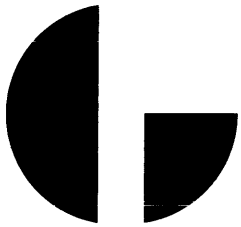


DATA GENERAL
CORPORATION
Southboro,
Massachusetts 01772
(617) 485-9100

INTRODUCTION

TO THE

REAL TIME OPERATING SYSTEM

ABSTRACT

This document provides an introduction to the DGC Real Time Operating System (RTOS). It also contains a basic description of the concepts and terms found in other documentation describing the use and structure of RTOS.

093-000093-01

Original Release - October 1973
First Revision - May 1974

PREFACE

The Real Time Operating System (RTOS) for the Data General family of computers and peripherals consists of a small, core resident, general purpose multitask monitor designed to control a wide variety of real time environments.

By using RTOS, user programs are relieved from the details of critical input/output device timings, data buffering, priority handling, and task scheduling. In addition, tasks are provided with a parallel processing capability plus intertask communication and synchronization facilities. Since RTOS is highly modular and reentrant, additional device handlers can be easily added to an RTOS system.

Communication with the RTOS executive takes place through a set of system and task control commands. Calling sequences, mnemonics, and operation of RTOS are identical to those in Data General's Real Time Disc Operating System (RDOS). This allows software development and debugging to be carried out on an RDOS system for later use in a core-only RTOS system.

TABLE OF CONTENTS

* Topics starred here and throughout the manual can be read for a broad overview of RTOS. Unstarred topics contain information in greater detail about RTOS.

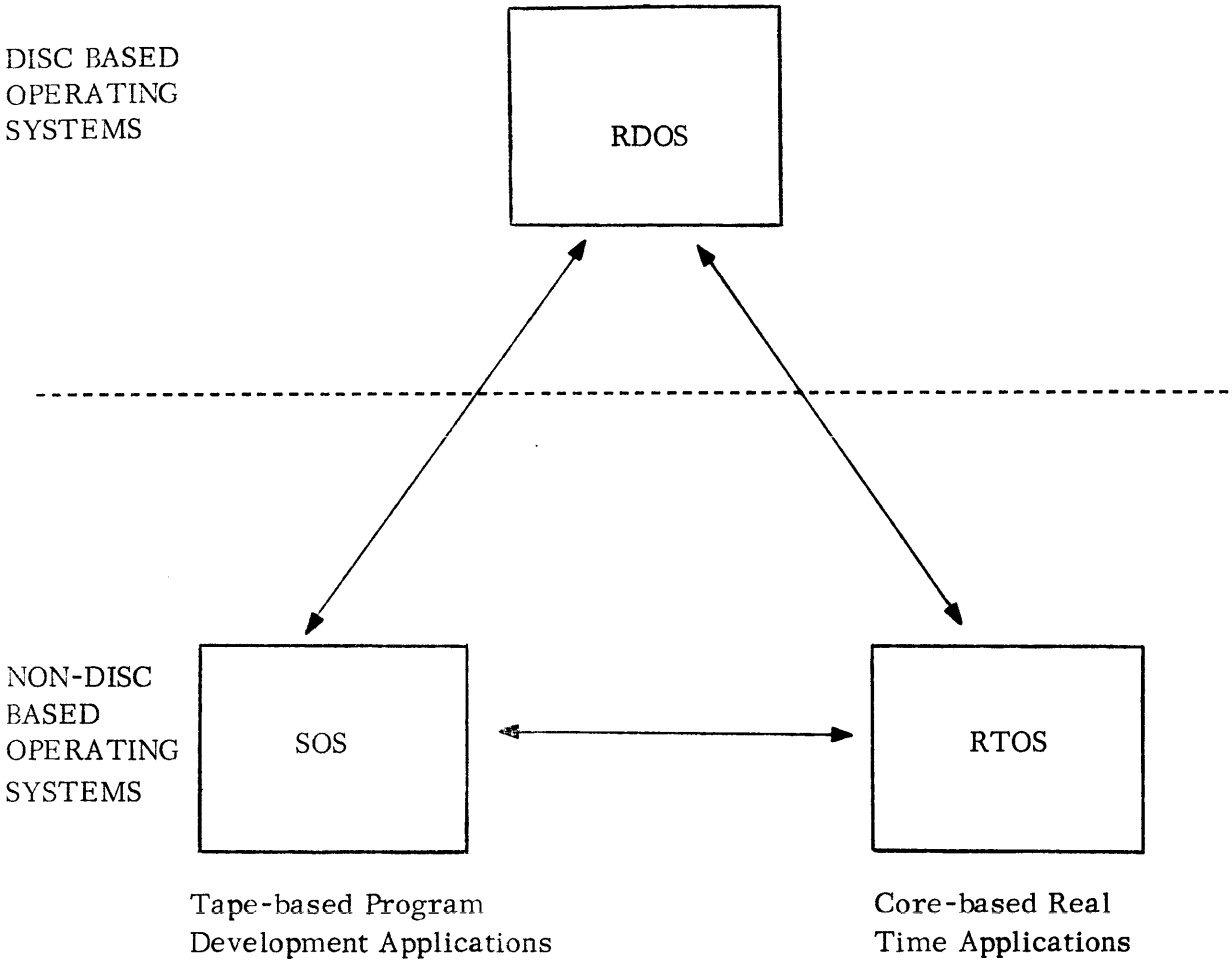
LIST OF FIGURES

* HOW TO USE THIS MANUAL

This manual uses modular instruction techniques to present a complex subject, the Real Time Operating System, in a simple, step-by-step fashion. Each topic is presented on a single page and is accompanied by a single-page illustration. There is a loose logical progression in the presentation of topics, yet each topic is truly self-contained. Thus there is no need to digest all the material in a single sitting; you may indeed read the chapters selectively according to your particular interests. If you would like a broad overview of the operating system, note that you may read only those chapters whose titles are starred.

No manual of this kind can present any topic in depth, and so after you have finished this primer you should consult one or more of the publications listed in the back of this manual following the glossary of terms. This list of publications gives you the titles and numbers of all DGC manuals describing RTOS and related utilities. Additionally, this list gives a précis describing what you may expect to find in each of these publications.

Disc-Based Program Development
and Real Time Applications



OPERATING SYSTEM COMPATIBILITY

*DATA GENERAL OPERATING SYSTEMS

The Data General family of computers is supported by a wide range of software designed to provide the user with a system that will meet his most demanding requirements. The first and possibly most important software package is the operating system, which interfaces user programs with the computer hardware.

Data General provides the user with three operating systems to help meet his application's needs in an economical and efficient manner. Thus, the user can concentrate efforts on problem solving -- application programs -- rather than on the intricacies of the hardware.

At the top of the operating system hierarchy is the Real Time Disk Operating System (RDOS), a powerful real-time multitasking disc based operating system that provides users with a system for interactive or batch program development and run-time support for demanding real-time environments. RDOS fully supports single processor configurations running in a single program, a partitioned memory foreground/background, or a fully protected dual programmed environment. Multiple processor configurations with shared discs or linked via a multiprocessor communications adapter provide the reliability necessary for critical real time applications.

For non-disc based applications or applications utilizing disc as a high speed bulk storage device, Data General provides the Real Time Operating System (RTOS). RTOS provides the user with a small and fast core resident, general purpose multitask executive that is a compatible subset of RDOS. Calling sequences, mnemonics, and operation are identical to those of RDOS so program development and testing can be done under RDOS for later use with RTOS. This compatibility also allows a user to develop programs for a non-disc system that can be easily expanded with the later addition of a disc.

For non-real-time applications, the Stand Alone Operating System (SOS) provides a program development and execution system that is also a compatible subset of RDOS. SOS provides the non-disc based user with a system of editors, assemblers, and loaders utilizing paper tape, cassette tapes, or magnetic tapes to speed user program development.

RTOS HIGHLIGHTS

- RTOS is a small, general purpose multitask real time operating system.
- RTOS is a compatible subset of RDOS allowing program development and debugging to be performed using a disc operating system.
- RTOS handles task scheduling and task priority.
- RTOS provides parallel task processing capability.
- RTOS provides for intertask communication and synchronization.
- * RTOS allows the execution of tasks to be monitored and controlled on either a priority or a task I. D. basis.
- * RTOS allows user tasks to communicate with the executive through a set of system and task commands.
- * RTOS controls and executes all input/output operations.
- * RTOS provides simple calling sequences to communicate with character, word, and block oriented devices.
- RTOS has handlers written for the console teletype, real time clock, paper tape readers and punches, line printers, card readers, plotters, cassettes and magnetic tape units, fixed and moving head disks, multiple teletypes, and communication and process I/O equipment.
- RTOS provides all necessary entry points so user specified device handlers can be easily added to the system.
- RTOS is a highly modular and reentrant software system provided as a relocatable package.

* REAL TIME SYSTEMS

A real time system is a hardware/software package that allows processing of information in a rapid and responsive manner. This response is swift enough to ensure that the results of the processing are available in time to influence the process or environment which is being monitored and controlled.

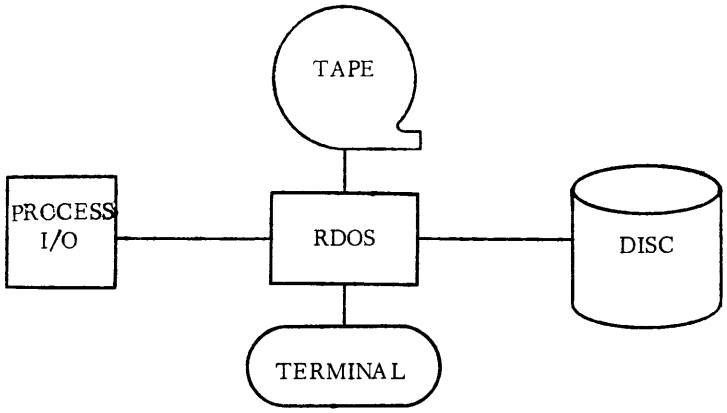
RTOS is an executive system designed to interface with user programs that have real time requirements. The standard input/output communications sub-system, together with the efficient scheduling and interrupt processing features of the executive system, provides an environment satisfactory for any real time program.

RTOS is an event-driven operating system. This means that the user does not have to execute special system calls to cause rescheduling to occur. RTOS maintains the highest priority (user-defined) task in execution which is capable of being in execution. This task, the highest priority ready task, will continue in execution until one of several events occurs:

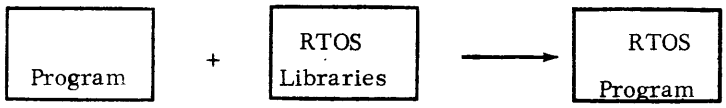
- the task terminates its own operation,
- the task becomes suspended while it awaits the occurrence of an event, e.g., the completion of an I/O operation,
- a higher priority task becomes capable of going into execution.

RTOS allows the user to implement his application on the computer quickly by providing the following aids:

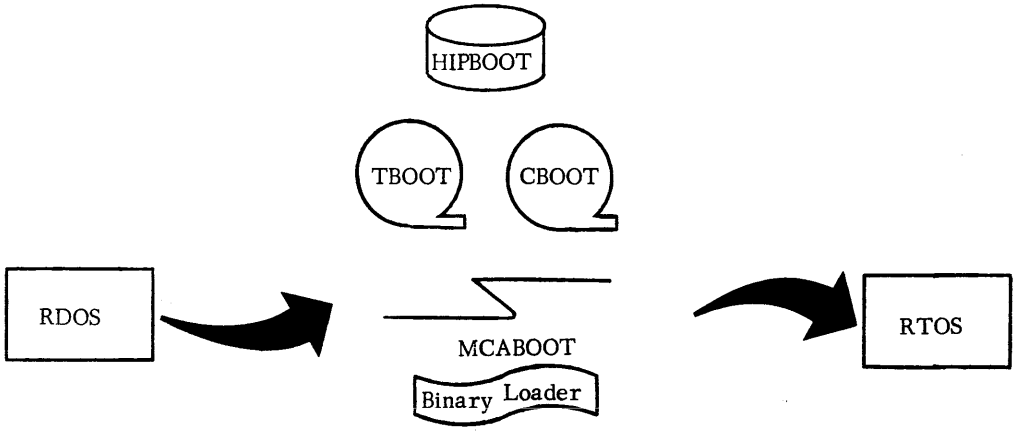
- standard device input/output handlers,
- a standard interrupt servicing program,
- all necessary executive functions to schedule the usage of the computer and peripherals effeciently.



