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

PROGRAM

Real-Time Operating System
Reference Manual

TAPES

Relocatable Binary: 089-000061
Relocatable Binary: 089-000062
Relocatable Binary: 089-000063
Relocatable Binary: 089-000064
Relocatable Binary: 089-000065
Relocatable Binary: 089-000066
Relocatable Binary: 089-000067
Relocatable Binary: 089-000068
Relocatable Binary: 089-000069

ABSTRACT

The Real-Time Operating System (RTOS) for the Nova/Supernova computer family consists primarily of a small, general-purpose multi-programming monitor designed to control a wide variety of real-time input/output devices. User programs are relieved from the details of I/O timing, data buffering, priority handling, and task scheduling. In addition, they are provided with a parallel processing capability plus inter-task communication and synchronization facilities. Communication with the monitor takes place through a small set of meta-instructions (consisting primarily of machine-language subroutine calls). The system, which is entirely core-resident, is highly modular and largely reentrant, allowing for the straightforward addition of special device handlers.

093-000056-02

REAL TIME OPERATING SYSTEM

Table of Contents

APPENDICES

FIGURES

CHAPTER 1

USERS GUIDE

1.1 INTRODUCTION

The Real-Time Operating System (RTOS) for the NOVA/SUPERNOVA computer family consists primarily of a small, general-purpose multi-programming monitor designed to control a wide variety of real-time input/output devices. User programs are relieved from the details of I/O timing, data buffering, priority handling, and task scheduling. In addition, they are provided with a parallel processing capability plus inter-task communication and synchronization facilities. Communication with the monitor takes place through a small set of meta-instructions (consisting primarily of machine-language subroutine calls). The system, which is entirely core-resident, is highly modular and largely reentrant, allowing for the straightforward addition of special device handlers.

RTOS is designed specifically for the handling of multiple user tasks. These tasks are created by means of one of the meta-instructions, and once initiated may be terminated at any time. A large number of common processing situations lend themselves admirably to this sort of operational control philosophy. Simple examples include the reading or writing of a block of data while simultaneously performing some other (possibly unrelated) operation, listening for input from several devices at the same time, shared device use by multiple tasks, sophisticated communications problems, etc. As will be seen from the meta-instructions available, RTOS is a small, rudimentary time-sharing system. Its parallel processing capability is, however, discussed here more as a means for writing control programs for a wide variety of I/O devices rather than as a way to run several computations in parallel, although the latter feature is clearly a part of the system.

1.2 RTOS TASK STATES

As a "TASK" is the basic logical unit which is controlled by RTOS, it is necessary at this time to describe the concept of a TASK, and define some of its operational states during existence. Essentially, a TASK is a logically complete program segment, operating at a specified priority level, whose execution may proceed simultaneously with, and independently of other TASKs (although communication and synchronization between TASKs is permitted and provided for in RTOS). Due to the serial nature of the computer, TASKs which appear to execute in parallel are in actuality executed in short (serial) segments. It is necessary, then for RTOS to maintain certain status information (primarily active registers) concerning all TASKs which are not currently in a state of execution. This information is retained within an information structure called the "TASK control block" (TCB), which is a seven word block of storage structured as follows:

USAGE	WORD
RTOS LINKAGE WORD	Ø
CARRY + PRIORITY	1
ACØ STORAGE	2
AC1 STORAGE	3
AC2 STORAGE	4
AC3 STORAGE	5
PC STORAGE	6

Figure 1-1 TCB FORMAT

The maximum number of TASKs which may be supported in any RTOS configuration is defined at RTOS assembly time (by specifying a value for the "JOBS" parameter). At any given time, each of these TASKs will exist in one of four states which are described in the following sections.

1.2.1 EXECUTING STATE

A TASK is executing when the central processing unit (CPU) has been restored to the state specified in the TASK's TCB. Note that when a TASK is in the executing state, it has control of the CPU and no TCB is required.

1.2.2 PENDING TASK

A TASK is pending when it is ready and available for execution. A TASK becomes pending if and only if:

a) it is being initiated for the first time (with the .FORK command),

or b) an RTOS meta-instruction is executed and a TASK of equal or higher priority is pending,

or c) a teletype break occurs (see description of the .BRK command),

or d) the real-time operation it was awaiting has occured/been completed.

1.2.3 SUSPENDED STATE

A TASK is suspended whenever it must await the occurrence or completion of some real-time operation. More specifically, this condition may exist if and only if:

a) an I/O operation (.IOX) has been initiated by the TASK and it is not yet complete,

or b) the TASK has given a .RCV request and is awaiting a corresponding .XMIT fron another TASK,

or c) the TASK has given a .XMIT request with the "@" option and is awaiting the corresponding .RCV from another TASK,

or d) the TASK has given a .WAIT request and is awaiting the passing of the specified number of clock cycles,

or e) the TASK has given a .BRK request and is awaiting the occurrence of the specified break character on a teletype.

1.2.4 DORMANT STATE

A TASK is dormant if it is neither suspended, pending, or executing. This situation exists if and only if:

a) the TASK has not yet been created in the system (with the .FORK command),

or b) the TASK has been terminated, either due to the execution of a .QUIT command or the break character being received or a .RCV being issued on a previously .RCV activated channel.

The various modes of transition between the four TASK states in RTOS are illustrated in Figure 1-2.

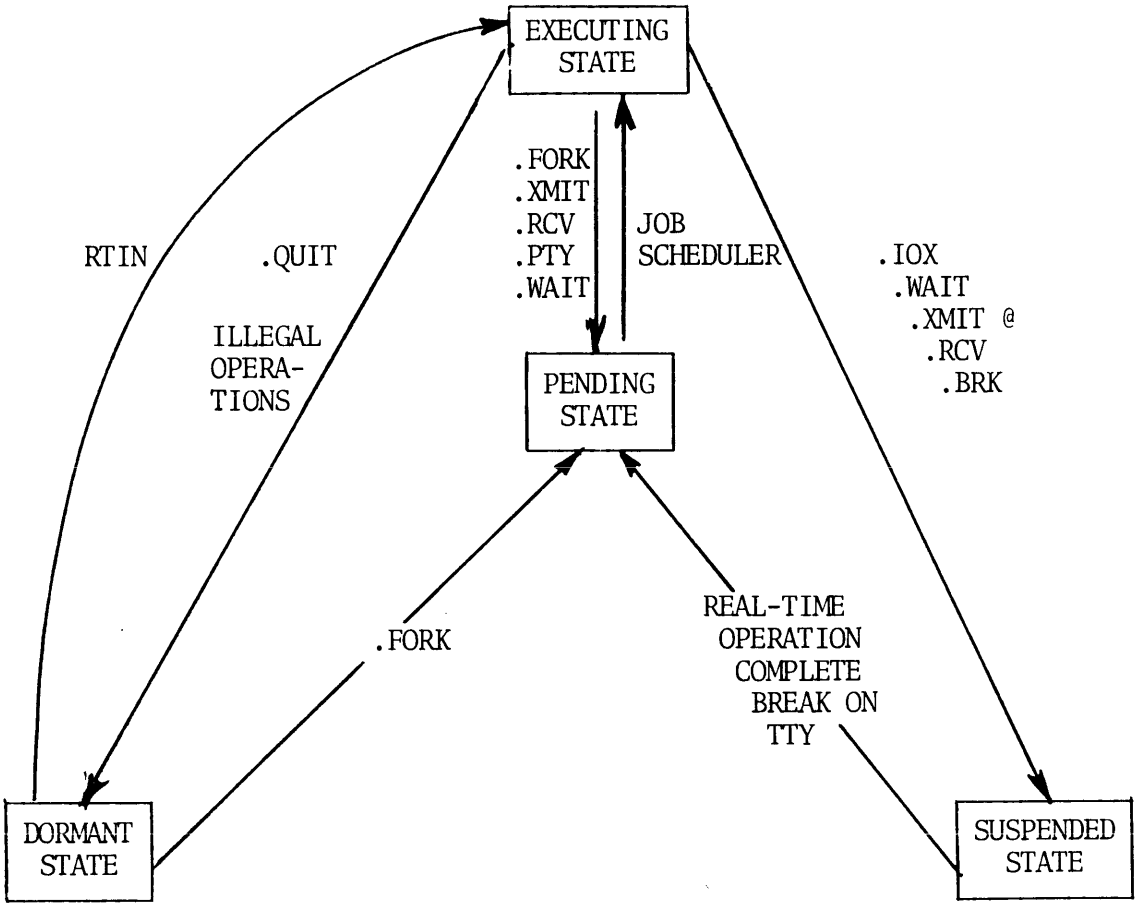


Figure 1-2 TASK STATE TRANSITION

1.3 RTOS META-INSTRUCTIONS

Communication between user TASKs and RTOS is effected by means of eight "meta-instructions". These meta-instructions, mostly subroutine calls, are declared as entry points (.ENT) to RTOS (which is supplied as a relocatable package, and thus must be declared as externals (.EXTN) to

the user program. Alternatively, an absolute binary tape of RTOS could be prepared for special situations, in which case the meta-instructions would be user-defined as transfers through the appropriate address vector on page zero.

It should be noted that all meta-instructions return control by means of the RTOS TASK scheduler, which will place the current TASK in the pending state if any TASKs of equal or higher priority are pending when the meta-instruction is executed. In addition, the TASK scheduler is activated whenever a TASK is transferred from the suspended to the pending state (i.e. - when a real-time operation has been completed). This technique is employed, rather than using the real-time clock to generate "time slices", in an attempt to reduce overhead in the system as well as provide some user control over the scheduling operation (which is desireable in a control environment). A true, "time-slicing" mode of operation is available to the RTOS user, however, and will be discussed under the description of the .WAIT instruction.

1.3.1 .IOX INSTRUCTION

```
.IOX
<logical device #>
<device control word>
<first data item pointer>
<data item count>
<error routine address>
<normal return>
```

This is the standard I/O operation in RTOS, initiating an activity on the specified device. If the device is currently busy (i.e., servicing an .IOX request from another TASK), execution of the requested operation will be postponed until the device becomes free. Once this execution begins, absolute control of the device is guaranteed to the TASK until the .IOX operation has terminated. In all cases, the execution of an .IOX causes the current TASK to be placed in the suspended state, making a transition to the pending state only when the requested function has terminated. Note that this may effectively be avoided, if

desired, by the logical structure created by the following command sequence:

```
┌─.FORK    ; create parallel process
│ .IOX     ; perform I/O operation
│ .QUIT    ; terminate TASK when I/O complete
└─→---     ; parallel process to avoid hang-up
```

The use of parallel processes (i.e., multi-TASK's) allows a TASK to communicate with a given device without having to concern itself with whatever other task may be using it. For example, a number of TASKs could "simultaneously" be reading some control variable through an A/D converter input. Needless to say, multiple TASKs generating output on "serial" devices such as the paper tape punch must take care to include identifying information within the output data block.

The device type on which the I/O operation will be performed is indicated by the "logical device number", the first parameter of the .IOX command. Logical device numbers for each class of devices (e.g., Ø = teletype, 1 = high speed paper tape reader, 3 = line printer) are specified in the individual device handler descriptions which are included in an appendix to this manual. It should be noted that device handlers may be capable of servicing multiple units of the same device type. In these cases, the specific unit is indicated by a device unit number which forms part of the device control word. The teletype is an example of a device whose handler can service multiple units.

The second parameter of the .IOX command is called the device control word and is used to specify all device-dependent options. In this section, the teletype device control word will be shown for illustrative purposes; other device control words are specifically described in the appendix on Peripheral Support.

An attempt has been made to keep these device control words as compatible as possible, so much of what is given regarding the teletype will hold true for other devices as well. The teletype device control word is shown in Figure 1-3.

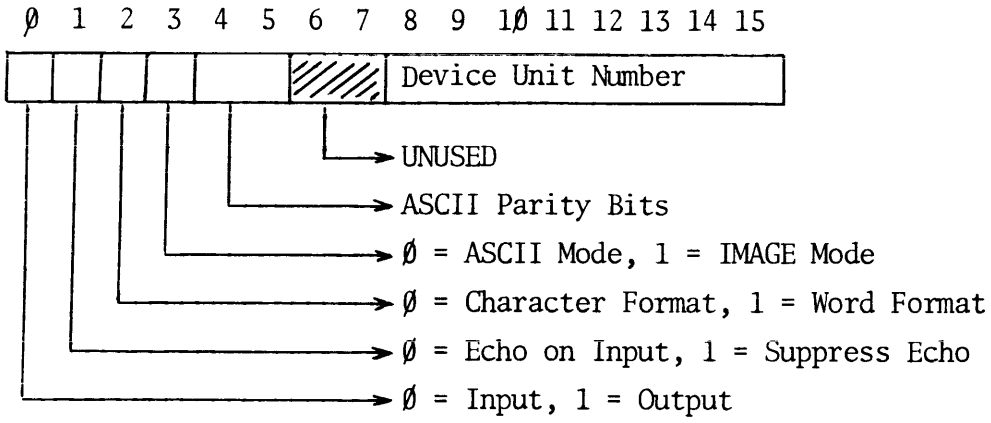


Figure 1-3 Teletype Device Control Word Format

The next two parameters of the .IOX command give the starting address of the data list and the data item count. These two parameters are either a byte pointer/byte count or a word pointer/word count depending on the format specified in the device control word.

