T H E U N I V E R S I T Y O F M I C H I G A N

Memorandum 30

The CAMA Operating System

L. J. Julyk

CONCOMP: Research in Conversational Use of Computers
F. H. Westervelt, Project Director
ORA Project 07449

supported by:

ADVANCED RESEARCH PROJECTS AGENCY
DEPARTMENT OF DEFENSE
WASHINGTON, D.C.

CONTRACT NO. DA-49-083 OSA-3050
ARPA ORDER NO. 716

administered through

OFFICE OF RESEARCH ADMINISTRATION ANN ARBOR

August 1970

ABSTRACT

The CAMA (Computer-Aided Mathematical Analysis) operating system is a program which controls the operation of an interactive processor. It is designed to operate in the environment of a large central computer which polls a small graphics terminal computer for user-input. The CAMA system is designed to handle a number of different and independent operations, and to perform operations in a priority-based, multiply-queued environment. It is self-expandable by the use of its macro facilities.

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

The CAMA (Computer-aided Mathematical Analysis)[8] operating system is a program which controls the operation of an interactive processor. It is designed to operate in the environment of a large central computer with a small graphic terminal computer connected to it by means of a 2000-baud telephone line. The CAMA system is designed to handle a large number of different and independent operations assigned to it until the user signals that he wants to do something else. When this happens, the normal processing operations are interrupted, the user generates priority operations of his own, and the CAMA system returns to whatever it was doing before the user interrupted. A user-generated interrupt is not processed immediately, but is held in the terminal computer until the central computer has completed its current operation, whereupon it processes the interrupt.

The operating system in the central computer operates in an asynchronous mode. That is, new tasks are initiated not on a fixed time schedule, but on the completion of the currently executing task. The central computer determines whether there is information waiting for it by polling the terminal computer.

When executing many tasks, the central computer polls the terminal computer after the completion of each task. However when the queues are empty, the central computer sends a message to the terminal computer which says in effect,

"Send me word if you want to do something." That is, the central computer no longer polls, but waits for the terminal computer to send information.

The supervisor performs many functions for the user. It allows him to insert commands and have them processed, and to enter various modes of operation, for example, a mode for defining macros, a mode for operating the interpreter, or a mode for defining various procedures or tasks. The supervisor also handles program interrupts, attention interrupts, and error messages from the central computer, all the time allowing the user to remain operating within the CAMA system without returning to the central computer controlling system. It also allows the user to run other programs such as the FORTRAN compiler or a user-generated program while still under CAMA command control. The supervisor dynamically loads and executes all programs from disk storage. The user may then unload all of these dynamically loaded programs by issuing the proper command.

2. GLOSSARY

Task

A task is a procedure which is executed according to its position in a queue. When a task is completed the control of the program returns to the CAMA supervisor to determine which task should be executed next.

During the execution of a task other tasks may be generated and put onto the queue. As soon as a task is put on the queue, operation returns to the mother task.

Normal Task Procedure

1. Operating in T_0
2. Occurrence causes new task to be generated
3. New task T_{N+1} is put on queue
4. Operation returns to T_0
5. Tasks T_0 through T_{N+1} are executed (unless priorities are established).

Wait Task

When a wait task is generated by the current operating task and put on the queue, operation is returned to the queueing system and not to the current task.

Wait Task Procedure

1. Operating in T_0
2. Occurrence causes a wait task to be generated
3. The new task T_{N+1} is put on queue
4. Operation is returned to Task T_1
5. Tasks T_1 through T_{N+1} are executed
6. Operation is then returned to T_0.

The purpose of a wait task is to take care of operations which are necessary for the completion of the current operating task T_0. As an example, suppose a pack has a data overflow while a store operation is taking place. The store operation is task T_0, wait task T_{N+1} is generated to expand the pack. The store operation is not continued until the pack is expanded.

Queue

A queue in the CAMA sense is a series of tasks waiting to be accomplished.

Queues occur in pairs. There may be n pairs of queues. Each pair consists of an active queue and a reserve queue.

Active queue

An active queue is the queue which is processed first when a queue pair is referenced. It is always processed before a reserve queue in the pair, except when the writer explicitly makes reference to the reserve queue.

Reserve Queue

In general, a reserve queue is executed only when the active queue is exhausted. The tasks on the reserve queue are usually of relatively little importance--garbage collection, for example--which can be done when no pressing activities are present.

Look Task

The look task is a special task operating in the central computer which looks to see if there is any information in the terminal computer. In effect it polls the terminal computer. The look task is executed after each task on the task queue. It is suspended when a data ready task is sent to the terminal computer. This occurs under two conditions:

1) if the panic flag has been set;

2) if the current queue pair runs down.

Queue Pair

A queue pair is an active queue and a reserve queue taken as a set. The reserve queue operates when the active queue has run down.

Data Ready Task

A data ready task is a task for the terminal computer. It is implemented either when the panic flag is set or when a queue pair runs down, and allows the system to operate in a read state (which is more economical) while waiting for data to be sent from the terminal computer.

Panic Flag

A panic flag is set by the command HALT. At the time of execution of the look task if the panic flag is set, tasking is not continued; it is suspended and the data ready task is sent to the terminal computer. The panic flag permits the user to:

1) survey the status of his present queues,
2) see where he stands in the operation of the program,
3) execute other commands to take care of such things as data collection,
4) pause when he is confused and needs time to look around a bit.

Front End

The front end is a procedure whose function is to dispatch information arising from the terminal computer and

to initiate procedures to process or store this information. The information from the terminal computer can be in the form of commands, graphic information, numerical information, or other types of data. The front-end procedure first determines the character of the information, decides which processor should handle it, and then relinquishes control to that processor. In general, the items sent to the processors are not tasks and can be processed immediately. Sometimes one of these procedures will generate tasks to be put on the queue to be processed later. During the processing of the front end, the operation of the queue is stopped and the processing takes place as determined by the front-end processor.

Command

A command is one of the group of words preceded by a percent sign or a colon which stimulates action that takes precedence over all other operations in the CAMA system. The key words for a command are preassigned, and the commands are defined as macros in such a way that the user can expand the command processor by devising new commands.[1]

The command language is processed through the interpreter. That is, a command is generated by using the macro processor to generate the code, which is then executed through the interpreter.

Scheduler

The task scheduler is a procedure for operating the current queue pair. In addition, it checks to see if a task

is in use and, if so, reschedules the task on the current queue from whence it came. It also handles loading of processing code, if necessary, as well as errors which might occur.

CAMA Macro

A CAMA macro is a macro written by a user or by a writer. It is written in the form of a prototype, as are many macros for assembly languages. The only variation is that the CAMA macro may be expanded in a number of base languages, among which are the command language, FORTRAN, ALGØL, or other standard languages. Macros can also be written in terms of languages created within CAMA.

Language

In CAMA, a language is a set of instructions whose interpretation is a function of the language named. For example, an ADD operation in the MATRIX language would be interpreted differently than, say, an ADD operation in the POLYNOMIAL language.

Writer

A writer is a skilled programmer who knows the intricacies of the CAMA system. He is distinguished from the user in that he has considerably more experience and therefore is able to manipulate the system internally, which the user cannot do.

User

A user is one who may know a little about the CAMA system but is essentially concerned with the operations as they have been predefined by the writer. The distinction between a user and a writer is not always clear-cut; an individual might perform as a writer in some cases and as a user in others.

Interpreter

The interpreter is a processor which interprets code dynamically.[1]

ATB - ACTIVE TASK BLOCK (see Fig. 1).

ATPL - ACTIVE TASK PUSH LIST, a pack of Type 4 on which the ACTIVE TASKS are stored.

CURRENT ACTIVE TASK - a task which is currently in operation.

ACTIVE TASK - a task which is pending operation due to the generation of a WAIT TASK or a READ which reads the TTY, light pen, or Grafacon.

TCB - TASK CONTROL BLOCK (see Fig 4).

WTL - WAIT TASK LINK (see Fig 5).

PRIORITY CONDITION - the priority upon which the QUEUE PAIR is to operate with (see command SET).

TASK PRIORITY - the priority given to a task when it is generated.

MOTHER TASK - task from which a wait task was generated.

PCB - PAUSE CONTROL BLOCK (see Fig. 2).

RCB - READ CONTROL BLOCK (see Fig. 3).

3. OPERATION OF THE CAMA SUPERVISOR

The operation of the CAMA supervisor is begun by issuing a RUN CAMA command to MTS. After MTS indicates that execution has begun, there is a period of waiting until the CAMA system is bootstrapped into the virtual memory. At the completion of this loading, the CAMA supervisor asks for the name of the data structure file where the user has stored his information or data. If an illegal file name is given, the supervisor continues to ask for it until a legal name is given. The supervisor does not check to see that this file has any structure in it; it requires only the legal name of a file. This can be a permanent file in the MTS system or it can be a temporary file.

Once the file name for the data structure is given, the CAMA supervisor checks to see if a master directory exists in this file. If one does not, the supervisor creates its own master directory. It also creates a number of packs which are necessary for its own operation. For example, it creates packs necessary for the operation of the CAMA queue pairs. It next sets a number of traps for such things as program interrupts and attention interrupts. When all of this work is completed, the CAMA supervisor will

print on the output terminal the words CAMA SYSTEM. At this point the user is in control. At this time the user is being buffered in the terminal computer and in the central computer. This means that messages being sent either way are not dependent upon whether the computer at the other end is ready to receive or not. Therefore the messages can be transmitted or held for transmission until the computer is ready to receive. The 338 or the terminal computer does not send anything to the central computer unless it is asked to. If, however, the central computer sends information to the terminal computer, the RAMP[2] system will set up a RAMP task to handle it.

Although the user is in control of the operation he must do a number of things before he can go very far. The initial loading of the CAMA system included only those subroutines that are needed for minimal operation. To attain certain specific objectives with the CAMA system, other routines will have to be loaded. These must be accessed from subroutine libraries, and the user must specify which libraries he wants before he can proceed. Once specified, these libraries need not be kept in virtual memory, however, but may be discharged by the user, thereby reducing his operating cost.

