REAL-TIME EXECUTIVE

RTX4

USER'S MANUAL

93410-90

FEBRUARY 1979

ComputerAutomation™

Engineering Reference Number
90-93410-00C0

NAKED MINI Division
HERTFORD HOUSE
DENHAM WAY, MAPLE CROSS
RICKMANSWORTH, HERTFORDSHIRE WD3 2XD

Telephone Rickmansworth 71211 Telex 922654

REVISION HISTORY

Revision	Issue Date	Comments
A0	May 1977	Original Issue
A1	August 1977	Miscellaneous revisions to manual
A2	October 1977	Miscellaneous revisions to manual
A3	October 1977	Revisions to demonstration program appendix
B0	January 1978	Additions to system services; changes to system tables
B1	April 1978	Production release
B2	October 1978	Clock option added, other minor revisions; manual reformatted and reorganized.
B3	December 1978	MDBUG4 and XDBUG4 options added; minor documentation errors corrected. Additions to rescheduling.
C0	February 1979	Production release; incorporating all previous changes.

PREFACE

This manual describes Computer Automation's Real-Time Executive (RTX4), a package of software modules designed to provide the overhead functions and scheduling services associated with a real-time, multi-tasking environment.

This manual is intended to serve both as a learning tool for programmers, new to RTX4, and as a reference source for experienced users. Section 1 provides an introduction to RTX4 and its use. System initialization, the first operation performed by the program, is discussed in Section 2. The fundamental concepts of tasks, activities, environment, and semaphores are elaborated in Sections 3-6. Special facilities provided to RTX4 users -- system clocks and the mailbox -- are described in Sections 7 and 8. The appendices provide a glossary, descriptions of the RTX4 tables, detailed information on the RTX4 services, M4D12 addressing format, RTX4 exceptions, configuration options, and a demonstration program.

The following Computer Automation Incorporated documents provide information related to the use of RTX4:

- OS4 System User's Manual (93460-90)
- NAKED MINI 4 Assembler User's Manual (93500-80)
- Input/Output Subsystem IOS4 User's Manual (93430-90)
- NAKED MINI 4 Debugging Monitor Reference Manual (93015-90)
- LSI 4/10, 4/30, or 4/90 Computer Reference Manual (20990-91, 20991-91, or 20945-91, respectively)

Contact your CAI representative for copies of these documents or any other CAI documents.

CONTENTS

APPENDIXES

FIGURES

TABLES

SECTION 1

BASIC CONCEPTS OF REAL-TIME SYSTEMS

1.1 INTRODUCTION

Computer Automation's Real-Time Executive (RTX4) provides a real-time, multi-tasking environment for user-written applications.

This introductory section discusses some of the concepts that are fundamental to real-time systems in general. Subsequent sections of this manual describe the use of RTX4 in particular.

1.2 AN ANALOGY

The following analogy of an auto repair garage and its activities, illustrated in Figure 1-1, introduces the basic concepts of a real-time system and the functioning of its component parts.

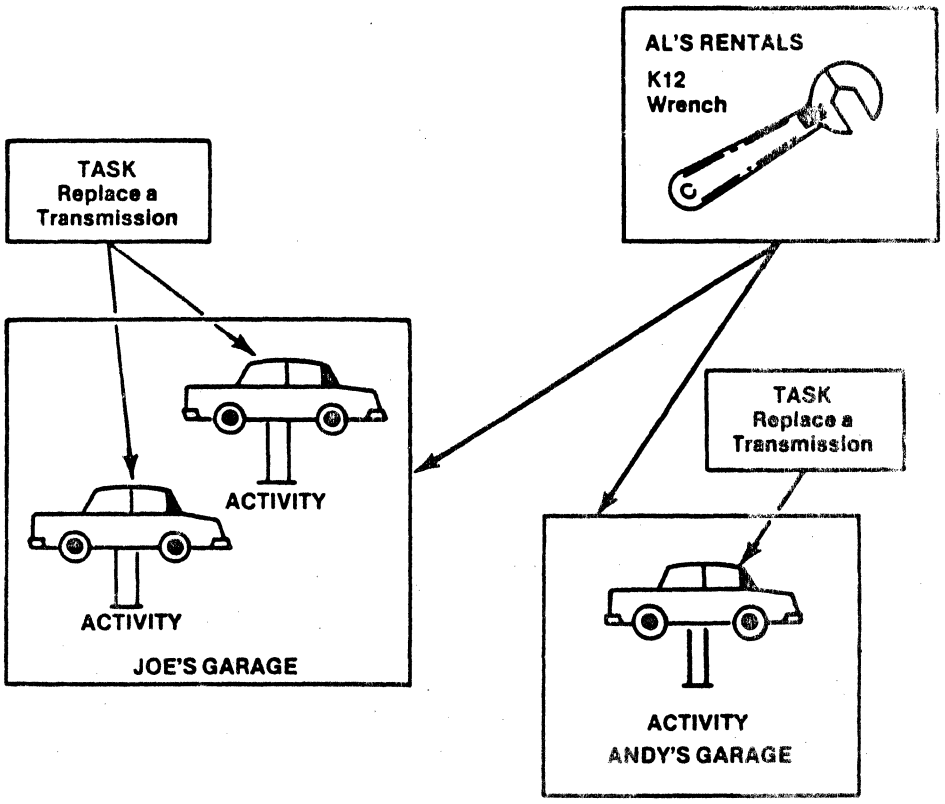


Figure 1-1. Analogy of a Real-Time System

Assume that all mechanics change transmissions in all cars the same way. We may think of "replacing a transmission" as a task. It consists of a series of well-defined steps using a certain set of tools. At many garages there is a fixed price for this task, and many mechanics know how to do it. The task of "changing a transmission" is independent of whether anyone is doing it; that is, if no one anywhere is replacing a transmission, there is still a well-defined task called "replacing a transmission."

There may be a place known as "Joe's Garage." It is an "environment" in which many mechanical tasks are performed. Joe tells his workers to perform mechanical tasks. In fact, he orders his task requests in a particular manner, probably one which maximizes his profits. Although he is in complete control of his own garage, he has no authority in Andy's Garage, across the street. He cannot tell Andy's mechanics what to do, even though he may know as well as Andy what needs to be done. There is no wall or other physical restraint between their shops, but the law forbids Joe to exercise authority in Andy's garage. His interaction with Andy's Garage is limited: he may exchange messages with Andy or his mechanics, or even have his own car fixed there.

If Joe tells one of his mechanics to replace a transmission, he has started an "activity," or an instance of the task "replacing a transmission." He may tell two mechanics to change two different transmissions at the same time. Then there would be two such activities being performed. In fact, Andy may request one of his mechanics to replace one at the same time, so the same task is being performed in two separate environments. Each instance of the task is an activity. Since all three mechanics can work concurrently, this situation describes a concurrently reentrant task.

Suppose that replacing a transmission requires a K12 wrench. The only K12 wrench in town is at Al's Rentals and must be reserved ahead of time. Then only one activity of "replacing a transmission" can be performed at one time. The multiple activities which Joe and Andy requested must be performed serially. This may slow down their shops' throughput, but if K12 wrenches are very expensive (as they appear to be, since neither Joe nor Andy has one), then this might be the most effective way to solve the problem.

Three terms are introducted in this analogy:

- A task is a set of rules, instructions and resources. It is generally created once, perhaps occasionally updated. It can be concurrently or serially reusable. A task which is concurrently reusable is said to be reentrant. A task's reentrancy is determined by its creator (and updater), since he knows what resources are required by it.

- An activity is a specific instance of performing a task. It uses three resources: CPU time, a task, and a context. All are allocated to the task when it begins. There may be many activities being performed at the same time, of the same or different tasks.

- An environment is a set of resources. A task may be performed in several environments, but each instance or activity can use only the resources that exist in the environment in which it is started. Thus, environment is a method of resource allocation.

A glossary of these and other terms used in this manual appears in Appendix A.