The last .IOX parameter is the address of an error program. If an error occurs during the processing of an .IOX meta-instruction, the error return address parameter is used as the return PC when the TASK is placed in the pending state.

1.3.2 .FORK INSTRUCTION

```
.FORK
<new task priority>
<new task address>
<return>
```

The .FORK meta-instruction is the basic mechanism for the creation of parallel processes (i.e., multiple TASKs) in RTOS. Its execution causes the creation of a new TASK (with its associated TCB) at a specified priority level. Priorities in the range 1-$255_{1Ø}$ may be used ($255_{1Ø}$ being the lowest priority); specifying a level of Ø causes the new TASK to be created with the same priority as the executing TASK. Both TASKs are placed in the pending state when the .FORK is executed, and the system scheduler determines which TASK will next be placed in the executing state. From this point onward, the two TASKs exist as separate

entities in the system; no structure or hierarchy is remembered. Much of the power of RTOS is inherent in the .FORK mechanism.

Note that if both the executing TASK and the new TASK created by the .FORK are of the same priority (new task priority = Ø in .FORK call) the previously executing TASK will be re-scheduled before the new one. This convention allows for the convenient initiation of I/O activities without suspending the corresponding computation (as illustrated in the .IOX description).

1.3.3 .QUIT INSTRUCTION

```
.QUIT
```

This meta-instruction is used to terminate the execution of the current (executing) TASK. The TASK is placed in the dormant state and will not become pending unless re-created by a .FORK command. .QUIT is implemented simply as an entry to the TASK scheduler, preserving none of the current TASK information.

1.3.4 .PTY INSTRUCTION

```
.PTY
<new priority level>
<return>
```

The .PTY meta-instruction is used to dynamically alter the priority of the current executing TASK. Priorities within the range Ø-$255_{1Ø}$ are permitted, being specified by the eight least significant bits of the new priority level parameter. Level $255_{1Ø}$ (377_8) is the lowest priority.

Note that level Ø, the highest RTOS priority level, may only be specified by a .PTY command (i.e., not by a .FORK operation). A number of special system operations take place at this level and the user should exercise discretion regarding attempts to run TASKs at this priority. System integrity will be maintained but any "compute-bound" operation at level zero will cause a serious degradation in I/O operating speeds.

1.3.5 .WAIT INSTRUCTION

```
.WAIT
<# of clock cycles>
<return>
```

This meta-instruction is used to delay the execution of the current TASK for a specified time interval. The executing TASK is placed in the suspended state for the indicated number of clock cycles, following which it is transferred to the pending state, which makes it available for scheduling. Specifying a zero or a negative number of clock cycles as the waiting time causes the current TASK to be immediately placed in the pending state, allowing the scheduling of any other TASKs of the same priority. This feature is provided as a convenient technique to force rescheduling.

The duration of a system clock cycle is set at 1Ø milliseconds in the distributed version of RTOS. Other frequencies are available (See the appendix on System Generation) and may be in use at a given system installation since it appears on the parameter tape.

It should be noted that the real-time clock interrupt routine which is used by RTOS does not directly call the scheduler (although it may do so indirectly by virtue of the fact that a TASK which has been suspended due to a .WAIT operation will cause rescheduling when it becomes active again). This means that if a number of "compute-bound" TASKs are active at the same priority level, the current one will remain running unless it executes a meta-instruction or a higher priority TASK forces rescheduling. This potentially undesireable (in some cases) situation would be avoided in a true "time-slicing" environment, which may be created by having a time slice clock task which performs the following instruction sequence:

```
.PTY        ;Priority set to highest or could be
<Ø>         ;set to one immediately below which
.WAIT       ;time-sharing is to take place
<n>         ;The time slice duration
JMP .-2
```

The implementation of the time slice RTOS system assures that a compute bound task will not lock out the activities of other tasks of the same priority. The number of real time clock interrupts (n) that must occur before the scheduler is entered is set by the user. Care must be exercised when choosing this time slice interval because it affects the system overhead.

1.3.6 .XMIT INSTRUCTION

```
.XMIT                          .XMIT
<channel #>      or            <@ channel #>
<return>                       <return>
```

This is the first of a pair of complementary meta-instructions which are provided for the purposes of TASK synchronization and inter-TASK communication. The synchronization is performed through a number of transmit/receive "channels", the available number being defined at RTOS assembly time (nominally eight in the distributed system, with zero being the first channel number).

The .XMIT command causes transmission of a "synchronization signal" over the specified channel. If an "@" sign is present in the channel number argument, the TASK will be placed in the suspended state until the .RCV signal has been received. Otherwise, the TASK will be allowed to continue. In either case, the contents of ACØ at the time of the .XMIT request are retained by RTOS as a single word "message" to the receiving TASK (this "message" could, of course, be the address of a more lengthy message stored somewhere in core memory).

It is intended that there be a one to one correspondence between .XMIT and .RCV requests. An .XMIT to an illegal channel number functions as a .QUIT. Second and subsequent .XMIT operations on the same channel (before the corresponding .RCV is executed) act as no-operations, a feature which may be used to ascertain first occurrences of specific conditions, critical timing situations, etc.

1.3.7 .RCV INSTRUCTION

```
.RCV
<channel #>
<return>
```

This meta-instruction forms the second half of the synchronization/ communication facility in RTOS. The .RCV command enables the reception of a "synchronization signal" sent by a .XMIT over the specified channel. Upon executing the command, the current TASK is placed in the suspended state until the signal is received, at which time it will be made pending and become available for execution again. Needless to say, if the signal (sent by .XMIT) had already been transmitted (and was therefore awaiting reception), the TASK issuing the .RCV would not enter the suspended state, but would be immediately available for scheduling. In all cases, when the TASK resumes execution AC3 will contain a copy of the "message" sent by the corresponding .XMIT operation.

Specifying either a negative or an illegal channel number in the .RCV request will be handled like a .QUIT. Second and following requests to a channel activated by a previous .RCV command will also cause a .QUIT operation.

It should be clear that the .XMIT/.RCV meta-instruction pair may be used for both inter-task communication (via the "message" word) and synchronization purposes. Note that a "join" operation as shown in Figure 1-4, logically complementing the .FORK instruction, is implemented simply by the use of a .XMIT/.RCV pair, the .XMIT being followed by a .QUIT.

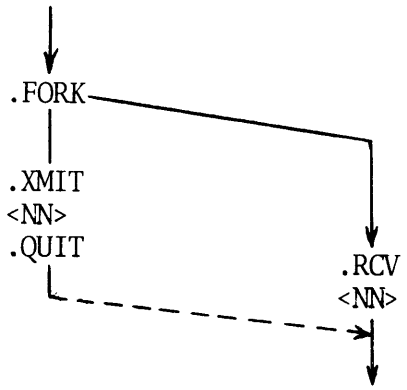


Figure 1-4 Inter-TASK Communication

1.3.8 .BRK INSTRUCTION

```
.BRK
<character code (ASCII)>
<return>
```

The .BRK meta-instruction is a special-purpose command which allows the input of a specified ASCII character to interrupt the normal operation of RTOS tasks. Any number of teletypes or teletype-like devices in the system may be connected to this facility. It is an assembly time parameter with the last word of the DUCB being set negative if the break feature is disabled and set equal to the teletype unit number if the break is enabled.

The TASK issuing the .BRK request is placed in the suspended state until the specified break character is typed on one of the enabled teletypes. At this time, any operation on this teletype will be terminated, suspended TASKs awaiting I/O on this teletype will be made dormant, the break TASK will be returned to the pending state, and the scheduler will be called. When the break TASK begins executing, AC3 will contain the unit number of the teletype which initiated the break request.

Additional .BRK requests, result in the previous .BRK request being terminated and an error message being sent back to the user in AC3. Its value is dependent on the value of the new break character. Possible values for AC3 are as follows:

AC3	-1	Break request terminated by another legal request
AC3	-2	Break request terminated by a new .BRK request with a negative break character.

The new .BRK request may also return back to the user TASK with AC3 set to -3. This indicates that the break character in the call was negative.

Note that the .BRK instruction is not intended to function as a general-purpose meta-command. Its use is primarily intended for and should be restricted to, the operation of a keyboard-oriented executive, or monitor TASK running at the user level.

CHAPTER 2

RTOS IMPLEMENTATION DETAILS

The primary function of RTOS is the supervisory control of four basic operations: TASK state transition, TASK synchronization/communication, real-time clock servicing, and input/output device servicing. Each of these areas of interest will be discussed separately.

2.1 TASK STATE TRANSITION

2.1.1 EXECUTING TASK

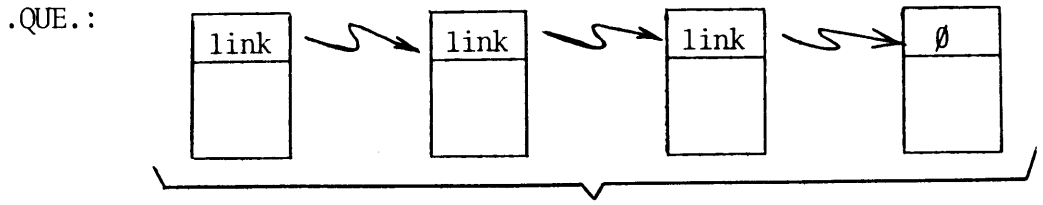
An executing TASK, by definition, has control of the central processing unit (CPU). The only parameter which RTOS must necessarily retain concerning such a TASK is its current priority. The variable location '.CPTY' always contains the priority of the current executing TASK.

When a TASK in RTOS is in the executing state, the CPU is said to be in USER MODE. Otherwise (i.e., RTOS is engaged in some system function such as TASK scheduling), the CPU is said to be in SYSTEM MODE. Due to the random nature of interrupts in a real-time environment, plus the desireability of reducing interrupt latency as much as possible, it was not feasible to make all system functions in RTOS fully reentrant. Instead, a variable location ('.SYS.') is used as a switch to denote USER MODE ('.SYS.' = $\emptyset$) or SYSTEM MODE ('.SYS.' $\neq \emptyset$). This technique protects against illegal reentrancy while still allowing for optimal priority scheduling.

2.1.2 PENDING TASK

Two queue structures are maintained to facilitate the manipulation of TASKs in the pending state. Of primary concern is the "pending TCB

queue", which is maintained by the TASK scheduler. TCBs contained in this queue are linked in order of decreasing priority (i.e., increasing numerical value). Location '.QUE.' contains the address of the first TCB in the chain, which "points" to the next in turn, and so on to the end of the queue. The final TCB entry has a zero pointer value to mark the end of the chain. The pending TCB queue, then, is structured as follows:



PENDING TCB's

The nature of a real-time environment dictates that suspended TASKs may become active at any time. Unfortunately, the routine to enter a TCB into the pending TCB queue cannot be reentrant, as correct chaining must be assumed at all times. Therefore, the (non-reentrant) TASK scheduler is given the unique right to insert TCBs into this queue. TCBs which dynamically become pending are reentrantly pushed into a special stack ('.JSTK') maintained for this purpose (this stack is pointed to by '.JPNT'). Each time the scheduler is called, it checks to see if any new TASKs have become pending (i.e., are in '.JSTK'). If so, the associated TCBs are entered into the pending TCB queue prior to raising to the executing state the highest priority pending TASK.

2.1.3 SUSPENDED TASK

Pointers to the TCBs for suspended TASKs are maintained within the various routines in RTOS which service real-time operations (i.e., routines handling the operations which caused the TASKs to become suspended). More detailed descriptions of these specific structures are covered elsewhere in this document.

When RTOS has determined that a suspended TASK can be made pending, the associated TCB is pushed into '.JSTK.' stack. Subsequently, this TCB will be entered into the pending TCB queue the next time the scheduler is called, and will be executed according to its relative priority.

2.1.4 DORMANT TASK

A dormant task is by definition unknown to RTOS, and therefore no information need be maintained concerning it. The .QUIT meta-instruction, which causes an executing TASK to become dormant, is merely a call to the system scheduler.

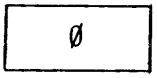
2.1.5 NULL TASK

When the RTOS TASK scheduler determines that the pending TASK queue is empty, it initiates a null TASK. This TASK is started by clearing location zero and the carry bit, restoring the system to USER MODE, and jumping to location zero. The null TASK will continue executing a "JMP Ø" in location Ø until a hardware interrupt causes a user TASK to be made pending as described above. When the computer is in the null TASK only the RUN, ION, and FETCH lights of the console will be lighted.

2.2 INTER-TASK COMMUNICATION

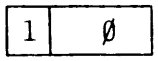
Two tables of length 'CHAN' are maintained by RTOS to handle .XMIT/.RCV inter-TASK communication. 'CHAN. is equal to the number of available .XMIT/.RCV channels and is set by the user at system generation.