In some cases the order of bringing these files into the virtual memory is most important. Actually the order depends upon the operation of the MTS loader.[3] The loader will scan the library files in the order specified.

Because it is a one-pass loader, it cannot handle back-references between libraries, that is, references to a library file that has already been scanned. If the order of loading is improper, MTS will not be able to find certain subroutines and will send a message to that effect. In order to resume operation, the user must give a command to the terminal computer to turn off the buffering

(CTRL-A CTRL-A TK 1375 0)

then type the appropriate answer to the loader, followed by a local command to turn buffering back on

(CTRL-A CTRL-A TK 1375 1)

when he has finished communicating to the loader.

Commands are issued in CAMA by typing a % or : as the first character, followed by the command name, followed by the parameters for the command. For example, to get a complete dump of the master list, one would type

%DUMP PTR=ON

(See COMMAND section for a complete description of this command and others presently defined in the CAMA system.)

To enter a specified mode in CAMA, one would type a left parenthesis as the first character, followed by the mode name, then a right parenthesis, and finally, any other information applicable to the mode. At this point the procedure for handling this mode will be dynamically loaded and any subsequent line of input will be directed to this procedure. For example, to enter the interpreter mode, one would type

(INTERPRETER) (MATRIX) (PROB5)

Note that only the first three characters of the mode name are used to identify it. Here the interpreter mode would be entered, with the default variable mode taken to be the MATRIX mode, and the default problem name taken as PROB5.

A mode is ended by entering a new mode or by typing (END), except in the case of the interpreter mode. This mode is ended by typing

(END) INTERPRETER

thereby allowing the interpreter to release its temporary variables. If one first establishes the interpreter mode and then establishes a second mode, the interpreter mode is held pending. Then, if the user ends the second mode with (END), he will return automatically to the interpreter mode. See the routine LPARIN for more details about mode setting. Modes may also be ended by giving the command to unload (i.e., %UNL). Currently three modes are available:

interpreter mode,[1]

macro mode (see routine STOMAC), and

procedure mode (see routine STOPRO).

The remainder of this report describes the current commands available under CAMA. In these commands, one or more blanks. or a comma with optional blanks on either side, serve as delimiters. Underlined values are the default values. Following the commands are descriptors of the routines which make up the CAMA supervisor, as well as comments in some cases about the internal structure of the CAMA supervisor.

4. CAMA COMMANDS

NAME: ALIB

PURPOSE: to add library files to dynamic loader's library table.

PROTOTYPE: ALIB LFN

PARAMETERS: LFN one or more LIBRARY FILE names separated by delimiters.

COMMENTS: the library files are added to the bottom of the dynamic loader's library table in the order given. Currently a maximum of ten library files may be used at any one time.

EXAMPLES: %ALIB A,B, C

files A, B, and C are added.

%ALIB A C,D

only D is added since A and C already exist.

NAME: DESTROY

PURPOSE: to destroy a pack.

PROTOTYPE: DESTROY P=pack name, L=list name

COMMENTS: (1) must confirm action by giving OK.

(2) see DESTP routine in Reference 4 before using this command.

EXAMPLE: %DESTROY PACK1

PACK1 defined in the master directory is destroyed.

%DESTROY PACK3, LIST5

PACK3 defined in the list LIST5 is destroyed.

%DESTROY L=LIST5

LIST5 is destroyed.

NAME: DLIB

PURPOSE: to delete library files from dynamic loader's library table.

PROTOTYPE: DLIB LFN

PARAMETERS: LFN zero or more library file names separated by delimiters.

COMMENTS: if no parameter is given then all the library files stored are released.

EXAMPLES: %DLIB C,B

files C and B are deleted.

%DLIB

all library files are deleted from table and their associated storage is released.

NAME: DPROB

PURPOSE: to define a problem in CAMA

PROTOTYPE: DPROB

COMMENTS: see Reference 5.

EXAMPLE: %DPROB

NAME: DTASK

PURPOSE: to delete a task from a queue.

PROTOTYPE: DTASK T=task name, Q=queue name or blank.

COMMENTS: none

EXAMPLES: %DTASK TASK1

causes the first occurrence of task TASK1 to be deleted from the active queue pair.

%DTASK TASK3, QUE5

causes the first occurrence of task TASK3 to be deleted from the queue pair QUE5.

NAME: DTATPL

PURPOSE: to delete a task from the ATPL.

PROTOTYPE: DTATPL task name or blank.

COMMENTS: none

EXAMPLES: %DTATPL PRT

deletes the task PRT from the ATPL if PRT is in ATPL.

%DTATPL

deletes the most current task in ATPL.

NAME: DUMP

PURPOSE: to dump the contents of a list, association table or a queue, or to obtain the header information on any pack.

PROTOTYPE: DUMP P=pack name, L=list name

T={ List | Association table | Queue={ Active | Reserve | Both } | Pause | ATPL | GRS } , PTR={ ON | OFF }

H={ ON | OFF } , A=association, Ø=object

V=valve

COMMENTS: underlined values are the default values.

EXAMPLES for dumping lists:

%DUMP

gives a dump of the pack names in the master directory.

%DUMP PTR=ØN

gives a dump of the pack names and their pack pointers in the master directory.

%DUMP LIST5 PTR=OFF T=LIST

gives a dump of the pack names in list LIST5.

%DUMP LIST5

equivalent to above command.

EXAMPLES for obtaining the header information for any pack:

The header information returned for a pack consists of:

NAME.....name of pack dumped.

LENGTH.....number of units pack is defined for.

TYPE.....type of pack (some positive number).[4]

USAGE COUNT.....current usage count.

DEFINED IN.....name of list where pack was defined.

LENGTH OF DATA.....current length of data in use (in bytes).

LINE NUMBER.....line number of pack indicating where it is defined on the disk.

%DUMP H=ON

gives the header information for the master directory.

%DUMP H=ON PACK1

gives the header information for the pack PACK1 which is defined in the master directory.

%DUMP PACK6, LIST12 H=ON

gives the header information for the pack PACK6 defined in list LIST12.

%DUMP L=LIST1 H=ON

gives the header information for the list LIST1.

EXAMPLES for dumping QUEUES:

%DUMP T=QUEUE

gives a dump of the tasks on the currently active queue pair. Dumps only the active queue member of the pair.

%DUMP T=Q==RESERVE

same as above except that only the reserve queue member of the queue pair is dumped.

%DUMP QUE5.63A T=Q==BOTH

gives a dump of the tasks on the queue QUE5.63A. Dumps both the active and reserve members of the queue pair. Note: queue dump consists of the names of the tasks on the queue, their associated priorities and the name of the task that they return to.

EXAMPLES for dumping association packs[4]:

%DUMP T=ASSOCIATION

gives a complete dump of the association table which connects all lists in the data structure.

%DUMP T=A V=MASDIR

dumps out only the associations with value equal to MASDIR in the association table which connects all lists in the data structure.

%DUMP ASSOC1 T=A O=M1

gives a dump of associations in the association table ASSOC1 which is defined in the master directory. Dumps

only those associations with object value equal to M1.

%DUMP ASSOC5, LIST3 A=A1, V=V2

gives a dump of the association table ASSOC5 defined in list LIST3. Dumps only those associations with association equal to A1 and value equal to V2. Note that T=A was not necessary here.

EXAMPLES for dumping list of tasks which have been PAUSEd (through the execution of a FORTRAN PAUSE type of statement):

%DUMP T=PAUSE

or equivalently

%DUMP T=P

EXAMPLES for dumping the ATPL:

%DUMP T=ATPL

EXAMPLE for dumping general registers:

%DUMP T=GRS

NAME: EMPTY

PURPOSE: to empty a pack.

PROTOTYPE: EMPTY P=pack name, L=list name.

COMMENTS: (1) lists cannot be emptied.

(2) must confirm action by giving OK.

EXAMPLES: %EMPTY A1, C3

pack A1 defined in the list C3 is emptied.

%EMPTY A5

pack A5 defined in the master directory is emptied.

NAME: FPAUSE

PURPOSE: to flush a PAUSEd task.

PROTOTYPE: FPAUSE name of task

COMMENTS: flushes a task which was PAUSEd by the execution of a FORTRAN PAUSE statement.

EXAMPLES: %FPAUSE TASK3.5

flushes the PAUSEd task TASK3.5.

NAME: GTQ

PURPOSE: go to a specified queue and begin task scheduling with this queue.

PROTOTYPE: GTQ queue pair name or blank.

COMMENTS: if the queue pair named does not exist it is created.

EXAMPLES: %GTQ QUE1.6

task scheduling is resumed with queue QUE1.6.

%GTQ

task scheduling is resumed with previously defined queue pair.

NAME: HALT

PURPOSE: to halt task scheduling.

PROTOTYPE: HALT

COMMENTS: see command RES.

EXAMPLES: %HALT

NAME: LIST

PURPOSE: to list a macro definition or a procedure.

PROTOTYPE: LIST macro name or procedure name, language name of macro, S=starting line number, E=ending line number.

COMMENTS: none

EXAMPLES: %LIST PRO1

lists all lines defined for the procedure PRO1

%LIST MACRO1, LANG3.5 S=-5.21

lists the lines starting at line number -5.21 to end of pack for the macro MACRO1 defined in language LANG3.5.

%LIST PRO2 3.5, 3.5

lists the line 3.5 only for the procedure PRO2.

NAME: MTS

PURPOSE: returns the user to MTS.

PROTOTYPE: MTS

COMMENTS: all MTS commands may then be processed with the exception of the RUN command. The MTS command RESTART brings the user back to the CAMA supervisor.

EXAMPLE: %MTS

NAME: PROT

PURPOSE: to protect or unprotect a pack.

PROTOTYPE: PROT name of pack, L=name of list where pack is defined,
P={ON / OFF}

COMMENTS: a protected pack cannot be destroyed.

EXAMPLES: %PROT PACK1

this protects the pack PACK1 which is defined in the master directory.

