicates that no physical page was available and one was desired, a note is made. Before exiting the routine, if this condition is noted, the routine reports a No Page event and gives up to T:REG. When control is returned, the physical pages will be allocated properly and the routine returns to the caller with the appropriate error status in the condition codes.

T:GVGPI sets an indication that the physical page is to be saved and calls on T:GVPI which does all the work. Upon returning, this routine returns to the caller.

T:RVSPI sets an indication to use the physical page provided and calls on T:RVPI which does all the work. Then it returns to the caller.

ID

Get and Release Virtual Pages Master/Internal

T:GVPM	Get Virtual Page Master
T:GVPI	Get Virtual Page Internal
T:FVPM	Free Virtual Page Master
T:RVPI	Release Virtual Page Internal

PURPOSE

T:GVPM is the interface between the get page CALs and T:GVPI. Its only function is to insure that the requested virtual page is in a user's data area, otherwise it returns.

T:GVPI is the general get virtual page routine. It does all the work for everybody.

T:FVPM and T:RVPI are the same except they are for releasing instead of getting virtual pages.

USAGE

Calling Sequence		Input	Output	Volatile Registers
BAL, 11	T:GVPM T:GVPI	7 = VP # 5 = 0 get PP = 1 no PP = 2 PP provided = 3 PP provided	———	All
BAL, 11	T:FVPM T:RVPI	7 = VP # 5 = 0 release PP = -1 return PP	3 = PP saved	All

SUBROUTINES

T:SGA and T:SGR are used to allocate and release swapper granules.
T:GPP and T:FPP are used to get and release physical core pages.
IV0 and DV0 are used to insert and delete entries from J:CL, JH:DA, JB:CMAP and JB:LMAP.
T:SAC to set access.
T:SXMAP and T:SXAC to load the Map and Access Registers.
T:REG is used to report No Disc event and give up control.

DESCRIPTION

T:GVPI checks that the virtual page, VP, is free and that the maximum number of pages allowed this user have not been allocated. The maximum number of pages allowed a user

is determined by the foloowing tests:

UB:PCT < 128

UB:PCT+PB:PSZ(APR)+PB:PSZ(AP0)+2(for COOP BUFs if allowed).
< limit supplied by SUPER or the default

UB:PCT+PB:PSZ(APR)+PB:PSZ(AP0)+2+PB:PSZ(TEL if on-line)
< SL:CORE and
SL:PCORE the physical core limit established at initialization.

If these conditions are fulfilled, the requested page is allocated to the user.

If the number of granules remaining from a previously allocated group of four granules (JBNRG) is zero, a swapper granule from the granule position specified in JB:NASP is requested from T:SGA. If none are available, E:ND, event no disk is reported and control is given up to T:REG. When control is returned, another attempt is made to obtain a granule. When a granule is obtained JB:NASP is updated to reflect the next granule group position and the number of remaining granules (JBNRG) is initialized to three. The disk address of the granule is placed in the next available entry of JH:DA and a seek command, with the physical address of the JH:DA entry, is added to the end of the command list. J:CLE is incremented by 2. If there is a granule remaining from a previously allocated group the granule allocation is not performed and no seek IOCD is constructed in the command list.

If this is a normal get page request, T:GPP is called to get a physical page from the monitor free page pool (MB:PPUT). If none were available or if this is a get virtual no physical request, NPMC, no page map constant is set up in place of the physical page and the Ready to Run flag in the user's flag table, UH:FLG is reset. If this request is get virtual and the physical has been provided, this logic is skipped.

The JX:LMAP chain is rippled down until the place is found where the new VP is to be linked. Each time a VP is passed in the chain; i.e., one ripple is done, the physical page address associated with that VP in CL is moved down one entry. When the ripple stops, the address of the new PP is set in the CL entry just vacated along with a Write order code. The J:VLCS, Virtual Link Chain Stop, it set to this VP if this VP is farther down the chain than where VLCS presently points.

If the PP is a NPMC, it is set in JX:CMAP. Otherwise, the PP is linked into the user's chain in MX:PPUT in the same order as LMAP. The user's page-count-needed total, UB:PCT, is incremented by one. The Pure-Procedure-must-be-Swapped flag in UH:FLG, is set since the rippling (we must assume) has resulted in the memory image becoming different from the swapping RAD image.