The first is the channel status table ('.CST.'), containing one entry per channel. This entry is in one of four different formats:



-- zero entry indicates channel inactive

-- indicates that a .RCV has been requested on the channel. 'ADDR' is the TCB address of the associated suspended .RCV TASK.



-- indicates that a .XMIT has been sent to the channel with no suspension of the TASK

-- indicates that a .XMIT has been sent to the channel, and the TASK is awaiting a corresponding .RCV request. 'ADDR' is the TCB address of the suspended .XMIT TASK.

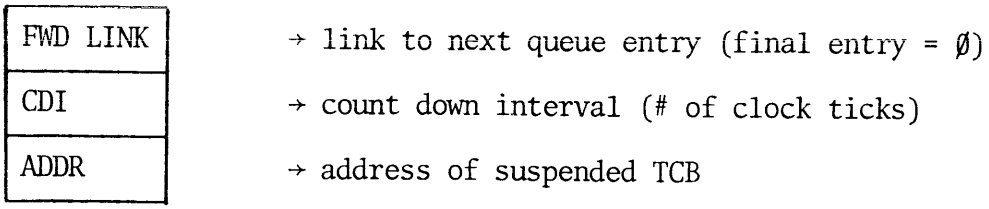
The second is the channel message table ('CMT'), also containing one entry per channel. When a .XMIT is initiated on a given channel, the 16-bit synchronization message (i.e., the contents of ACØ at the time of the .XMIT) is stored in the relevant channel entry in the message table 'CMT'.

2.3 REAL-TIME CLOCK SERVICING

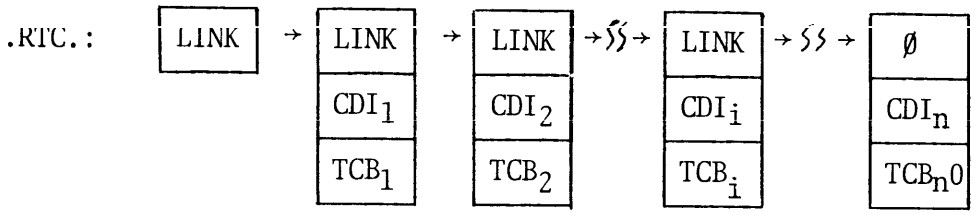
The real-time clock supplied with the Nova family of computers is capable of being operated at four different frequencies, one of which is selected as the standard RTOS clock time. This is done at RTOS assembly time, by specifying an appropriate value for 'FREQ'; it may also be accomplished during system run-time by modifying the 'FREQ' entry in the device control block for the real-time clock - '.CUCB+3.'. Each "tick" of the clock is used to decrement an appropriate count down interval value, which, when its value reaches zero, allows the suspended TASK to be made pending.

In order to handle multiple simultaneous .WAIT requests, a "clock count down interval queue", '.CLK.', is maintained, structured in a similar fashion to the pending TCB queue. Each queue entry contains three

words, defined as follows:



Location '.RTC.' in the clock service routine contains the address of the first entry in the interval queue, thereby creating the following structure:



Each count down interval value is, in reality, an incremental count down value. In other words, the total elapsed time before the suspended TASK represented by TCB_i is made pending is given by x clock "ticks", where,

$$x = CDI_1 + CDI_2 + CDI_3 + ... + CDI_i$$

The routine to handle the clock count down interval queue, much as in the case of the pending TCB queue routine, can not be reentrant, as the queue requires protection from multiple simultaneous accesses. However, the real-time clock must be allowed to run continuously as long as the interval queue is not empty in order to prevent error accumulation in multiple-interval timing situations. In order to alleviate these problems, an "overcount" state has been provided in the clock service routine, operating as follows: should RTOS (i.e., the .WAIT service routine) be accessing the queue when the interrupt service routine also desires access, the overcount state is entered and the clock continues to run. When the interval queue again becomes available, additional counts are provided according to the number of overcount

cycles which occurred thereby maintaining accuracy with minimal processing overhead.

2.4 INPUT/OUTPUT SERVICING

All user I/O is accomplished by a request to RTOS using the .IOX command. The .IOX section of RTOS is responsible for allocating the use of all standard devices of the system. It provides for referencing all I/O devices by logical unit numbers instead of physical unit numbers. The assignment of the logical unit numbers is made at system generation time (see Appendix A).

Each I/O device on the system has a handler associated with it. Control is transferred to the handler from the .IOX processor. If the required device is already busy, the request is stacked for later execution. If the device is not busy, the device is activated to input or output the first character or word. The task requesting the I/O operation is suspended until the transmission is completed.

If the request is illegal for some reason (e.g., illegal logical unit number, negative word count, etc.), control is returned to the user program at the error return address.

Device handlers with two exceptions (real-time clock and teletype), are written as separate, relocatable subroutines which are linked to RTOS at load time by the relocatable loader. The two exceptions are made for the following reasons:

a) the real-time clock is not treated as a standard peripheral (it communicates with .WAIT rather than .IOX) and is not an optional device.

b) the teletype is assumed to be present in most systems and its handler is general-purpose and flexible enough to provide major services for almost all other device handlers.

Each I/O handler performs operations and functions on a specific device only. This means that a handler must be provided for each hardware

device in the computer system. Handlers have been written and are supplied within the basic RTOS package for the high speed paper tape reader and punch, the incremental XY plotter, and the analog to digital converter. Handlers for the card reader, line printer, fixed head disk, magnetic tape, data communications system, and other devices interfaced to the Nova computer will be made available as they are developed.

CHAPTER 3

DEVICE HANDLER IMPLEMENTATION

3.1 .IOX INSTRUCTION

The .IOX meta-instruction of RTOS provides the linkage between user TASKs and the individual device handlers. The calling sequence has been briefly described in section 1.3.1.

3.1.1 LOGICAL DEVICE NUMBER

The device type on which the I/O operation will be performed is indicated in the first parameter of the .IOX calling sequence and referred to as the logical device number. Logical device numbers for each class of device on the system have been arbitrarily assigned in the released version of RTOS as follows:

Ø	Teletype keyboard and printer
1	High Speed paper tape reader
2	High Speed paper tape punch
3	Line printer
4	Incremental plotter
5	Card reader
6	Fixed head disk
7	Analog to digital converter
8	Magnetic tape
9	Data communications multiplexer

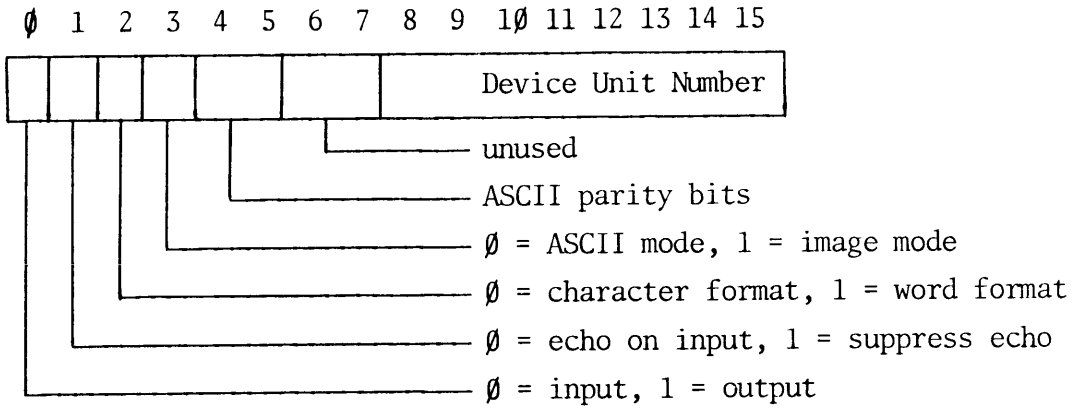
The assignment of logical device numbers may be changed at system generation time by the user modifying the order of the table of device handler addresses starting at location 'HANTB'.

If the logical device number is outside the above range (negative or greater than 9_{10}) or it specifies a device that has nct been declared as part of the hardware configuration at system generation time, an illegal device number error will occur. The error return will be made with AC3 = 0.

3.1.2 DEVICE CONTROL WORD

The device control word format will be discussed bit by bit in this section. This control word is used to specify all the device-dependent options to the device handler. In this description, the teletype device control word will be used for illustrative purposes; other device control words are specifically described with the device handler description in the appendix on Peripheral Support. An attempt has been made to keep these control words compatible, so much of what is said regarding the teletype will hold true for other devices as well.

The teletype device control word is structured as follows:



The following conventions have been adopted in RTOS:

Bit Ø Input / Output Mode Switch

0 = input mode

1 = output mode

(Used only where device has both input and output mode like a teletype. Paper tape reader is always in input mode and this bit is not tested.)

Bit 1 Echo Indicator Switch

0 = echo on input

1 = suppress echo

(Used only for teletype where characters typed on the keyboard may be echoed on the teleprinter.)

Bit 2 Data Format Switch

Ø = character format

1 = word format

Character format is available for I/O of data items occupying 8 bits or less. The items are stored/ retrieved in byte format (two bytes per word - right half first) and the data item pointer and data item count are interpreted as the byte address and byte count respectively.

Word format is available for I/O of data items occupying 16 bits or less. The items are stored/retrieved in right justified form from successive memory locations (unused bits are cleared by the device on input and ignored on output). The data item pointer and count are interpreted as a word address and word count respectively.

Bit 3 Transmission Mode Indicator

Ø = ASCII mode

1 = Image mode

In ASCII mode 8 bit characters are handled according to the following conventions:

i) characters are stored/retrieved with parity generation/checking as specified by the ASCII parity bits.

ii) nulls are ignored on input unless listed as terminators by the system.

iii) input is terminated by any character in the list of ASCII system terminators or by the data item count going to zero, whichever occurs first. (ASCII system terminators are simply a list of ASCII characters masked to 7 bits).

iv) output is terminated by a null byte, the data item count going to zero, or (in the special case of a teletype) the input of a character, whichever occurs first.

In image mode, data items are stored/retrieved in character or word format (as specified) exactly as received by the hardware (no parity generation/testing, etc.) Input is terminated only by the data item count going to zero, while output is terminated by the item count going to zero or in the case of the teletype by the operator striking any key.

Bits 4 and 5 — ASCII Parity Bits

Control words for devices which are capable of handling the ASCII mode of transmission have two bits reserved for the specification of the parity operation required. The various possibilities are as follows:

i) Input
- ØØ - mask to 7 bits (no parity check)
- Ø1 - mask to 7 bits (check even parity)
- 1Ø - mask to 7 bits (check odd parity)
- 11 - same as ØØ

ii) Output
- ØØ - channel 8 set to zero
- Ø1 - even parity in channel 8
- 1Ø - odd parity in channel 8
- 11 - channel 8 to one

Bits 6 and 7 — Unused

Bits 8-15 — Device Unit Number

This number is applicable only in the case of multi-unit device handlers such as the teletype or magnetic tape. The device unit number specifies the specific unit (of a particular device type) which is being referred. In the case of teletypes, both keyboard and printer are considered as being a part of the same device unit. The console teletype (hardware device codes 1Ø and 11) is device unit number zero.

3.1.3 DATA ITEM POINTER

The data item pointer is a byte address if the data is being input or output in character format and is a word address if the data is input or output in word format.

3.1.4 DATA ITEM COUNT

The data item count is a byte count if the data is being input or output in character format and is a word count if the I/O operation is in word format.

3.1.5 ERROR ROUTINE ADDRESS

If an error occurs during the processing of an .IOX meta-instruction, the error routine address parameter is used as the return PC when the TASK is placed in the pending state. The contents of AC3 at the time of the return are used to specify the error. Error indications differ from device to device; possibilities for the teletype are as follows:

AC3 = n	Input of character "n" caused termination of the output
AC3 = Ø	Illegal device unit number in the .IOX calling sequence
AC3 = -1	Parity error on input
AC3 = -2	Negative word count in the .IOX calling sequence
AC3 = -3	Teletype unit number error in the .IOX calling sequence.

A number of devices have no defined error conditions (e.g., high speed paper tape punch, incremental plotter, etc.). The error routine address parameter in these cases is merely a dummy parameter provided to maintain .IOX format compatibility. The calling sequence to these devices could be as follows:

```
.IOX
<DEVICE #>
<CONTROL WORD>
<DATA POINTER>
<DATA COUNT>
<. + 1>
<RETURN>
```

3.2 DEVICE HANDLERS

That portion of RTOS which deals with the processing of .IOX commands is of primary interest to the real-time user wishing to add his own (or modify the standard) I/O device handlers. These handlers, with two exceptions (real-time clock and teletype), are written as separate, relocatable subroutines which are linked to RTOS at load time by the relocatable loader.

A device handler consists of three major sections: the device

initialization routine, which also includes a "next I/O" segment, the interrupt service routine, and the device unit control block (DUCB).