1.3 INTERRUPT PROCESSING

A real-time system must respond to external events that occur asynchronously to processes within the computer. In this usage, "asynchronously" means essentially the same thing as "randomly," but the events are not truly random because various aspects of them are predictable (maximum interval, time window, relative sequence, etc.). However, their exact timings are not known, so real-time programs cannot afford to wait for them by looping. The accepted method for relating the outside world to a computer is through hardware interrupts. The computer can perform other functions until the interrupt occurs. Interrupts allow the effective and efficient utilization of a computer in a real-time environment.

However, current computer architecture and programming conventions are sequentially oriented, and interrupts make it impossible for an entire application to be seen as a single sequential process. Every time an interrupt occurs the order of execution of computer instructions is changed, if only temporarily.

Figure 1-2 illustrates how conceptual execution transfers when interrupts occur. In this example, two types of interrupts (perhaps from different devices) occur at two different times during the processing of the mainline program. Each time, execution is shifted from the mainline, to the interrupt, to the routine to process that particular type of interrupt, then back to the mainline program.

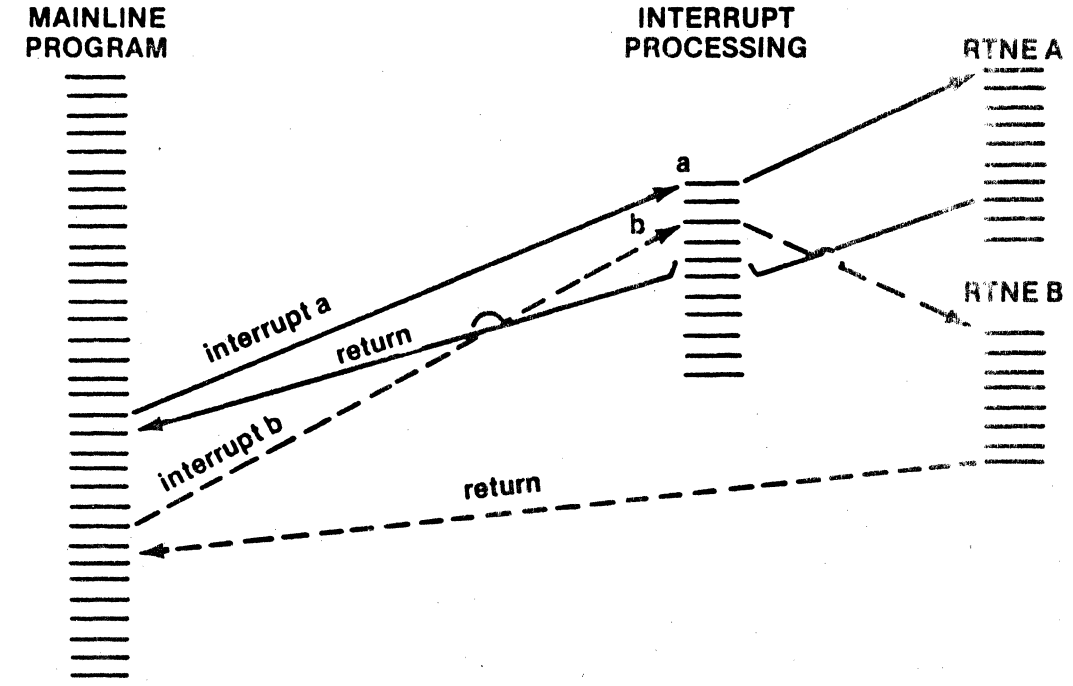


Figure 1-2. Real-Time Interrupt Processing Concepts

The problem can be simplified by viewing the system as two or more sequential processes that execute concurrently. This ideal can be met by providing the programmer with two or more separate computers that can communicate with each other through status flags. Currently, this approach is economically unfeasible.

A real-time executive can simulate the existence of multiple processors, allowing the programmer to treat his application as a collection of sequential processes, yet requiring only one CPU. Some CPU time and some memory are occupied with the overhead of this simulation, but the incremental costs of additional CPU time and memory in a single computer are relatively small.

1.4 PROGRAMMING BY FUNCTIONS

One way to optimize the use of a real-time executive is by subdividing a program into its component functions. Each function can process when it is required and when the necessary hardware and other environmental factors are available.

For example, consider a small program that reads from cards and prints their contents on a line printer using two buffers so that the reading and printing operations can overlap in time. The asynchronous outside events in this case are the completion of reading a card and the completion of printing a line. They are asynchronous not only because of small variations in the mechanical devices, but because of human factors such as reloading the card hopper, emptying the card stacker, changing printer paper, etc.

It is possible to solve this problem using standard sequential programming techniques or primitive interrupt responses. A simple alternative solution is shown in Figure 1-3. This solution involves two processes: Process A reads cards into the buffers, and Process B prints the contents of the buffers. Each process consists of seven steps. The first six steps are standard RTX4 system service requests consisting of two words of memory each. Step 7 is a jump to the start of the process so the process is repeated. Most importantly, the flow of processing can be followed easily; there are no conditional tests, no branching.

Four system services are requested. "Read" and "print" are calls to the Input/Output Subsystem[1] that return only when the operation is complete. "Signal" and "wait" are complementary synchronization services: a process that waits for a condition resumes as soon as it is signalled. This service is known as a semaphore. Semaphores are discussed in a later section.[2]

No combination of external events can result in a card not being printed or being printed twice. Note how easy it is to verify that fact. Also, Process A and Process B could just as well be in separate computers with some shared memory. That is the advantage of a real-time executive: it provides concurrent processing without the hardware cost of multiple processors. Whenever the card reader is empty or both processes are waiting, other processes can use the CPU time.