Tests determine which area (context, data, dynamic data or pure procedure), the VP is in and the count for that area, JB:PCP is incremented. The access code appropriate for that area is set into the user's access image, J:AC, by T:SAC unless it is context in which case it is skipped. T:GAJP is called in case AJIT, additional JIT page, is necessary as a result of this request increasing J:CL, the user's command list, to a size unable to fit any longer in JIT. T:GAJP makes the test and when necessary gets the AJIT page and moves the CL.

If a physical page, PP, was not obtained, the condition codes are all set and the routine exits. Otherwise T:SXMAP and T:SXAC are called to load the affected mapping and access registers. If however the Special-JIT-Access flag of UH:FLG is set and the VP is in the same area as JIT, then the special JIT access image word is loaded into the access/register instead of calling on T:SXAC. Then the condition codes are reset and the routine exits.

T:RVPI does nearly the opposite of T:GVPI. It makes sure the page is in use, and then ripples down JX:LMAP until it finds the VP and moves the PP memory address up J:CL until the VP's entry is wiped out. It unlinks the VP from LMAP and picks up the PP from CMAP and sets NPMC into that entry of CMAP. It also takes the PP out of the user's PPUT chain. It still must set the pure-procedure-must-be-swapped flag. It calls on T:FPP to release the physical page, PP, back to MX:PPUT unless this is a save-the-PP request. It then increments the number of remaining granules and, if the result is equal to four, it picks up the last DA entry in JH:DA and calls T:SGR to release the group of four granules and update JB:NASP. If the number of remaining granules is not four, or after the group of four has been returned, it decrements the appropriate area count and MB:PCT, calls on T:SAC to set the access to 11, calls on T:SXMAP and T:SXAC to load the map and access registers, and exits.

ID

Get and Free Physical Core Pages

T:GPP Get Physical Page
T:FPP Free Physical Page

PURPOSE

T:GPP gets a free physical page, PP, by getting the head, M:FPPH, of the monitor's free page pool and setting the next page in the chain into the head. If the head was 0 it returns that as an indication that it could not get a page.

T:FPP releases a PP back to the monitor's free page pool by linking it onto the last page of the chain and updating the tail, M:FPPT.

USAGE

Calling Sequence	Input	Output	Volatile Registers
BAL, 11 T:GPP	____	3 = free PP#	3, 4
BAL, 11 T:FPP	3 = PP # to release	____	4

(Page 100 has been deleted)

ID

Swap RAD Granule Allocation and Release

T:SGA	Swapper Granule Allocation
T:SGAJIT	Swapper Granule Allocation with no associated user
T:SGR	Swapper Granule Release
T:SGRNU	Swapper Granule Release with no associated user

PURPOSE

T:SGA Allocates a swapper granule from the swapper granule pool at the granule position specified if possible, or if not, the next higher position and so on until one is found. (The swapper granule pool is initialized with one of every four granules available.)

T:SGR Releases back to the free swapper granule pool the specified granule and returns its position. (The specified granule must be a multiple of four.)

T:SGAJIT and T:SGRNU are the same as SGA and SGR except that there is no relevant user to provide swap RAD index. This occurs when SYSMAK initialized the tables and when SSS obtains a granule for a user JIT.

USAGE

Calling Sequence		Input	Output	Volatile Registers
BAL, 11	T:SGA	1 = granule position requested	15 = disk adr of granule allocated	1, 2, 12-15
BAL, 11	T:SGR	15 = disk adr of granule to release	1 = granule position of released granule	1, 2, 12, 13
BAL, 11	T:SGAJIT	2 = swap RAD index		
BAL, 11	T:SGRNU	2 = swap RAD index		

DATA BASE

The user's swap RAD index, from UB:SWAPI, is used to obtain the appropriate entries from the following tables which describe the RAD and are used in the allocation routines.

NAME	CONTENTS				USAGE
RAD TYPE	0	1	2	3	
MB:GAM1	X'3F'	7	7	7	GRANULE ADDR. MASK
MB:GAM2	1	3	7	15	(SGP Wds/Gran)
MB:GAM3	-1	-2	-3	-4	Shift for SGP index to Gran. Pos.
MB:GAM4	6	3	3	3	Shift for track to Gran. Addr.
MB:GAM5	-7	-4	-4	-4	Shift for Disc Add. to Track No.
MB:GAM6	X'7F'	X'F'	X'F'	X'F'	Sector Addr. Mask
MB:GPT	41	6	6	6	Gran. Per Track