3.2.1 DEVICE INITIALIZATION

The device initialization routine is called by the .IOX processor by using the device handler entry point orientated in logical device code order in RTOS in a table called 'HANTB'. These entry points are the interrupt service routine addresses. The location preceding each entry point in the device handler must contain the address of the initialization routine, and is used as the transfer mechanism by the .IOX processor. This can be easily seen by reviewing some of the supplied device handlers.

The device initialization routine tests for device availability. If the device is available, it initiates the operation. If it is unavailable, the request is queued by the "IOSTK" routine. The initialization section of the handler also has an entry point which may be used by the interrupt routine to activate an "next I/O" operation for the device, a mechanism which should become apparent after reviewing a supplied device handler.

3.2.2 DEVICE INTERRUPT SERVICING

The interrupt service routine is responsible for responding to the interrupts which a device generates while in progress. It is pointed to by two tables in RTOS: one oriented in hardware device code order in the interrupt processor 'INTP', called 'DISP' and one oriented in logical device code order, called 'HANTB'. If a hardware device has not been included as part of the hardware configuration its entry in the 'DISP' table is replaced by the address of a routine 'NODEV' which will clear the interrupting device and dismiss the interrupt.

The interrupt processor when entered first checks whether the power monitor option caused the interrupt if it is part of the hardware configuration. If it was the power monitor, the power fail handler '.PWR'

is entered. If the power monitor did not cause the interrupt or is not part of the system, the processor will read in the device code of the interrupting device and branch through the correct entry in the dispatch table.

Interrupt service is performed primarily by the system routine entitled 'PRIOR' (called by JSR @.SERV), which rearranges the system interrupt priorities and saves the machine environment in the "PRIOR" parameter block. The first parameter in the 'PRIOR' call is the new priority interrupt mask to be used while the device is being serviced. If desireable, this parameter may be altered dynamically by user programs, but extreme caution is urged.

The program then issues suitable I/O instructions to input or output the next data item and test the status of the device being serviced. In the case of character orientated devices like the teletype, paper tape reader and punch, or plotter most of the necessary servicing routines are supplied within RTOS.

3.2.3 DEVICE UNIT CONTROL BLOCK (DUCB)

Every device, or more specifically, every unit of every device type in the system is defined by static and dynamic information contained in the Device Unit Control Block (DUCB). The DUCB table is contained as part of the relocatable device handler.

A typical DUCB (in fact, the model on which other DUCB's are based) is that for the teletype, which appears as follows:

```
.TTYØ:      Ø          ; queue block address, Ø if inactive   (DTCBA)
            Ø          ; variable storage location            (DTEMP)
            JMP Ø,3    ; subroutine return instruction
            Ø          ; address of get-store character
                         routine                              (DGSR)
            1Ø         ; device code for TTI                  (DVCDE)
            Ø          ; data pointer                         (DDADR)
```

```
Ø           ; data count                                    (DCNT)
Ø           ; Ø = queue available,
              1 = queue busy                                (DQBSY)
Ø           ; Ø = ASCII mode, 1 = image mode                (DCMDE)
Ø           ; address of proper parity routine              (DPRTY)
.TTIO       ; address of "next I/O" routine                 (DNIOR)
TTØØ + 3    ; interrupt data block address (out-
              put)                                          (DIDBO)
.TERM       ; address of input terminator list              (DTERM)
TTIØ + 3    ; interrupt data block address (in-
              put)                                          (DIDBI)
Ø           ; error return address                          (DERTN)
Ø           ; device operating mode (Ø - 2)                 (DMODE)
Ø           ; -1 = break disabled, else unit #              (DBRK)
```

Given at the right of the comments is a list of parameter names by which RTOS refers to these DUCB entries. These same names are made into equates and used in the device handlers.

The first DUCB entry is set to zero when the device is inactive. Activating a device (via an .IOX meta-instruction) results in this entry being chained to the TCB of the TASK awaiting service. If a device is already busy when the .IOX request is made, the new TCB is merely chained to the list of TCBs waiting for the device. This operation is performed by the IOSTK routine (called by JSR @.STAK), and results in a structured equivalent to the pending TCB queue.

The "subroutine return instruction" provides an "execute" facility in order for the device handlers to execute specific instructions (e.g., I/O commands) without destroying their re-entrant nature. The instruction is merely stored preceding the return command and a JSR to the instruction is issued. As the DUCBs are unique to each device, handler re-entrancy is preserved.

The next location of the DUCB is used to save the address of a get/store character/word routine. This address can be a fixed entry or it can be taken from the table '.BTAB' in RTOS by the initialization section of

the handler. This table has the following entries:

```
.BTAB+1:    SCHRP       Store character (byte mode)
            GCHRP       Get character (byte mode)
            STCHR       Store character (word mode)
            GTCHR       Get character (word mode)
```

The fourth DUCB location contains the device code for the hardware device which the DUCB describes. The next two entries following this are for the data item pointer and data item count. These are either in byte or word format depending on the word format indicator (bit 2 of the control word).

The queue availability switch is set when a call to the device initialization section of the handler is made. It is used by the routine 'IOEND' which handles the end of I/O operation interrupt so that it does not change the unit availability switch while the initialization routine is testing it. It is reset before the operation is initiated if the device is available or after the request is stacked in the pending queue for that device.

The mode indicator has the same meaning as bit 3 of the device control word. The address of the parity routine is taken from a table of parity checking generating subroutine addresses starting at '.PTAB+1'. The address to be taken from this table is dependent on whether the operation is input or output and the value of the ASCII parity bits. The table has eight entries as follows:

```
BIT7        ;Mask to 7 bits (input)
BIT7        ;Mask to 7 bits (output)
TEVEN       ;Test even parity
GEVEN       ;Generate even parity
TODD        ;Test odd parity
GODD        ;Generate odd parity
BIT7        ;Mask to 7 bits (input)
CHN8        ;Set channel 8 to one (output)
```

The next I/O routine address is stored in the twelfth entry of the DUCB table. It serves as the entry point for starting the next pending I/O operations for the device. Entries 13 and 15 provide the addresses of the output and input interrupt data blocks respectively. These data blocks are usually the part of the 'PRIOR' parameter block that contains

the machine status before it was interrupted.

The fourteenth entry of the DUCB provides the address of a list of terminator characters that will terminate ASCII input. This list must be terminated by a -1. The list used can be set up by the user. The current version of RTOS has a list of ASCII terminator characters starting at '.TERM' as follows:

```
.TERM: 177        ;Rub out, delimiter
       15         ;Carriage return, control M
       12         ;Line feed
       33         ;Escape
        3         ;End of text, end of message, control C
       -1
```

The error return address is taken directly from the .IOX calling sequence if the device can generate an error condition. The operating mode is usually fixed at the time of handler design. For example the card reader can only operate in input mode and so this parameter would be Ø while it would be a 1 for the high speed paper tape punch to indicate output mode. The teletype can have an echo on input, input only, or output only mode and so this parameter is a variable in the case of the teletype.

It should be noted that the final entry in the DUCB shown above is necessary only for the teletype (the .BRK facility) and will normally not be used for other device handlers. It should also be clear that some of the DUCB entries are not required for some devices types (e.g., output only devices do not require an input interrupt data block address or input terminator list, devices with no error return conditions require no error return address, etc.). Unused entry locations may be used for storage of constants or handler variables.

3.2.4 DEVICE OPERATION COMPLETE ROUTINE

It is important to be aware of the fact that when termination of an .IOX operation causes a device to become free, another request may be waiting to be activated. In order to properly handle the complexity of interacting interrupt levels at this point a "pseudo-TASK" (with priority

level Ø) is created by RTOS to initiate the next operation request. This "pseudo-TASK" requires a TCB like any other TASK, and thus must have been provided for when the 'JOBS' parameter was set during RTOS system generation. A TCB must be provided for each interrupting hardware device including the teletype and real time clock. Failure to provide sufficient TCB storage will cause a HALT at location 'QSPOP+4' or a branch to a user program at '.SHLT' depending on the value of 'SHALT'.

3.3 RTOS DEVICE SERVING PROGRAMS

Many of the subroutines within RTOS necessary for servicing the teletype have been written re-entrantly. The reason for this is that at least the console teletype is assumed to be present in most systems and its handler, if developed in a general-purpose and flexible enough manner, will provide major services for almost all other device handlers that are included or may be developed for system hardware devices. A brief description of the four most common routines appears in the following sections.

3.3.1 DEVICE CONTROL BLOCK SETUP (.DUCB)

This subroutine is entered with AC3 containing the address of the control block for the device being initiated and the device control word is contained in ACØ. The routine sets the ASCII/image mode and operating mode indicators, claculates and saves the addresses for the get/store character and parity generating/checking routines, and obtains the data item pointer and count from the .IOX calling sequence.

After the DUCB is initialized, the first I/O operation is initiated. If the operation is an input, the routine obtains the device code from the DUCB and by using the temporary storage and return provided in the devices DUCB, it executes an NIOS to start the device. If output is requested another subroutine 'OUT' will be called to get the character, generate the correct parity, and output the first character to the device. It should be noted that since this routine was developed for the teletype which only has the device code of the keyboard in the DUCB,

any other device handler trying to use '.DUCB' for initiating output operations must supply a device code one less than its true code.

3.3.2 INPUT HANDLER (.CHIN)

This re-entrant routine is used to handle input interrupts on the teletype and similar byte oriented devices. It is entered with the previous machine status saved and the new character as it came from the device in ACØ.

The routine first checks whether the device is being run in ASCII or image mode. If it is in ASCII mode, a check is made to see whether a null character has been entered in which case it is ignored. A parity check is then performed and if not correct the error return is taken. If correct parity is found, the routine then checks whether a match between the list of terminator characters and the input character (now masked to 7 bits) is present. If it is the input operation is complete and if not, the character is stored in the input buffer, and another request made if the data count has not gone to zero.

3.3.3 OUTPUT HANDLER (.CHOT)

This re-entrant routine is entered with AC2 containing the address of the device unit control block and is used to handle output interrupts on the teletype and similar character oriented devices.

The data count is checked upon entry to this routine. If it has gone to zero, the output operation has been completed and the end of output completion section 'IOEND' is entered. If data remains to be output, the next character is obtained and the output device started using the routine 'OUT'. The interrupt is then dismissed with the routine 'DISIN'.

3.3.4 INPUT/OUTPUT REQUEST SETUP (.TTIO)

This routine, used by the teletype .IOX handler and referred to as the

"next I/O" routine, is entered when it is determined that the requested device is now available for the next I/O request. It sets the I/O queue indicator of the DUCB to the available status, restores AC2 to the address to point to .IOX + 1 and stores the error return address in the DUCB.

3.3.5 MACHINE STATUS SAVE ROUTINE (PRIOR)

'PRIOR' is a module within the RTOS system that an interrupt connected routine calls to save the environment of the TASK that was interrupted. It also provides a mechanism whereby an interrupt servicing routine may operate at the priority of the hardware device.

The calling sequence is shown below and shows that 'PRIOR' stores the environment of the interrupted routine in the storage block just following the JSR, to the "PRIOR" subroutine. An entry point '.SERV' in the RTOS program is provided for use by the device handlers.

```
JSR PRIOR
NMASK       ; New hardware priority word
0           ; Old priority word storage
0           ; Carry storage
0           ; AC0 storage
0           ; AC1 storage
0           ; AC2 storage
0           ; AC3 storage
0           ; Program counter storage
DUCB        ; Address of device control block
```

The subroutine 'PRIOR' is entered with AC2 and AC3 having been previously saved in locations 'SAC2' and 'SAC3', respectively, by the interrupt processor 'INTP'.

3.3.6 MACHINE STATUS RESTORE ROUTINE (.DISN)

The RTOS executive routine '.DISN' provides an interrupt service routine with the means to perform the following functions:

i) restore the operating environment of the interrupted routine,

ii) restore the hardware interrupt priority, and,

iii) return to the interrupted routine.

It is entered with AC2 containing the address of the interrupt data block. The interrupt data block address is taken as the address of where the carry bit is stored, i.e., three locations past the JSR PRIOR.

3.3.7 INPUT/OUTPUT REQUEST STACKER (IOSTK)

This system routine is called when the I/O request must be queued for the device because a previous task is already using the device. An entry point '.STAK' within the RTOS program is provided for use by the device handlers.

3.3.8 END OF I/O OPERATION ROUTINE(.IOEND)

This routine is used to handle the end of I/O operations for a device handler at either the interrupt or system level. When entered on the interrupt level, the address of the interrupt data block will be in AC0. At the non-interrupt level, AC0 will contain a zero.

Upon entry, if it is found that the DUCB is not available, this routine will create a job at priority zero to perform the end of I/O operation at the non-interrupt level.

If while in the .IOEND routine, it is determined that another request is pending, the routine will create a job of priority zero to start the next I/O operation on the device.

CHAPTER 4

SYSTEM INITIALIZATION

The initialization program 'RTIN' is used to initialize stacks and clear switches in the RTOS system. The first section of the routine zeros system switches and sets all the device handlers to the available state. The system is initially set to a priority of zero, the hardware mask is made zero, the system is set in USER MODE, the real time clock is set inactive, the clock queue and the TCB queue pointers are both set to zero.