1. Speed program development by using an RDOS system.



2. Reload the developed program with RTOS libraries.



3. Bootstrap the RTOS program.

RTOS PROGRAM DEVELOPMENT

The compatibility which exists between Data General's RDOS and RTOS operating systems can be extremely important to users engaged in program development. The speed and convenience of working under RDOS can be brought to bear on programs which will eventually be run without the benefit of a disc operating system. Moreover, even though RTOS is a strictly core-resident operating system, it does support the disc for bulk storage and uses a file structure which is compatible with that used by RDOS.

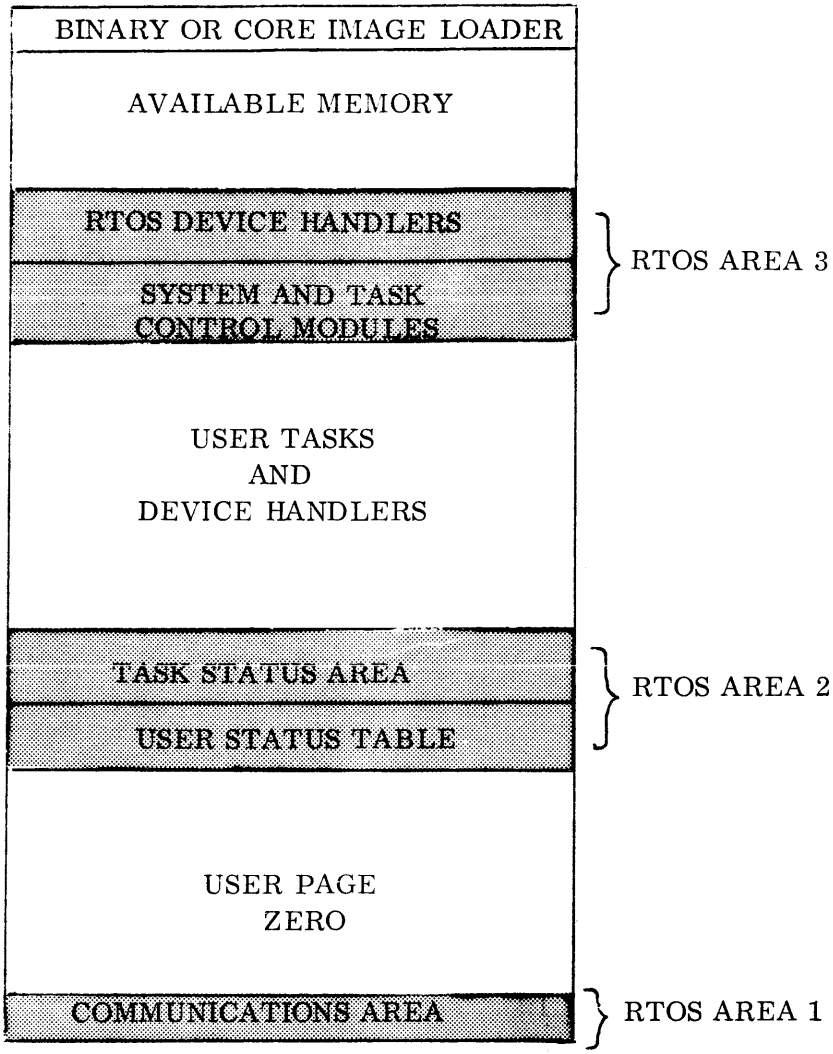
By utilizing those features of RDOS which will be available under RTOS, a user may perform all program development under RDOS. All stages of program development, from the initial creation and editing of source files through the execution and debugging of save files, will thus be expedited by using RDOS. Users wishing to use a high level language as a program development tool can use the real time FORTRAN IV calls which are available to both RTOS and RDOS.

After testing and debugging a program under RDOS which is to be run under RTOS, the program is merely reloaded with the RTOS libraries instead of those containing the RDOS modules. This will produce an RTOS program which is identical in operation to its RDOS counterpart. The RTOS program, however, will be entirely core-resident and can be run on systems with or without a disc.

After a program has been reloaded with the RTOS libraries, it must be bootstrapped into execution. The method of bootstrapping the program depends upon the device configuration of the system under which it will be running. There are four distinct kinds of systems, and thus four different methods of bootstrapping the systems:

- Disc systems
- Magnetic Tape or Cassette systems
- Multiprocessor systems
- Paper Tape systems.

If the RTOS system contains a disc unit, the RDOS disc bootstrap program, HIPBOOT, can be used to activate the RTOS program. HIPBOOT runs independently of RDOS and provides the most convenient means of executing an RTOS program. Systems with either a magnetic tape or cassette unit can use a tape bootstrap program (TBOOT or CBOOT, as appropriate) to load and execute an RTOS program. If one CPU running under RDOS and one or more idle CPUs are linked by a multiprocessor communications adapter (MCA), an RTOS program can be transmitted from the RDOS system to one or more of the idle CPUs. This transmission would be accomplished by means of the MCA bootstrap program, MCABOOT. Finally, paper tape systems use the bootstrap and binary loaders to load and execute an RTOS program.



RTOS CORE CONFIGURATION

ORGANIZATION AND CORE CONFIGURATION

The RTOS executive, composed of system and task control modules, constitutes the main framework of the operating system. It provides routines to process interrupts and dispatch them to device interrupt servicing modules, to process system calls that initiate I/O functions, to define new interrupt processing routines or get/change the time of day or date, and to process task calls that control the flow of execution through user written tasks.

RTOS occupies three areas in memory. The lowest 20_8 memory locations are used as a communications area for both interrupts and tasks wanting to access the other two areas.

The second area begins at location 400_8. It contains a User Status Table (UST) that provides information describing the total program such as its size, starting address, starting address of task queues, and whether the debugger is present. Above the UST is the task status area followed by device and I/O channel control tables.

User task and device drivers are loaded above the second RTOS area. Immediately above the user task area is the RTOS system and task control modules followed by RTOS device handlers. These last modules are extracted from the RTOS libraries to satisfy unresolved references made by user tasks. This area contains only those task and device modules necessary to control the user program environment.

```
CORE STORAGE (IN K WORDS) 12

RTC FREQ (0=NONE, 1=10HZ, 2=100HZ, 3=1000HZ, 4=LINE)4
LINE FREQ (0=50HZ, 1=60HZ) 1

TASKS(1-255)  ? 10
CHANNELS(1-63)  ? 8

RESPOND WITH NUMBER OF UNITS
DISK(0-1)  ? 1
    DISK STORAGE (IN K WORDS) 128

    DISK FILE STRUCTURE
    1ST BLOCK ? 6
       END BLOCK  ? 200
          NAME  ? A
          NAME  ? FILEA
    1ST BLOCK ? 201
       END BLOCK  ? 506
          NAME  ? FILEB
    1ST BLOCK ? 507
       END BLOCK  ? 511
          NAME  ? FILEC

DKP(0-4)  ? 0
MTA(0-8)  ? 0
CAS(0-8)  ? 0
PTR(0-2)  ? 1
PTP(0-2)  ? 1
LPT(0-2)  ? 1
    COLUMN SIZE (80, 132) 80
CDR(0-2)  ? 0
PLT(0-2)  ? 0
QTY(0-64) ? 0
TTYS(0-3) ? 1
MCA(0-15) ? 2
    #  TIMEOUT RETRIES  (0-65535)  ? 200

RESPOND WITH 0 FOR NO, 1 FOR YES

AUTO RESTART  ? 1

HIGH PRIORITY INTERRUPTS  ? 0
USER SUPPLIED DRIVERS  ? 0

   SUMMARY OF RTOS SYSGEN

   CODE     DCT NAME    NAME

    06        MCTDC
    07        MCRDC
    10        TTIDC      $TTI
    11        TTODC      $TTO
    12        PTRDC      $PTR
    13        PTPDC      $PTP
    14        RTCDC
    17        LPTDC      $LPT
    20        DSKDC

SYSGEN OKAY ? 1
OUTPUT FILENAME ? SYS1
```

SAMPLE RTOSGEN DIALOGUE

SYSTEM GENERATION AND LOADING

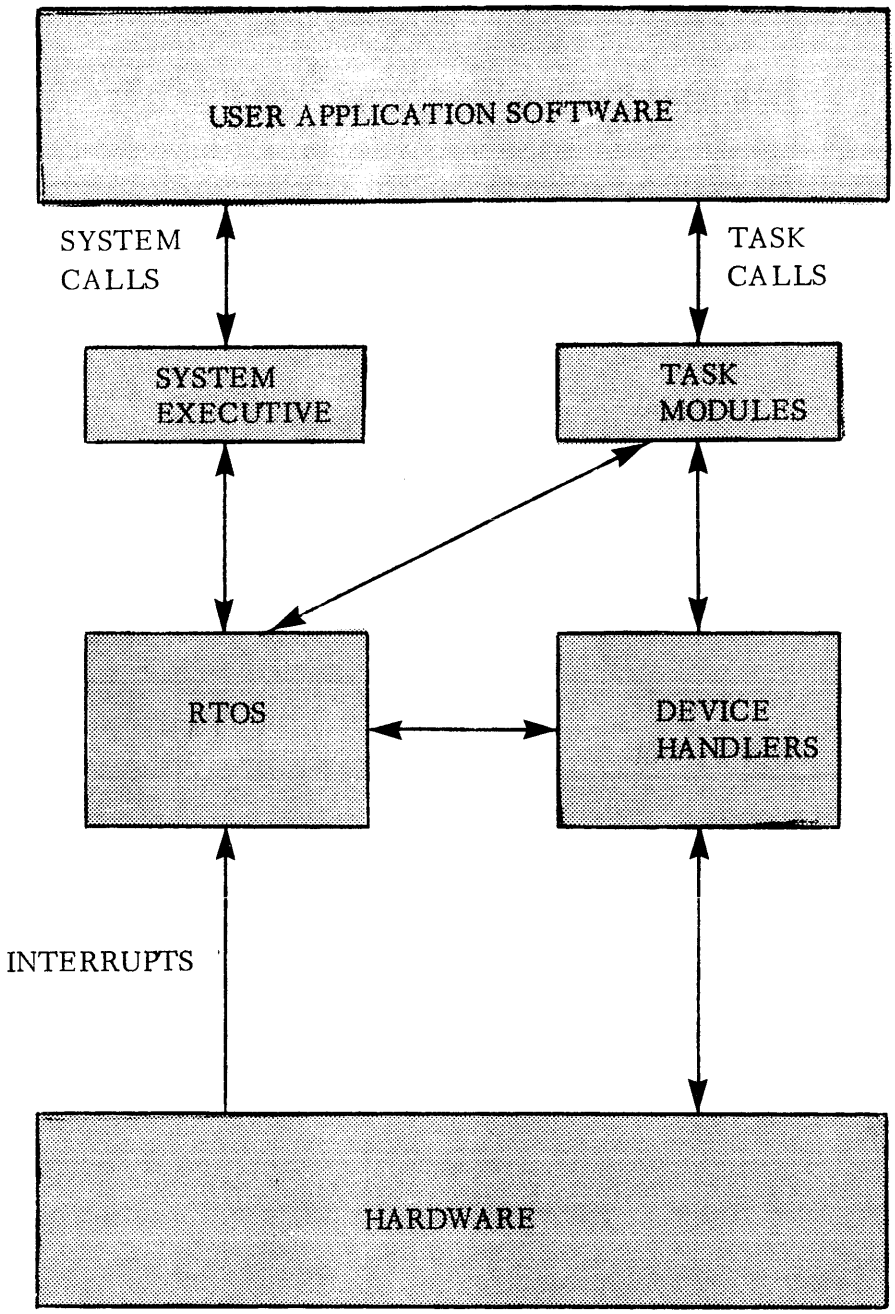
In real time environments, individual user requirements vary from installation to installation either due to hardware differences or to differing requirements in each application. These differences may be diverse combinations of standard Data General-supplied hardware, special user-constructed interfaces, different core memory sizes, or system throughput and environment considerations.

Thus to promote efficient utilization of core memory in each installation, each RTOS system must be tailored to the specific requirements of the installation. This tailoring process is called system generation. A utility program, RTOSGEN, is provided to perform system generation. RTOSGEN can be executed as a stand-alone program or it can be executed under RDOS. This utility provides the first step in the tailoring process by asking a series of questions on the teletype console to which the user must respond. At the end of each question series, RTOSGEN outputs a relocatable binary module on paper tape, magnetic tape or cassette, or a disc.

The relocatable binary module must be loaded with the application software relocatable binaries and with the RTOS libraries. This process, system loading, is the second and final step performed to produce an executable RTOS program. Relocatable loading can be performed either under a stand-alone loader, under SOS, or under the RDOS relocatable loader. The RDOS loader, like all RDOS utilities, is the most convenient to use. If loading is done under RDOS, a single command line like the following could be input to the system via the teletype console:

```
RLDR/C  RTOS  USER1  USER2  RTOS1.LB  RTOS2.LB )
```

The RTOS system generation module is named RTOS in this illustration; the user application programs are named USER1 and USER2. When relocatable loading is finished, the RTOS program is fully executable. Several different system loads with different RTOS modules could be performed to produce numerous RTOS programs tailored to individual application requirements.