%PROT P=OFF PACK3,LIST3

this command unprotects the pack PACK3 defined in list LIST3.

%PROT L=LIST3

this protects the list LIST3.

NAME: REL

PURPOSE: to release the virtual memory used by the data structure.

PROTOTYPE: REL

COMMENTS: this command sets up a task to release the virtual memory used by the data structure and to save any part of the data structure that was changed while in virtual memory.

EXAMPLE: %REL

NAME: RES

PURPOSE: to restart task scheduling.

PROTOTYPE: RES

COMMENTS: see command HALT.

EXAMPLES: %RES

NAME: RPAUSE

PURPOSE: to restart a PAUSEd task.

PROTOTYPE: RPAUSE name of task.

COMMENTS: restarts tasks which were PAUSEd by the execution of a FORTRAN PAUSE statement.

EXAMPLES: %RPAUSE TASK3.5

restarts the PAUSEd task TASK3.5.

NAME: RTQ

PURPOSE: to return to a specified queue pair and begin task scheduling with it only after the current queue pair has run down.

PROTOTYPE: RTQ queue-pair name or blank.

COMMENTS: if the queue pair named does not exist it is created.

EXAMPLE: %RTQ QUE6

task scheduling is resumed with the queue pair QUE6 only after current queue pair has run down.

%RTQ

task scheduling is resumed with the queue pair previously defined when current queue pair runs down.

NAME: RUN

PURPOSE: to dynamically run programs within CAMA.

PROTOTYPE: RUN string

COMMENTS: same as MTS RUN command[3] except for the handling of the PAR= option.

EXAMPLES: %RUN *PERMIT PAR='FILE RO'

note that the parameters for PAR= have been enclosed in primes.

%RUN *FORTRAN SCARDS=FILE(3.5,LAST-10)
SPUNCH=PUN(LAST+1) PAR=SML

note that when only one parameter is given for PAR= it need not be enclosed in primes.

%RUN *ASMG SCARDS=FILE1+FILE2(1,10)
+(30.5,LAST-2)+FILE3 SPUNCH=-PUNCH
0=*SYSMAC 2=MLIB PAR='B,SIZE==100,NX'

note the use of the double equal-sign in the PAR= parameter list.

NAME: SAVE

PURPOSE: to save a pack onto disk storage if it has been changed while in virtual memory.

PROTOTYPE: SAVE pack name, L=list name.

COMMENTS: if a list is saved then everything connected below the list is saved also.

EXAMPLES: %SAVE

saves the complete data structure.

%SAVE PACK1

saves pack PACK1 defined in the master directory.

%SAVE PACK5, LIST12

saves pack PACK5 defined in list LIST12.

%SAVE L=LIST12

saves the list LIST12.

NAME: SET

PURPOSE: to set certain options in CAMA.

PROTOTYPE: SET PC=priority condition

PRINT= {ON / OFF}

COMMENTS: priority condition =0=> process tasks in order which they are stacked (i.e., first on, first off).

priority condition =-1=> process highest priority tasks first.

priority condition =n>0=> process tasks with priority equal to n first.

EXAMPLES: %SET PC=0

%SET PC=5.25 PRINT=ON

the PRINT=ON => all internal data structure and task-handling comments which would normally not be printed are printed to aid the user in possible trouble-shooting.

NAME: UNL

PURPOSE: when the Active Task Push List runs down, UNL unloads all the subroutines which were dynamically loaded.

PROTOTYPE: UNL

COMMENTS: turn off the current mode unless it is the INTerpreter mode.

EXAMPLES: %UNL

Additional commands and subroutines may be found in References 1-6.

5. TASK-HANDLING ROUTINES

NAME: ANSWER

PURPOSE: to answer a read in CAMA

CALLING SEQUENCE: CALL ANSWER (PBUF, HL, SW)

ARGUMENTS: PBUF pointer to buffer.

HL (half-word integer) length of line in buffer.

SW (half-word integer) switch

=0=>normal return

=4=>EOF

RETURN CODE: none

COMMENTS: reads are answered in CAMA by typing a slash "/" followed by required text. FORTRAN formatted reads are protected from errors in typing in data.

The routine ANSWER reestablishes the task state of the task which issued the read and returns the data obtained, thereby restarting the task which generated the read. (See READ routine.)

ANSWER is not to be called by the user.

NAME: CAMSET

PURPOSE: to send a task to the PDP-8 to indicate whether or not the 8 is to store display file names sent to the 8.

CALLING SEQUENCE: CALL CAMSET(SW)

ARGUMENTS: SW integer switch with the value 0 or 1.

RETURN CODE: none

COMMENTS: SW=0=> do not store names

=1=> store names

NAME: DFANS

PURPOSE: to answer a DF read

CALLING SEQUENCE: CALL DFANS(PBUF, HL)

ARGUMENTS: PBUF pointer to buffer

HL (half-word integer) length of line in buffer.

RETURN CODE: none

COMMENTS: DF reads (i.e., reads generated internally within the DF package of routines) are

answered in CAMA by inserting a small 'd' at the beginning of each line to be transmitted. This is done within the PDP-8. Once the response has been obtained the DFANS routine reestablishes the task which issued the DR read and returns control to it. (See READ routine.)

DFANS is not to be called by the user.

NAME: DFCBL

PURPOSE: to obtain the starting address and length of the current Display File (DF) construction buffer.

CALLING SEQUENCE: CALL DFCBL (START, LEN)

ARGUMENTS: START starting address of buffer (integer)
LEN current active length of the buffer in bytes (integer).

RETURN CODE: RC=4 no buffer or buffer is empty.

COMMENTS: none

NAME: DFPTB

PURPOSE: stores a PACK in the current Display File (DF) construction buffer.

CALLING SEQUENCE: CALL DFPTB(PTR)

ARGUMENTS: PTR pack pointer

RETURN CODE: RC=4 no buffer or pack is empty

COMMENTS: none

NAME: DL

PURPOSE: to dynamically load and execute a subroutine so that it can use CAMA variables.

CALLING SEQUENCE: CALL DL (SNAME, NRC, ARG1, ..., ARGM, &1, ...N-1,&N)

SNAME 8-character name of subroutine.

NRC number of return codes possible for subroutine SNAME plus one (integer).

ARG1,..., ARGM for subroutine SNAME.

&1,...,&N-1 returns for subroutine SNAME.

&N return for dynamic loader.

RETURN CODE: see DLR routine.

COMMENTS: the actual arguments to subroutine SNAME are pointed to by the arguments of DL.

NAME: DLAL

PURPOSE: allows the addition of one library file name to the library name table in DLR.

CALLING SEQUENCE: CALL DLAL('FNAME ',&1)

ARGUMENTS: 'FNAME ' - character string which is a library file name. String length 12 bytes max.

RETURN CODES: &1 - control is returned to this statement for one of the following reasons:

(1) illegal character in file name (***DL** ILLEGAL FILENAME),

(2) the file does not exist (***DL** FILE DOES NOT EXIST),

(3) the library file name table is full (***DL** LIBTAB FULL),

(4) the file is not a library file (***DL** BAD LIB FILE).

COMMENTS:

(1) The trailing blank can be omitted if the file name is 12 characters long.

(2) The library name table can hold 10 file names.

(3) The library file structure is expected to be like that produced by GENLIB.

(4) The legal characters in the file name are the same as are allowed for MTS files.

NAME: DLEAL

PURPOSE: deletes all library file names from library file name table in DLR.

CALLING SEQUENCE: CALL DLEAL

ARGUMENTS: NONE

RETURN CODE: NONE

COMMENTS: NONE

NAME: DLEOL

PURPOSE: deletes one library file from the name table in DLR.

CALLING SEQUENCE: CALL DLEOL('LFNAME ',&1)

ARGUMENT: 'LFNAME ' - character string which is the name of a library file. String length 12 bytes or less.

RETURN CODE: &1 - control is returned to this statement for one of the following reasons:

(1) the library name table was empty (***DL** LIB TAB EMPTY),

(2) the file name was not found in the table (***DL** FILE NOT IN LIBTAB),

(3) the file name is illegal (***DL** ILLEGAL FILE NAME).

COMMENTS: (1) The trailing blank is not necessary if the file name is 12 characters long.

(2) The characters allowed in the file name are the same as for MTS file name.

(3) The library name table is automatically garbage-collected.

NAME: DLR

PURPOSE: allows a subroutine to be dynamically loaded from a library file and executed.

CALLING SEQUENCE: CALL DLR ('STRING ',M,A1,A2,...,An,&1,&2...,&M)

ARGUMENTS: 'STRING ' - a character string which is the name of the subroutine that is to be dynamically loaded. FORTRAN subroutine name rules apply:

(1) first character alphabetic A-Z,

(2) succeeding characters alphabetic or integer digits 0-9,

exception (3) 8-byte character length allowed.

M - an integer value such that M-1 return codes are for the subroutine which is to be called; the Mth return code is for DLR.

A1,A2,...,AN- a list of variable names which would normally appear as arguments for the subroutine.

RETURN CODE: &M - control is returned to this statement for one of the following reasons:

(1) the object module for the subroutine was not found in the library (***DL** OBJ MOD NOT IN LIBR),

(2) no library files have been specified by the user (***DL** NO LIBRARY),

(3) the subroutine name does not start with an alphabetic character (***DL** ILLEGAL CHAR IN OBJ MOD NAME).

COMMENTS: (1) The time delay for the first call on a subroutine is approximately the same as for a normal load. Subsequent calls on that same subroutine have a very small time delay.

(2) If an illegal character occurs in a subroutine name after the first character, that character and all succeeding characters are replaced by blanks. No comment is printed.

(3) The trailing blank in 'STRING ' is necessary only if the name is less than 8 characters.

NAME: DLUNL

PURPOSE: unloads all object modules that have been loaded by DLR.

CALLING SEQUENCE: CALL DLUNL

ARGUMENTS: NONE

RETURN CODES: NONE

COMMENTS: (1) Selective unloading is not allowed at this time.