---

[1] Input/Output Subsystem IOS4 User's Manual
[2] Section 6

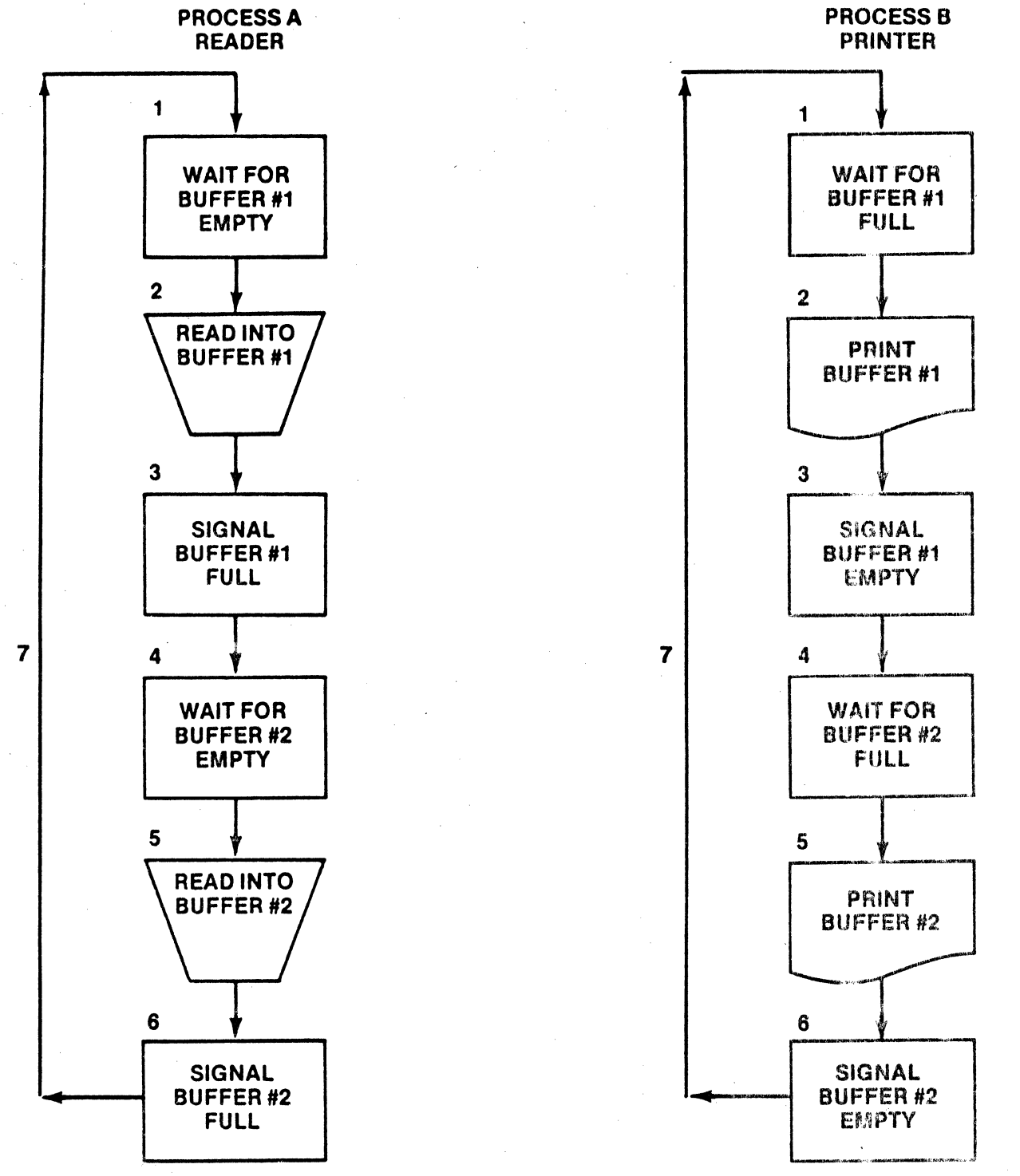


Figure 1-3. Example Processes

SECTION 2

RTX4 USAGE

2.1 INTRODUCTION

RTX4 is a package of software modules designed to provide the overhead functions and scheduling services associated with a real-time, multi-tasking environment. IOS4[1] is a subsystem of RTX4 which provides the user with a device-independent method of I/O device management and support.

The general organization of RTX4 and IOS4 is shown in Figure 2-1. RTX4 controls all aspects of priority scheduling, timing, interrupt servicing, I/O control, inter-task communication, and all necessary queuing functions. Modular construction allows the user to select only those portions of RTX4 required for his particular application.

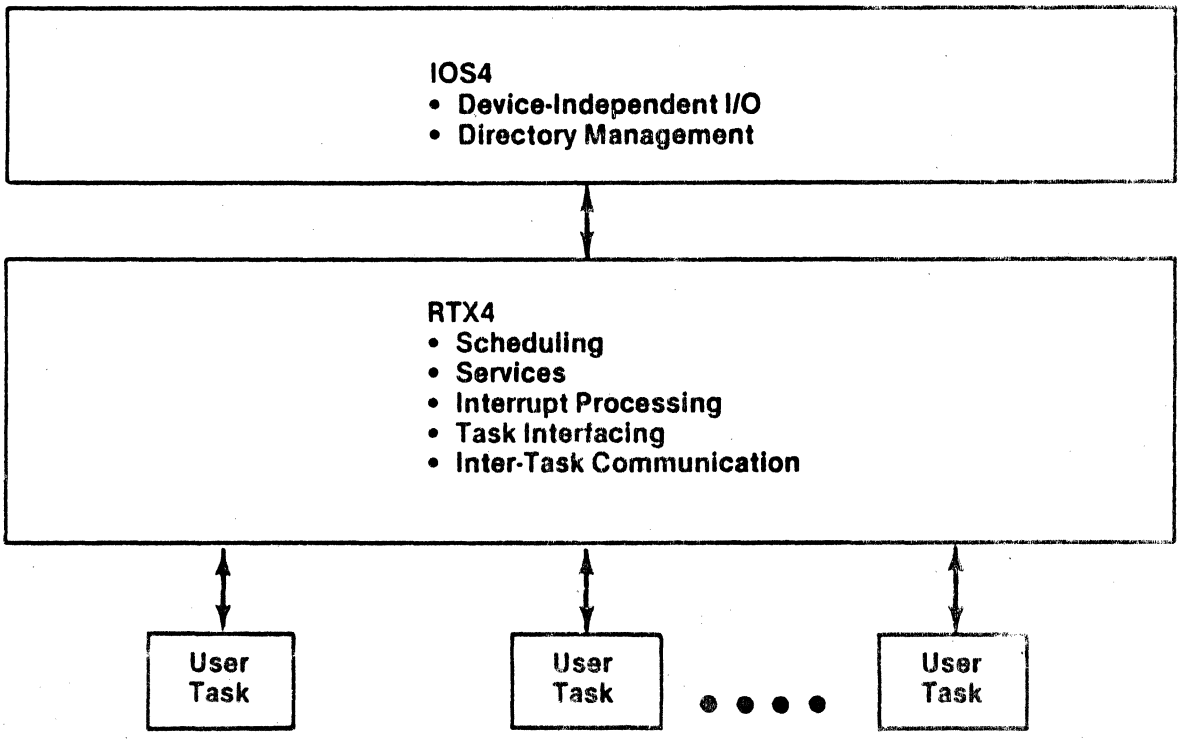


Figure 2-1. RTX4 Organization

[1] Input/Output Subsystem IOS4 User's Manual

Some of the particular features of RTX4 are:

- Allows the application program to be designed as a number of interrelated or subordinate tasks.
- Allows the application program to schedule CPU usage by dynamically defining and redefining the priority and seniority levels of several activities in the application using RTX4 service routines.
- Allows real-time response to external events by providing the interrupt instructions which transfer control to the interrupt service routines.
- Allows the various tasks in the application to communicate between themselves (or with RTX4) through RTX4 communication routines.
- Provides clock services for obtaining time-of-day information and/or for controlling the timing of activities.
- Allows dynamic memory management.

2.2 RTX4 SYSTEM SOFTWARE

The RTX4/IOS4 system software is available on floppy diskettes or on paper tapes. The floppy diskettes are intended for use with the OS4 program development system.[1] The paper tapes are provided for users of the OMEGA4 program development system.[2]

The contents of the product diskettes and paper tapes are discussed below. The procedures for using each medium in its corresponding program development system are outlined later in this section.[3]

2.2.1 System Software Diskettes

When RTX4 is to be used with OS4, it is delivered as five floppy diskettes, as follows:

- RTX4 Product Diskette — Diskette ID Number: F41001; CAI Part Number: 93410-01

  This diskette contains the RTX4 library file (RTX.LIB), the RTX4 demonstration program source, object, and binary files, and a JCL file for assembling the demonstration program. (A listing of the demonstration program appears in an appendix[4] to this manual.) A Help file describes the diskette's contents.

---

[1] OS4 System User's Manual
[2] OMEGA4 System User's Manual
[3] Subsection 2.4
[4] Appendix E

- RTX4 Macros Diskette — Diskette ID Number: F42501; CAI Part Number: 93425-01

  This diskette contains the user and development macro files. These files are described in Appendix H.

- IOS4 Product Diskette — Diskette ID Number: F43001; CAI Part Number: 93430-01

  This diskette contains the IOS4 library file (IOS.LIB).

- SFM Product Diskette — Diskette ID Number: F44001; CAI Part Number: 93440-01

  This diskette contains the SFM library file (SFM.LIB), the SFM demonstration program source, object, and binary files, and a JCL file for assembling the demonstration program source file.

- Standalone LABEL Diskette — Diskette ID Number: F44101; CAI Part Number: 93441-01

  This diskette contains the standalone OS4 disk labeling program. This program can be used to label disks in SFM format. Its use is described in the IOS4 User's Manual.[1]

2.2.2 System Software Paper Tapes

When RTX4 is to be used with OMEGA4, it is delivered as a set of paper tapes. These tapes contain the same items as the floppy diskettes described above (minus Help and JCL files and the standalone labeling program) except that each file is provided on a separate paper tape. Each paper tape has its own CAI part number, as follows:

CAI Model Number	File
93410-20	RTXDEMO.ASM
93410-30	RTXDEMO.OBJ
93410-40	RTXDEMO.BIN
93410-51	RTX.LIB
93420-60	GEN.MAC
93420-61	RTX.MAC
93420-62	RTXD.MAC
93420-63	IOS.MAC
93420-64	IOSD.MAC
93420-65	SFM.MAC
93420-66	SFMD.MAC
93430-51	IOS.LIB
93440-20	SFMDEMO.ASM
93440-30	SFMDEMO.OBJ
93440-40	SFMDEMO.BIN
93440-50	SFM.LIB

[1] Input/Output Subsystem IOS4 User's Manual

2.3 RTX4 MACROS

RTX4 provides three types of macros: macros for generating internal tables, macros for requesting system services, and macros which generate service request parameter lists.