PROGRAM EXECUTION FLOW

* PROGRAM EXECUTION FLOW

RTOS directs all activity in a real time application. RTOS receives requests from the user program and initiates suitable actions to satisfy these requests. RTOS also receives control when an interrupt occurs and passes control to the identified device handler while it holds the real time application in suspension.

There are two principal ways a user program can signal RTOS to perform desired actions for the program. These are system and task calls, and they are issued as program instructions. They activate program logic within either the system or task modules.

System calls generally request RTOS to perform I/O operations; to return to the user the value of system data such as the time of day, date, or the current clock frequency; or to define user interrupt servicing routines.

Task calls perform user task management functions such as task initiation, suspension, activation, and termination on an individual or group basis using task priorities or other identifiers.

Provision is made in RTOS for a user device handler to send a message to a task in the user application program. This facility permits external events to influence the multitasking environment even though it is being held in suspension while interrupt servicing occurs.

Clock/Calendar Commands:

.DELAY	Suspend a task for a specific time interval.
.GHRZ	Get the real time clock frequency.
.GDAY	Get the current date.
.GTOD	Get the time of day.
.SDAY	Set the current date.
.STOD	Set the time of day.

User Defined Interrupt Servicing Programs Commands:

.DUCLK	Define a user clock routine.
.IDEF	Identify a user interrupt device.
.IRMV	Remove a user interrupt device.
.RUCLK	Remove a user clock .

Console I/O Commands:

.GCHAR	Get a character from the keyboard.
.PCHAR	Output a character to the teletypewriter.
.WCHAR	Wait for a specific character to be input.

Device Control Commands:

.GMCA	Get the current CPU's MCA number.
.INIT	Initialize a magnetic tape or cassette unit.
.RLSE	Release a magnetic tape or cassette unit.

Input/Output Control Commands:

.APPEND n	Append to a device.
.CLOSE n	Close a file or device.
.GCHN n	Get a free channel number.
.MTDIO n	Perform free format I/O on a magnetic tape or cassette unit.
.MTOPD n	Open a magnetic tape or cassette unit for free format.
.OPEN n	Open a device or file on a channel I/O number.
.RDB n	Read one or more disc blocks.
.RDL n	Read a line of ASCII characters.
.RDS n	Read data in image format.
.RESET	Close all opened devices and files.
.WRB n	Write one or more disc blocks.
.WRL n	Write a line of ASCII characters.
.WRS n	Write data in image format.

Memory Size Commands:

.MEM	Determine the amount of available memory.
.MEMI	Change the top of available memory.

System Idle Commands:

.ERTN	Idle the system abnormally.
.RTN	Idle the system normally.

SYSTEM CALLS

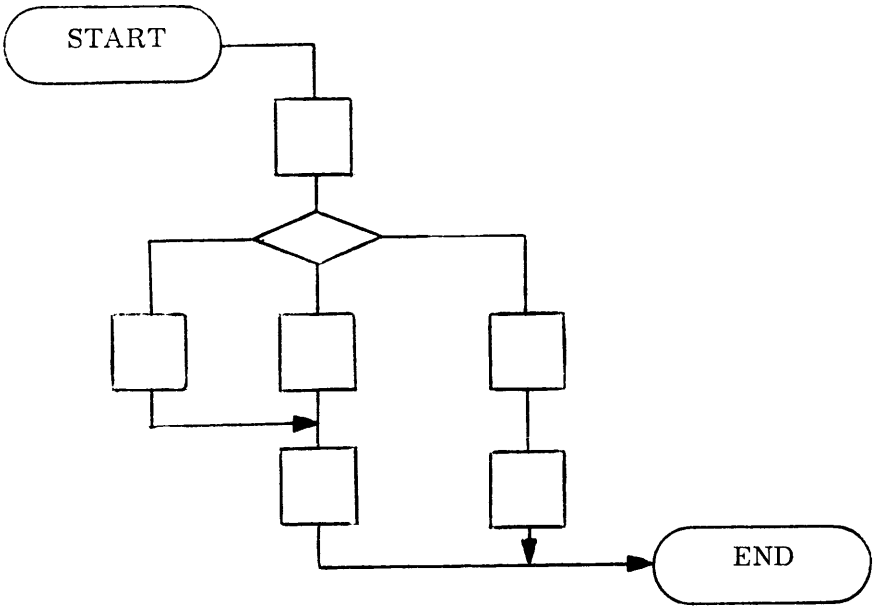
Users can communicate with RTOS by making system calls to RTOS subroutines to perform such functions as input/output, obtaining or changing system information like the time and date, and initializing devices. When an RTOS subroutine is entered, control passes to the task monitor, which saves the task environment before passing control to RTOS. Upon return from the system, the user's AC3 is set to the User Stack Pointer (USP). The contents of AC3 can conveniently be saved in the USP before a system call is made and restored automatically on return.

System calls are made in either of two forms:

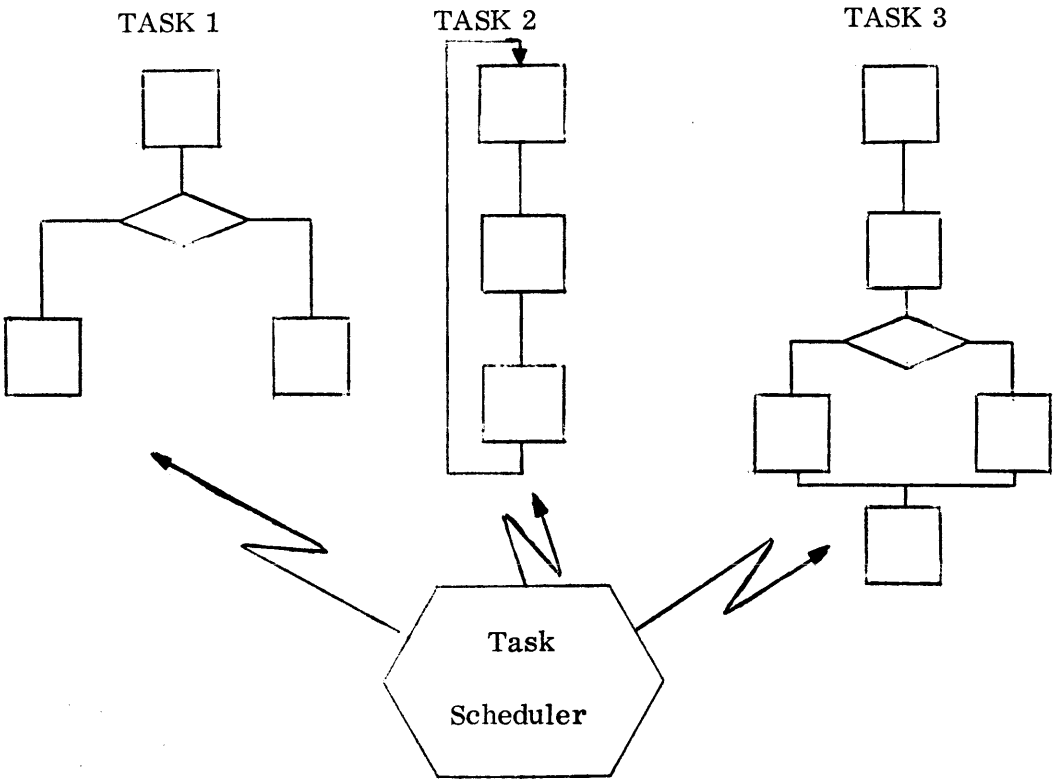
```
.SYSTM                  .SYSTM
command                 command n
exceptional return      exceptional return
normal return           normal return
```

The mnemonic .SYSTM, which must precede each command word, is recognized by both extended and RDOS assemblers as a subroutine call. The specific system command is assembled as the word following this mnemonic. In the first call format above, the command appears alone as the second word in the calling sequence. If the command requires arguments, they are passed in the accumulators AC0, AC1, and/or AC2. In the second call format, n is a positive integer or mnemonic equated to a positive integer representing an I/O channel number. (The channel number is a logical link to an opened file or device.) Additional arguments, if any, are passed in AC0, AC1, and/or AC2.

Any system command requiring a channel number need not specify it in the command itself. If an octal 77 (or the mnemonic CPU) is specified as the channel number with the command, the system uses the channel number passed in AC2. This facility provides a flexible runtime device selection method. In addition, passing arguments through the accumulators, instead of inline, makes it possible to write routines that can be shared by many tasks.



SINGLE TASK ENVIRONMENT



MULTITASK ENVIRONMENT

* MULTITASKING SYSTEMS

A task is a logically complete execution path through a user program demanding the use of system resources such as CPU and I/O device control. Single task environments are already familiar to users of non-real time operating systems. System utility programs such as assemblers, compilers, or relocatable loaders are all examples of programs executing in a single task environment. In summary, a single task environment is a program with a single unified path connecting all its program logic, no matter how complex the logic branches.

In a single task environment, the processor idles when the program initiates an I/O operation. The program is activated when the operation is complete. During the time the I/O operation is being performed, no useful user processing occurs. It was the problem of efficiently performing seemingly unrelated functions in a nonsequential manner to utilize the processor and I/O devices more fully that led to the notion of multitask real time operating systems.

A single real time program can have from one to a virtually unlimited number of logically distinct tasks. Each task performs a specified funtion asynchronously and in real time. CPU control is allocated by the RTOS Task Scheduler to the highest priority task that is ready to perform or continue performing its function.

Examples of multitask environments are process control operations; communications systems involving line control, message reception, queuing and switching; and multiterminal data base systems. The individual tasks within a process control system, for example, might be data collection, data alarming, control operations, periodic and demand data logging, operator console control, and control of line to supervisory computer.

EXECUTING	-	The task has control of the central processor unit (CPU).
READY	-	The task is ready and available for execution but cannot gain control of the CPU until all higher priority tasks existing in the READY or EXECUTING state are completed or go into a SUSPENDED state.
SUSPENDED	-	The task is awaiting the occurrence or completion of some system call or other real time operation.
DORMANT	-	The task has not been initiated (made known to RTOS) or its execution was completed and it is now idle.

TASK STATES

TASK STATES AND PRIORITIES

User multitask programs run under RTOS have one task initiated for them by the initialization phase of RTOS. To create a multitask system, the first task initiates the second and subsequent tasks by issuing the appropriate task monitor calls.

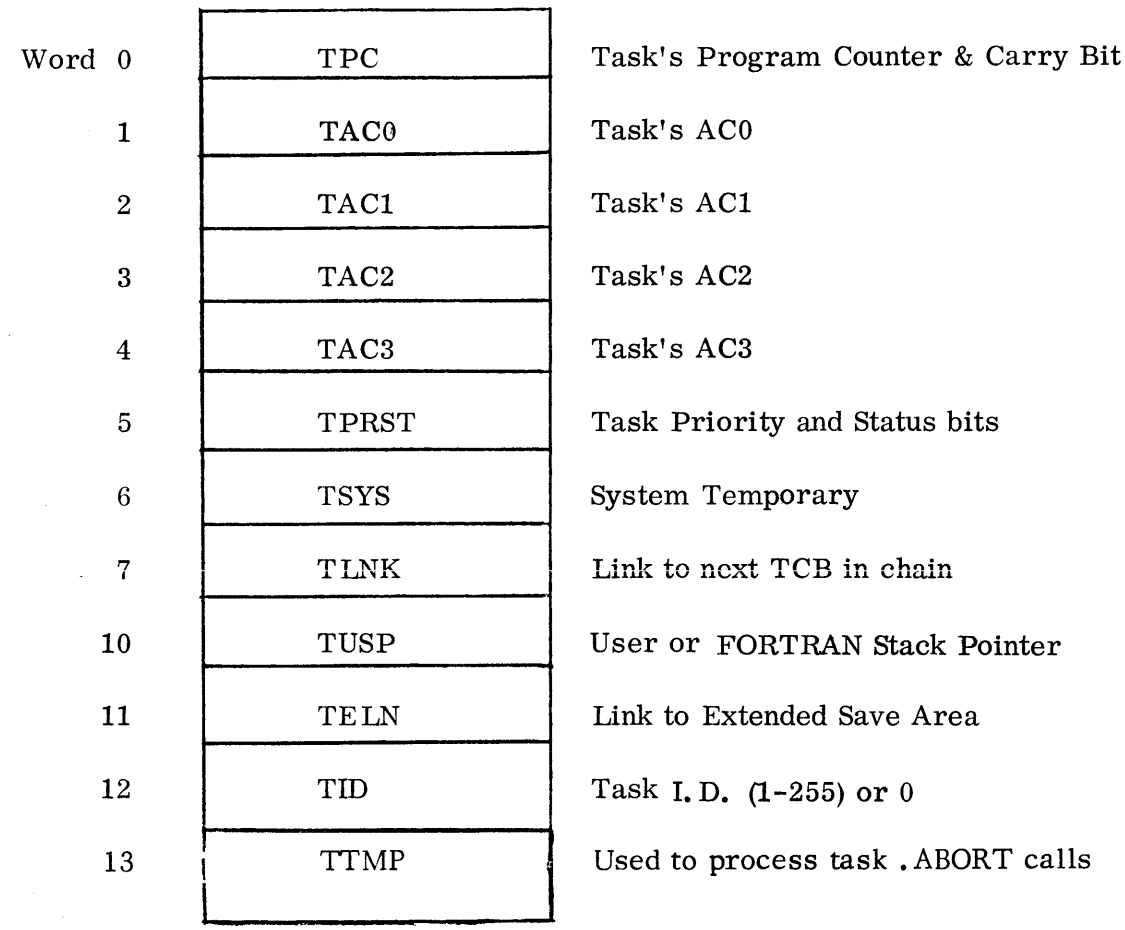
When a multitask environment is established, the Task Scheduler must decide which task should be executing. To enable the scheduler to function, each task is assigned a priority at the time of initiation. RTOS permits 256 levels of task priority in the range 0 through 255, with priority 0 being the highest. Several tasks can exist at the same priority level.