All the .XMIT/.RCV channel status table entries are made inactive by setting them all to zero. The task control block and clock queue stacks are initialized by linking each block together in a form similar to that described in section 2.1.2 for pending TASK. The break request is set inactive by setting the break character '.BCHR' to -1.

The last operations performed by the initialization program are to do an IORST, enable the interrupt facility by the INTEN instruction, and to jump to the start of a user TASK which must be given a label 'START' which is declared as an entry point in the user program.

The queue availability switch of each device handler must be zeroed by the initialization program. The address of this location (the first in the device's DUCB table) is available to the initialization program by means of a second entry point in the handler and usually defined using the first four characters of the handler interrupt entry point followed by a digit "1".

APPENDIX A

SYSTEM GENERATION

System generation allows the user of the Real Time Operating System to define the characteristics of his operating environment (the numbe of .XMIT/.RCV channels, I/O devices on the system, speed of the real time clock, etc.).

The input to an RTOS system generation is the set of relocatable source and binary paper tapes of RTOS, the initialization program (RTIN), and handlers for standard I/O devices. When source tapes are ordered in the customer software package, the user can easily change the hardware configuration parameters, the operating system specifications, and add or modify device handlers.

The RTOS system supplied supports a hardware configuration consisting of at least 4K core memory, the real time clock, the high speed paper tape reader and punch, and the console teletype. The operating system allows sixteen user TASKS, and eight .XMIT/.RCV communication channels. The real time clock is set to operate at 100 Hertz.

A.1 RTOS PARAMETER TAPE

The parameter tape provides a definition of all system variables that must be set by the user to tailor RTOS to his installation. The system/device definitions provided on this tape are described in section D.2.

Unless the user is adding new device handlers not provided for in the released RTOS system or is changing the standard logical device numbers of the system, the parameter tape is the only tape that must be modified by the user. A reassembly of RTOS and the initialization program using the new parameter tape is required to tailor the operating system to the user's environment.

A listing of the RTOS Parameter Tape is provided with the supplied program package. Note that all symbols have been declared using the .DUSR pseudo-op. They will therefore not appear in the assembly's symbol table output. Further, loading of the parameter tape can be skipped on pass 2 or saved permanently as part of the assembler.

Most of the system definition parameters described in section D.2 must be set to a value of Ø or 1 depending on whether the device or option is present or not on the system. Other parameters can take on values greater than 1 and will now be described in more detail than appears in D.2.

CHAN — Number of .XMIT/.RCV channels on the system. It should be given a value greater than zero. For each channel that is defined, an entry is made in each of the channel message and status tables.

FREQ — Real time clock frequency.

0 = AC line frequency
1 = 10 Hertz
2 = 100 Hertz
3 = 1,000 Hertz

JOBS — Maximum number of parallel TASKS allowed in the system. This must be set to the total number of user TASKS and hardware devices tied to the interrupt facility.

SHALT — System Resources Depleted Parameter
A "Ø" implies the system should HALT if enough TASK control blocks were not defined because the value of 'JOBS' was too small.
A "1" implies the system should branch to a user supplied program with an entry point called '.SHLT' if the above condition occurs.

TTYS — Number of teletypes in the system.
It is assumed that all Nova computer systems will have at least a console teletype and so this value should be set to at least "1".
If additional teletypes are added to the system this value should be increased and the user must remember to set up the DUCB tables and entry points in the logical device number and interrupt entry point tables.

A.2 PROGRAM ASSEMBLY

Once the user has obtained a new parameter tape, he can then proceed to assemble the Real Time Operating System and the initialization program 'RTIN'. Each of these programs requires that the parameter tape be the first tape in the assembly.

These programs should be assembled using either Extended Assembler (091-000017) or the DOS Assembler (088-000001).

If the user desires to assemble the RTOS system absolutely using the absolute assembler (091-000002), the code for relocation, interprogram communication, and conditional assembly must be removed. Location statements must be inserted into RTOS, the initialization, and the device handler programs to provide starting addresses. An equate table must be made for interprogram communication.

A.3 SYSTEM LOADING

After generating relocatable binary tapes for all RTOS system and user programs, the user is ready to load his system. The Extended Relocatable Loader (091-000038) is first loaded by the standard binary loader and is then used to load the relocatable binary tapes. For operating instructions the Relocatable Loader Manual (093-000039) should be referenced.

The starting address of the user TASKs must be labelled 'START' and declared as an entry point (.ENT).

A.4 SYSTEM STARTUP

The system should be started at location 'INIT' after loading all the user TASKs, RTOS system programs and required device handlers. The initialization program when completed branches to the start of the user TASKs at 'START'.

APPENDIX B

WRITING A DEVICE HANDLER

The purpose of this appendix is to outline a step by step procedure for adding a new device handler to the Real Time Operating System. The I/O calling sequence, the device handler implementation, and the subroutine entry points within RTOS that can be used have all been described in Chapter 3: Device Handler Implementation.

For the purpose of illustration, we shall call the device entry point .DEV and its definition parameter will be DEVICE. The label DEVICE will be edited into the RTOS parameter tape and given a value of "1" to tell the system that this device is to be included in the system.

To clarify what is said in the following sections, the user planning to implement his own device handler should review some of the supplied handler listings.

B.1 Entry Point Definition

The device handler entry point must be defined in two tables within the RTOS system. The first table 'HANTB' is a list of entry points oriented in logical device code order. The supplied system has defined logical unit numbers for most of the standard I/O devices that can be supplied with a Nova computer. (See section 3.1.1: LOGICAL DEVICE NUMBER). This table could be expanded by adding the new device handler entry point at the end of the table or modified by changing one of the present device assignments to the new device.

To include the new device as logical device number ten at the end of the present list, the code

```
.IFE   DCOM
.IOBAD
.ENDC
```

```
.IFN    DEVICE
.EXTN   .DEV
.DEV                            ;DEVICE 1Ø10
.ENDC
```

should be added to the existing table. It must precede the label 'HANTE' which is used to point to the end of the handler table.

To include the new device as logical device number four instead of the incremental plotter, the code

```
.IFN    PLOT
.EXTN   .PLT
.PLT                            ;DEVICE 4
.ENDC
.IFE    PLOT
```

should be changed to read

```
.IFN    DEVICE
.EXTN   .DEV
.DEV                            ;DEVICE 4
.ENDC
.IFE    DEVICE
```

The second table which must be supplied with the interrupt servicing routine entry point is the interrupt dispatch table 'INTP'. The entry in this table must correspond to the device code assigned to the hardware device. If the device code was $3Ø_{(8)}$, the code that must be added to the end of the table after

```
.IFN    DCOM
.EXTN   .DCOM
.DCOM                           ;Data Communication (Code24)
.ENDC
```

is as follows:

```
.IFE    DCOM
NODEV
.ENDC

NODEV                           ;CODE 25
NODEV                           ;CODE 26
NODEV                           ;CODE 27

.IFN    DEVICE
.DEV                            ;DEVICE CODE 3Ø
.ENDC
```

B.2 INITIALIZATION

Initial entry to the device handler is provided through the .IOX meta-instruction. The portion of RTOS that processes the .IOX call creates a TASK control block, checks that the device number is within the allowed range and is defined on the system, checks that the word count is greater than or equal to zero, and then branches to the initialization section of the device handler. If during the .IOX call processing, the count is found negative, the .IOX call is set up to take the error return with AC3 = -2. If it was not a legal device number, the error return is made with AC3 = Ø.

The entry to the initialization section is provided by putting its address immediately preceding the interrupt entry point. The .IOX servicing program loads AC3 with the interrupt entry point address '.DEV' from the 'HANTB' table and does a JMP @-1,3 to enter the initialization section of the handler.

Upon entering the initialization program, all system status information has been saved in a TASK control block whose address is contained in AC2. Accumulator zero contains the device control word from the .IOX calling sequence. The program must now check whether the device is available to service the new request or the request must be stacked to await completion of previous requests.

To insure that the DUCB is not modified by the interrupt servicing section of the device handler while it is being accessed from the initialization section of the handler, the eighth entry ('DQBSY') of the DUCB is set non-zero to indicate the queue is busy. The check is then made by loading the first entry of the DUCB. A zero value means the device is inactive and the desired I/O request can be initiated immediately. A non-zero value means the request must be put into the stack of TASKS awaiting to perform I/O on the called device. The TASK can be put in the stack by the 'IOSTK' routine which may be entered by a jump indirect through a page zero variable called '.STAK' and defined as an entry point in the RTOS program.

If the device is available to service the request, the initialization program then branches to the "next I/O" section of the handler.

B.3 NEXT I/O REQUEST SERVICING

This section of the handler is used to complete the DUCB setup for the I/O request once the device is available. It is entered directly from the initialization section of the handler if the device was available at the time of the .IOX call or from the interrupt servicing section if the previous request was completed. The address of this routine is the eleventh entry of the DUCB table and is also used by the 'IOEND' routine of RTOS when the end of the previous I/O operation is being completed and additional requests are stacked for the device.

The next I/O routine must be entered with the accumulators setup as follows:

AC0 = device control word (2nd .IOX parameter)
AC1 = 0
AC2 = address of the task control block
AC3 = address of the device unit control block (DUCB)

This occurs automatically when the end of I/O operation of RTOS is performed and must be the case when entering from the initialization section.

The routine should first save the TCB address as the first entry in the DUCB and zero the I/O queue availability switch in the DUCB. It must set up the data pointer, data count, and error return storage locations within the DUCB from the .IOX call parameters. The device control word is used to initialize the device operating mode switch, the address of the get/store character routine, the address of the proper parity routine, and the character mode indicator.

The device operating mode switch (parameter 'DMODE') and the character mode indicator (parameter 'DCMDE') can be set directly from the control word, their values being 0,1,2 or 0,1 respectively. The address of the get/store character routine can be obtained from the table '.BTAB' in

RTOS and stored as the 'DGSR' parameter in the DUCB. The correct entry is determined by examining the input/output mode and the data format switches of the control word. The address of the parity routine can be obtained from the table '.PTAB' in RTOS and stored as the 'DPRTY' parameter in the DUCB. The correct entry is determined by checking the input/output mode switch and the ASCII parity bits of the control word.

After the DUCB has been completely initialized, the routine should initiate the first I/O operation. If the device is operating in the input mode, this need only be an NIOS instruction to start the device. If output mode is used, the routine should get the first word or character and transmit it to the device. After starting the device in either input or output mode, the program should return to the TASK scheduler by the .QUIT meta-instruction.

If the handler is being developed for a teletype-like device (byte oriented), many of the routines written into RTOS can be used. This is the case for the high speed paper tape reader/punch and the plotter handlers. Routines that are commonly used and have been written re-entrantly like '.DUCB' and '.TTIO' have been described in sections 3.3.1 and 3.3.4 respectively.

B.4 INTERRUPT SERVICING

The interrupt servicing section of the device handler is entered directly from RTOS when it responds to the hardware interrupt. It performs the entry via an indirect jump through the interrupt dispatch table 'DISP' by getting the device code from the interrupting device through an interrupt acknowledge command 'INTA' and adding this number to the address 'DISP'.

Standard practice in handler development is to branch to the subroutine 'PRIOR' as the first operation in the servicing program. This can be performed by an indirect jump through the page zero entry point

'.SERV' and is used to rearrange the hardware priorities, save the system status, and enable the interrupt facility again. The return from this subroutine is to 1Ø locations past the JSR with AC2 containing the address of the device unit control block. For a detailed description of the subroutine 'PRIOR', see section 3.3.5.

If it was an input operation, the routine must read the character/word from the device, perform proper parity checking, save it in the input buffer, and test if more input is required and restart the device if necessary. If the operation was performing output, the routine must test if further characters/words are to be sent to the device and if so, initiate the next output operation.

If another I/O operation must be performed, the interrupt should be dismissed after the operation is initiated. This can be done by entering the '.DISN' routine of RTOS with AC2 pointing to the interrupt data block. This address is stored in the 'DIDBO' parameter of the DUCB for an output device and in the 'DIDBI' parameter for an input device. The '.DISN' routine will restore the hardware priorities and machine status to that found previous to the occurrence of the interrupt.

To initiate another I/O operation on a byte-oriented device like the teletype special re-entrant routines have been supplied in the RTOS program. These routines '.CHOT' and '.CHIN' can be used to initiate the next output or input operation respectively, in a manner similar to the supplied high speed paper tape punch and reader handlers.

If the device is not byte-oriented or cannot be handled like a teletype, the user writing the handler must provide the necessary code to start the next I/O operation. This may have been written for the "next I/O" servicing section or may have to be supplied for the interrupt "next I/O" request.

For an output operation, the next I/O portion must test if further output is required and if it is not, enter an end of output operation

phase. This routine is supplied in RTOS with the entry point being defined as '.CHRO'. If further output is required, the routine must use the get character/word subroutine, whose address has been entered in the DUCB as the 'DGSR' parameter, to get the next data item, to send it out to the device, and to dismiss the interrupt via the '.DISN' routine.