2.3.1 Table-Generating Macros

RTX4 involves a number of internal tables. RTX4 generates some of these tables automatically; others are generated in response to macros defined by the user in his program. These macros and the tables they generate are listed below.

Macro	Table Generated	Purpose of Table
TDB:A	Task Descriptor Block	Describes the needs and attributes of a task.
INIT:A	Initialization Block	Provides some information about the environment and supplies the address of the first task to be executed.
ECB:A	Environment Control Block	Defines user-occupied space to RTX4 and unit assignment to IOS4.
SDB:A	Semaphore Definition	Defines a semaphore for controlling synchronization of Block tasks.
MDB:A	Mailbox Definition Block	Defines a mailbox facility for communication between activities.

These macros are described in detail in subsequent sections of this manual. The structures of all RTX4 internal tables, including those listed above, are presented in an appendix.[1]

2.3.2 Service Macros

RTX4 provides a number of services which greatly simplify programming for a real-time environment. To invoke one of these services, the program executes the corresponding service request macros as listed below.

Macro	Service Invoked
R:BGIN	Initiates an execution instance of a task; i.e., creates an activity.
R:END	Completes an execution instance of a task; i.e., terminates an activity.

---

[1]Appendix B

Macro	Service Invoked
R:GPRI	Reads an activity's priority.
R:SPRI	Changes an activity's priority.
R:SATD	Given the ASCII time and date will set in binary.
R:GATD	Reads, in binary, the time and date and converts to ASCII.
R:CINT	Causes the current activity to return on a console interrupt.
R:ABUF	Allocates a buffer.
R:RBUF	Releases a buffer.
R:SIG	Signals a semaphore.
R:WAIT	Waits on a semaphore.
R:ITIC	Initiates a timer to cause a semaphore to be signalled after a specified number of Real-Time Clock ticks.
R:MTIC	Modifies a previously-initiated tick clock timer request.
R:CTIC	Cancels a previously-initiated tick clock timer request.
R:PAUS	Drops the seniority of an activity.
R:AWAL	Initiates a timer to cause a semaphore to be signalled at an absolute time.
R:IWAL	Initiates a timer to cause a semaphore to be signalled after a specified time interval has elapsed.
R:CWAL	Cancels a previously-initiated wall clock timer request.
R:STOD	Sets the binary time of day.
R:GTOD	Reads the binary time of day.
R:SEND	Sends a message from one task to another.
R:RECV	Receives a message sent by another task.

These macros are described in detail in subsequent sections of this manual.

The arguments to system requests are sometimes defined as values and sometimes defined as pointers to lists of values. For instance, in the R:BGIN request, the argument is a pointer to a list of parameters needed for the R:BGIN service. One of those parameters is a priority descriptor which can be expressed as a 16-bit integer.

Compare this to the R:SPRI request, whose argument is a priority descriptor rather than a pointer to one. The priority descriptor may be placed in the second word of the service request, or in the X register when the service request is made. If the descriptor is to be placed in some other memory location, it must be referenced indirectly.

When arguments to a service request macro must be specified in a list rather than directly in the macro, the programmer can call the appropriate list-generating macro. These macros are:

BGIN:A generates an argument list for a BGIN:A request.

MAIL:A generates an argument list for an R:SEND or R:RECV request.

TICK:A generates an argument list for an R:ITIC, R:MTIC, or R:CTIC request.

WALL:A generates an argument list for an R:AWAL, R:IWAL, or R:CWAL request.

These macros are described with the corresponding request macros in subsequent sections of this manual.

All service request arguments, whether specified directly in the request or in a list, are expressed in M4D12 format, i.e.:

[*]Mem(X,Y)

where:

[*] denotes an optional asterisk immediately preceding the memory address to indicate indirect addressing.

Mem is a memory address in the range 0-65535 inclusive or external.

(X,Y) denotes indexing, which is always optional and may specify either X or Y or both, in either order, separated by a comma, and the whole enclosed in parentheses.

This format permits a wide range of addressing modes. In simple systems, the direct and indirect modes may satisfy all programming needs. In more complex systems, a programmer may wish to place his argument pointer or value in a register or in his Y-scratchpad. These last two options are especially useful in writing reentrant tasks. The allowed addressing modes are:

- Direct addressing to anywhere in memory
- Indirect addressing to anywhere in memory
- P-relative addressing to within ± 4096 words of the current P register value
- Pre- or post-indexing with an offset of ± 4096
- A combination of the above

The following are addressing mode examples:

ABLE	direct reference to a label
*BAKER	indirect reference through a label
CHARLY(X)	post-indexed reference
DELTA(Y)	pre-indexed reference
ECHO(X,Y)	pre- and post-indexed reference
*FXTRT(X,Y)	indexed indirect reference

See the assembler manual[1] for more information on M4D12 format.

Macros are the preferred form for making service requests. Alternatively they can be made using a system trap (STRAP) instruction. The STRAP instruction in the first word of a service request traps to location :A4 where RTX4 proceeds to process the request. The first word also specifies the service being requested and the meaning of the second word. The second word can contain a value, an address, an indirect pointer, or a complex M4D12 pointer, depending on the service request and the contents of the first word. Together, these two words provide a simple, yet flexible, means of requesting services and providing argument values and lists.

2.4 RTX4/IOS4 PROGRAM DEVELOPMENT

The general procedures for developing an RTX4/IOS4 application program are like the procedures for developing any other type of user program: the programmer designs his program, creates the appropriate symbolic text, translates that symbolic text to an object module or modules via an assembler or compiler, links the object module(s) with any required library programs, loads and executes the linked program, and performs any necessary debugging.

The RTX4/IOS4 programmer can perform these processes in either the OS4 system or the OMEGA4 system. The OS4 and OMEGA4 user's manuals[2] provide details on how program development procedures are performed in those systems. This subsection presents some guidelines that apply to developing an RTX4/IOS4 application program in particular.

The OS4 user's manual outlines a suggested procedure for creating an RTX4/IOS4 application development system. For the reader's convenience, this discussion is repeated in an appendix[3] to this manual.

---

[1] NAKED MINI 4 Assembler User's Manual
[2] OS4 System User's Manual and OMEGA4 System User's Manual
[3] Appendix E

2.4.1 Designing the Program

The first step in designing an RTX4/IOS4 application program is to divide the problem into a suitable number of tasks. A task, as introduced in the auto repair shop analogy presented earlier,[1] is a set of instructions for performing a particular function. For example, the two processes shown in Figure 1-3[2] are tasks to perform the functions of reading and printing cards.

An application system can consist of one or more tasks. There is sometimes a "best" way of dividing a system into tasks, but there is seldom an "only" way. The decision to break the card reading/printing problem into the two separate tasks simplified the programming problem. Solving it as a single task would have been unnecessarily difficult; solving it as three or more tasks would be unnecessary. Figure 2-2 presents another example.