The default task is initiated at the highest priority and since it is the only task in the system, it receives control. When this task initiates the second and subsequent tasks, the Task Scheduler is called upon to put the highest priority task into execution. Other tasks that have been initiated but are of lower priority are said to be ready to run.

The executing task becomes suspended when it makes a call to the operating system to perform some function such as an I/O transfer or get time of day. The task remains suspended until the operation is completed, at which time it becomes readied.

Thus we have seen that tasks under the RTOS system can exist in either of four states:

- Tasks are in control of the CPU and are EXECUTING their assigned instruction paths.
- Tasks are READY and are waiting to become the highest priority task available for execution.
- Tasks are SUSPENDED waiting to be readied in response to a system or I/O call being completed.
- Tasks are not known to the system and are in a DORMANT state.



TASK CONTROL BLOCK LAYOUT

TASK ENVIRONMENTS

As discussed in the previous section, a task within an RTOS system can be either in an idle or active state. If the task is in the DORMANT state, the system has no knowledge that it exists even though the code remains part of the program. When a task is active (in the EXECUTING, READY, or SUSPENDED state), certain status information must be maintained about each task to enable the operating system to manage the environment and for the Task Scheduler to keep the highest priority ready task in the EXECUTING state.

This status information about each active task is contained within an information structure called a Task Control Block (TCB). There is one TCB for each task in the active state (no TCB for an idle task). TCB's are used to store active registers and other priority and status information when the task exists in either the READY or SUSPENDED state. The TCB of the executing task is allocated to the task but the TCB remains unused by the system until the task relinquishes control of the CPU. If the task becomes readied or suspended, its TCB is then used to store its status information. If, on the other hand, the rescheduling resulted from the task terminating its own execution, its TCB is placed into a pool of available TCBs.

The TCBs of ready and suspended tasks are linked together in chains. These chains are organized in order of decreasing task priority. This means that the Task Scheduler need only look at the top of the active chain to select the highest priority task. Each TCB in the active chain is connected by its link word to the next TCB in the chain. Among equal priority tasks, a round-robin scheduling of system resources is performed. Whenever a task has its TCB entered in the active chain, the task is assigned automatically the lowest priority within a priority level. The last TCB in the chain has a link of -1.

Unused TCBs in the system are linked together to form an inactive chain of available TCBs. Except for the link words, these TCBs are empty until a task is initiated; then, a free TCB is removed from this chain, filled with information about the task, and placed on the active chain.

TASK CALL SUMMARY

Task Initiation and Termination :

.TASK	Initiate a task.
.KILL	Terminate the calling task.
.AKILL	Terminate all tasks of a specified priority.
.ABORT	Abort a task's operation.

Task Execution Control:

.SUSP	Suspend the calling task.
.ASUSP	Suspend all tasks of a specified priority.
.ARDY	Ready all tasks of a specified priority.
.PRI	Change the calling task's priority.

Intertask Communication/Synchronization :

.XMT	Transmit a message.
.XMTW	Transmit a message and wait for it to be received.
.IXMT	Transmit a message from an interrupt routine.
.REC	Receive a message.

Task Control by I. D. :

.IDST	Obtain task's status by I. D.
.TIDK	Terminate a task by I. D.
.TIDR	Ready a task by I. D.
.TIDS	Suspend a task by I. D.
.TIDP	Change a task's priority by I. D.

User Interrupt Processing:

.SMSK	Set an interrupt mask.
.UCEX	Exit from a user clock routine.
.UIEX	Exit from a user interrupt routine.
.UPEX	Exit from a user power-fail routine.

TASK CALLS

Unlike system calls, task calls consist of single word instructions; all arguments for task calls are passed in the accumulators. Not all task calls have error returns and those that do not have error returns, do not reserve error return locations.

The general form of a task call in a program is:

AC0, AC1, AC2 contain required input arguments.

TASK CALL MNEMONICS
error return
normal return

AC0, AC1, AC2 contain output values after call
processing; AC2 is set to the error code, if any.
AC3 contains the User Stack Pointer (USP).

Each task call has an associated modular package of relocatable binary code in the RTOS library needed to perform the call processing. This feature - modularity - enhances core utilization since only those modules selected by the user occupy program space at load time.

Upon returning from processing a task call, RTOS gives program control to the Task Scheduler so that it can maintain the highest priority READY task in the EXECUTING state.

An example of part of a program that uses task calls follows.

```
        .
        .
        .
LDA 0 TPRI          ; LOAD TASK I.D./PRIORITY
LDA 1 TADDR         ; PICKUP TASK ADDRESS
.TASK
JSR ERROR           ; PROCESS ERROR
        .
        .
        .
LDA 0 MESAD         ; PICKUP MESSAGE ADDRESS
LDA 1 MESS          ; GET MESSAGE
.XMTW               ; TRANSMIT IT TO CALLER
JSR ERROR           ; PROCESS ERROR
        .
        .
        .
```

FORTRAN TASK CALL SUMMARY

Task Initiation and Termination:

ITASK, FTASK	Initiate a task.
*START	Start a task after a time delay.
*TRNON	Execute a task at a specified time.
*KILL, EXIT	Terminate a task in an orderly fashion.
AKILL	Terminate a class of tasks in an orderly fashion.

Task Execution Control:

SUSP	Suspend the operation of a task.
ASUSP	Suspend the operation of a class of tasks.
*WAIT, FDELY	Suspend a task for a specific time interval.
ARDY	Ready a class of tasks.
PRI	Change the priority of a task.

Intertask Communication/Synchronization:

XMT	Transmit a message to a task.
XMTW	Transmit a message to a task and await its receipt.
REC	Receive a task message.

Task Control by I.D.:

* STTSK	Obtain the status of a task.
* ABORT	Terminate a task's operation abruptly.
* RELSE	Ready a suspended task.
* HOLD	Suspend a task.
* CHNGE	Change the priority of a task.

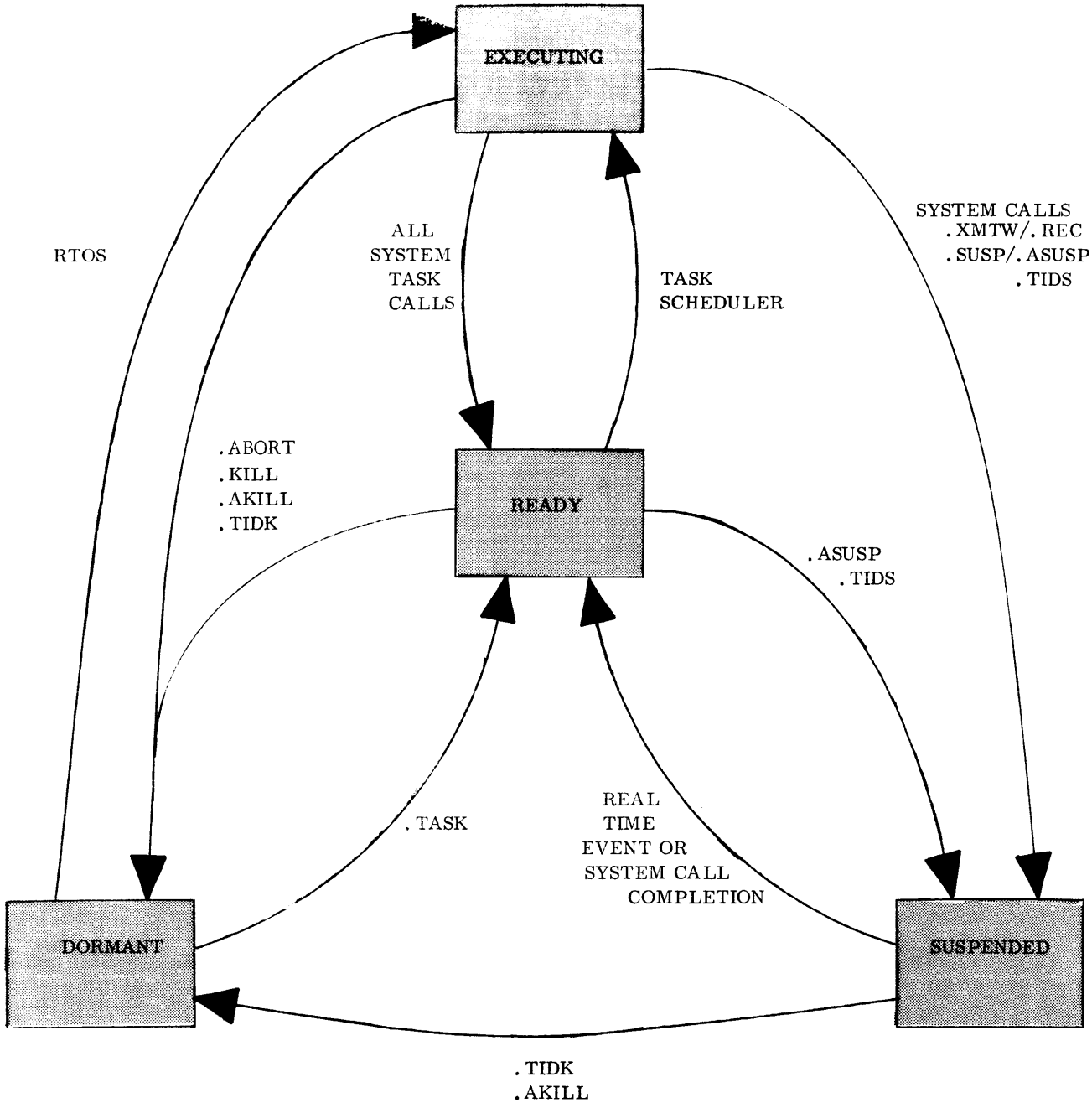
* REAL TIME FORTRAN IV PROGRAMMING

The benefits of programming in a high level language are many, and in essence they can all be reduced to one advantage: dollar savings. Programs written in high level languages reduce software production cost and total system production time, and they avoid expensive training costs. Because of their inherent improved readability, high level languages improve the uniformity of program coding and make program logic more visible ... features which decrease program maintenance costs.

Economies obtained from the use of high level languages in applications programs typically reduce needed manpower by 45%, even though high level languages increase core requirements by 20% and sometimes increase program execution time by up to 45%. Reduced hardware costs have inspired new interest in FORTRAN as a popular development tool in real time programming, and Data General provides several ANSI FORTRAN software packages for this use. As you might expect, Data General's FORTRAN packages run compatibly. FORTRAN program development can be accomplished with ease on an RDOS system for use on a system which will run under RTOS. Both versions of FORTRAN contain features required in real time applications:

- ANSI FORTRAN compatibility,
- reentrancy at run time,
- full multitasking support,
- Global/named common areas,
- in-line assembler coding capability,
- I/O access capability to bulk storage devices,
- subroutine library for analog and digital process I/O,
- bit/string manipulations,
- expanded debugging aids.

The Instrument Society of America (ISA) has participated actively in the proposed design of real time FORTRAN extensions. Standards proposed by the ISA were discussed at the eighth workshop on the standardization of industrial computer language, commonly called the "Purdue Workshop". The ISA recommends FORTRAN language extensions in recognition of the fact that it is a popular high level language which could simplify real time applications programming. Starred FORTRAN call names listed on the preceding page are those calls available in DGC FORTRAN which conform to the recommendations of the Purdue Workshop.