NAME: DRUN

PURPOSE: allows the user to suspend the execution of one main program and then execute another

main program with all the logical I/O devices reassigned. DRUN effectively allows the MTS command $RUN to be re-entrant.

CALLING SEQUENCE: CALL DRUN('STRING%',&1,&2,&3,&4)

ARGUMENTS: 'STRING%' - a string of characters identical in format to that following a $RUN command. 'MAP', 'NOMAP', MAPFDNAME, and execution limits are not allowed. % (percent) is the terminator for the string. Maximum string length is 255 bytes.

RETURN CODES: %1 - control is transferred to this statement if an error was detected in parsing 'STRING%'

(***DR** PARSING ERROR).

%2 - control is transferred to this statement if a call to error was trapped

(***DR** TRAPPED CALL TO ERROR).

%3 - control is transferred to this statement if a call to MTS, SYSTEM, or QUIT was trapped

(***DR** TRAPPED CALL TO MTS/SYSTEM /QUIT).

%4 - control is transferred to this statement

if the return code from the executed program is greater than zero (***DR** RC>0 FROM EXECT PROG).

COMMENTS:

(1) Prototype: (in FORTRAN)

```
CALL DRUN('*USERS%')
CALL DRUN('*FORTRAN SCARDS=-Z PAR=SML%')
```

(2) The default reassigned values for SCARDS, SPRING, and SERCOM are *SOURCE*, *SINK*, and *SINK* respectively. All other logical I/0 devices are unassigned (just as in MTS for the TTY).

(3) The symbol % (percent) is not allowed in FDNAMES that are given in 'STRING%'.

(4) The size of DRUN is 3856 bytes (approx. one page).

NAME: DTATPL

PURPOSE: to delete a task from the ACTIVE TASK PUSH LIST (ATPL).

CALLING SEQUENCE: CALL DTATPL(NAME)

ARGUMENTS: NAME 8-character name of task

RETURN CODE: RC=4 task not found

COMMENTS:

(1) If NAME is blank then the current active task will be deleted.

(2) If the task is found then this routine does not return to caller.

(3) When a task is deleted, any pending PAUSE, DF read, or trapped READ is flushed.

(4) All MOTHER tasks connected to the deleted task are also deleted.

NAME: DTASK

PURPOSE: schedules tasks within a queue pair.

CALLING SEQUENCE: CALL DTASK

ARGUMENTS: none

RETURN CODE: none

COMMENTS: The TASK SCHEDULER (DTASK) is a procedure which schedules tasks on a priority basis.

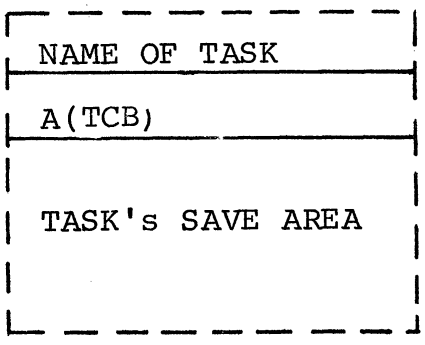
The current QUEUE PAIR is obtained and the ACTIVE QUEUE is referenced. If the ACTIVE QUEUE is empty then the RESERVE QUEUE is referenced. When both queues are found to be empty, DTASK returns with the NOTASK flag set. If one of the queues is not empty then the TASK to be processed is selected according to the priority condition which has been set. A priority condition of zero means that the next task on the queue is to be processed regardless of its priority. For a condition which is some positive number, then only tasks with this priority will be processed regardless of their position on the QUEUE. If there is no task with this priority, then the priority condition is reset to zero and the ACTIVE QUEUE is referenced again. If the priority condition is a minus

one, then those tasks with the highest priorities are processed first regardless of their position on the QUEUE. The priority condition may be set by the user by issuing the command

%SET PC=number.

Once a task has been selected, the ACTIVE TASK PUSH LIST (ATPL) is referenced to see if this new task is already in use (i.e., pending a WAIT TASK or the answer to a READ which requires TTY, light pen, or Grafacon response). If it is in use then the task is requeued; otherwise an ACTIVE TASK BLOCK (ATB) is created for this task, and processing continues as shown in flow chart for TASK SCHEDULER.

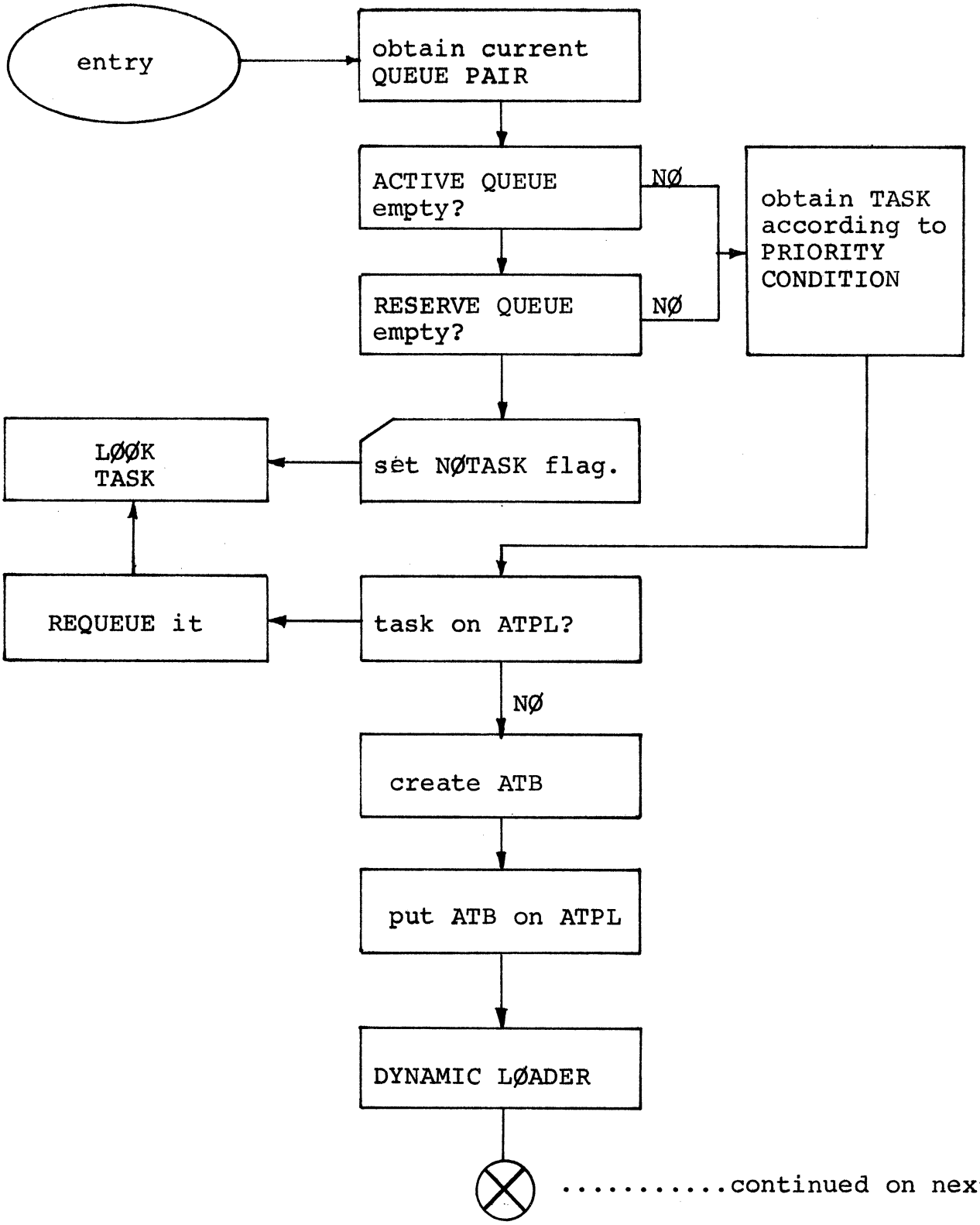
The ACTIVE TASK BLOCK has the following internal format:



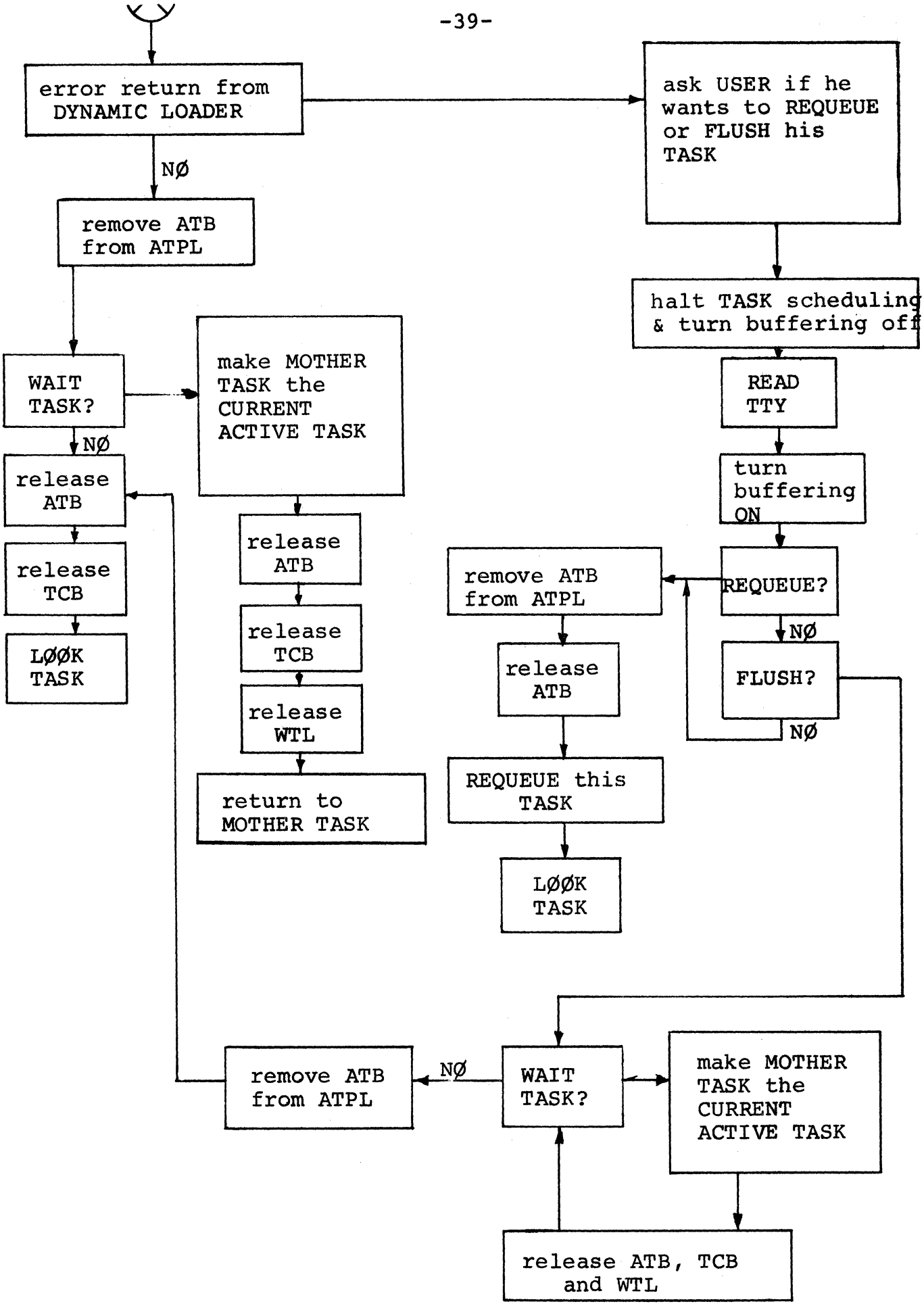
84 bytes

Figure 1. ACTIVE TASK BLOCK

Flow Chart for TASK SCHEDULER (DTASK)



.....continued on next page



Flow Chart for TASK SCHEDULER (DTASK)

NOTE: When DTASK has an error return from DLR it asks the user if he wants to requeue or flush the task which caused the error. Task scheduling is halted at this point, and after the user answers DTASK he must give the command %RES in order to restart task scheduling.

NAME: EBOCT

PURPOSE: convert EBCIDC representation of numbers to OCTAL.

CALLING SEQUENCE: CALL EBOCT(NUM,HOCT)

ARGUMENTS: NUM EBCDIC representation of number (full-word integer)

HOCT resulting octal number (half-word integer)

RETURN CODE: none

COMMENTS: none

NAME: ERRCODE

PURPOSE: program to snatch error code from IBCOM#.

CALLING SEQUENCE: call ERRCØDE

ARGUMENTS: none

RETURN CODE: none

COMMENTS: ERRCODE is not to be called by user.

NAME: FEND

PURPOSE: to dispatch data from the terminal to the

proper interpreter on the basis of its first character.

CALLING SEQUENCE: CALL FEND(PT2,HL)

ARGUMENTS: PTR pointer to buffer

HL length of data in buffer (half-word integer)

RETURN CODE: none

COMMENTS: (1) does not return to caller.

(2) its action is depicted in the flow chart for FRØNT END.

FEND is not be called by the user.

NAME: FIOCSERR

PURPOSE: I/0 recovery routine for FIOCS#

CALLING SEQUENCE: CALL FIOCSERR

ARGUMENTS: none

RETURN CODE: none

COMMENTS: FIOCSERR is not to be called by the user.

NAME: FPAUSE

PURPOSE: to flush a task which has been paused by the execution of a FORTRAN PAUSE statement.

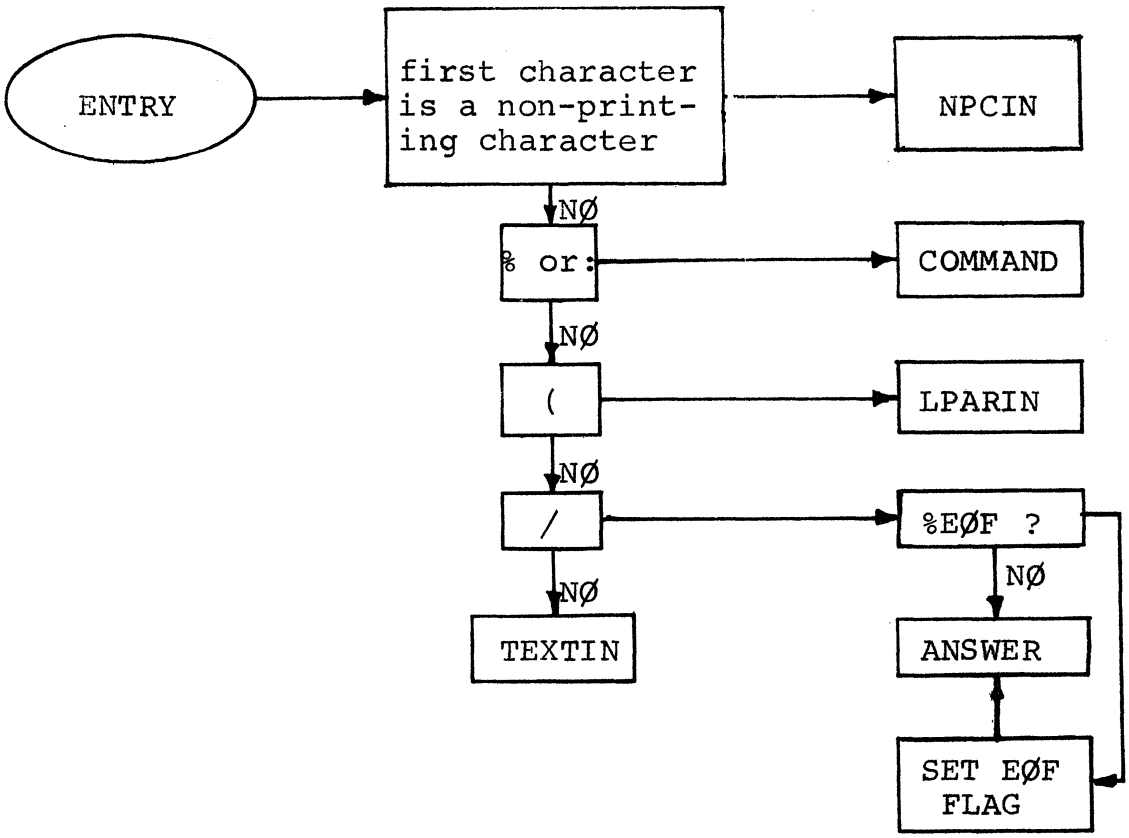
CALLING SEQUENCE: CALL FPAUSE(NAME)

ARGUMENTS: NAME name of paused task.

RETURN CODE: none

COMMENTS: does not return to caller.

Flow Chart for FRONT END (FEND)



NAME: FRDNL#, FWRNL#, DIOCS#

PURPOSE: NAMELIST and DEFINE FILE FORTRAN I/0 trap.

CALLING SEQUENCE: see FORTRAN's FRDNL#, FWRNL#, and DIOCS#

ARGUMENTS: see FORTRAN's FRDNL#, FWRNL# and DIOCS#

RETURN CODE: see FORTRAN's FRDNL#, FWRNL# and DIOCS#

COMMENTS: none

NAME: GTQUE

PURPOSE: to establish a new QUEUE PAIR or go to a previously established QUEUE PAIR.

CALLING SEQUENCE: CALL GTQUE(NAME)

ARGUMENTS: NAME 8-character name of QUEUE PAIR

RETURN CODE: RC=4 did not change QUEUE PAIR.

COMMENTS: If NAME is blank then go to previously established QUEUE PAIR.

NAME: HDINFO

PURPOSE: prints out the header information of a pack.

CALLING SEQUENCE: CALL HDINFO(PTR)

ARGUMENTS: PTR pointer to a pack

RETURN CODE: none

COMMENTS: see DUMP command.

NAME: IBCOM#

PURPOSE: intercept FORTRAN's IBCOM#

CALLING SEQUENCE: see FORTRAN IBCOM#

ARGUMENTS: see FORTRAN IBCOM#

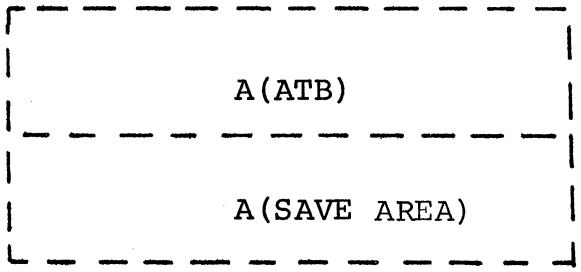
RETURN CODE: see FORTRAN IBCOM#

COMMENTS: IBCOM# is the main interception routine for FORTRAN I/0 (see TASKIBC).

If a pause statement in a FORTRAN program is executed, the paused task is held until the user flushes or restarts the task on command.

IBCOM# creates a PAUSE CONTROL BLOCK (PCB) with the following format:

Figure 2. PAUSE CONTROL BLOCK (PCB)



and places the PCB on a stack (see routines RPAUSE and FPAUSE).

NAME: IBCOMERR

PURPOSE: I/0 recovery routine for IBCOM#

CALLING SEQUENCE: called by FORTRAN's IBCOM#

COMMENTS: IBCOMERR is not to be called by user.