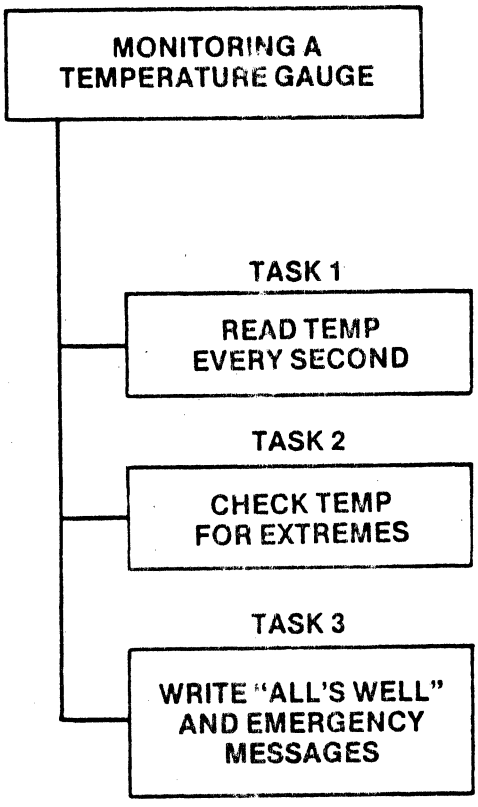


Figure 2-2. Dividing an Application into Tasks

[1] Subsection 1.2
[2] Subsection 1.3

The rules for dividing an application into tasks cannot be spelled out, unfortunately. As a guideline, whenever synchronization with another process (internal or external) requires excessive conditional testing, create a new task which performs only the synchronization. This guideline may result in a hierarchical structure of tasks, which in many situations is an excellent solution. Another way of viewing this guideline is to think of functions which, if performed in another computer, would simplify the problem in the main computer. Such functions should be performed in another task.

After dividing the problem into tasks, design the operation of each task.

2.4.2 Coding the Program

The elements of an RTX4/IOS4 can be coded in any order, but the program must include at least the following elements:

- Initialization Block[1]

  This table should be the first element of the program, as it provides information required by RTX4 when execution begins. It directs RTX4 to the first task to be executed and to the Environment Control Block (ECB). As described below, the ECB describes the environment in which the program will run. The Initialization Block also determines the size of the System Freepool. As described later,[2] the Freepool is a region of memory that RTX4 uses for its internal tables and control blocks. To generate the Initialization Block, the programmer codes an INIT:A macro.

- Task Descriptor Block(s)[3]

  For each task defined in the program, the programmer generates a Task Descriptor Block (TDB) by coding a TDB:A macro. In general, the macro call should be near the code of the task it describes. While not required by RTX4, this approach minimizes the number of external references required.

- Environment Control Block[4]

  The Environment Control Block (ECB) describes the program's resources to RTX4 and unit assignment to IOS4. The ECB also contains the heads of several lists generated by the program.

If the user wishes to use a nonstandard Unit Assignment Table, he must include the appropriate UAT:AA, UAT:EE, and UAT:ZZ macros.[5]

---

[1]Subsection 5.2
[2]Subsection 5.3
[3]Subsection 3.4
[4]Subsection 5.4
[5]Input/Output Subsystem IOS4 User's Manual

The programmer can code his program as a single module or as multiple modules. A convenient modular structure for large application programs is to code each task, including its Task Descriptor Block (TDB:A macro), as a separate module. Only one module must include the Initialization Block (INT:A macro) and the last module to be loaded must include the Environmental Control Block. (ECB:A macro) at the end of that module.

The programmer normally includes the directive:

```
LOAD      DEBUG4
```

when coding a new RTX4/IOS4 application program. This directive causes the DEBUG4 system[1] to be loaded with the program, providing facilities for debugging the program. This directive can appear in any program module.

Figure 2-3 illustrates the typical structure of an RTX4/IOS4 application program.

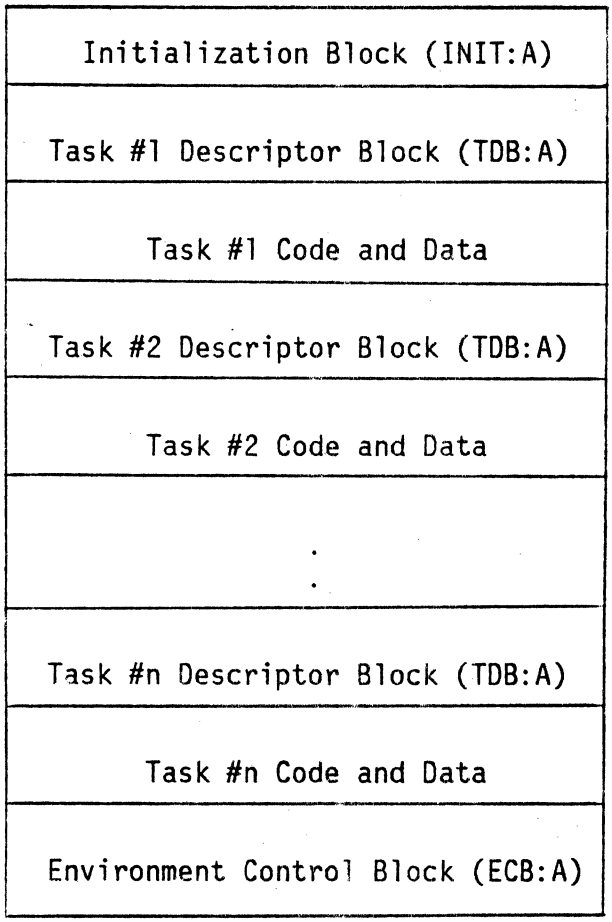


Figure 2-3. User Program Structure

[1] NAKED MINI 4 Debugging Monitor Reference Manual

2.4.3 Assembling the Program

RTX4/IOS4 application program modules can be assembled in any order. The files GEN.MAC, RTX.MAC, IOS.MAC, and SFM.MAC must be specified as the macro definition file for each assembly.

2.4.4 Linking the Program

The next step is to link all of the user-coded modules with the necessary library files.

NOTE

The module containing the Environment Control Block (ECB) must be the last program module linked because it must contain the heads of several lists generated by the program.

Following the user-coded modules, library files should be linked in the following order:

SFM.LIB — Provided on the SFM product diskette or paper tape; required only if the program uses Standard File Manager capabilities.

IOS.LIB — Provided on the IOS4 product diskette or paper tape; required only if the program invokes IOS4 services.

RTX.LIB — Provided on the RTX4 product diskette or paper tape; required for all RTX4/IOS4 application programs.

The program may be linked absolute or relocatable and may reside in memory at address :100, if two DIO boards are used :200, or greater.

2.4.5 Loading and Executing the Program

Once all of the program modules have been linked, the programmer can load and execute his program.

When a linked RTX4/IOS4 application program is loaded, it appears in memory as diagrammed in Figure 2.4. The area between the end of the program and the end of memory is called the Environment Memory Pool[1]. This space is used for scratch-pad and stack space requested by the program.

If DEBUG4 is loaded with RTX4, DEBUG4 receives initial control when the user's program is executed. The user can start the program by using DEBUG4 to transfer to location :80. If an exception[2] occurs, control returns automatically to DEBUG4. The user can access DEBUG4 at any time by transferring to location :7E.