TASK STATE TRANSITIONS

*TASK EXECUTION CONTROL

When an RTOS program is initiated, it gives control to a single user task after initializing the RTOS status tables. It could be viewed as taking the first task from the DORMANT state directly to the EXECUTING state. This task must initiate other tasks to set up the desired multitask environment. Additional tasks as they are initiated are put into the READY state.

To insure that the highest priority READY task is always the task in the EXECUTING state, task calls cause the issuing task to move from the EXECUTING state to the READY state. READY tasks await their turn in a READY queue organized by priority. The Task Scheduler takes the first task in this queue (the highest printing task, by definition) and puts it in the EXECUTING state.

SUSPENDED tasks are those that were once in the READY or EXECUTING state. A task may become suspended for one of the following reasons:

- It issued a suspend call, .SUSP, .ASUSP, or .TIDS.
- It suspended itself for a specified time delay, .DELAY.
- It is waiting a message from another task, .REC.
- It has issued a transmit-and-wait call, .XMTW.
- It is waiting the completion of a .SYSTM call.

Just as a number of different events may suspend a task, several events and task calls can cause a SUSPENDED task to be put into the READY state.

- The completion of a .SYSTM call (such as a request for I/O or the expiration of a time delay).
- The posting of a message for a task awaiting its receipt, .XMT, .XMTW or .IXMT.
- The request for a message previously sent and being waited for, .REC.
- The readying of a task by task calls, .ARDY or .TIDR.

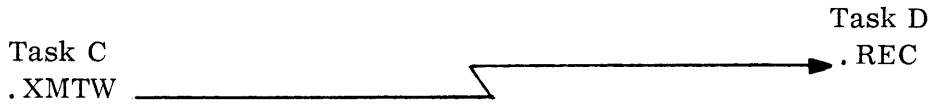
Tasks enter the DORMANT state by any of the following:

- The calling task terminated itself, .KILL.
- A task with a given I.D. was terminated, .TIDK.
- All tasks of a given priority level were terminated, .AKILL.

If all tasks are terminated either individually or on a group basis, the entire system is placed into an idle state, essentially halting the activity of the system although the system remains capable of responding to device interrupts.

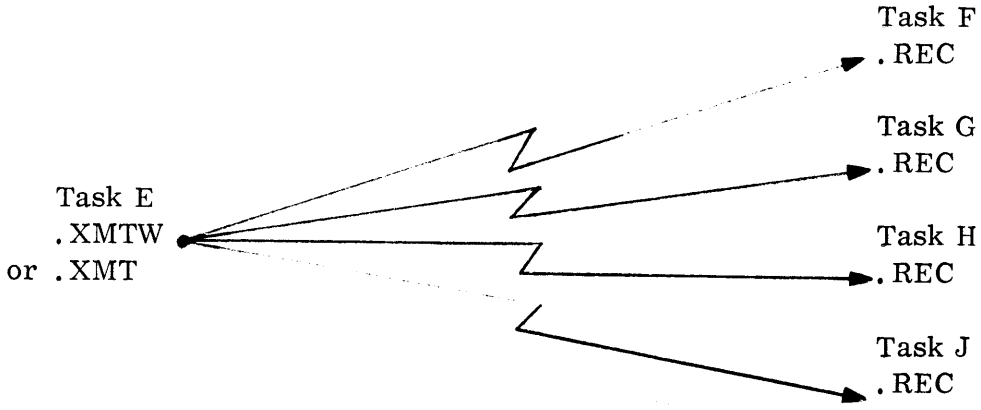
Task A sends message and goes into the READY state. Task B is SUSPENDED until the message is received.

INTERTASK COMMUNICATION



Neither task proceeds until the message is passed from Task C to Task D.

TASK SYNCHRONIZATION



GROUP TASK CONTROL

INTERTASK COMMUNICATION/SYNCHRONIZATION

Even though tasks operate asynchronously, it is often desirable for one task to be capable of "talking" to another task. Tasks communicate with one another under RTOS by sending and receiving one-word messages in agreed-upon core locations (message address). One-word messages can, of course, be pointers to larger messages if the tasks agree beforehand on the use of such a technique.

A transmitting task can simply deposit the message in an agreed-upon location (.XMT), or the caller can deposit the message and wait until it is received (.XMTW). To receive such a message, another task issues a .REC task call. If the transmitting task has not yet sent the message when the .REC call is issued, the receiving task is SUSPENDED until the message is sent. If the message has already been sent, the receiving task accepts the message and is put into the READY state.

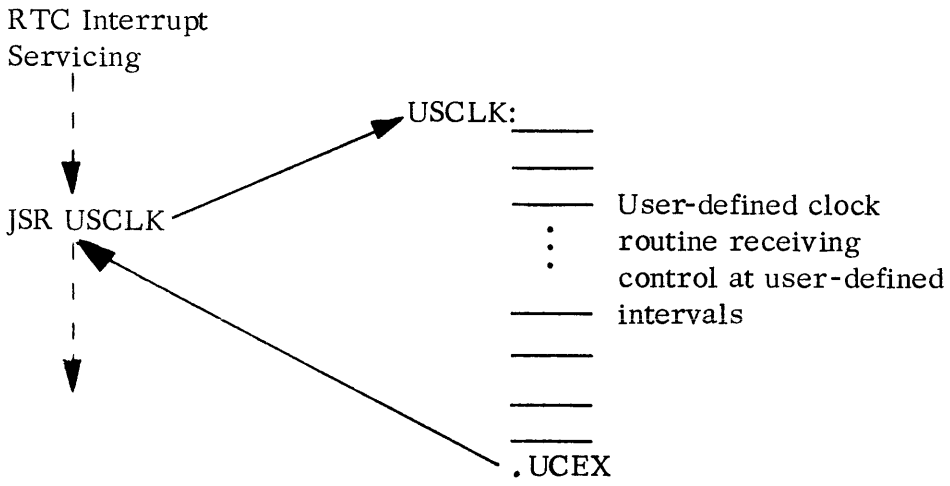
If the message was sent via the .XMTW task call, both the receiving and transmitting tasks are put into the READY state.

It is also possible to transmit a message directly from an interrupt servicing routine to a user task via the .IXMT call. This call is very useful for activating tasks based on the occurrence of external events indicated by interrupts.

More than one task can wait for the same message by issuing .REC calls using the same message address. The tasks are put into the READY state by another task issuing either a .XMT or .XMTW call.

```
PEND:   .SYSTM          ; Suspend calling task for a specified
        .DELAY          ; time interval.
```

```
TOD:    .SYSTM          ; Obtain time of day for use
        .GTOD           ; in task scheduling.
```



TASK TIMING CONTROL

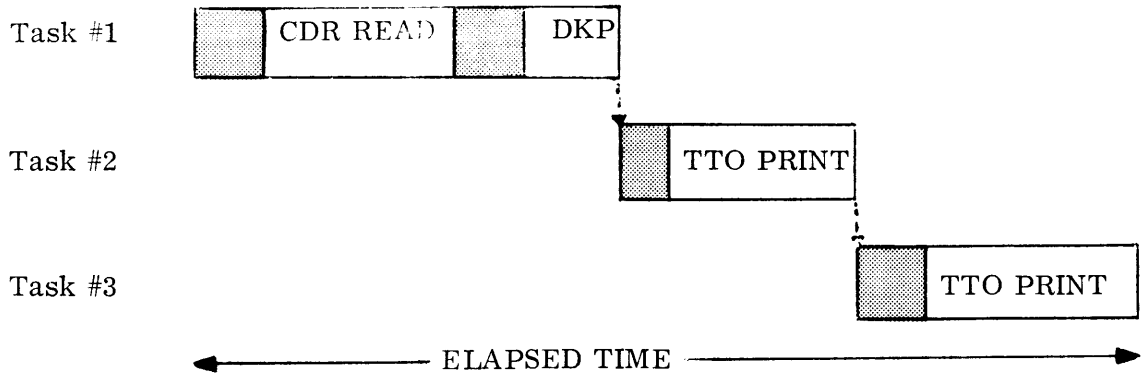
TASK TIMING CONTROL

RTOS offers several facilities for user tasks to perform time-related functions. For example, users can implement their own time-slicing or round-robin allocation of CPU control. By issuing the system call .DELAY, a task can suspend itself for a specified time period that is a multiple of the real time clock frequency. (The frequency of the real time clock can be determined by the system call .GHRZ.)

RTOS also maintains a system clock and calendar for tasks that should be scheduled on a time-of-day basis. Tasks can obtain or set the date or the correct time in seconds, minutes, and hours. By periodically monitoring the time-of-day clock, tasks can be scheduled for such funtions as performing hourly scans or printing shift reports.

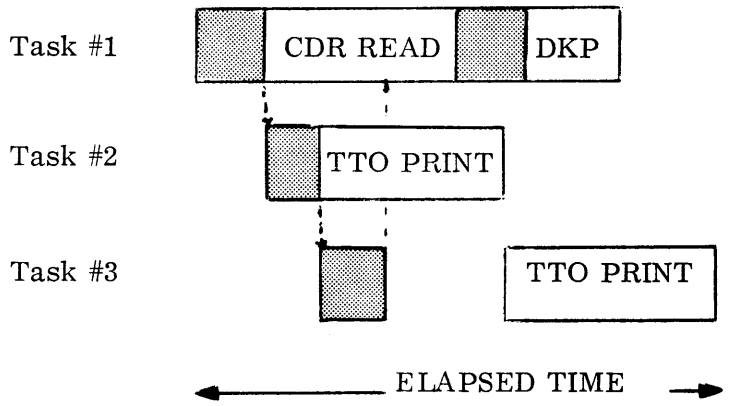
Finally, RTOS provides a pair of system commands permitting the definition and removal of a user clock, driven by the system clock. This user clock receives control at user-defined intervals. Thus, the user program can execute a short routine at the real time clock interrupt level. The routine can perform time-out control for other device drivers or allow task control based on milliseconds instead of seconds.

Single Task Operating System



NOTE: One task at a time executes.
CPU stays idle when tasks wait for I/O.

Multitask Operating System



NOTE: One task at a time executes.
CPU time is given to lower priority task when task waits for I/O. Same throughput as shown above, yet shorter elapsed time!

THROUGHPUT COMPARISONS

* INPUT/OUTPUT CONTROL

An important function of any real time operating system is the efficient handling of input/output operations. Optimum usage of machine devices and central processor time in the accomplishment of tasks is the real reason for designing and implementing a multitask system.

Since I/O devices are slow compared to the internal speed of the computer, they must be programmed to overlap their operations with computations, when possible, in order to:

- Increase usable CPU time
- Greatly increase efficiency of I/O operations
- Provide more throughput of data

The responsibility of RTOS I/O control routines is to react during normal program execution to the structuring of I/O requests, making assignments of requests to machine devices when they are idle, and queuing requests for devices that are busy. Through the queuing facility, RTOS achieves maximum and continuous overlap of multitasks without direct intervention by the tasks themselves.

All input and output of data via RTOS-supported devices must be through system I/O commands. Although the system does not reject any user I/O command, the issuance of such commands by a user would be both risky and unnecessary since a full complement of system I/O commands is provided within RTOS.

System I/O commands require a channel number to be given in the second field of the command word. This channel number is associated with a particular device or file when the device or file is first opened by the system command .OPEN. Once this association is made, all subsequent I/O commands pertaining to the file or device require only the channel number.

INPUT/OUTPUT COMMAND MODES

TYPE	CALL	DATA	QUANTITY
CHARACTER	.GCHAR/.PCHAR .WCHAR	ASCII	SINGLE CHARACTER
LINE	.RDL/.WRL	ASCII	FIELD TERMINATED BY CARRIAGE RETURN, FORM FEED, NULL, OR 132 CHARACTERS
SEQUENTIAL	.RDL/.WRS	BINARY	FIELD SIZE CONTROLLED BY BYTE COUNT
DIRECT BLOCK	.RDB/.WRB	BINARY	MULTIPLE 256-WORD DATA BLOCKS
FREE FORMAT	.MTDIO	BINARY	WORD COUNT (2-4096)
USER-WRITTEN DRIVER	NIOS, NIOC, NIOP DIA, DIB, DIC DOA, DOB, DOC	ASCII or BINARY	DEVICE INDEPENDENT

RTOS provides six basic modes for reading and writing data: character, line, sequential, direct block, free format, and user-written driver I/O.