NAME: LPARIN

PURPOSE: to clear or set modes in CAMA

CALLING SEQUENCE: CALL LPARIN(PBUF,HL)

ARGUMENTS: PBUF pointer to buffer

HL length of data in buffer (half-word integer)

RETURN CODE: none

COMMENTS: the LEFT PARENTHESIS INTERPRETER (LPARIN) is a procedure which establishes mode operations for the CAMA system. Predefined modes are stored in the LPARPACK which is Type 3 (association table). The format of LPARPACK is

A	O	V
3-character mode name	8-character long name of mode	8-character name of procedure to handle this mode

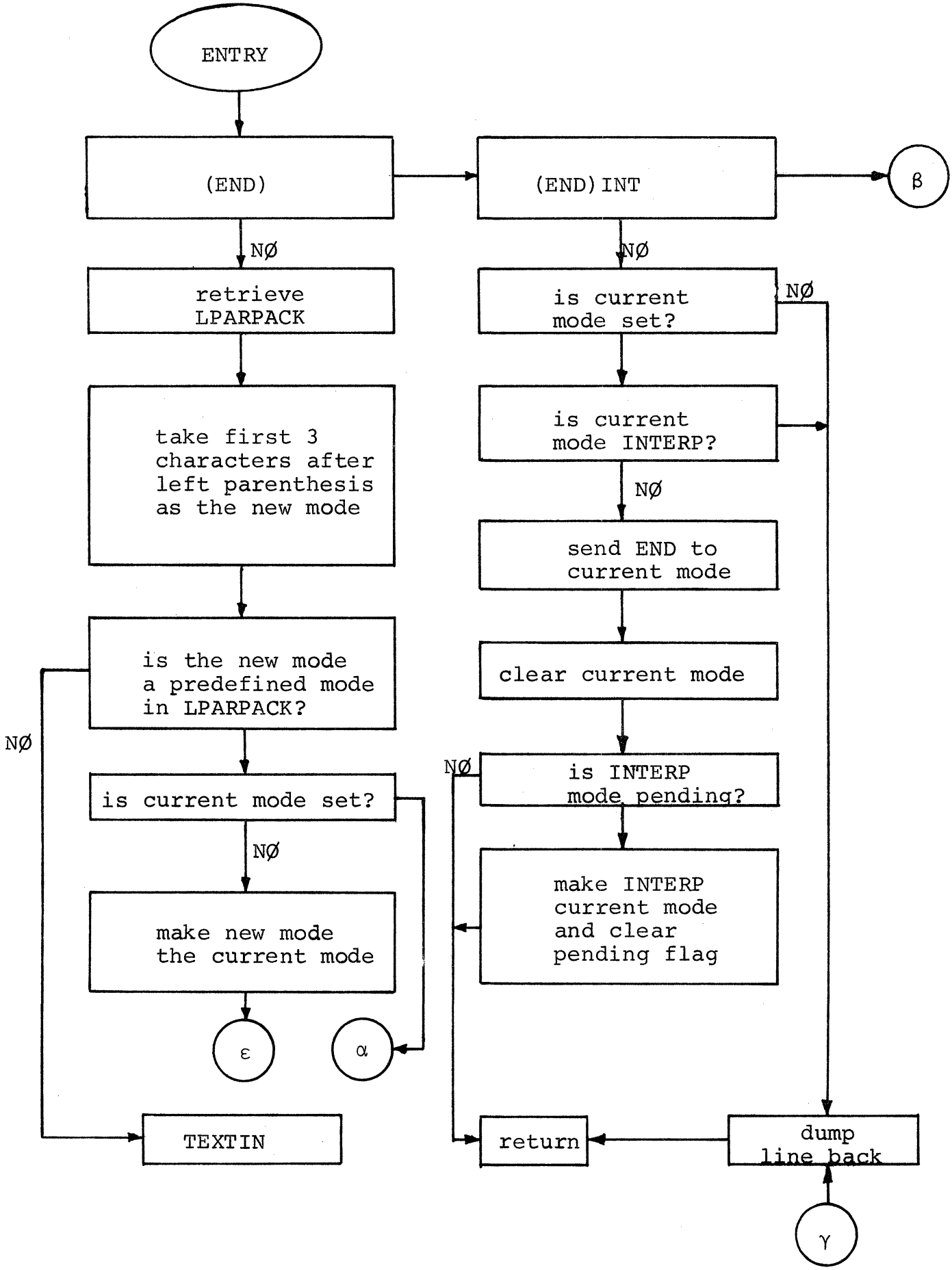
A mode is established by typing a left parenthesis in column one followed immediately by 3 or more characters, a right parenthesis followed by optional data.

If the mode is a legal mode as stored in LPARPACK, then an (END) is sent to the current mode, unless the current mode is the INTERPRETER MODE in which a flag is set to indicate that the INTERP mode is pending. A user may end any other mode by typing (END), and if the INTERP

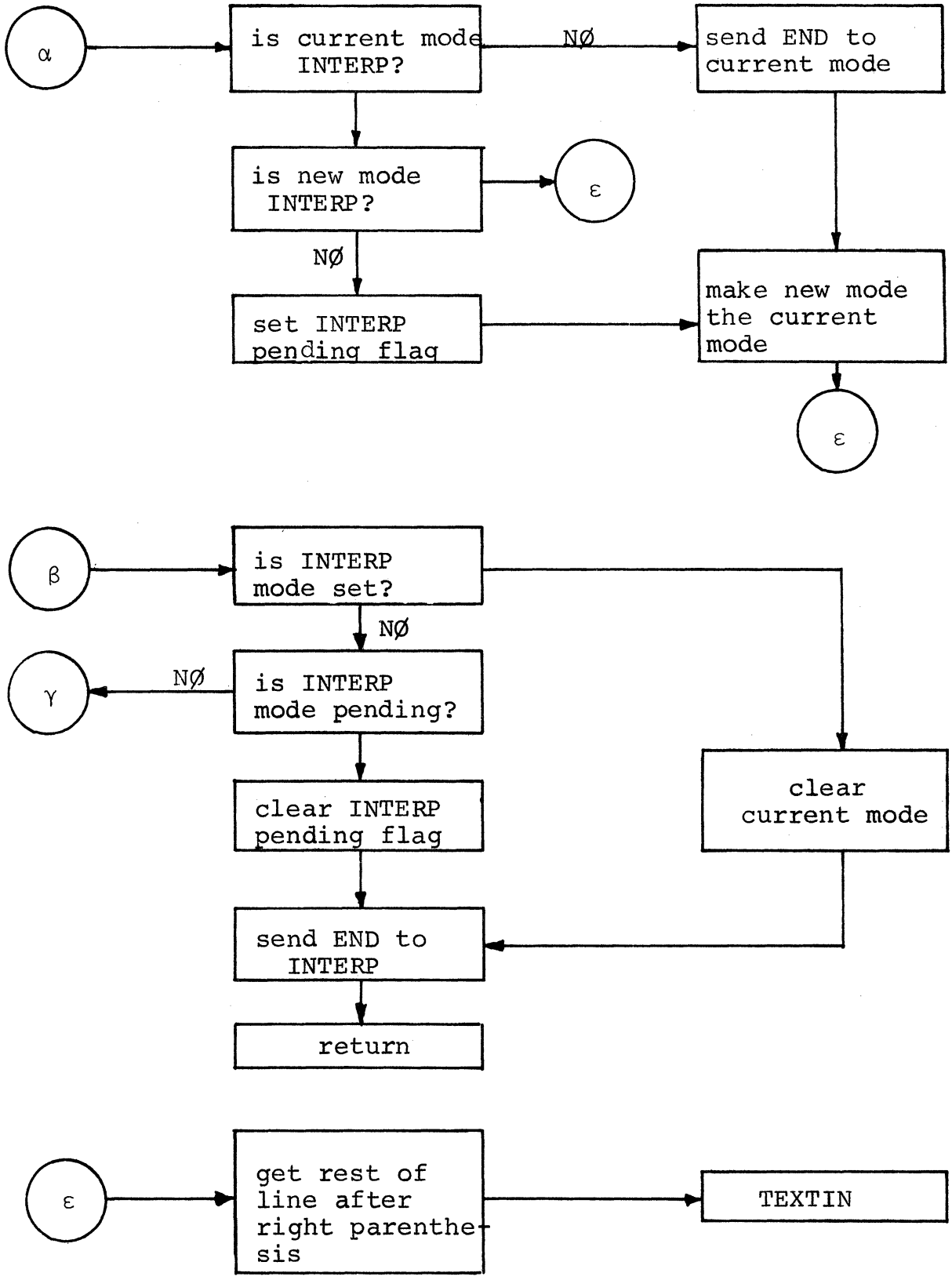
mode is pending it will be re-established as the current mode. This prevents the user from losing his system variables when going from INTERPRETER mode into a new mode (see INTERP description).[1] If he really wants to release these variables then he must type (END)INTERP. Once a mode has been established the rest of the text or any subsequent text is directed to the proper procedure by the TEXTIN routine. Whenever the command UNL is issued by the user, an (END) is sent to the current mode unless it is the INTERP mode. The purpose of sending the (END) to the current mode is so that the current mode can release any temporary storage that it may have acquired or do anything else which might be necessary to close itself out. All mode subroutines must have the same argument list as LPARIN and must accept (END).

The following flow chart describes LPARIN's operation.

Flow Chart for LEFT PARENTHESIS INTERPRETER (LPARIN)



...continued on next page


α
is current mode INTERP?
NØ
send END to current mode
is new mode INTERP?
ε
NØ
set INTERP pending flag
make new mode the current mode
ε
β
is INTERP mode set?
NØ
γ
NØ
is INTERP mode pending?
clear INTERP pending flag
clear current mode
send END to INTERP
return
ε
get rest of line after right parenthesis
TEXTIN

NAME: MTS

PURPOSE: return user to MTS

CALLING SEQUENCE: CALL MTS

ARGUMENTS: none

RETURN CODE: none

COMMENTS: this routine allows user to go to MTS with buffering in PDP-8 turned off. By giving a RES command in MTS, CAMA will be restarted with buffering turned back on.

NAME: NPCIN

PURPOSE: to interpret lines coming from the terminal with a non-printing character as their first character.

CALLING SEQUENCE: CALL NPCIN(PBUF,HL)

ARGUMENTS: PBUF pointer to buffer

HL length of data in buffer (half-word integer)

RETURN CODE: none

COMMENTS: the NON-PRINTING CHARACTER INTERPRETER (NPCIN) is a procedure operating in the central computer to direct the flow of data from the internal responses of the PDP-8 produced by programs and actions taken within the PDP-8. For example, all light pen or Grafacon hits are

directed to the proper places by preceding the first character of the response with a small 'd'.