NAME	CONTENTS				USAGE
RAD TYPE	0	1	2	3	
MB:SWAPS	0	1	2	3	Shift for Gran. Pos. to SGPX
MB:DWT	41	12	24	48	DW size of SGP

Where RAD TYPES are

TYPE	RAD	PSA (HEX)
0	7212	
1	7232	$0<PSA\leq 80$
2	7232	$80<PSA\leq 100$
3	7232	$100<PSA\leq 200$

DESCRIPTION

The swapper granule pool, M:SGP, is initialized by SYSGEN or during system initialization to indicate the availability of every fourth granule. T:SGA starts at the specified granule position (word in M:SGP), looking for an available granule and increasing it until one is found. If there is none, CC4 is set and the routine exits. When one is found, the word (equivalent to a granule position) of the table containing it is exchanged with a zero. The least significant bit is extracted. Its position from the right is determined which becomes the band part of the address and the position of the word of the table indicates the sector part. The word is restored to the table with that least significant bit reset to indicate that it has been allocated. Its address is set up and the routine exits.

T:SGR checks the input granule address to insure it is the first of a group of four granules. If it is not, it is not returned to M:SGP and the program returns with CC4 set to 1. If it is the first granule of a group, T:SGR breaks up the address into the band and sector parts. The band number is used to create a bit number from the least significant bit.

This bit is selectively stored into the word specified by the sector position divided by 2. If there are any users in state DP (disk page), event E:DPA, disk page available, is reported.

ID

T:GAJP Get AJIT Page

PURPOSE

T:GAJP determines whether an AJIT page is required and obtains it if it is.

USAGE

Calling Sequence	Input	Output	Volatile Registers
BAL, 0 T:GAJP	12 = command list length	—	0-4, 6, 11-15

SUBROUTINES

T:SGA is used to get a swapper granule.
T:REG is used to report a no disk or no physical page event and give up.
T:SGR is used to release a swapper granule.
T:GPP is used to get a physical page.
T:SXMAP is used to load the mapping registers.

DESCRIPTION

T:GAJP immediately returns if the specified command list length is less than the maximum length in JIT called JCCL or if an AJIT page has already been obtained as indicated by J:AJ≠0. If it is necessary to get an AJIT page, the JIT's disk address is changed to the second granule of the group allocated for the JIT, and the AJIT is assigned to the first granule of the group.

UB:PCT, the user's total-pages-needed count is incremented. A physical page is requested of T:GPP. If none is available, an NPMC, No Page Map Constant, is set in JB:CMAP, the Ready to Run flag is reset and an event no-core, E:NC is given to T:REG. When control is returned, the Swapper has set a physical page in CMAP and linked it in MB:PPUT in the user's PP chain, and the Ready to Run flag is set. If there was no physical page, the Swapper recognizes the condition by the CMAP entry = NPMC and J:AJ = 0.

If the PP was obtained, it is linked into MB:PPUT in the user's chain, set into JB:CMAP, and the mapping register loaded.

In either case, the context area count is incremented by 1. The physical page number is set into J:AJ and its address set into J:CLPA. The disk address is set in UH:AJIT.

UTS TECHNICAL MANUAL

ID

ALLOCAT

PURPOSE

ALLOCAT controls the allocation of file and symbiont granules through the use of in core granule and cylinder stacks and non-resident HGP bit maps.

ALLOCAT is master mode ghost program. It is loaded with the public HGPs, generated by PASS2 of SYSGEN, and with the Monitor's REF/DEF stack. This allows it direct access to the Monitor and all its functions.

Replacing public HGPs in memory are PUSH/PULL stacks of disk addresses. There are four stacks for public granule allocation:

1. PFA on RAD
2. PFA on Pack
3. PER
4. Public cylinders

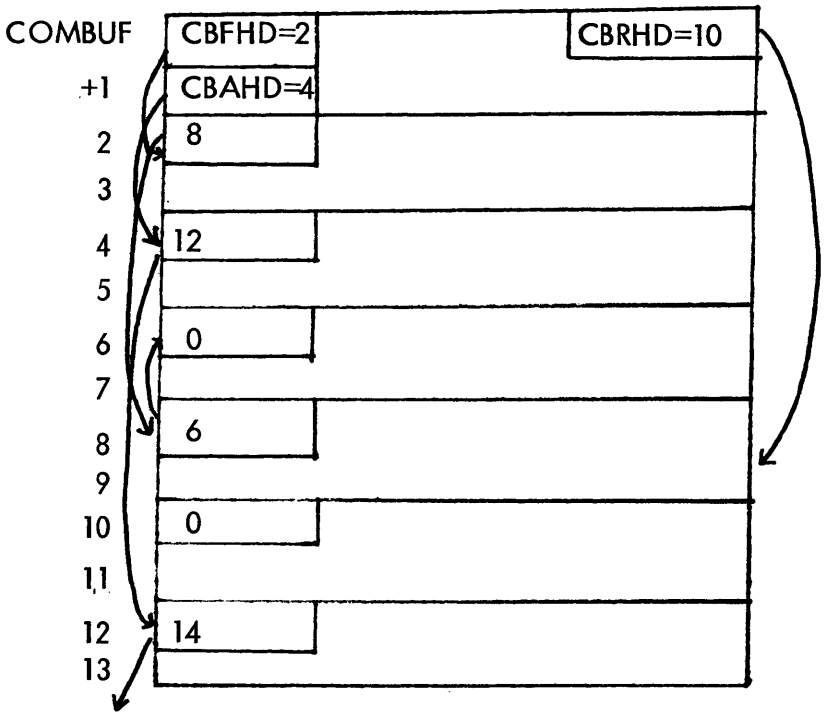
All GET and RELEASE granule routine calling sequences are identical to previous versions. The GET and RELEASE background granule, symbiont granule, and public cylinder routines are replaced by PULLs and PUSHs, respectively, of disk addresses into the appropriate stacks. For each allocation stack, two values are kept to control the calling of ALLOCAT and its actions when called. They are as follows:

1. A low and high threshold expressed as a single value: BUFTHRSH
2. An optimum fill :BUFSOPT

ALLOCAT is called if either the number of granules in any stack falls below the low end threshold for that stack or if the number of granules in the stack rises so that there remains less than the threshold space in the stack. After each PUSH and PULL, the space count and word count in the stack pointer doubleword are compared with threshold data values for that stack. When either is less T:GJOBSTRT will be called to wake ALLOCAT. When ALLOCAT is called, it adjusts all the stacks to their optimum fill level.

ALLOCAT pulls disk addresses from the stacks and releases the corresponding granules to its HGPs or allocats granules from its HGPs and pushes their disk addresses. Other functions require the presence of the HGPs and thus ALLOCAT. These explicit tasks are communicated to ALLOCAT through two-word communication buffers. The buffers are chained to one of three heads: CBFHD, the head of the free chain; CBAHD, the head of ALLOCAT's chain; and CBRHD, the head of the chain being used for releasing files. The buffers are linked through the first byte. A zero terminates a chain. (The zeroth buffer is not used as a buffer, but contains the headers themselves).

The figure below shows a sample of communication buffer linking.



In this example, buffers 2, 8, and 6 are free. Buffers 4, 12, 14 ... contain messages for ALLOCAT and buffer 10 is being used to release file.

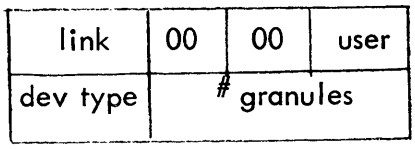
When ALLOCAT's services are required, the calling routine gets a buffer and calls a subroutine, ALLOQ, which adds the buffer to ALLOCAT's chain and wakes it, if necessary. If the routine needs to wait for a response, it blocks the associated user program with an E:QFAC (queue for ALLOCAT) event. ALLOCAT is swapped in and reads the chain of messages. If there is no response necessary, ALLOCAT frees the buffer; otherwise, ALLOCAT unchains the buffer from its own chain and leaves the answer in the buffer. Users are unblocked when ALLOCAT has

been swapped out (E:UQFAC). The calling routine must have remembered any buffer addresses containing answers and rechain the buffers to the free chain.

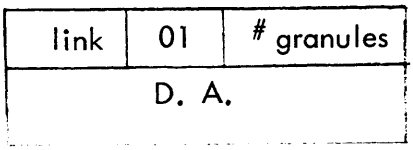
If no buffers are available, the user is blocked by ALLOQ until ALLOCAT makes some available, and will be unqueued when ALLOCAT is swapped.