In Character mode, a single character is transferred between the console teletype and AC0. The character received is stored right adjusted in AC0 with bits 0-8 cleared. The send character call transfers a character in AC0, bits 9-15, to the console. The third character call, .WCHAR, allows a task to be suspended until it receives a single, specific character from a teletype keyboard.

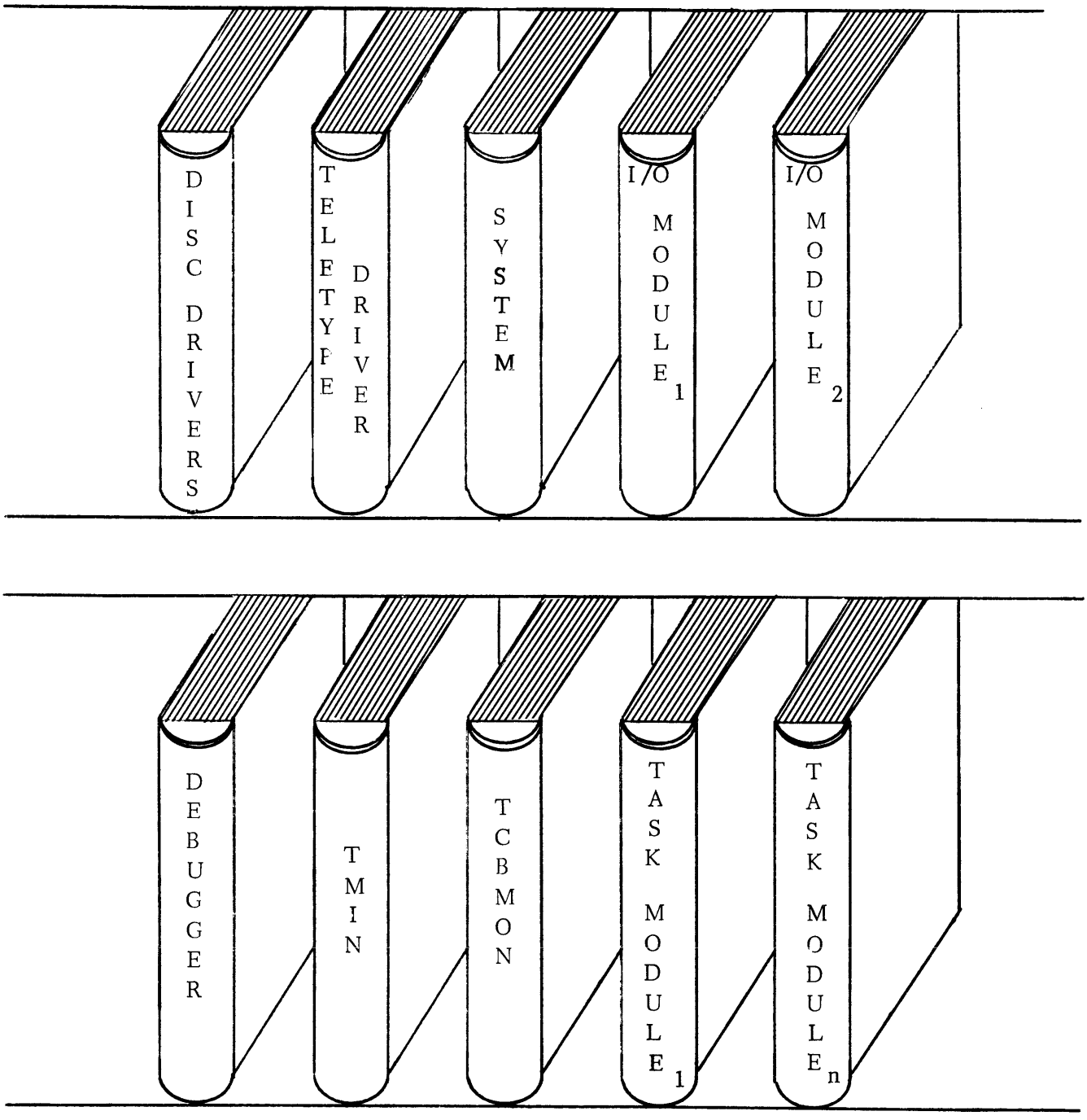
In Line mode, data read or written is assumed to consist of ASCII character strings, terminated by a carriage return, form feed, or null. Reading or writing continues until one of these three characters is detected. RTOS handles all device-dependent editing at the device driver level. For example, line feeds are ignored on character input devices and are supplied after carriage returns on all character output devices unless checking is suppressed. Furthermore, neither reading nor writing requires a byte count, since reading continues until a terminator is detected and writing proceeds until a terminator is written.

Sequential mode provides unedited data transfers. In this mode, no assumption is made by the system as to the nature of the information. Thus this mode is always used for processing binary data and can also be used for processing ASCII data (provided no editing of this data is required). Sequential mode transfers require specific byte counts to satisfy read or write requests. All character-oriented devices can be used in sequential data transfers.

In Direct Block mode, binary data is transferred directly betwen a disc file and a user core buffer. This mode allows single or multiple block transfers to or from contiguous files on fixed or moving head discs. This transfer takes advantage of the multiple block read/write capability of the hardware to process data transfers quickly.

In the Free Format mode, the operation of magnetic tape and cassette units can be controlled directly from the user program. This mode permits reading or writing of data in 2- to 4096- word records, spacing of the unit forward or backward 1 to 4095 data records or to an end-of-file, writing of an end-of-file , initiating of a rewind operation, or reading of the unit status. Free format mode allows tapes to be formated easily for IBM, CDC, UNIVAC, or other computer systems.

The User-Written Driver mode permits assembly language I/O instructions to be issued from device driver routines written by the user. These routines could be written to support either high priority interrupt devices or other devices which would receive ordinary interrupt service by the system.



RTOS SYSTEM LIBRARIES

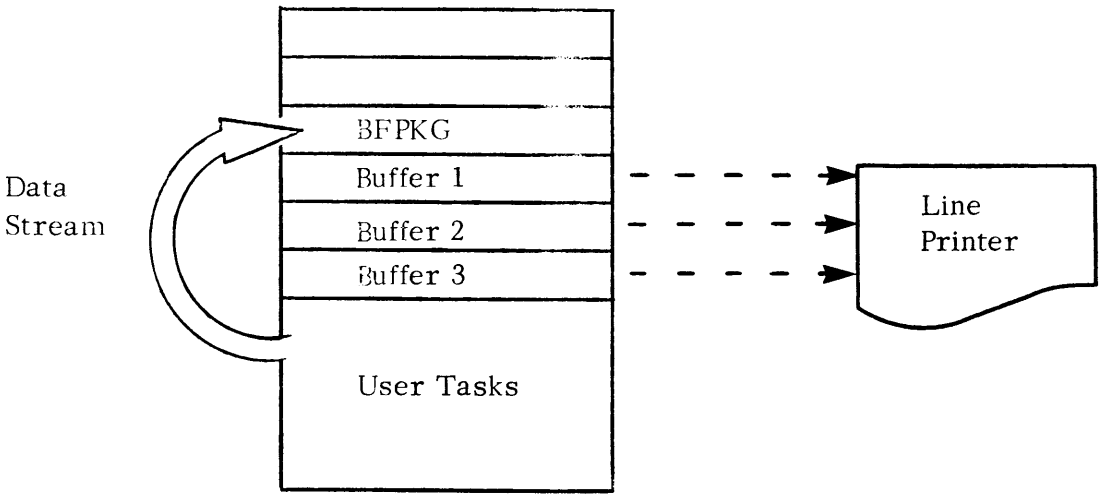
SYSTEM LIBRARY

The system libraries RTOS1.LB and RTOS2.LB are a collection of program modules that support user programs run under RTOS. These modules can be likened to volumes on a library shelf. Each user program needing one or more of these modules selects them from the library, leaving behind those of no current use. Because system modules are placed in the library, user program core requirements are greatly reduced.

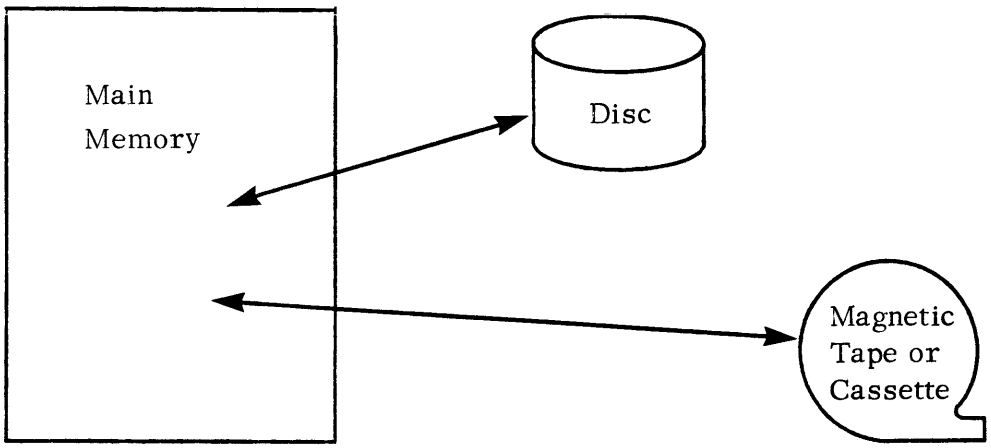
The Relocatable Loader acts as the system "librarian," extracting modules from the library. The Loader removes only those modules named in external pseudo-ops in the relocatable binary file created at system generation or in user-written tasks. This procedure ensures that, except for the Task Scheduler, only those task modules required for program operation are loaded and results in a net savings in total core used. All modules taken from the library are loaded after user-written tasks; thus, loading proceeds from low to high core.

The RTOS system library contains the multitask and single task schedulers, TCBMON and TMIN; command processing modules for each task and system call type; the buffered I/O package, BFPKG, which performs buffered asynchronous line and sequential data transfers; and drivers for each device supported by RTOS.

The system library also contains a symbolic program debugger utility. IDEB disables interrupts making it suitable for debugging real time program environments where it may be necessary to freeze program control.



BUFFER CONTROL PACKAGE, BFPKG



DISC FILE AND TAPE I/O CONTROL

BUFFER CONTROL PACKAGE

The RTOS system library provides a module which permits buffered line and sequential I/O transfers. The Buffer Control Package utilizes tasking concepts in addition to standard system I/O calls to fill or empty two or more buffers asynchronously (overlapped buffering) and, therefore, provide a constant supply of input or output data.

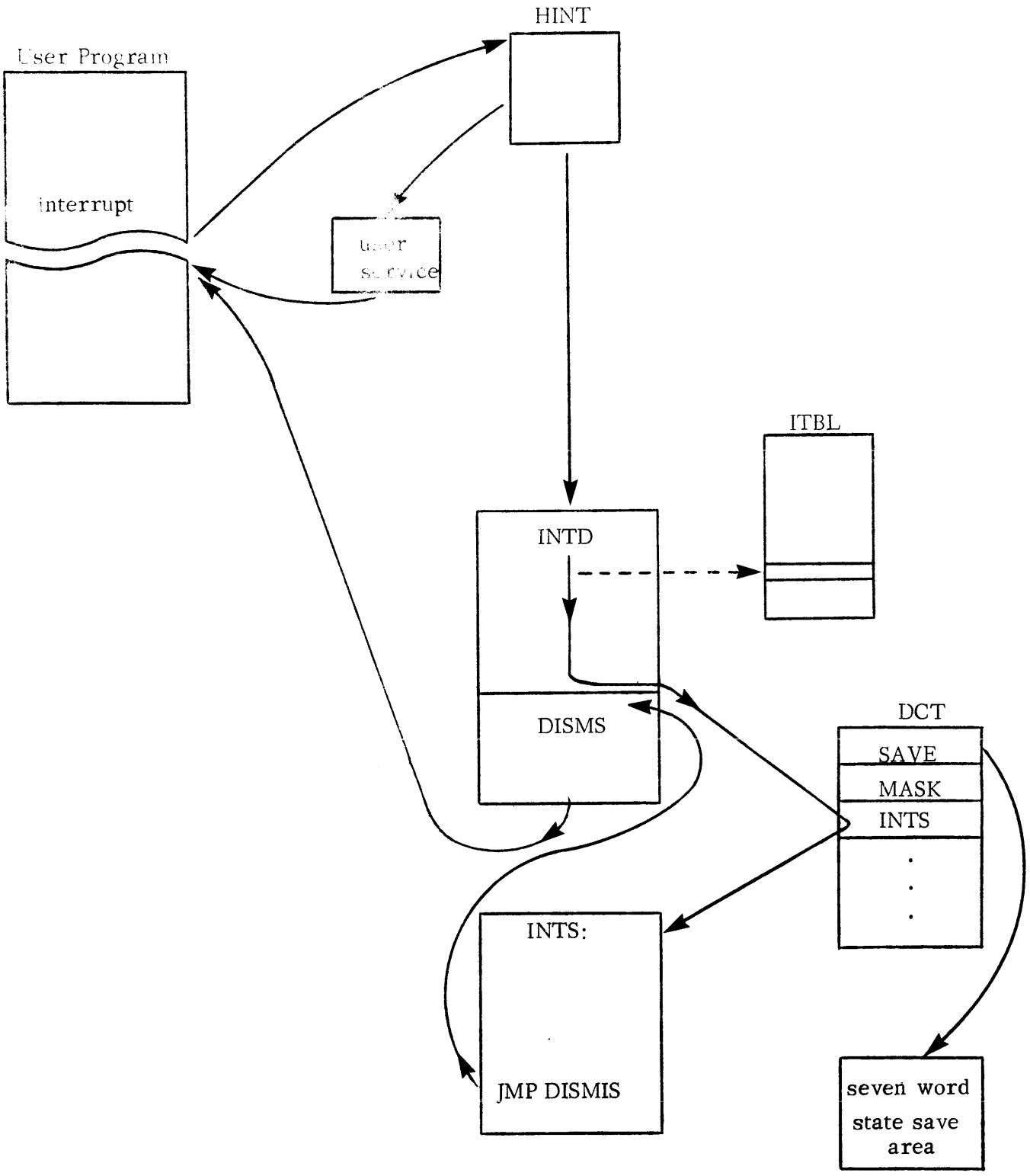
DISC FILE INPUT/OUTPUT CONTROL

RTOS supports contiguously organized disc files that are completely compatible with those available under RDOS. RDOS compatibility allows an RTOS system to collect data for later processing under a disc operating system. Names and sizes of these files are specified at system generation time.