The flow chart, NON-PRINTING CHARACTER INTERPRETER, describes the action taken by NPCI

NAME: PROG

PURPOSE: to send a PDP-8 program to the PDP-8 from the central computer.

CALLING SEQUENCE: CALL PROG(PTR)

ARGUMENTS: PTR pointer to data pack where PDP-8 program is stored.

RETURN CODE: none

COMMENTS: none

NAME: PROG2

PURPOSE: used to store PDP-8 programs in a data pack.

CALLING SEQUENCE: CALL PROG2(PTR,FDUB)

ARGUMENTS: PTR pointer to data pack where PDP-8 program is to be stored.

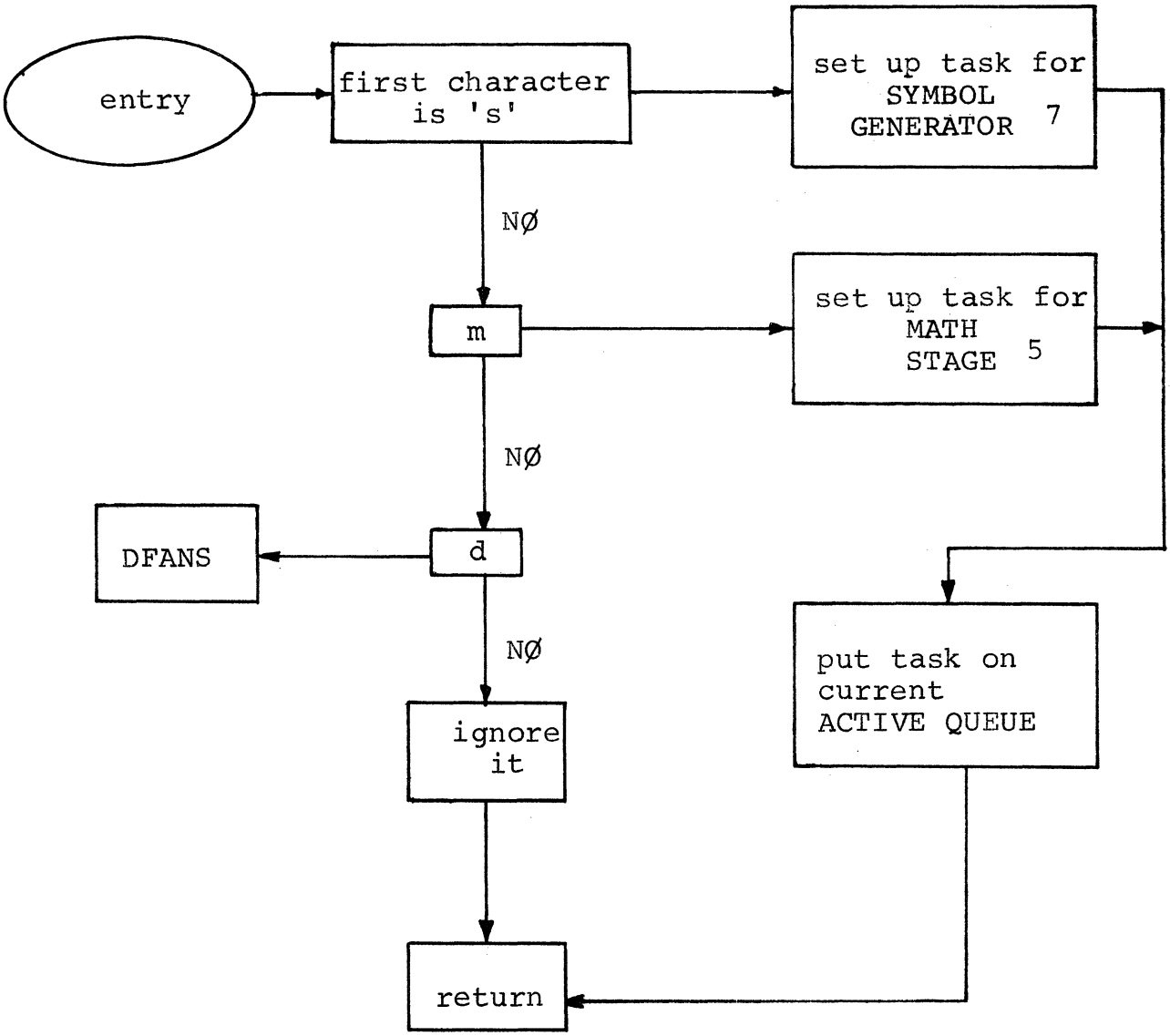
FDUB FDUB of file to be read.

COMMENTS: none

NAME: READ

PURPOSE: to intercept calls to READ, READ#, SCARDS, and SCARDS# routines when buffering is on.

Flow Chart for NON-PRINTING CHARACTER INTERPRETER (NPCIN)



CALLING SEQUENCE: standard calling sequence used in FORTRAN.

ARGUMENTS: standard arguments used in FORTRAN calling sequence.

RETURN CODE: see FORTRAN

COMMENTS: the following conventions are assumed:

(1) calls to SCARDS and SCARD# are trapped.

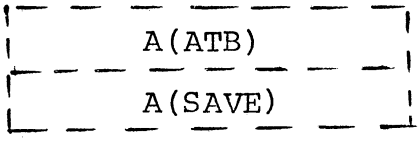
(2) calls to READ or READ# with a FDUB or LDN which is nonexistent, unassigned, or connected to the terminal are trapped.

(3) all other conditions allow the READ to fall through.

(4) if a call to READ has a FDUB which is connected to the terminal then this read is trapped as a DF read.

DF reads are answered via the DFANS routine. All other trapped reads are answered via the ANSWER routine which requires a "/" (slash) as the first character. When a read is trapped a READ CONTROL BLOCK (RCB) is generated with the following format:

Figure 3. READ CONTROL BLOCK (RCB)



where A(ATB) is the address of the ATB for the task which generated the read, or zero if not generated by a task. A(SAVE) is the address of the save area for the read.
When a read is trapped its RCB is put on a stack and control is returned to the CAMA supervisor (see ANSWER, DFANS, and IBCOM#).

NAME: REL

PURPOSE: to release the data structure from virtual memory and save any packs which have been changed.

CALLING SEQUENCE: CALL REL

ARGUMENTS: none

RETURN CODE: none

NAME: REQUE

PURPOSE: to requeue a task

CALLING SEQUENCE: CALL REQUE

ARGUMENTS: none

RETURN CODE: none

COMMENTS: this program requeues the task in which it was called in. That is, if REQUE is called within a task, then that complete task will be requeued.

NAME: RESTOR

PURPOSE: The complement to SAVE, i.e., restores the contents of the general registers and the values of a list of local variables.

CALLING SEQUENCE: CALL RESTOR (A1,A2,...,AM)

ARGUMENTS: A1,A2,...,AM a list of variables whose values are to be restored. Each variable must be a full word and aligned on a full-word boundary. M should be less than or equal to N in SAVE.

RETURN CODE: none

COMMENTS: The values for the variables are restored in the order that they were saved. No mode or adcon checking is made. See SAVE routine.

NAME: RPANIC

PURPOSE: to decrement the panic flag or HALT flag in CAMA

CALLING SEQUENCE: CALL RPANIC

ARGUMENTS: none

RETURN CODE: none

COMMENTS: In order to restart tasking operations the panic flag must be zero

NAME: RPAUSE

PURPOSE: to restart a task which has paused by the execution of a FORTRAN PAUSE statement within the task.

CALLING SEQUENCE: CALL RPAUSE(NAME)

ARGUMENTS: NAME name of paused task

RETURN CODE: none

COMMENTS: does not return to caller.

NAME: RTQUE

PURPOSE: to establish a new QUEUE PAIR or go to a previously established QUEUE PAIR only after the current QUEUE PAIR is empty.

CALLING SEQUENCE: CALL RTQUE(NAME)

ARGUMENTS: NAME 8-character name of QUEUE PAIR

RETURN CODE: none

COMMENTS: if NAME is blank, then go to previously established QUEUE PAIR. The routine RTQUE generates the task TASKRTQ which actually does the work.

NAME: RUN

PURPOSE: to call DRUN with buffering off.

CALLING SEQUENCE: CALL RUN (same as DRUN)

ARGUMENTS: same as DRUN

RETURN CODE: see DRUN

COMMENTS: see DRUN

NAME: SAVE

PURPOSE: allows the user to make FORTRAN subroutines recursive by saving the contents of the general registers and the values of a list of local variables.

CALLING SEQUENCE: CALL SAVE (A1,A2,...,AN)

ARGUMENTS: A1,A2,...AN a list of variables whose values are to be saved. Each variable must be full-word (four bytes) and aligned on a full-word boundary. The mode may be real, integer, or logical.

RETURN CODES: none

COMMENTS: none

NAME: SETPRI

PURPOSE: to set the priority condition for DTASK

CALLING SEQUENCE: CALL SETPRI(PRI)

ARGUMENTS: PRI priority condition times 1000 (integer)

RETURN CODE: none

COMMENTS: PRI = 0 => process tasks in order in which they are stacked.

PRI = -1 => process highest priority tasks first.

PRI = n>0 => process only tasks with priority of n.

NAME: STOMAC

PURPOSE: to store macros.

CALLING SEQUENCE: CALL STOMAC(BPTR,HL)

ARGUMENTS: BPTR pointer to buffer

HL (half-word integer) length of line in buffer.

RETURN CODE: none

COMMENTS: typing (MACRO) in CAMA will establish the store macro mode. The syntax of this mode is

ƀ(ƀ macro mode ƀ)ƀ(ƀ language name ƀ)

or a line number followed by a line of text. Examples:

(MAC) (M1) (L1)

(MAC) established the store macro mode with the macro name taken as 'M1' and its language name as 'L1'. If a macro name or language name is longer than 8 characters only the first 8 continuous nonblank characters between the parentheses are used. If the macro was previously defined a comment is printed to alert the user.