---

[1]Subsection 5.5
[2]Appendix C

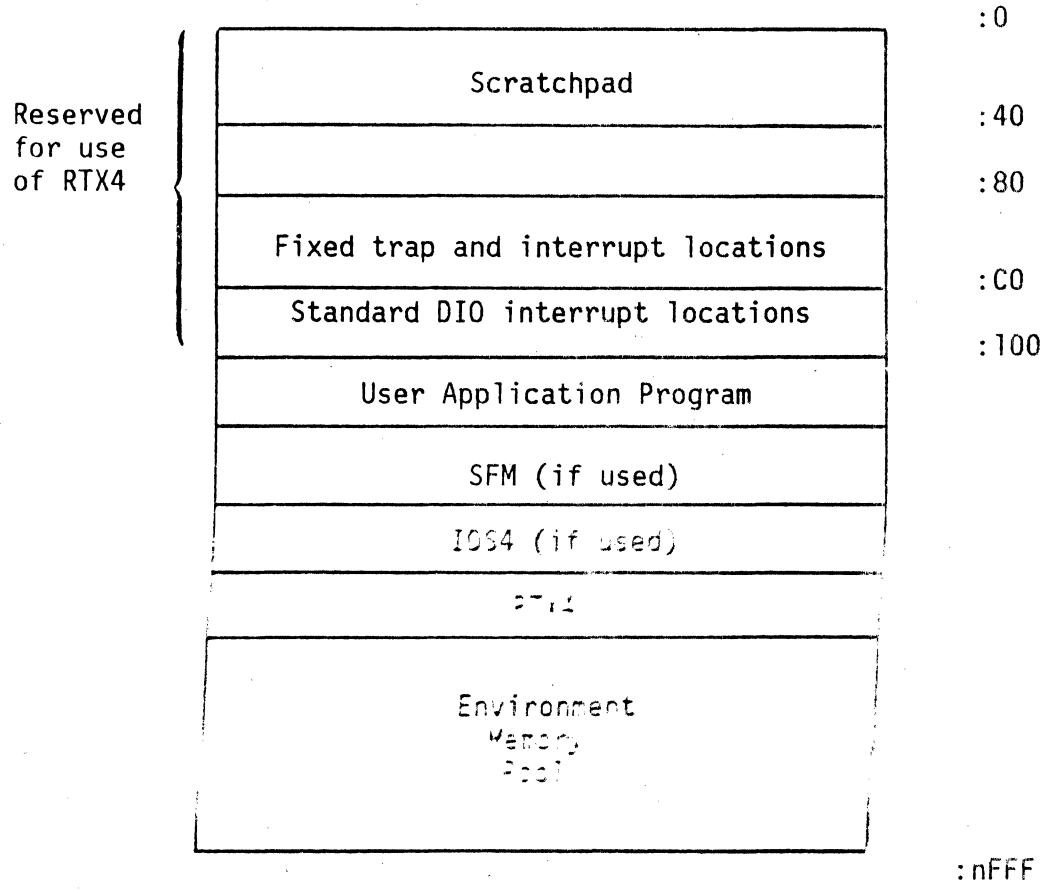


Figure 2-4. Map of All Memory

2.4.6 Debugging the Program

RTX4 contains many blocks of information on a variety of lists. The programmer can examine these lists using the Z command. The heads of these lists and the meaning of the contents of the blocks are presented on the following page.

Label	Location	Contents
R:ECBH	:20	Head of the Environment Control Block list.
R:RDY	:22	List of Activity Control Blocks currently ready for activity execution. R:ACT usually points to the first ACB on this list.
R:ACT	:21	Activity Control Block for the current activity.
R:INTQ	:23	List of Activity Control Blocks of activities which have been readied for execution and are awaiting merger into the R:RDY list the next time through the dispatcher.
R:FPH	:25	Head of the Freepool list of available blocks.
R:FPT	:26	Tail of the Freepool list of available blocks.
R:CCBH	:2B	Head of Tick Clock Control Block list.
R:WCBH	:2C	Head of Wall Clock Control Block list.
R:TODU	:30	Time of day upper 16 bits.
R:TODL	:31	Time of day lower 16 bits.

The user can also examine the system trap locations which can identify the user's last system request.

SECTION 3

TASKS

3.1 INTRODUCTION

A task is an ordered collection of machine instructions that perform a particular function.

3.2 TASK RESOURCES

Several resources are associated with a task, including the initial register context, Y-scratchpad, and the user's stack.

3.2.1 Initial Register Context

The contents of the A, Q, X, and possibly Y registers of the task which begins another task form the initial register context of the new task. The initial register context provides communication between the original task and the task to be started. It can determine the function to be performed, the location of data areas and buffers, etc. If the required information does not fit into the registers, the registers can point to memory locations which contain the information.

3.2.2 Stack

Each execution instance of a task must have its own stack. The stack is used by RTX4 for several purposes, and may be used by the programmer for subroutine linkages using the JSK and RSK instructions.[1]

The amount of stack space required for a task is the sum of the spaces required for program use and system use. The amount used by the program depends on the maximum depth of subroutine nesting (not the total number of subroutines). The amount used by the system depends on what system services are requested. If no services are requested, the system requires 14 words for handling interrupts (only 7 if a significantly higher maximum interrupt latency is acceptable). If any system services are requested, 8 more words are required. Additional stack locations are required for many services. The number of additional stack locations are listed as a part of the documentation of each service. Also, the use of JSK and PUSH requires 7 words of stack space.

---

[1] NAKED MINI 4 Assembler User's Manual

As illustrated in Figure 3-1, the K register points to the top of the stack area currently being addressed and the L register points to the lower limit of the stack.

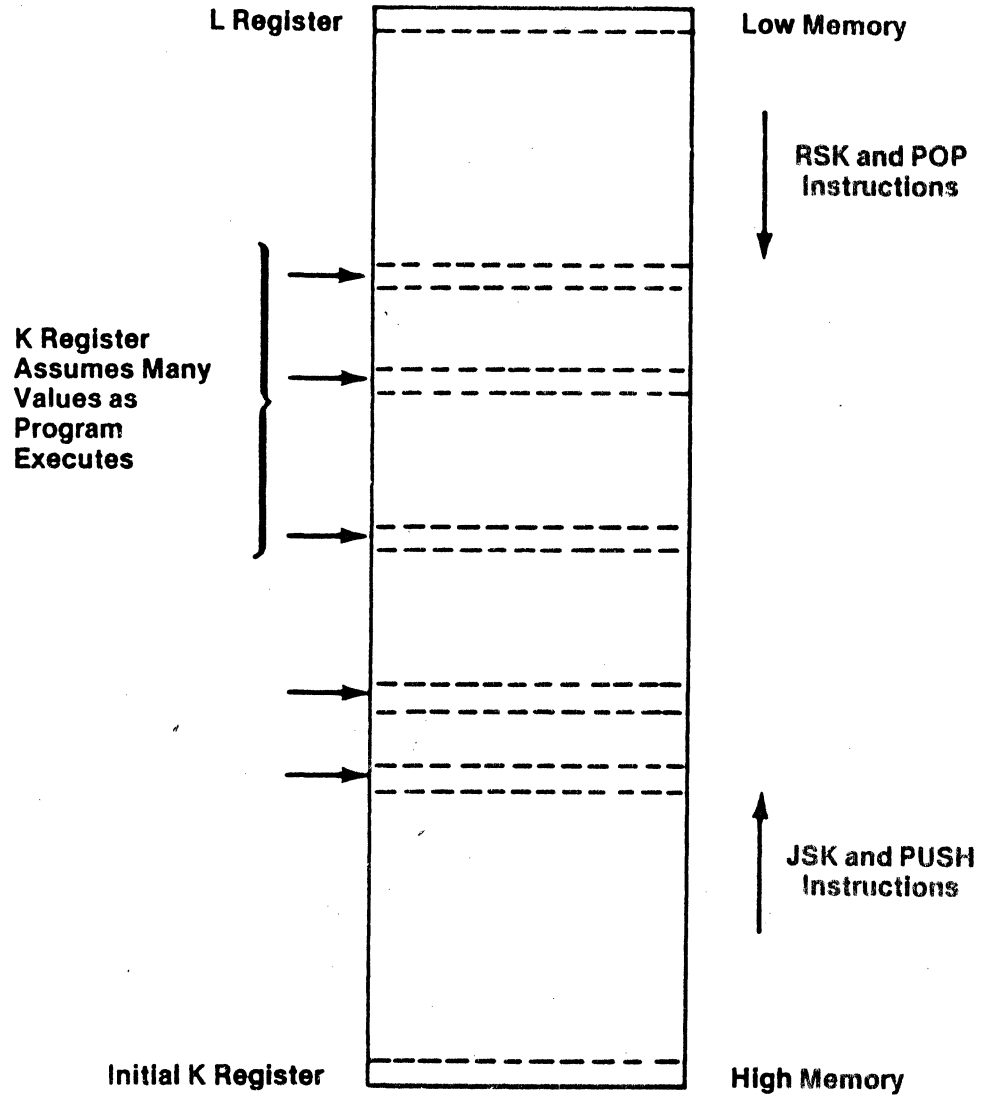


Figure 3-1. Stack

3.2.3 Y-Scratchpad

Each invocation of a task can have its own scratchpad area of any length. This area is called the Y-scratchpad because it is reached via the Y register. The size of the Y-scratchpad area is user-defined.