CASSETTE/MAGNETIC TAPE I/O CONTROL

A method of free format input/output permits the reading or writing of data on a word-by-word basis to a cassette or magnetic tape unit. This mode provides users with the means of accessing data in variable size records (from 2 to 4096 words in length) within tape files.

Free format I/O commands also permit a tape reel to be spaced forward or backward 1 to 4096 records or to the start a new data file, and allow the reading of the transport status word.



FLOW OF CONTROL DURING INTERRUPT SERVICING

INTERRUPT SERVICING

When an interrupt is detected by the CPU, the currently executing program is suspended and control goes to the interrupt servicing portion of RTOS. RTOS considers interrupts as originating from three distinct kinds of devices: standard or system devices, user devices, and high priority devices. Thus when an interrupt is detected, RTOS first determines the category of the interrupt before proceding to service it.

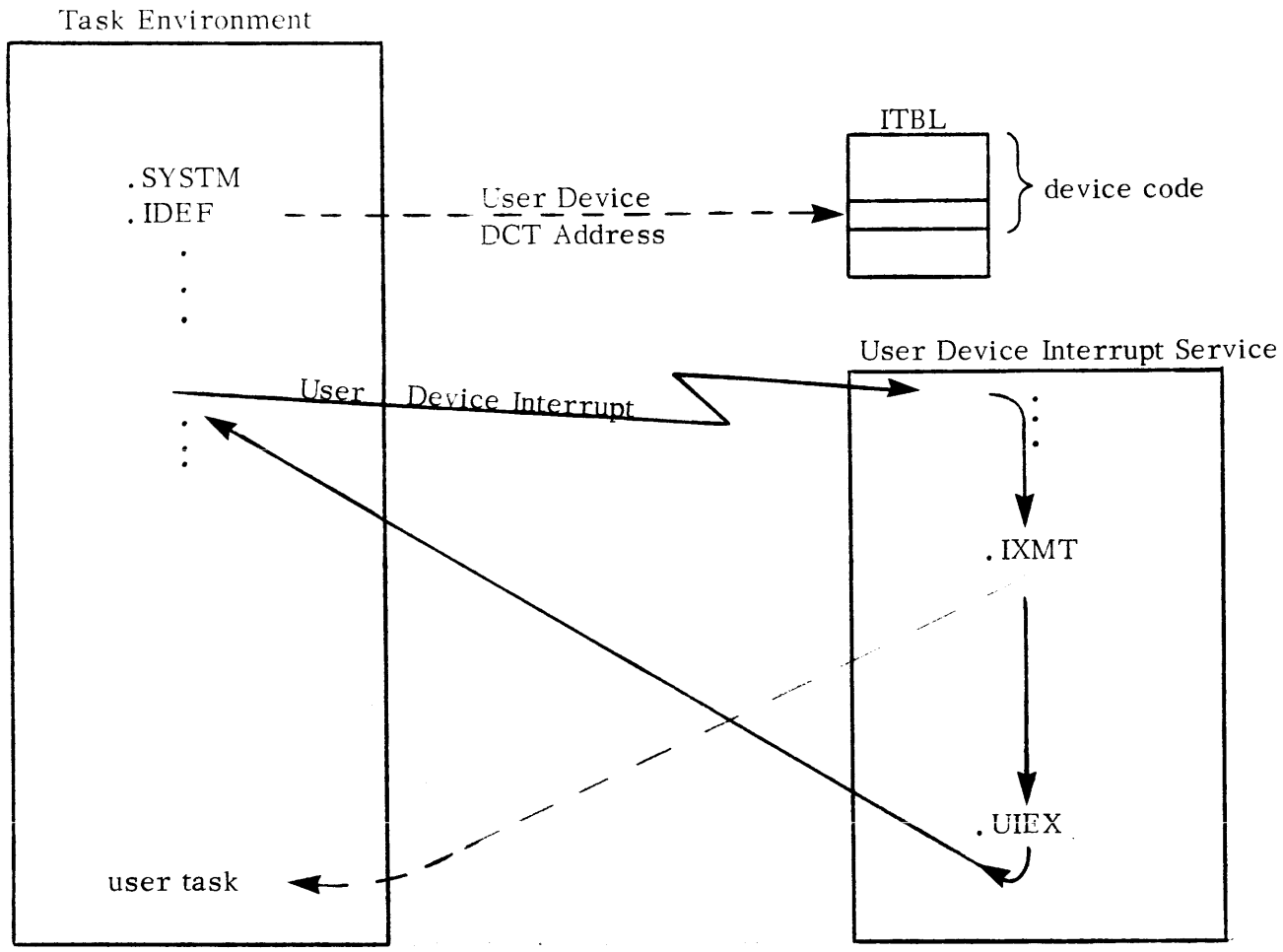
Standard devices are those devices like the disc and line printer for which RTOS provides interrupt service. Standard device support is extended by RTOS when the system is generated. RTOS also permits user devices to be announced at system generation time, and will treat interrupts received from such devices as though they were standard devices.

RTOS also provides a convenient method of providing interrupt service to non-standard devices at run time. These non-standard or user devices are not announced at the time the RTOS system is generated.

High priority interrupts, as the name implies, are interrupts generated by devices requiring the speediest interrupt service possible. Typical high priority devices are the power fail/auto restart option and the real time clock. RTOS permits users to specify other devices as being of high priority also. Such devices would require special interrupt service routines written by the user, and the names (and device codes) of these devices would be specified by the user at system generation time. The high priority interrupt routine, HINT, receives control when any interrupt occurs. This program determines whether or not the interrupt device is a high priority device. If it is, control branches to this device's service routine. High priority service routines are short and quickly executed so that service can be provided swiftly and control may be returned to the user program promptly.

If the interrupting device is not recognized by HINT, control is passed to the interrupt dispatch routine, INTD. This routine directs control to the appropriate service routine for all standard and user devices (i.e., all devices not listed in HINT). INTD directs control to the proper routine by using the device code of the interrupting device as an index into an interrupt branch table (ITBL). The entry in this table is the address of a device control table (DCT) associated with the servicing routine. Each standard device and user device has a DCT. The first three entries of each DCT are as follows:

Words	Mnemonic	Contents
0	DCTSV	Address of 7-word state save area.
1	DCTMS	Interrupt service mask.
2	DCTIS	Device interrupt service routine address.



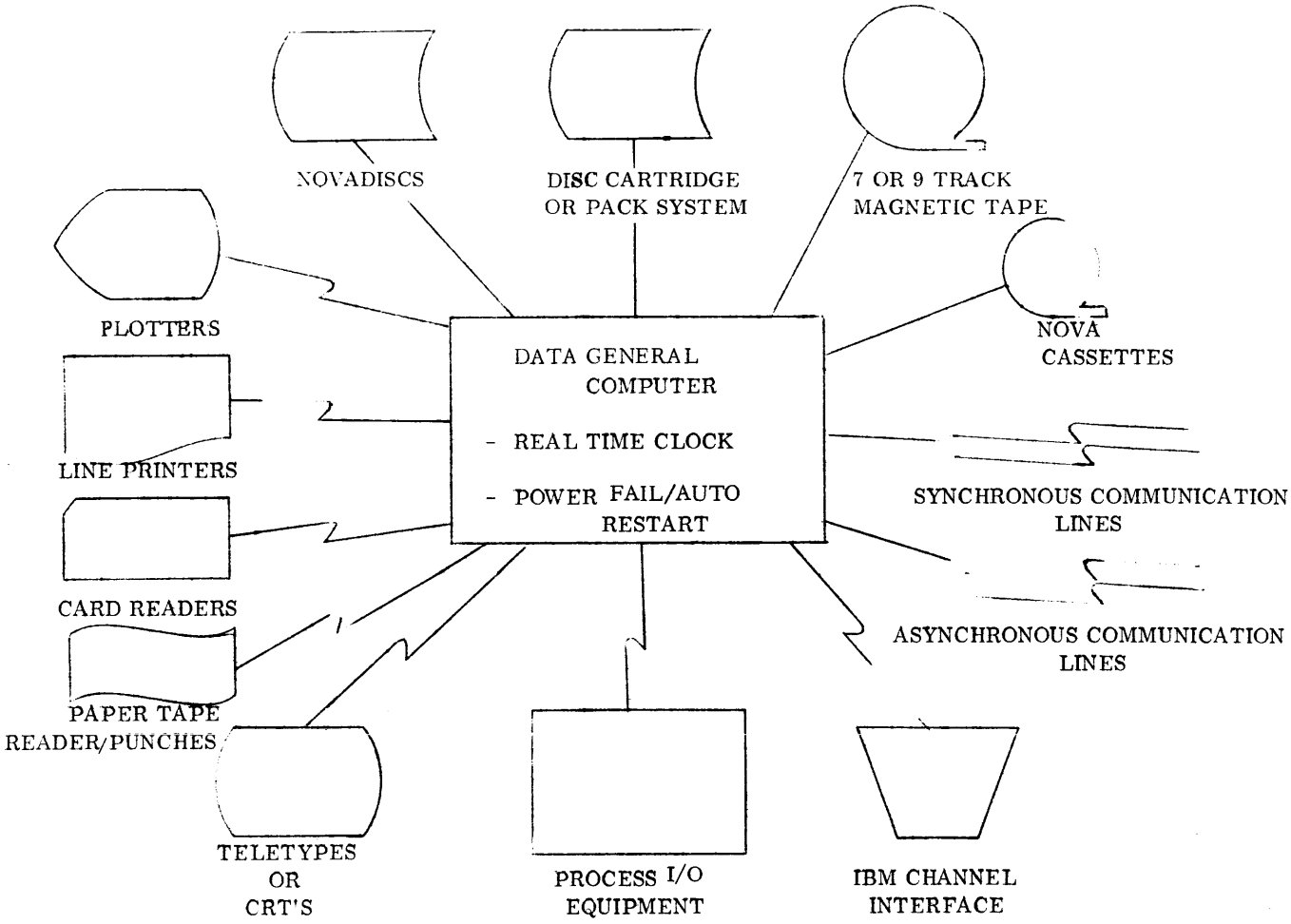
USER INTERRUPT CONTROL FLOW

*USER INTERRUPT PROCESSING

In addition to providing interrupt servicing for standard devices, RTOS provides a simple software interface for non-standard devices. This interface is provided through an abbreviated DCT (the first three entries of a standard DCT) which supplies the address of a 7-word state save area, the hardware interrupt mask to be set while servicing the user interrupt, and the address of the user interrupt servicing routine. The system stores the program counter, accumulators, carry, current hardware mask, etc. in the state save area before transferring control to the interrupt service routine. The interrupt service routine is written by the user, and it contains all program code necessary to process the interrupt.

A system call, .IDEF, is used to insert a pointer to a user DCT into the interrupt vector table (.ITBL), identifying the device to the system. To remove this entry from the table, the system call .IRMV is issued with the device code passed as a parameter.