The format for each message buffer is as follows:

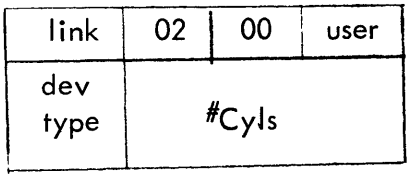
1. Get n Contiguous Background Granules



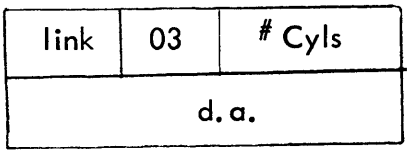
2. Release n Contiguous Background Granules



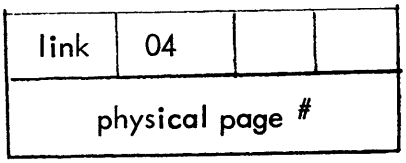
3. Get N Contiguous Background Cylinders



4. Release n Contiguous Background Cylinders



5. Release Buffer or Disk Addresses



DESCRIPTION

1. Adjust Stack Logic

Each stack has an associated threshold value (BUFTHRSH) which determines when to wake ALLOCAT. If the stack space count or word count are less than the given threshold ALLOCAT is called.

If no granules are available for a user, the event, E:QFAC, is issued causing the user to go to a queue, SQFAC, to wait until ALLOCAT has adjusted the stacks. If a symbiont requires a granule it must queue itself. SACT reactivates the symbiont periodically.

When ALLOCAT finishes its tasks, it posts a flag which causes the swap scheduler to swap it out next. At its next entry, the swap scheduler sets up the necessary tables and drives directly to the outswapper. When the outswap is complete, an event is reported (E:UQFAC) causing the SQFAC queue to be emptied.

Everytime ALLOCAT is called it automatically adjusts all stacks to their optimal depths (no communication buffer required). The stack pointer doublewords for the four public stacks begin at BUFSPD and are contiguous. BUFSOPT is a byte table telling their optimal depth. If granules are being added to the stack, they are put at the bottom causing granules already in the stack to be pulled first. All disk addresses added have the high order bit set to flag them as unusable until the HGP is updated on RAD (end of ALLOCAT's outswap). This assures that a granule is never used until it has been recorded in the HGPs on RAD. After the outswap, the flag bits are reset, before SQFAC is emptied.

2. Get n Contiguous Background Granules

GNBG calls ALLOQ and blocks the user with the event E:QFAC. ALLOCAT attempts to allocate the requested granules. It then removes the communication buffer from its own chain (but does not add it to the free chain) and puts the disk address in the second word (d.a. = 0 implies could not allocate). The user is unblocked after the outswap.

3. Release n Contiguous Background Granules

RNBG will call ALLOQ and continue. ALLOCAT releases the granules and frees the communication buffer.

4. Get n Contiguous Background Cylinders

GNCYL parallels GNBG.

5. Release n Contiguous Background Cylinders

RNCYL parallels RNBG.

6. Release Buffer of Disk Addresses

CLOSE builds chains of buffers containing disk addresses to be released. When the chain is complete, the disk addresses are released to the stacks, if they will fit, (the fastest method for release of small files). Otherwise, CLOSE acquires a communication buffer, frees the physical page, calls ALLOQ, and continues. If no communication buffer is free, it blocks with the E:QFAC event and retries when unblocked.

ALLOCAT releases the granules and removes the buffer from its chain, placing it in the CBRHD chain of buffers of disk addresses being released. At the E:UQFAC event (end-of-outswap) a routine checks for buffers. If there is one, it releases the page and communication buffer if it is the end of the chain; otherwise, it reads in the next buffer full of disk addresses. At the end-action from the read it recalls ALLOQ and repeats the process until all buffers in the chain have been processed. This process causes a swap of ALLOCAT for every 508 granules released. This occurs infrequently enought to be an insignificant addition to system overhead.

7. Keep Count of Granules

ALLOCAT, upon first entry (at boot and recovery), counts the number of available public granules and records them in a table in memory called GRAVAIL, the permanent available granule counts.

GRAVAIL	# Granules	PFA RAD
+1	# Granules	PFA Pack (granule allocated)
+2	# Granules	PER
+3	# Cylinders	PFA Pack (cylinder allocated)

These are updated every time ALLOCAT adjusts stacks. These counts are used for the DISPLAY DISC key-in and for triggering symbiont truncation. ALLOCAT also keeps the same counts in itself in AGRAVAIL.