Although NAKED MINI 4 Family computers have a 64-word scratchpad (the first 64 words of memory), these words cannot be used by application programs in the RTX4 system. They are used by RTX4 for critical program sequences, list heads, and other uses that increase throughput and reduce overhead.

Any normal scratchpad use can be performed in Y-scratchpad, including indexed indirect references and direct references from any memory location. The Y-scratchpad is allocated to a task when it begins execution. The address is placed in the task's Y-register as part of the initial register context The program refers to locations in Y-scratchpad by including the pre-indexing symbol (Y) on operands which are to fall into Y-scratchpad.

Two options are available for allocating Y-scratchpad space. The task may use the Y-scratchpad space of the task which began it; in this case, the Y register value is simply passed as part of the initial register context along with the A, Q, and X registers. Or, the programmer may request that the Y-scratchpad space be allocated dynamically when the task is begun.

In either case, the programmer is free to load the Y register with the address of his own Y-scratchpad region.

It is the user's responsibility to avoid referencing locations which fall outside the allocated Y-scratchpad; RTX4 cannot perform any limit checking. The greatest volume of Y-scratchpad is required for reentrant or recursive programming where data areas must be allocated for each execution instance. All memory reference instructions can refer to Y-scratchpad, including single-word memory reference instructions (64-word range), system request parameters (4096-word range), and double-word memory reference instructions (65,536-word range).

Figure 3-2 illustrates Y-scratchpad allocation and how it is accessed and used in a task.

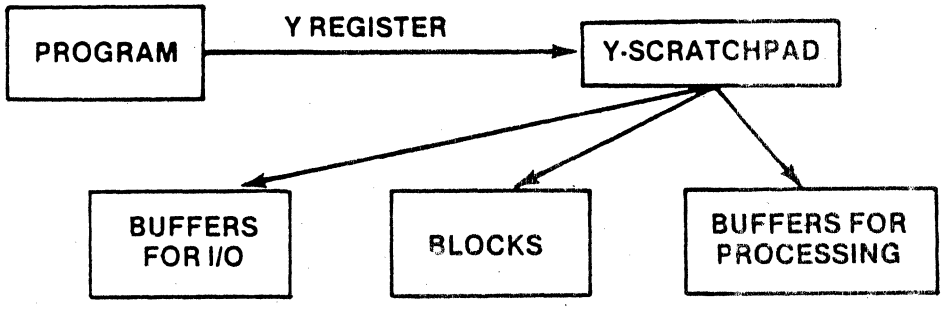


Figure 3-2. Y-Scratchpad Allocation and Access

3.3 SERIAL/REENTRANT TASKS

A task may be serial or reentrant in its use. If a task is serial, one activity (execution) of the task must be completed before the next activity can begin. A reentrant task can support several activities executing concurrently.

A reentrant program significantly reduces memory size for some applications. For example, consider a data entry system consisting of four CRT terminals connected to one computer and a disk. An operator sits at each terminal entering data from questionnaire forms into the computer.

One approach to building this system would be to create four copies of the data entry program, as illustrated in Figure 3-3. Such a solution reduces development time and speeds execution time.

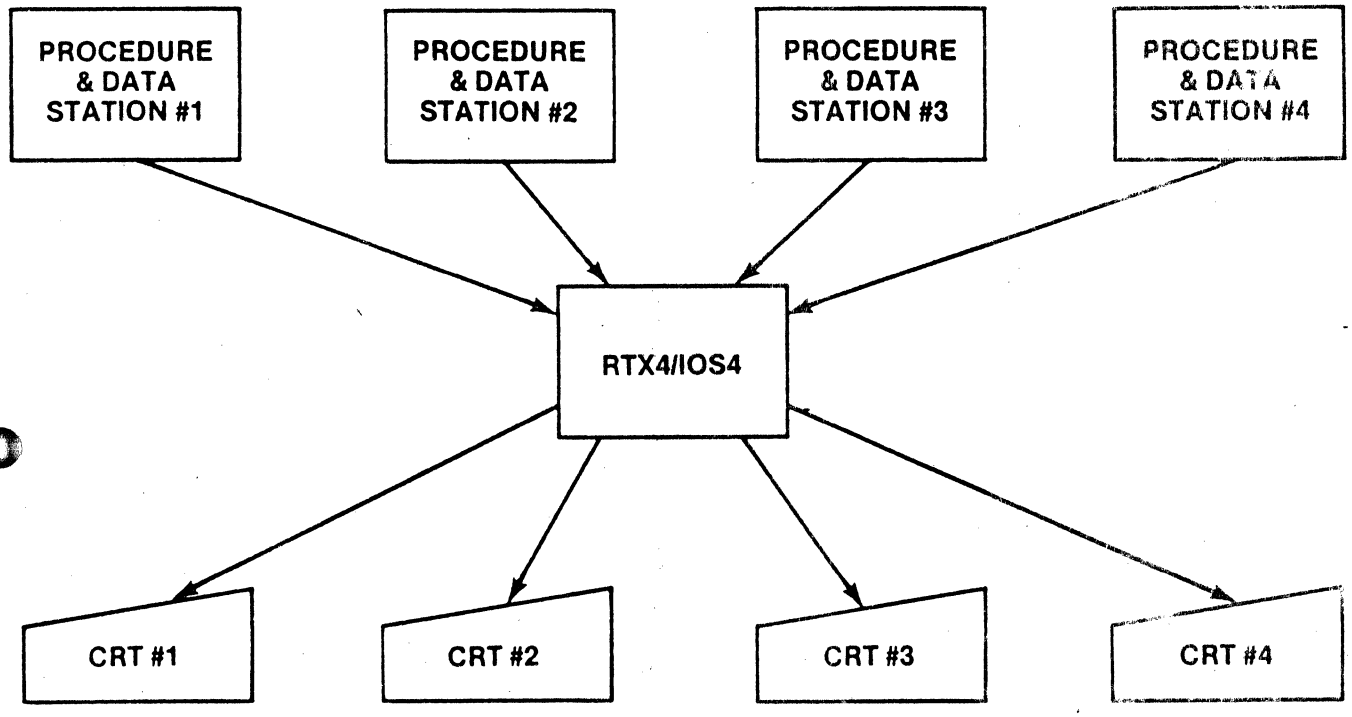


Figure 3-3. Serial Approach

The reentrant approach, illustrated in Figure 3-4, requires a slightly longer development time and may run slightly slower, but the resulting system uses much less memory and is easier to maintain and expand.