The user interrupt service routine is considered to reside outside the task environment, since the task environment is held in suspension when any interrupt is detected by the system. Thus user drivers cannot issue either system or task calls, with the exception of three calls which have the format of task calls but are not routed through the task scheduler: .IXMT, .SMSK, and .UIEX. Call .SMSK permits the current interrupt mask to be modified, and .UIEX is issued to exit from the user routine and to return to the task environment. Call .IXMT provides the user routine with the facility of activating user tasks and transmitting messages to them. This is done by transferring a nonzero message from the user service routine to a user task via .IXMT. If the task has not issued a .REC for the message, the .IXMT call simply posts the message so that it can be retrieved when the task does issue the .REC call. If the user task is suspended and is awaiting the message, the .IXMT call will cause that task to be readied when control is returned from the service routine to the task environment.



RTOS HARDWARE CONFIGURATION

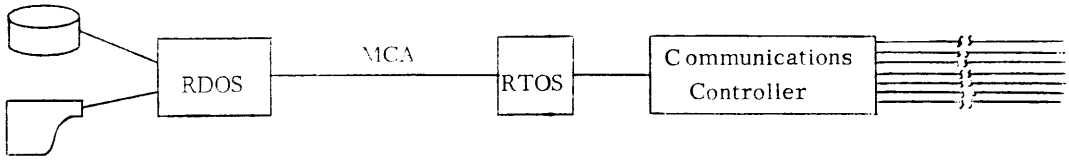
* MULTIPLE DEVICES AND UNITS

The Real Time Operating System (RTOS) is capable of supporting multiple-character and block-oriented devices. Some of the standard peripherals and controllers currently supported by RTOS on a single system are:

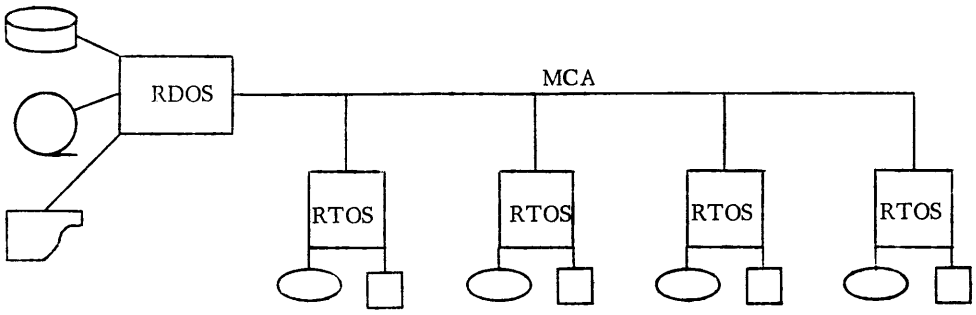
Novadiscs	-	up to two million words
Moving Head Discs	-	up to four units: Diablo cartridge or Century 111 or 114 disc packs
Magnetic Tapes	-	up to eight units on a single controller
Cassettes	-	up to eight units on a single controller
Synchronous Lines	-	either single (4074 or 4015) or multiple synchronous communication lines (4073)
Asynchronous Lines	-	either a single line (4029 asynchronous line adapter), up to 64 lines (4060 multiplexor) or up to 1024 lines (4100 multiline asynchronous controller).
Process I/O	-	analog to digital converters
	-	digital to analog converters
	-	digital input/output
Teletypes	-	up to three
Paper Tape Readers	-	up to two
Paper Tape Punches	-	up to two
Card Readers	-	up to two
Line Printers	-	up to two
Plotters	-	up to two (80 or 132 column)

Multiprocessor Communications Adapter

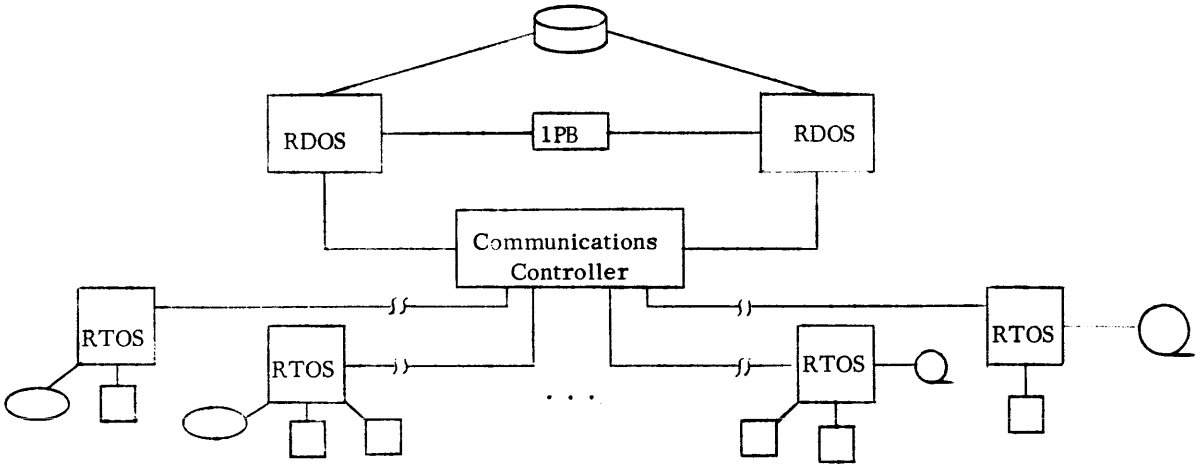
Power Fail/Automatic Restart



Data Acquisition and Analysis



Host/Slave Process Control



Dual Host Processors for Redundant Operations

* MULTIPROCESSOR SYSTEMS

There is more to building a multiprocessor system than stringing several processors together. Unless both the hardware and the software have been designed for use in a multiprocessor system, the system cannot operate with efficiency. Data General's philosophy of maintaining compatibility between operating systems, high level languages, utilities and bootstrap programs, and disc file structures lays the groundwork for building multiple processor systems with efficiency and economy.

Although endless varieties of multiprocessor configurations can be designed, in general each system will balance total system cost with system reliability. Processes which can tolerate occasional interruption due to power failure can be resumed by means of the power fail-auto restart option under both RTOS and RDOS. Processes where continuity of operation is critical can be continued with minimal delay in systems with redundant processors operating under RDOS. "Watch-dog" timer hardware permits one RDOS processor to check the operation of another RDOS processor every second. Failure by either processor to service its real time clock within a second's time causes an interrupt to be generated which alerts the working processor to the alarm condition. The working processor can then take control of peripheral equipment via an I/O bus switch and can examine a common disc data base to determine how the process should be continued.

Common multiprocessor systems will incorporate processors running under both RTOS and RDOS. A simple multiprocessor configuration might include one processor to act as a data concentrator: this data would then be passed on, via an MCA line, to a second processor which would reduce and analyze the data and print summary reports at periodic intervals. The processor performing the data concentration function could run under RTOS, and the other processor could run under RDOS.

A larger multiprocessor system could contain five processors: one host running under RDOS, and four satellite processors running under RTOS. Each satellite processor would perform a process control function, could display current control parameters on a CRT, and would send periodic data to the host via an MCA line.

A yet larger multiprocessor system might include two processors running redundantly under RDOS to insure continuity of operation, and up to 15 satellite processors communicating with the host pair via synchronous or asynchronous communications lines. In each of the illustrated systems, the RDOS processors could be run in a foreground/background mode to provide a means of on-line computational power or continuing software development while the host process was executed without interruption. Software development could be performed in either assembly language or in real time FORTRAN on either processor.

*****

* Support Literature

17-000001, Synchronous Communications Package. This application note describes in depth a general purpose software package that can be used to control the Synchronous Line Adapter, type 4074.

17-000003, Buffered I/O Package in RDOS/RTOS. This application note describes BFPKG, a module in RTOS1.LB that provides asynchronous data buffering in main memory for user programs. BFPKG can be used by both RTOS and RDOS.

17-000004, Remote Synchronous Terminal Control Program. This application note describes how the Remote Synchronous Terminal Control Program, RSTCP, allows a Data General computer with peripherals to be operated as a remote intelligent data terminal. RSTCP is supported by both RTOS and RDOS.

17-000005, Multiline Asynchronous Controller Software Package. This application note describes a general purpose subroutine package used to control the operation of a multiline asynchronous multiplexor. This multiplexor can be used with one or more Nova computers to control multiple asynchronous lines. The subroutine package can be run under RTOS or RDOS.

17-000006, User Device Driver Implementation in RTOS. This application note describes in depth the techniques required to add a device driver to RTOS on a user level or system level.

17-000008, IBM 360/370 Channel Controller Software Package.

17-000009, Dual Processor/Shared Disk Configurations.

17-000010, Workshop on Standardization of Industrial Computer Languages.

17-000011, Implementation of an On-line System in Real Time FORTRAN.

17-000012, FORTRAN IV Assembly Language Interface.

17-000013, Synchronous Controller Software Package. This application note describes a general purpose subroutine package used to control the operation of the multiline synchronous controller, type 4073.

17-000014, Asynchronous Multiplexor Software Package. This application note describes a general purpose subroutine package used to control the operation of the multiline asynchronous multiplexor, type 4060.

* Support Literature (Continued)

93-000018, Symbolic Text Editor Manual. This manual describes the use and operation of the symbolic text string editor under RDOS. This editor is needed to produce and correct source files for assembly, compilation, etc.

93-000040, Extended Assembler Manual. This user manual describes the use and operation of the Extended Relocatable Assembler under RDOS.

93-000044, Debug III User's Manual. This document explains the use of the symbolic debugger with RTOS; this debugger disables interrupts allowing the user to look at a dynamic real-time environment.

93-000053, FORTRAN IV User's Manual. This document describes DGC FORTRAN IV, including an exposition of its real time extensions.

93-000056, Real Time Operating System User's Manual. This manual is the primary document to be consulted by users of RTOS.

93-000068, FORTRAN IV Run Time Library User's Manual. This document describes the FORTRAN IV run time library routines in detail as well as methods for interfacing these routines to assembler language programs.

93-000074, Library File Editor Manual.

93-000075, Real Time Disk Operating System User's Manual. This is the primary document to be consulted by users of the Real Time Disc Operating System. It describes relationships between RDOS and RTOS.

93-000080, Extended Relocatable Loaders Manual.

93-000081, Macro Assembler Manual. This manual describes the RDOS Macro Assembler. This assembler's functions are a compatible superset of those provided by the Extended Relocatable Assembler.

93-000083, Introduction to the Real Time Disk Operating System.

93-000084, Octal Editor Manual. This manual describes an RDOS utility used to examine or modify RDOS disc file space.

93-000085, FORTRAN 5 User's Manual.

* Support Literature (Continued)

93-000092, Stand-alone Disk Editor Manual. This manual describes the use of a disc editor that can be used to examine or modify all disc space. A large portion of this manual is devoted to a discussion of directory structures of RDOS.

93-000096, FORTRAN 5 Run Time Library User's Manual. This manual describes the FORTRAN 5 run time library routines as well as methods for interfacing these routines to assembler language programs.

*****

Glossary of Terms

Bootstrap	A technique for loading the first few instructions of a routine into storage, then using these instructions to bring in the rest of the routine.
Device	A hardware component of the system with unique operational characteristics.
Device independence	The ability of a task to communicate with a device independent of the device's uniqueness.
Dormant	The state of a task that has not been initiated (made known to RTOS) or whose execution was terminated or completed.
Executing	The state of a task which has the highest priority of all tasks and is in control of the central processor unit.
File	A collection of related data (such as a disc or magnetic tape file) treated as a unit and addressable by an alphanumeric identifier.
Multitasking	The ability to support more than one active task within an address space.
Program	The contents of a complete address space.
Ready	The state of a task that is ready and available for execution but which is awaiting the execution of one or more higher priority tasks before it can gain control of the CPU.
Suspended	The state of a task which is awaiting the occurrence or completion of a system or task call or some other real-time event.
System Generation	A procedure used to customize an operating system to the available hardware and to the expected user application. RTOSGEN is the RTOS system generation program.

Glossary of Terms (Continued)

Task	A unique execution path within an address space.
Task calls	Requests to the Task Monitor to effect task state changes. This also causes the Task Monitor to pass control to the highest priority ready task.
Task Monitor	A collection of subroutine modules that schedule and manage calls from user tasks. Task Monitor is equivalent to Task Scheduler.
Task state	The status of a task in the RTOS environment. A task can exist in one of four states: dormant, ready, suspended, or executing.
TCB (Task Control Block)	A block of memory containing the task's state variable and control information.
USP (User Stack Pointer)	A page zero location that is always preserved when control is passed from one task to another. USP is one of the system state variables (like the program counter and the accumulators) and makes re-entrant coding possible.
UST (User Status Table)	A 30-word area (starting at location 400) that records all information pertinent to the execution of the entire RTOS program.

*****