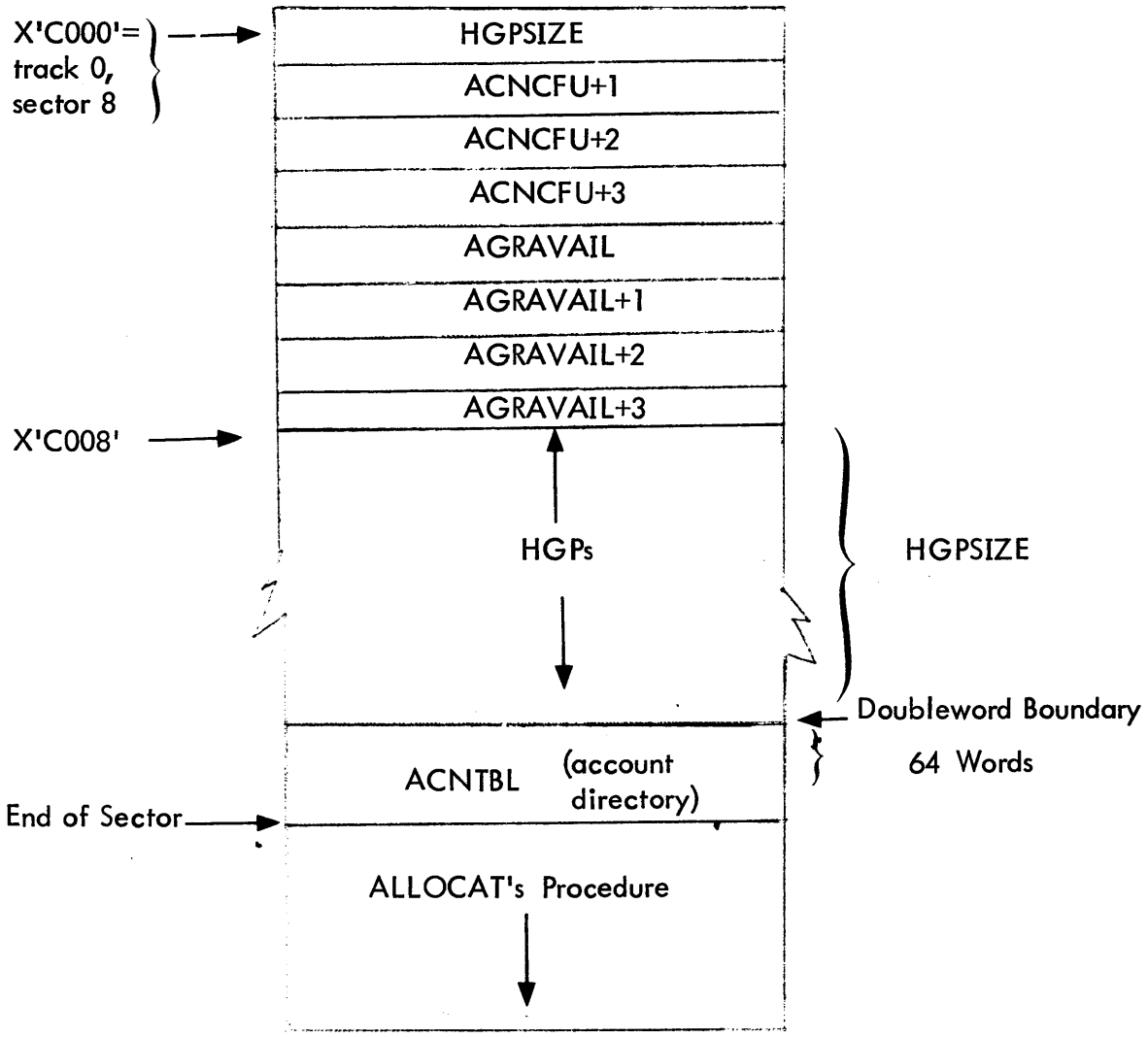
Two other counts GRANMIN and GRANRESET are modified to trigger automatic FPURGE.

8. Emptying Stacks

The stacks are emptied for quiescence by setting the optimum stack size, BUFSOPT, to zero for all stacks and waking ALLOCAT to adjust them.

DATA BASES

ALLOCAT's data is loaded as follows to facilitate locating it, its size and the ACNCFU information:

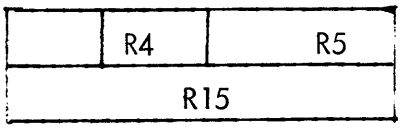


Detailed Description ALLOCAT

ALLOINIT is ALLOCAT's start address. It executes a master mode CAL. It then releases any FPOOls and IPOOLs allocated to it by STEP. It next releases the swapper granules allocated for its outswap, sets its disc address table back to home base, forces its JIT to track 0 sector 4, its AJIT if any to track 0 sector 2 and sets its swap device index to 0 (system resident swapping rad). It next reinitializes the account FCU and sets up the granule counts through HGPCNT.