For an input operation, the next I/O routine must check the parity of the newly entered data item if operating in the character mode and then to store the data into the input data buffer via the routine whose address is the 'DGSR' parameter of the DUCB. The routine must then test if further input is required and if it is not, enter an end of input operation phase. A routine for this purpose is supplied in RTOS with an entry point defined as '.CHRI'. If additional input data is required, the device should be reactivated and the interrupt dismissed via the RTOS routine starting at entry point '.DISN'.

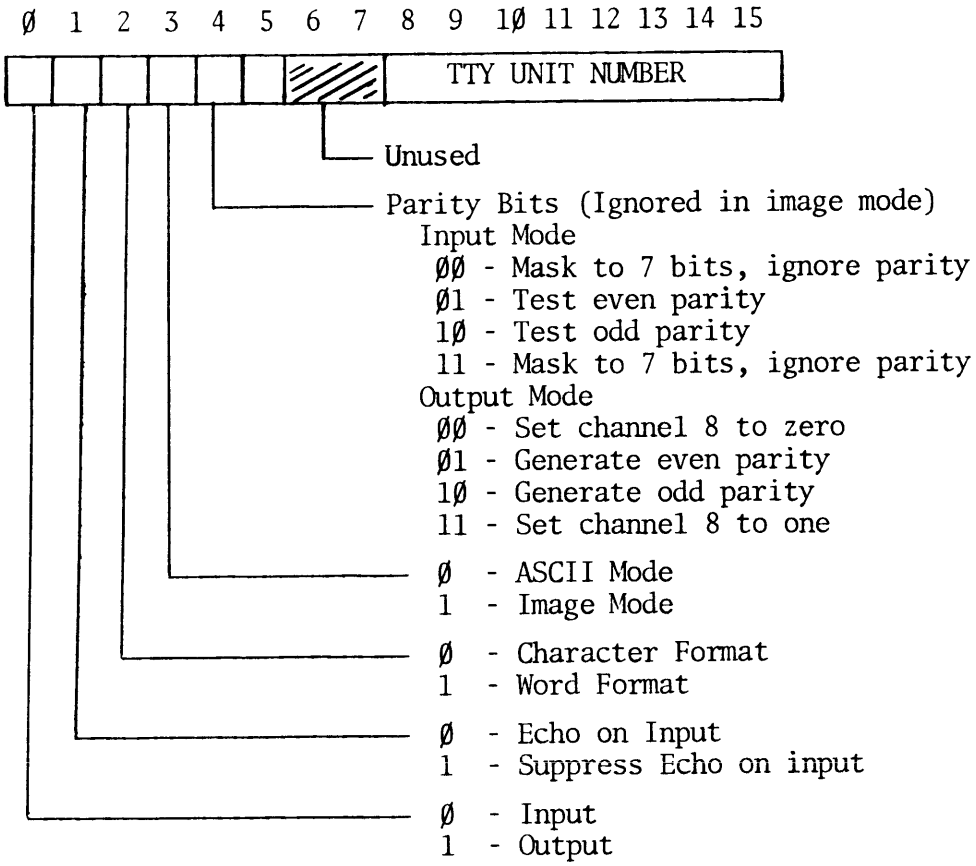
APPENDIX C

PERIPHERAL SUPPORT SUMMARY

This appendix contains a summary sheet for each of the standard device handlers. The sheet gives the format taken by the device control word in the .IOX calling sequence. It also provides a summary of the possible normal and error returns from the handler and the meaning that AC3 takes in these cases.

Additional sheets for other device handlers will be added to this appendix as the handlers are made available. Handers currently available include:

TELETYPE DEVICE CONTROL WORD



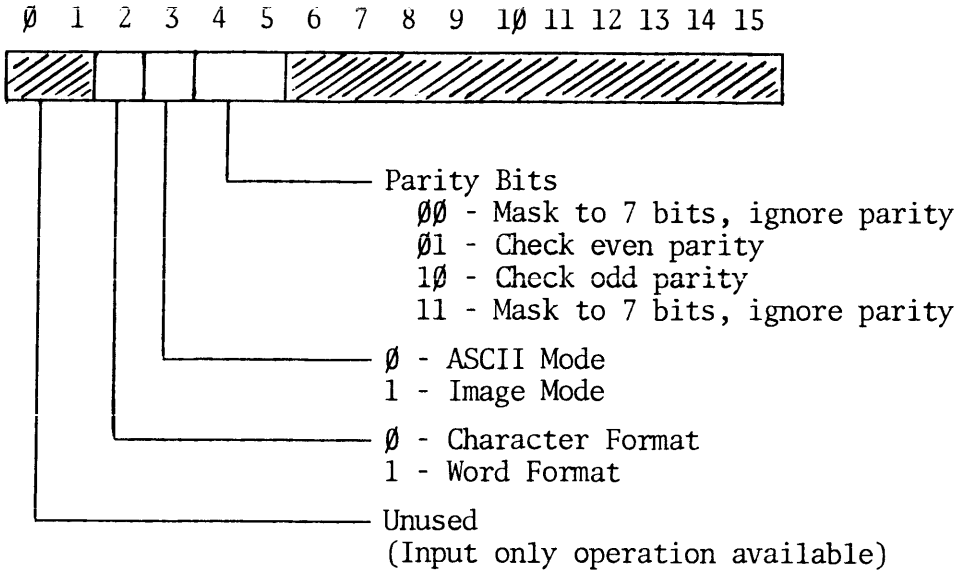
NORMAL RETURN CONDITIONS

AC3 = -1	Data count went to zero
AC3 = n	Input of character "n", either a null or from the list of terminators, caused termination of input
AC3 = Ø	Data output was terminated by a null character

ERROR RETURN CONDITIONS

AC3 = n	Input of character "n" caused termination of the output
AC3 = Ø	Illegal device unit number in .IOX calling sequence
AC3 = -1	Parity error on input
AC3 = -2	Negative word count, in .IOX calling sequence
AC3 = -3	Teletype unit number error in .IOX calling sequence.

HIGH SPEED PAPER TAPE READER CONTROL WORD



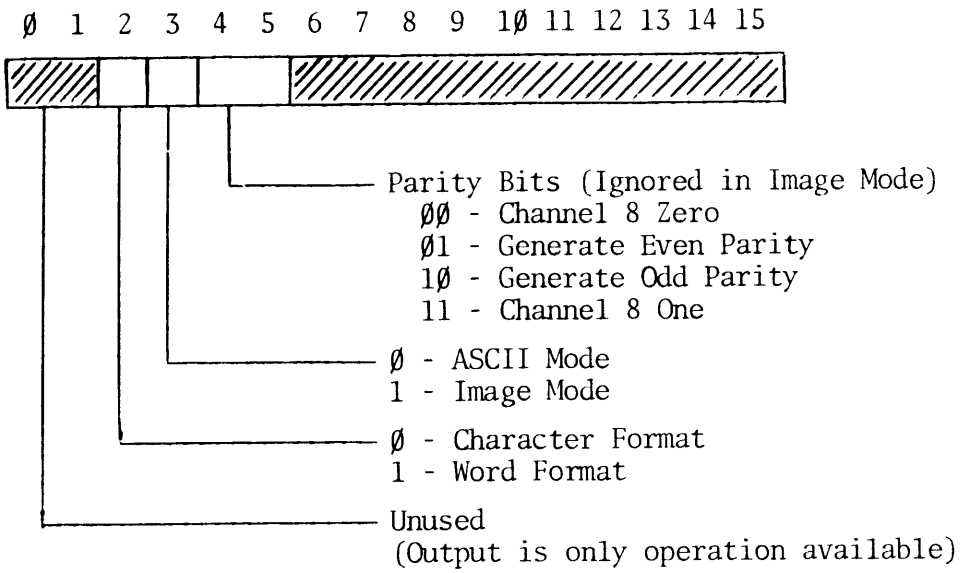
NORMAL RETURN CONDITIONS

AC3 = -1	Data count went to zero
AC3 = n	Input of character "n", either a null or from the list of terminators, caused termination of input.

ERROR RETURN CONDITIONS

AC3 = Ø	Illegal device unit number in .IOX calling sequence
AC3 = -1	Parity error on input
AC3 = -2	Negative word count in .IOX calling sequence

HIGH SPEED PAPER TAPE PUNCH CONTROL WORD



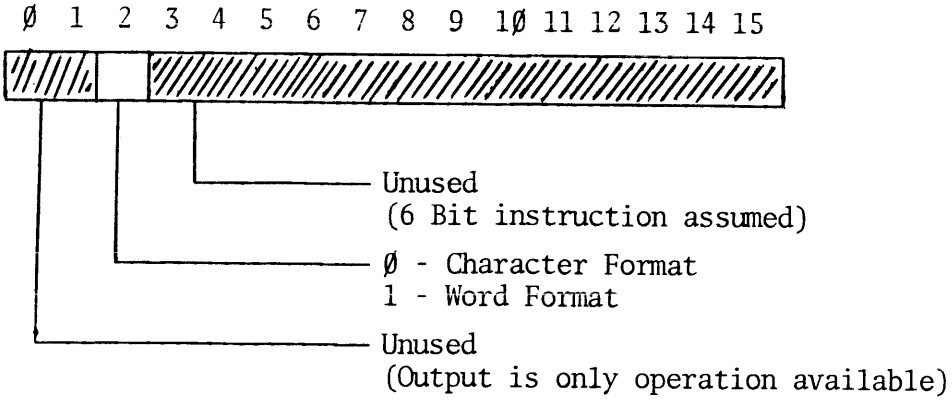
NORMAL RETURN CONDITIONS

AC3 = Ø	Data output was terminated by a null character
AC3 = 1	Data count went to zero

ERROR RETURN CONDITIONS

AC3 = Ø	Illegal device unit number in .IOX calling sequence
AC3 = -2	Negative word count in .IOX calling sequence

PLOTTER CONTROL WORD



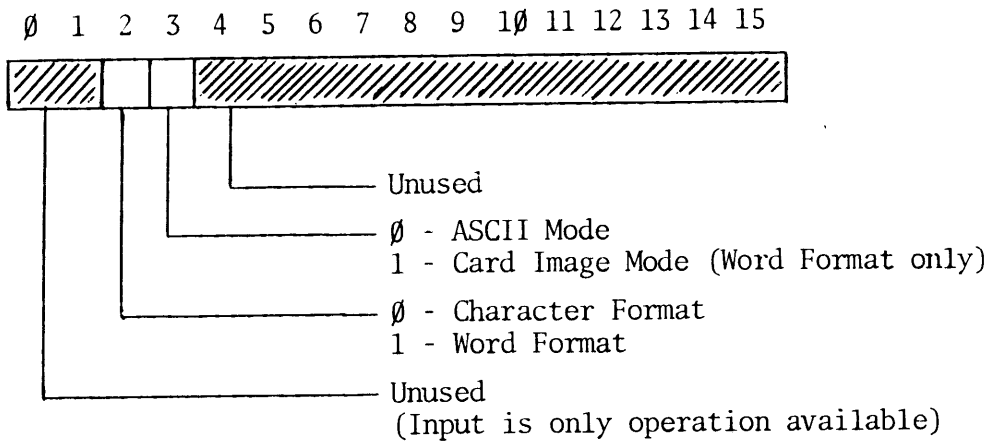
NORMAL RETURN CONDITIONS

AC3 = Ø	Data output was terminated by a null character
AC3 = -1	Data count went to zero

ERROR RETURN CONDITIONS

AC3 = Ø	Illegal device unit number in .IOX calling sequence
AC3 = -2	Negative word count in .IOX calling sequence.

CARD READER CONTROL WORD



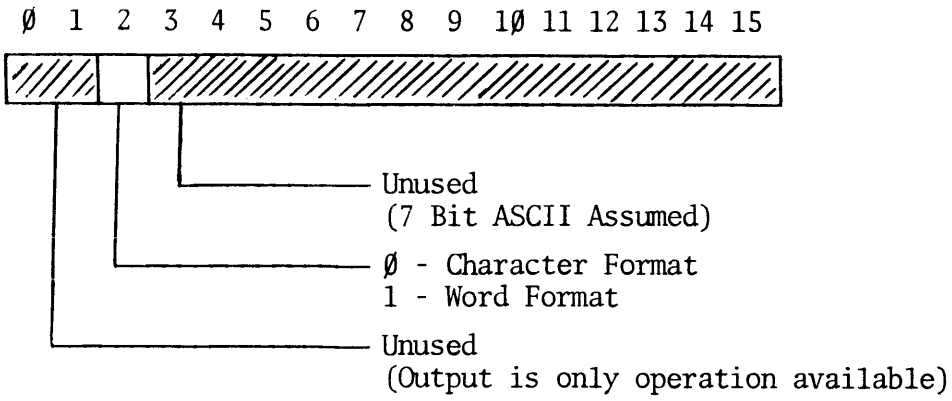
NORMAL RETURN CONDITIONS

AC3 = -1	Card read correctly

ERROR RETURN CONDITIONS

AC3 = Ø	Illegal device unit number in .IOX calling sequence
AC3 = -1	Card reader is not available (not on line)
AC3 = -2	Negative word count in .IOX calling sequence
AC3 = 1	A card has failed to move properly through the reader or an error has been detected in the circuitry.
AC3 = 2	A card was not brought in from the hopper
AC3 = 4	The card hopper is empty or the stacker is full.