(M2) the macro name is taken as 'M2' defined in the language 'L1'.

()(L3) the macro name 'M2' is defined in the language 'L3'.

9.361 FN P# '&P P#

the line FN P# '&P' P# is entered into macro 'M2' defined in language 'L3' with line number 9.36J

The line number range is -99999.9999≤n≤+99999.999

3.5,211 DO 215 I=J,N

the line

211 DO 215 I=J,N

is entered for line 3.5.

2

line 2 is destroyed.

(END) will terminate the store macro mode.

NAME: STOPRO

PURPOSE: to store a procedure

CALLING SEQUENCE: CALL STOPRO(BPTR,HL)

ARGUMENTS: BPTR pointer to buffer

HL (half-word integer) length of line in buffer.

RETURN CODE: none

COMMENTS: typing (PROCEDURE) in CAMA will establish the store procedure mode.

The syntax is

ø(ø procedure name ø)

or a line number followed by a line of text.

EXAMPLES: (PRO) (P1)

(PRO) establishes the store procedure mode with the procedure name taken as P1.

(P2)

P2 is taken as the procedure. If a procedure already exists with the given name, then a comment is printed to alert the user.

9.5

deletes line 9.5

9.3(FORTRAN) N=3.5*2

enters line with line number 9.3.

(END) will terminate the store procedure mode.

MAME: SPANIC

PURPOSE: to increment the panic flag or HALT flag in CAMA

CALLING SEQUENCE: CALL SPANIC

ARGUMENTS: none

RETURN CODE: none

COMMENTS: a call to SPANIC stops tasking operations.

NAME: SPEW

PURPOSE: unloads subprograms which were dynamically loaded in CAMA.

CALLING SEQUENCE: CALL SPEW

ARGUMENTS: none

RETURN CODE: none

COMMENTS: the task TKSPEW is generated to handle the unloading.

NAME: TASK

PURPOSE: to put a task on the QUEUE and return.

CALLING SEQUENCE: CALL TASK(0,QUE,PTR)

CALL TASK(1,QUE,PRIORITY, PROTECTION, TASKNAME, ARG1,...,ARGN)

ARGUMENTS: QUE=0 put task on ACTIVE QUEUE

=1 put task on RESERVE QUEUE

PTR pointer to TCB

PRIORITY the priority that this task is to have times 1000 (integer)

PROTECTION=0=> unprotected

=1=> protected from attention interrupts.

TASKNAME 8-character name of task

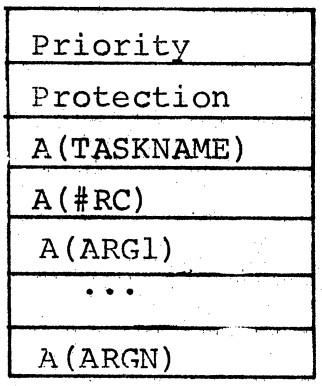
ARG1 first argument of the task

ARGN nth argument of the task

RETURN CODE: none

COMMENTS: (1) when the first argument of the routine TASK is zero then it is assumed that the third argument is a pointer to a user-set-up TCB. If, however, the first argument is one, then the TASK routine will set up the TCB. The TASK CONTROL BLOCK (TCB) has the following format:

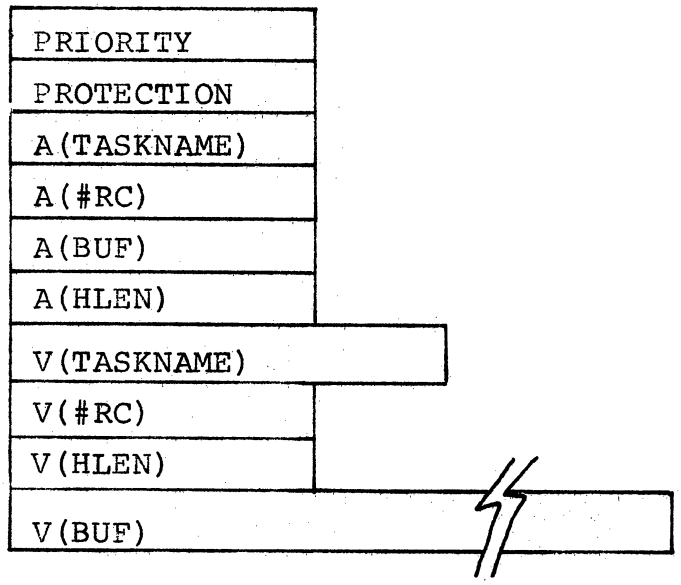
Figure 4. TASK CONTROL BLOCK (TCB)



#RC=actual number of RC for the routine

TASKNAME + 1

(2) The address of pointer to arguments may or may not point to within the TCB. For example, one might have



where A(...)= address of

V(...) = value of

This can be done only if the user (or writer) sets up the TCB. Note that the TCB is automatically destroyed upon returning from the completion of the task. Therefore if the user sets up the TCB, he must get space dynamically.

NAME: TASKIBC

PURPOSE: a task to handle IBCOM#s when they pile up.

CALLING SEQUENCE: generated within IBCOM#

ARGUMENTS: see IBCØM#

RETURN CODE: none

COMMENTS: if a FORTRAN read is pending, then this implies that FORTRAN's I/O is in use. Since it is not reentrant, any subsequent call to IBCØM# must be requeued until FORTRAN's I/O is available.

NAME: TASKRTQ

PURPOSE: task to handle RTQUE

CALLING SEQUENCE: generated within RTQUE

ARGUMENTS: see RTQUE

RETURN CODE: see RTQUE

COMMENTS: requeues itself until current QUEUE PAIR is empty.

NAME: TEXTIN

PURPOSE: to dispatch text to the current mode set.

CALLING SEQUENCE: CALL TEXTIN(PBUF,HL)

ARGUMENTS: PBUF pointer to buffer

HL Length of data in buffer (half-word integer)

RETURN CODE: none

COMMENTS: if no mode has been set then the line is dumped back with a question mark followed by the text.

NAME: TKSPEW

PURPOSE: task to handle SPEW

CALLING SEQUENCE: generated within SPEW

ARGUMENTS: see SPEW

RETURN CODE:

COMMENTS: (1) TKSPEW requeues itself until the ATPL is empty.

(2) Before unloading takes place the current mode is cancelled unless it is the INTERPRETER mode.

NAME: WTASK

PURPOSE: to put a task on the queue and return when the task has been completed.

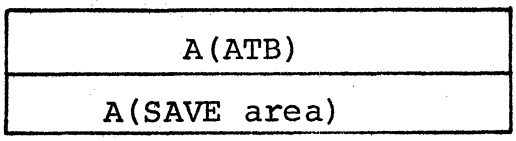
CALLING SEQUENCE: CALL WTASK(O,QUE,PTR)

CALL WTASK(1,QUE,PRIORITY,PROTECTION,
TASKNAME,ARG1,...,ARGN)

ARGUMENTS: same as TASK routine

RETURN CODE: return codes of TASKNAME

COMMENTS: see TASK routine. Note that here, if a wait task (WTASK) is generated within a task or wait task, a WAIT TASK LINK (WTL) is generated and has the following format: Figure 5 . WTL



where A(ATB) is the address of the ATB of the task from which the wait task was generated, and A(SAVE area) is the address of the save area supplied by the MOTHER TASK.

NAME: ZPANIC

PURPOSE: to zero the panic flag in CAMA

CALLING SEQUENCE: CALL ZPANIC

ARGUMENTS: none

RETURN CODE: none

COMMENTS: forces a restart of tasking operations in CAMA.

REFERENCES

1. Dingwall, T., Julyk, L., and Wolf, L., The CAMA Interpreter, Memorandum 36, Concomp Project, University of Michigan, Ann Arbor, August 1970.

2. Mills, D., RAMP: A PDP-8 Multiprogramming System for Real-Time Device Control, Memorandum 5, ibid., May 1967, 24 pp.

3. MTS (Michigan Terminal System) Manual, Computing Center, University of Michigan, Ann Arbor, 1967.

4. Julyk, L., and Wolf, L., The CAMA Data Structure, Memorandum 29, Concomp Project, University of Michigan, Ann Arbor, August 1967, 49 pp. + appendices.

5. Goodrich, Mrs. S., CAMA: Define-Problem Command, Memorandum 28, ibid., June 1970, 31 pp.

6. Dingwall, T., Julyk, L., and Wolf, L., The CAMA Macro Processor, Memorandum 35, ibid., August 1970, 31 pp. + appendices.

7. Bisgrove, Mrs. J., and Goodrich, Mrs. S., Symbol Generation, internal memorandum, ibid., 22 June 1970, 8 pp.

8. Julyk, L., and Wolf, L., CAMA General Description, Memorandum 33, ibid., August 1970, in press.

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)*

UNIVERSITY OF MICHIGAN
CONCOMP PROJECT

2a. REPORT SECURITY CLASSIFICATION

Unclassified

2b. GROUP

3. REPORT TITLE

THE CAMA OPERATING SYSTEM

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

Memorandum

5. AUTHOR(S) *(First name, middle initial, last name)*

L. J. Julyk

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
August 1970	65	8

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
DA-49-083 OSA-3050	Memorandum 30
b. PROJECT NO.	
c.	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*
d.	

10. DISTRIBUTION STATEMENT

Qualified requesters may obtain copies of this report from DDC.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	Advanced Research Projects Agency

13. ABSTRACT

The CAMA (Computer-Aided Mathematical Analysis) operating system is a program which controls the operation of an interactive processor. It is designed to operate in the environment of a large central computer which polls a small graphics terminal computer for user-input. The CAMA system is designed to handle a number of different and independent operations, and to perform operations in a priority-based, multiply-queued environment. It is self-expendable by the use of its macro facilities.

DD FORM 1 NOV 65 1473

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
CAMA EXECUTIVE SYSTEM						
INTERACTIVE COMPUTER GRAPHICS						
PRIORITY-BASED TASK SCHEDULER						
TASK						
MATHEMATICAL ANALYSIS						