ALLOCAT is where ALLOCAT begins execute in each time it is called. It saves the ACNCFU critical data and runs the chain of command buffers executing each special request. It sets up the registers as indicated.

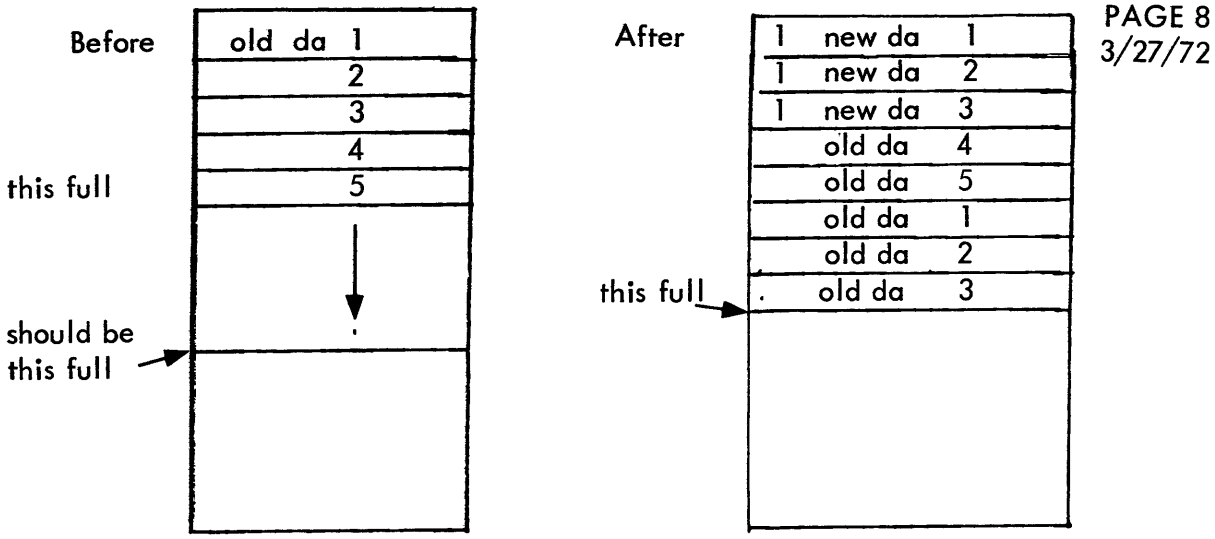
R1 - COMPUF index



When these requests are all honored control goes to ADJSTKS, the stack balancing logic.

ADJSTKS cycles through the four stacks. For each stack it picks up the depth from BUFSOPT and subtracts the current number of words in the stack. If the result is negative the stack is too full and control goes to EMPTYIT. If the result is 0 ALLOCAT moves to the next stack. If it's too full control falls through to FILLIT.

FILLIT adds BUFTHRSH, the threshold value to the number of disc address to be added. This biases the stacks toward fuller than BUFSOPT if it was getting empty. It then gets the required granules one at a time, flags them with a high order bit, exchange the new disc address with the one at the bottom of the stack and that old disc address at the top until enough granules have been added to the stack.

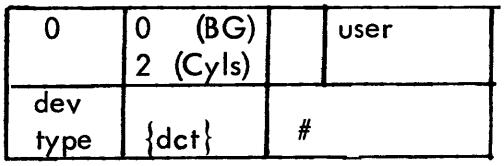


If the granules cannot be acquired and the stack is empty the high order bit of the stack pointer double word BUFSPD is posted to inform GRAN that none are available (LOCKSTK).

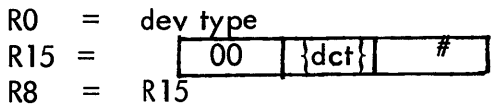
EMPTYIT biases the stack depth by subtracting BUFTARSH from the number to take out. It then pulls the disc addresses from the stack and releases them.

When all the stacks are adjusted ALLOCAT checks to see if FILL should be awakened. It then saves the in-core account directory in ALLYAC and goes to sleep after setting ALLOOUT to request an outswap.