LINE PRINTER CONTROL WORD



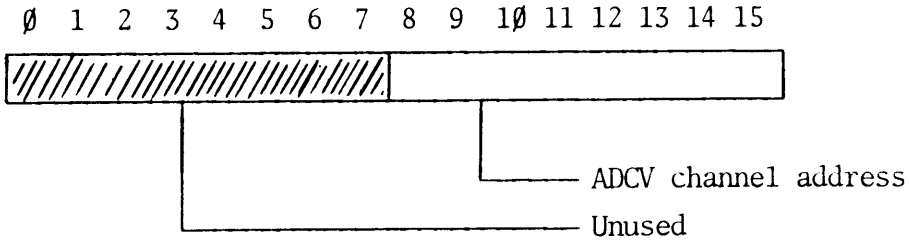
NORMAL RETURN CONDITIONS

AC3 = Ø	Data output was terminated by a null character
AC3 = -1	Data count went to zero

ERROR RETURN CONDITIONS

AC3 = Ø	Illegal device unit number in .IOX calling sequence
AC3 = -1	Line printer not available (not on line, power off or out of paper)
AC3 = -2	Negative word count in .IOX calling sequence

ANALOG TO DIGITAL CONVERTER CONTROL WORD



NORMAL RETURN CONDITIONS

AC3 = -1	Data count went to zero

ERROR RETURN CONDITIONS

AC3 = Ø	Illegal device unit number in .IOX calling sequence
AC3 = -2	Negative word count in .IOX calling sequence.

MULTIPLE TELETYPE SYSTEM

Provided with RTOS is a handler for two additional teletypes. These are included in the program if at assembly time the parameter tape specifies that TTYS is greater than one. Only a device unit control block and the interrupt servicing logic for the teletype input and output are necessary to implement an additional teletype under RTOS.

The two teletypes referred to as TTY Unit Numbers 1 and 2 are implemented in the .MTTY handler service device codes 50,51 and 40,41 respectively.

To implement other teletypes on the system, tape six of RTOS would need to be modified so the necessary entries are made in the interrupt and the teletype unit dispatch tables.

The control word for the additional teletypes has exactly the same format as for the console teletype described on page C-2.

APPENDIX D

RTOS VARIABLE DEFINITIONS

The Real Time Operating System has a set of reserved words consisting of the eight meta-instructions (.IOX, .WAIT, etc.), the system definition parameters (JOBS, TTYS, TAPE, etc.), the device handler names (.PTR, .DSK, etc.), and the entry points defined in RTOS (.JOB., .STAK, etc.) and the initialization program (START). These reserved words cannot be used as variable names in user programs except as their usage pertains to RTOS. The meaning and usage of these variables is defined below.

D.1 RTOS META-INSTRUCTIONS

.BRK - is a meta-instruction (to be used as a special purpose command) which allows input of a specified ASCII character to interrupt the normal operation of RTOS.

.FORK - is the RTOS meta-instruction for the creation of parallel processes (i.e., additional TASKS).

.IOX - is the meta-instruction that handles standard I/O operations within RTOS, initiating an activity on the specified I/O device.

.PTY - is the meta-instruction to alter the priority of the current TASK, within the range $\emptyset$-255_{10} ($\emptyset$ being the highest).

.QUIT - is the RTOS call to terminate a TASK.

.RCV - is the second half of the meta-instruction pair for TASK synchronization and communication. It enables the reception of a "synchronization signal" sent by an .XMIT instruction over a specified channel.

.WAIT - is the meta-instruction used to delay the execution of the current TASK for a specified time interval.

.XMIT - is the first half of a complementary pair of meta-instructions provided for TASK synchronization and inter-TASK communication. It causes transmission of a "synchronization" signal over a specified channel.

D.2 RTOS SYSTEM DEFINITION PARAMETERS

These parameters are all defined on the system parameter tape that is tailored to each installation and is used when assembling the RTOS mainline and initialization programs.

NAME	VALUE	PLACES USED	DESCRIPTIONS
A2D	Ø or 1	RTOS RTIN	Analog to digital converter definition.
CARD	Ø or 1	RTOS RTIN	Card reader definition.
CHAN	+ ve	RTOS RTIN	Number of .XMIT/.RCV channels in the system.
DCOM	Ø or 1	RTOS RTIN	Data communications multiplexer definition.
DISK	Ø or 1	RTOS	Fixed head disk definition.
FREQ	Ø,1,2,3	RTOS	Real time clock frequency.
HSP	Ø or 1	RTOS RTIN	High speed paper tape punch definition.
HSR	Ø or 1	RTOS RTIN	High speed paper tape reader definition.
JOBS	+ ve	RTOS RTIN	Maximum number of parallel tasks allowed in the system.
PLOT	Ø or 1	RTOS RTIN	Incremental plotter definition.
PRINT	Ø or 1	RTOS RTIN	Line printer definition.
PWRFL	Ø or 1	RTOS RTIN	Power monitor / auto restart option definition.
SHALT	Ø or 1	RTOS	System resources depleted definition.

NAME	VALUE	PLACES USED	DESCRIPTIONS
TAPE	Ø or 1	RTOS RTIN	Magnetic tape definition.
TTYS	> or =1	RTOS	Number of teletypes in the system.

D.3 DEVICE HANDLER NAMES

The following list of names is reserved for device handlers in RTOS. As additional devices are interfaced to the Nova family of computers and new device handlers become available, this list of names will be expanded. The first name is the interrupt entry point while the second name is the start of the device unit control block in the handler.

.ADCV,	.ADC1	-	analog to digital converter (4032)
.CDR,	.CDR1	-	card reader (4016)
.DCOM,	.DCM1	-	data communications multiplexer (4026)
.DSK,	.DSK1	-	fixed head disk (4019)
.LPT,	.LPT1	-	line printer (4034)
.MTA,	.MTA1	-	magnetic tape (4030)
.PLT,	.PLT1	-	incremental plotter (4017)
.PTP,	.PTP1	-	high speed paper tape punch (4012)
.PTR,	.PTR1	-	high speed paper tape reader (4011)
.PWR		-	power monitor / automatic restart option (XX06)
.TTI1	.TTY1	-	teletype unit 1 (device codes 50,51)
.TTO1			
.TTI2	.TTY2	-	teletype unit 2 (device codes 40,41)
.TTO2			

D.4 SYSTEM ENTRY POINTS

NAME	VALUE	PLACES USED	DESCRIPTION
.BCHR	-ve = inactive +ve = break char.	RTOS RTIN	Break character storage.
.BTAB	.NREL address	RTOS Device Handlers	Address of table of get/ store character/word sub-routine addresses.
.CHIN	.NREL address	RTOS Device Handlers	Routine to handle an input interrupt from a byte oriented device.
.CHOT	.NREL address	RTOS Device Handlers	Routine to handle an output interrupt from a byte oriented device.
.CHRI	.NREL address	RTOS Device Handlers	Routine to dismiss an input interrupt at the end of an input operation.
.CHRO	.NREL address	RTOS Device Handlers	Routine to dismiss an output interrupt at the end of an output operation.
.CLK.	.NREL address	RTOS RTIN	Address of the clock count down interval queue.
.CPNT	.NREL address	RTOS RTIN	Address of the first entry in the clock pending stack.
.CPTY	$Ø$-255_{10}	RTOS RTIN	Priority of the currently executing TASK.
.CST.	.NREL address	RTOS RTIN	Address of .XMIT/.RCV channel status table.
.CSTK	.NREL address	RTOS RTIN	Address of the stack of clock queue entries.
.CUCB	$Ø$ = inactive $\neq Ø$ clock active	RTOS RTIN	Clock active switch.
.DCB1	.NREL address	Device Handlers	Entry point in the .DUCB routine to initiate the first I/O instruction

System Entry Points (cont'd)

NAME	VALUE	PLACES USED	DESCRIPTION
.DISN	.NREL address	RTOS Device Handlers	Routine to restore the previous machine status and dismiss an interrupt.
.DUCB	.NREL address	RTOS Device Handlers	Routine to set up the device unit control block for .IOX calls.
.IOER	.NREL address	RTOS Device Handlers	.IOX unit number error return subroutine.
.IOEND	.NREL address	RTOS Device Handlers	Address of a routine to handle end of I/O operations
.JOB	.NREL address	RTOS RTIN	Address of pending job TCB's.
.JPNT	.NREL address	RTOS RTIN	Address of first entry in job pending stack.
.JSTK	.NREL address	RTOS RTIN	Address of stack of TCB addresses to be put in the pending queue.
.PMSK	$\emptyset$-177777_8	RTOS RTIN	Hardware mask for currently executing TASK.
.PTAB	.NREL address	RTOS Device Handlers	Address of table of parity checking/generating subroutine addresses.
.QPNT	.NREL address	RTOS RTIN	Address of first entry in job queue stack.
.QSTK	.NREL address	RTOS RTIN	Address of pending job TCB address storage area.
.QUE.	.NREL address	RTOS RTIN	Address of first TCB to be queued.
.RTC.	.NREL address	RTOS RTIN	Address of first entry in clock count down interval queue.
.SERV	.ZREL address	Device Handlers	Address of the interrupt priority controlling routine (PRIOR).

System Entry Points (cont'd)

NAME	VALUE	PLACES USED	DESCRIPTION
.SHLT	.NREL address	RTOS	Start of a user program to handle the case when too few tasks were defined (also SHALT = 1)
.STAK	.ZREL address	Device Handlers	Address of the I/O job stacker routine (IOSTK)
START	.NREL address	RTIN	Starting address of user TASKS, branched to after completion of initialization.
.SYS.	Ø = User 1 = System	RTOS RTIN	RTOS operating mode switch.
.TERM	.NREL address	RTOS Device Handlers	Address of a list of termination characters used to terminate input.
.TTIO	.NREL address	RTOS Device Handlers	Next I/O routine for the teletype interrupt handler.
.TTYØ	.NREL address	RTOS RTIN	Address of teletype unit Ø device unit control block.
.TTYI	.NREL address	RTOS .MTTY	Address of general routine to pre-process all teletype input
.TTYO	.NREL address	RTOS .MTTY	Address of general routine to pre-process all teletype output.

E.1 DEMONSTRATION PROGRAM NO. 1

This program demonstrates the usage of all eight meta-instructions of the Real Time Operating System. To do this, it creates a number of parallel tasks performing data input and output to the teletype, suspends execution of the program for ten seconds, and uses the break feature to allow the user to restart the program at any point during its execution cycle. It uses the real time clock and console teletype so it can be run on the minimum hardware configuration necessary for using RTOS.

The new user of RTOS should have thoroughly read the RTOS Reference Manual previous to attempting to follow the logic of this example.

Once the system is initialized, the user TASK priority is set to 100 octal. A restart TASK is then initiated at a higher priority (10 octal) to await the user typing a control C character on the teletype. When this break character is input, a complete re-initialization of the system will be performed.

The initial TASK then initiates a parallel TASK (TASK1) to request the user to enter a ten character name on the teletype. This TASK is at the same priority as the initial TASK and so does not begin execution until the initial TASK has been completed or in this case causes a system rescheduling after initiating the output of a message "RTOS TEST PROGRAM" via an .IOX meta-instruction. This TASK, after completely outputting the message, terminates itself by a .QUIT.