GETNCYLS and GETNBG get in contiguous cylinders or background granules respectively. The COMBUF contained

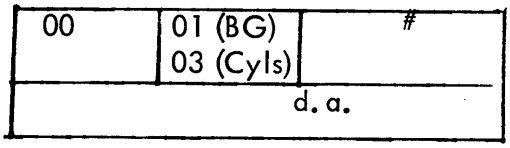


The registers are set up as follows:



GNCYL or GNBG is called. The returned d.a. is put into the COMBUF's second word and the COMBUF is unqueued from ALLOCATs chain but not freed.

RELNCYLS and RELNBG release n contiguous cylinders or background granules respectively. The COMBUF contains



Registers are set up as follows:

R8 = d. a.
R15 = #

RNCYL or RNBG is called. The COMBUF is released.

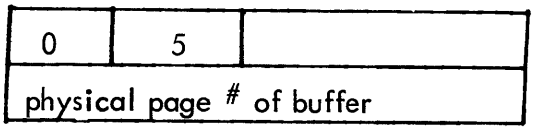
UCB unqueues a COMBUF from ALLOCAT's chain

RCB unqueues it and chains it to CBRHD, the rad buffer chain

FCB unqueues it and chains it to CBFHD, the free chain

R1 points to the next COMBUF upon exit.

RELBUF releases a buffer full of disc addresses. The COMBUF contains



RELBUF gets and releases a virtual page to locate an available virtual page. It then SAP CAL's the physical page into that virtual page. The granules are released and RCB is called to return the COMBUF to the buffer chain.

HGPCNT runs through the chain of HGP's counting the number of each type of granule or public cylinders into AGRAVAIL, its copy of the counts. When there are no more HGP's the counts are Ø transferred to GRAVAIL, the resident copy.

GBPG, GSG, GBG, GNBG, GCYL, GNCYL, RSG, RBG, RNBG, RCYL and RNCYL set up registers and interface with GRANSUB to accomplish the task. See GRANSUB documentation.

GRANSUB

PURPOSE

GRANSUB is loaded both with ALLOCAT for use on public devices and with the resident monitor for private devices and various necessary internal services. Its function is primarily to do the actual work with the HGPs (get 1 or more granules or cylinders or release them). It also maintains the counts of available public storage.

INTERFACE

ENTRY	REGISTERS	FUNCTION
CMNGG	R15 = 1 0 dct # granules R 0 = device type R 2 = 5 for PER 6 for PEA	Common get granule
GPVCYL	None	Get 1 private cylinder
GNPVCYL	R 15 = # granules	Get n private cylinders
GNNAT	None	Get NVAT
CMNRELA	R 8 = d.a. R15 = # granules to release	Common release granule
RPVCYL	R 8 = d.a.	Release private cylinder
RNPVCYL	R 8 = d.a. R15 = # cylinders	Release private cylinders
RNVAT	R 8 = d.a.	Release NVAT

DETAILED DESCRIPTION

CMNGG checks R15 for a DCT index. If specified it will start the search in the corresponding HGP. If not it starts with the first HGP. Each HGP is checked for the correct device type. If correct and not cylinder allocated control goes to CMNGGZ. If cylinder allocated the number of granules requested gets contested to a number of cylinders.

GTNIT moves through the HGP's and allows for the possibility of more than one device type, which causes a re-search of the HGPs.

CMNGG2 sets up 23 with the appropriate stack index for granule counting purposes. It then initiates a quick scan of the HGP word by word for the first non-zero word. If the HGP is empty it gets flagged as such.

L199 through GGB find the appropriate number of contiguous granules. Registers are

R7 = HGP
R5 = relative word in HGP
SR1 = number of bits left to look at in HGP
D2 = Bit being checked
R3 = Type of granule being seeked
0 = PFA rad
1 = PFA rad
2 = PER
3 = public cylinder
4 = private
D4 = # granules required
D4 - R4 = # acquired
R1 = # words x 32 remaining in HGP

The logic searches for the n contiguous granules. If found the da. is returned. If less than n are found followed by one which is not free, RELCALL is called to release the ones acquired and the search continues. If a device boundary is reached the ones acquired are also released.

CMNRELA releases n granules/cylinders. After error checks it converts the d. a. to a relative word and bit and sets up a pointer to the last bit to be released. At RECALL the registers are:

D4 - R4 = to release
R7 = HGP
R5 = word to start releasing at →R6
D2 = bit to release 1st →R3

R4 is incremented to D4. IThe number of granules/cylinders released is added to the appropriate available granule counter.