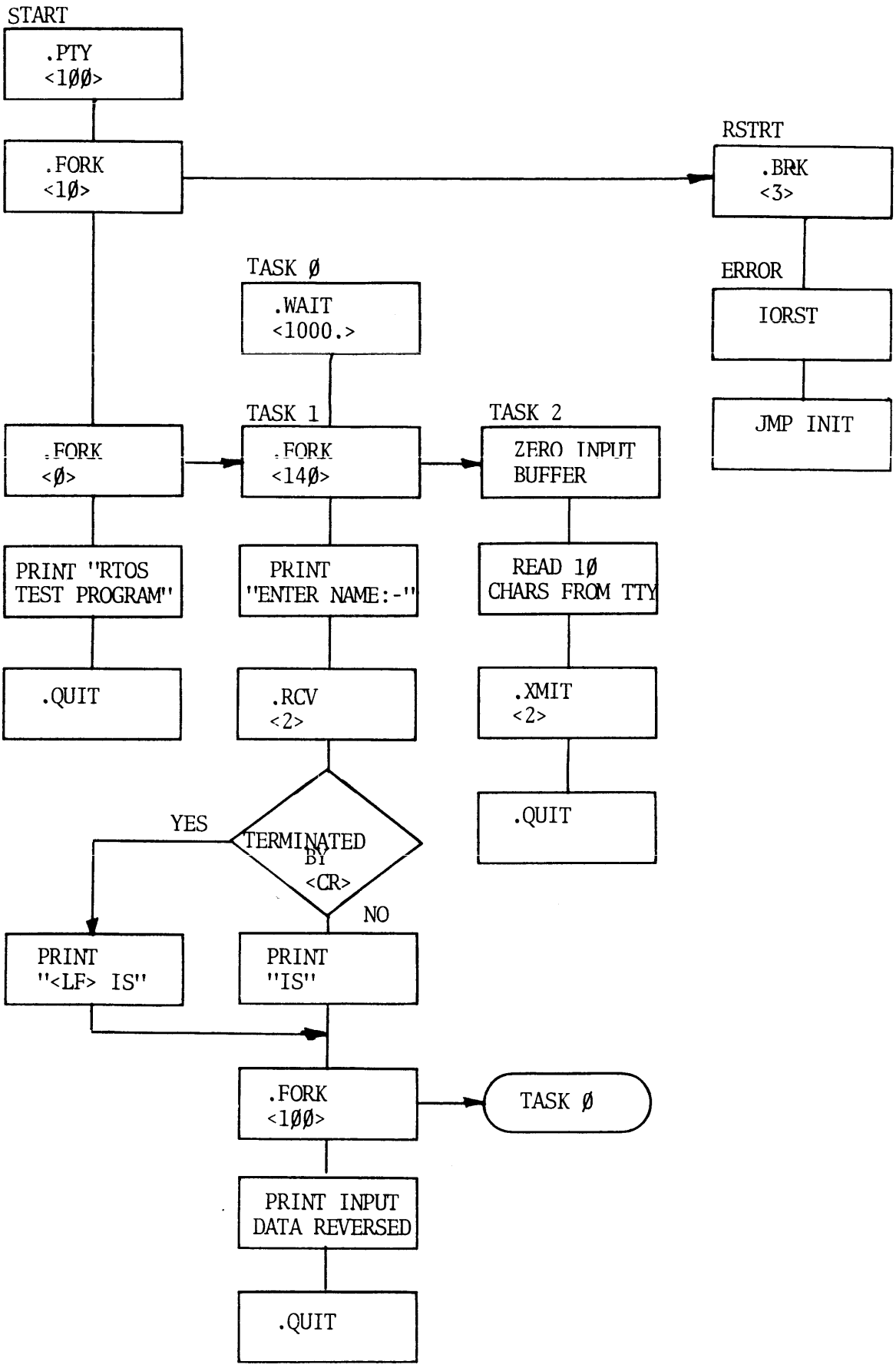
The first function performed by TASK1 is to initiate a TASK (TASK2) at priority 14Ø (octal) to input up to ten characters from the teletype. Because TASK1 is higher priority than TASK2, it is scheduled for execution after completion of the .FORK instruction and starts the output of a message to the operator to enter a name "ENTER NAME:-". TASK1 is now in a suspended state until the message is completed and TASK2 is put in the executing state. TASK1 after completely outputting this message issues a .RCV meta-instruction on channel 2 to wait for completion of the data input function of TASK2.

After completing the input phase (ten characters entered or terminated by one of the ASCII terminator characters: carriage return, line feed, escape, end of text, or rubout), TASK2 transmits the contents of AC3 when it returned from the .IOX instruction to TASK1 via channel 2. To use the .XMIT instruction, the message must be moved into ACØ. It then performs a .QUIT operation.

TASK1 upon receiving the message over channel 2 is again put into the pending state. When activated, it checks the message sent from TASK2 and contained in AC3. If TASK2 input was terminated by a carriage return, TASK1 types "<LF> IS" and if not, it types "IS". TASK1 then initiates a parallel TASK (TASKØ) at priority 1ØØ (octal) to wait ten seconds before requesting another name from the user. After TASKØ is initiated, execution of TASK1 resumes and outputs the characters of the name entered by the user in the reverse order to which they were entered. After the output is complete, a .QUIT instruction is executed to terminate the TASK.

Included in the following pages are a program flow chart, program listing, and a copy of the program output.

DEMONSTRATION PROGRAM NO. 1



```
                        ;REAL TIME OPERATING SYSTEM
                        ;DEMONSTRATION PROGRAM # 1

                        .TITL DEMO1

                        .ENT START
                        .EXTN .IOX,.PTY,.QUIT,.BRK
                        .EXTN .RCV,.XMIT,.WAIT,.FORK
                        .EXTN INIT

                ;INITIALIZATION TASK

                        .NREL

000000'177777   START:  .PTY            ;SET INITIAL PRIORITY
000001'000100           100             ;TO 100(8)

000002'177777           .FORK           ;CREATE RESTART TASK
000003'000010           10              ;PRIORITY 10(8)
000004'000017'          RSTRT

000005'000002'          .FORK           ;CREATE PARALLEL TASK
000006'000000           0               ;AT SAME PRIORITY (100(8))
000007'000026'          TASK1

000010'177777           .IOX            ;OUTPUT MESSAGE TO TTY
000011'000000           0               ;"<15><12><12>RTOS TEST"
000012'100000           OUTP            ;TERMINATED BY NULL CHARACTER
000013'000000=          MESS1*2
000014'000100           100
000015'000021'          ERROR

000016'177777           .QUIT           ;TERMINATE INITIAL TASK
```

```
↑↑↑      DEMO1

                    ;SYSTEM RESTART TASK

 00017'177777  RSTRT:  .BRK              ;BREAK CHARACTER IS CONTROL C
 00020'000003          3                 ;IT WILL RESTART SYSTEM

                    ;ERROR HANDLING ROUTINE (RESTART)

 00021'062677  ERROR:  IORST             ;ERROR ROUTINE
 00022'002401          JMP @.+1          ;RE-INITALIZE SYSTEM
00023'177777          INIT

                    ;DELAY TASK (WAIT TEN SECONDS BEFORE REDOING)

 00024'177777  TASK0:  .WAIT             ;WAIT 10 SECONDS
 00025'001750          1000.             ;BEFORE RESTARTING TASK1

                    ;MESSAGE OUTPUT TASK

 00026'000005' TASK1:  .FORK             ;CREATE PARALLEL TASK
 00027'000140          140               ;AT PRIORITY 140(8)
00030'000114'         TASK2

 00031'000010'         .IOX              ;OUTPUT MESSAGE
 00032'000000          0                 ;"<15><12>ENTER NAME :-"
 00033'100000          OUTP              ;TO PROMPT OPERATOR
 00034'000026=         MESS2*2
 00035'000100          100
 00036'000021'         ERROR

 00037'177777          .RCV              ;WAIT FOR MESSAGE FROM TASK2
00040'000002          2                 ;TRANSMITTED IN AC3

 00041'024412          LDA 1,CR          ;TEST IF INPUT TERMINATED
 00042'166404          SUB 3,1,SZR       ;BY CARRIAGE RETURN
 00043'000411          JMP NOLF          ;NO

 00044'000031'         .IOX              ;PRINT "<LF> IS "
 00045'000000          0
 00046'100000          OUTP
 00047'000050=         MESS3*2
 00050'000100          100
 00051'000021'         ERROR
 00052'000410          JMP REVER

 00053'000015  CR:     15                ;CARRIAGE RETURN CHARACTER

 00054'000044' NOLF:   .IOX              ;PRINT " IS "
 00055'000000          0
 00056'100000          OUTP
 00057'000051=         MESS3*2+1
 00060'000100          100
 00061'000021'         ERROR
```

```
00062'020055-  REVER:  LDA  0,LAST      ;SETUP TO REVERSE BUFFERS
00063'040020           STA  0,INC1
00064'040021           STA  0,INC2

00065'102400           SUB  0,0         ;ZERO THE OUTPUT BUFFER
00066'024056-          LDA  1,TEN
00067'042020           STA  0,@INC1
00070'125404           INC  1,1,SZR
00071'000776           JMP  .-2

00072'020054-          LDA  0,FIRST     ;REVERSE THE INPUT BUFFER
00073'040030           STA  0,DEC1      ;PUTTING THE RESULTS
00074'024056-          LDA  1,TEN       ;IN THE OUTPUT BUFFER
00075'022030           LDA  0,@DEC1
00076'101004           MOV  0,0,SZR
00077'042021           STA  0,@INC2
00100'125404           INC  1,1,SZR
00101'000774           JMP  .-4


00102'000026'          .FORK            ;CREATE PARALLEL TASK
00103'000100           100              ;TO START CYCLE AGAIN
00104'000024'          TASK0            ;IN TEN SECONDS

00105'000054'          .IOX             ;OUTPUT THE ENTERED NAME
00106'000000           0                ;IN REVERSE ORDER
00107'120000           OUTP+WORD
00110'000041-          OUTPUT
00111'000100           100
00112'000021'          ERROR

00113'000016'          .QUIT            ;TERMINATE TASK1

                ;DATA INPUT TASK

00114'020054-  TASK2:  LDA  0,FIRST     ;ZERO INPUT BUFFER
00115'040030           STA  0,DEC1
00116'102400           SUB  0,0
00117'024056-          LDA  1,TEN
00120'042030           STA  0,@DEC1
00121'125404           INC  1,1,SZR
00122'000776           JMP  .-2

00123'000105'          .IOX             ;INPUT UP TO TEN
00124'000000           0                ;CHARACTERS FROM THE TTY
00125'022000           EVEN+WORD        ;EVEN PARITY, WORD FORMAT
00126'000027-          INPUT
00127'000012           10.
00130'000021'          ERROR

00131'161000           MOV  3,0         ;AC0= RETURN AC3 VALUE
00132'177777           .XMIT            ;TRANSMIT MESSAGE TO TASK1
00133'000002           2

00134'000113'          .QUIT            ;TERMINATE TASK2
```

```
                        ;MESSAGE AND DATA STORAGE AREA OF PROGRAM

                                .ZREL

                        MESS1:  .TXT !<15><12><12>RTOS TEST PROGRAM!
00000-005015
00001-051012
00002-047524
00003-020123
00004-042524
00005-052123
00006-050040
00007-047522
00010-051107
00011-046501
00012-000000

                        MESS2:  .TXT !<15><12>ENTER NAME :- !
00013-005015
00014-047105
00015-042524
00016-020122
00017-040516
00020-042515
00021-035040
00022-020055
00023-000000

                        MESS3:  .TXT !<12> IS !
00024-020012
00025-051511
00026-000040

      000012    INPUT:  .BLK 10.          ;INPUT BUFFER
      000012    OUTPUT: .BLK 10.          ;OUTPUT BUFFER
00053-000000            0
00054-000041-   FIRST:  OUTPUT            ;INPUT BUFFER ADDR.+1
00055-000040-   LAST:   OUTPUT-1          ;OUTPUT BUFFER ADDR.-1
00056-177766    TEN:    -10.              ;BUFFER LENGTH (NEGATIVE)

      002000            EVEN=   2000      ;EVEN PARITY
      100000            OUTP=   100000    ;OUTPUT OPERATION
      020000            WORD=   20000     ;WORD FORMAT
      000020            INC1=   20        ;AUTO-INCREMENTING
     000021            INC2=   21        ;REGISTERS
      000030            DEC1=   30        ;AUTO-DECREMENTING REG.

      000021'           .END ERROR
```

```
---
CR          000053'
DEC1        000030
ERROR       000021'
EVEN        002000
FIRST       000054-
INC1        000020
INC2        000021
INIT        000023'X
INPUT       000027-
LAST        000055-
MESS1       000000-
MESS2       000013-
MESS3       000024-
NOLF        000054'
OUTP        100000
OUTPU       000041-
REVER       000062'
RSTRT       000017'
START       000000'
TASK0       000024'
TASK1       000026'
TASK2       000114'
TEN         000056-
WORD        020000
.BRK        000017'X
.FORK       000102'X
.IOX        000123'X
.PTY        000000'X
.QUIT       000134'X
.RCV        000037'X
.WAIT       000024'X
.XMIT       000132'X
```

DEMONSTRATION PROGRAM NO. 1

Operation

```
RTOS TEST PROGRAM
ENTER NAME :- RTOS TEST  IS  TSET SOTR
ENTER NAME :- DEMO NO. # IS # .ON OMED

RTOS TEST PROGRAM
ENTER NAME :- DEMO NO.

RTOS TEST PROGRAM
ENTER NAME :- DEMO NO 12 IS 21 ON OMED
ENTER NAME :-

RTOS TEST PROGRAM
ENTER NAME :-       IS

RTOS TEST PROGRAM
ENTER NAME :- BACKWARDS IS SDRAWKCAB
ENTER NAME :- FORWARDS IS SDRAWROF
ENTER NA

RTOS TEST PROGRAM
ENTER N

RTOS TEST PR

RTOS TEST PROGRAM
ENTER NAME :- ABCDEFGHIJ IS JIhGFEDCBA
ENTER NAME :- ABCDEFGHIJ IS JI

RTOS TEST PROGRAM
ENTER NAME :-  IS
ENTER NAME :- EVE IS EVE
ENTER NAME :- ABCDEFGh
 IS hGFEDCBA

RTOS TEST PROGRAM
ENTER NAME :-
```