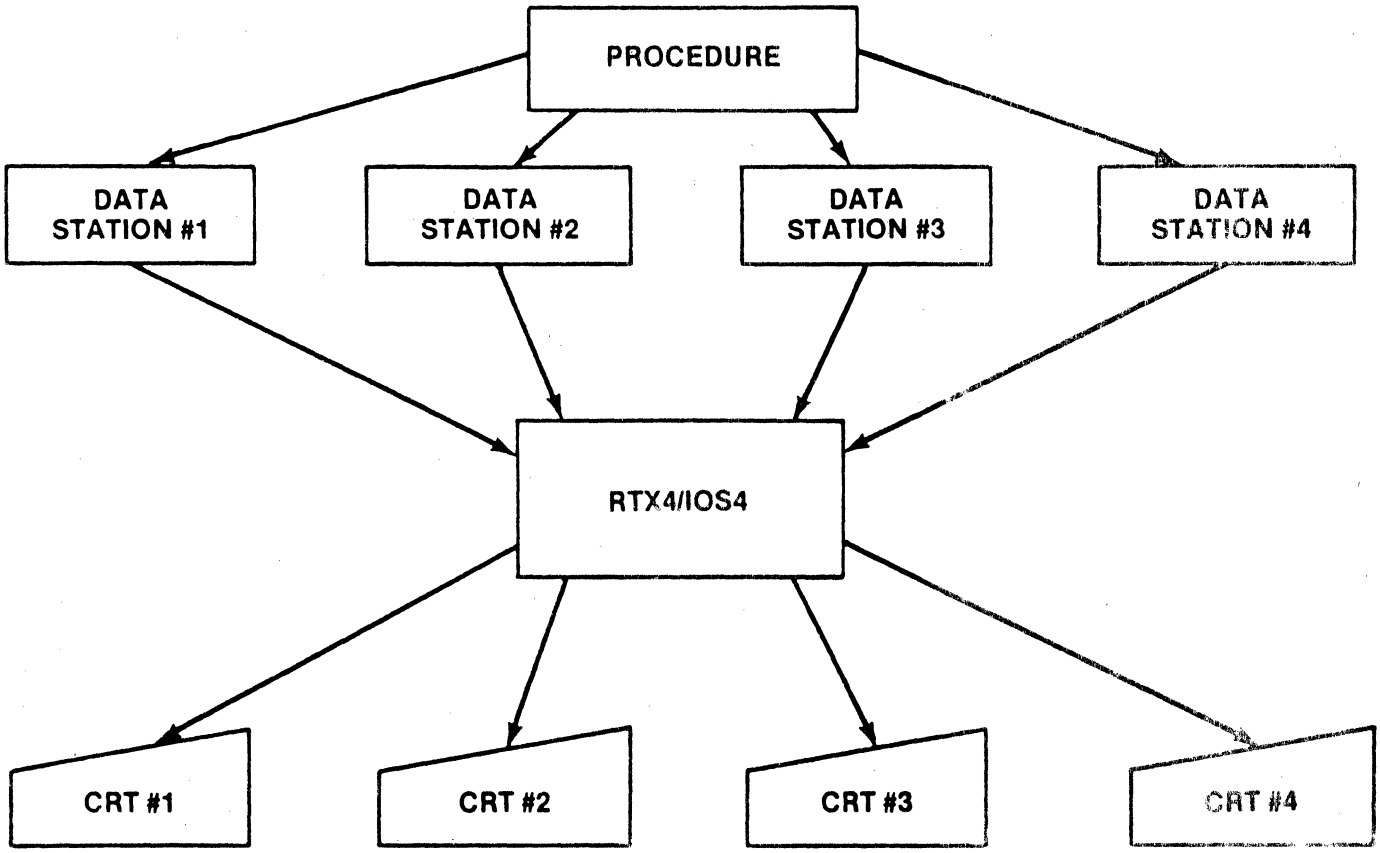


Figure 3-4. Reentrant Approach

3.3.1 Serial Tasks

In a serial task, resource allocation is simple, but several options are available. The simplest method, with the lowest CPU overhead, is to allocate space for the stack along with the task and tell the system where it is. RTX4 then faces no dynamic allocation problems. This method also protects that memory space from being available for other uses when the task is inactive. Alternatively, the programmer may ask RTX4 to allocate the stack and Y-scratchpad space dynamically when the task begins. He must supply the lengths of each, but RTX4 determines their locations. In either case, when the task begins, the Y, K, and L registers are set to point to the Y-scratchpad and stack.

3.3.2 Reentrant Tasks

The terms "procedure" and "data" are fundamental to reentrant programming. Essentially, the procedure is the unchanging part of the reentrant task and the data is the variable portion.

A procedure includes all portions of a program which do not change during program execution. It includes all items such as machine instructions, literals, data constants, and fixed address pointers which determine the process to be performed.

Data in the reentrant task refers to all items such as variables, temporary cells, stacks, and buffers which may change during program execution. It includes all program memory locations which do not qualify as procedure.

These terms are the basis of the following concepts of reentrant programming:

- Procedure and data are treated separately.
- Each activity of the reentrant task has its own allocated region for data.
- All activities of the reentrant task share the same procedure.

Suppose that the example data entry system presented earlier[1] requires 10K words of procedure and 4K words of data for each data entry station (CRT). The serial approach requires 61K words of memory:

5K*	RTX4/IOS4
40K	Procedures (4*10K)
16K	Data (4*4K)

The reentrant approach requires only 33K words of memory:

5K*	RTX4/IOS4
10K	Procedure
16K	Data

*Approximate figure

Such savings are common when reentrant programming is appropriate.

Initial resource allocation for a reentrant task has few options. The stack must be allocated dynamically by RTX4. Y-scratchpad also must be allocated dynamically if it is required. Additionally, the following rules for writing the task must be followed to ensure reentrancy:

- The Y-scratchpad address (the initial contents of the Y register) must be kept in a register at all times.
- All references to variables and temporary cells must be relative to a register, usually the Y register.
- All subroutine linkages must be made using JSK and RSK. Variable parameter passing must be through the registers.

All of the above factors concerning the task must be considered in writing the code. After writing the task, the programmer summarizes his decisions for RTX4 by creating a Task Descriptor Block, described in the following subsection.[2]

---

[1]Subsection 3.3
[2]Subsection 3.4

3.3.3 Memory Requirement Guide

Typiecal memory sizes to accommodate component parts are listed below:

IOS 3K
SFM 3.2K
RTX 2.3K

3.4 TASK DESCRIPTOR BLOCK

A Task Descriptor Block (TDB) is used to describe the needs and attributes of task to RTX4. If the Y-scratchpad and stack areas are to be allocated by RTX4, they are identified in the TDB. Each Task Descriptor Block is generated by the TDB:A macro.

3.4.1 TDB:A Macro

The TDB:A macro can occur anywhere in the task, but is usually placed at the beginning. TDB:A has five required parameters: label of the TDB, starting address of the task, address of stack space, and amount of stack space. The macro also has two optional parameters: flags and concurrent usage. The formats of the TDB:A macro are:

```
TDB:A   label,start,yscratch,stackad,stackamt
TDB:A   label,start,yscratch,stackad,stackamt,flags
TDB:A   label,start,yscratch,stackad,stackamt,flags,usage
```

where:

label	Label to be assigned to start of TDB.
start	Starting address of task.
yscratch	Amount of Y-scratchpad to be used by the task. If zero, the Y register of the calling task is used. Must be zero (or omitted) for a serial task.
stackad	Address of preallocated stack. If zero, stack space is allocated by RTX4. Must be zero (or omitted) for a reentrant task.
stackamt	Amount of stack space used by the task.
flags	None currently defined.
usage	Maximum number of concurrent activities of this task. DEFAULT = 1

To omit a parameter, enter two consecutive commas (,,).

In RTX4, each activity must have a stack for storing return addresses for a JSK. The stack is also used by the system to save the context of an activity that is making a system request or is interrupted. Also, some service routines use this area for storage of return addresses.

The user must specify the amount of stack space required. He can allocate the stack space himself in his program and he can supply the address in the TDB:A stackad parameter. As an alternative, the user may allow RTX4 to allocate the stack dynamically by specifying zero in the stackad parameter. In either case, he must specify the amount of stack space required through the TDB:A stackamt parameter. However the stack space is allocated, when the activity begins, the K register marks the top of the stack area currently being addressed and the L register marks the lower limit of the stack.

The amount of stack space needed depends on the use of the stack by both the user's program and the system. Space is calculated as follows:

7 words	To save the context of an activity when an interrupt occurs.
8 words	To save the context of an activity when a system service request is made, and call the appropriate service routine.
n words	The maximum used by any called system service routine (amount given in the description of each service).
7 words	To prevent a hardware stack exception trap after a PUSH or JSK.
n words	For the user program (e.g., subroutine calls).

System service routines are executed as part of the activity requesting the service, and they use the stack of the requesting activity. Therefore, if an interrupt occurs during the execution of a service routine, 15 words of context are on the stack.

A stack exception trap occurs when, after a PUSH or JSK has been executed, less than seven words of stack space remain. This situation can occur when a system service routine is interrupted, because both the current context and the user's context at the time of the request would be on the activity stack. The stack exception trap processing checks for this special case and resumes processing if it is found. This extra processing can cause excess interrupt latency. An additional seven words of stack space prevents this problem.

3.4.2 Examples

The following TDB:A macro creates a Task Descriptor Block for a serial task:

```
TDB:A   SBLOCK,START,,TSTACK,70
```

A 70-word stack, starting at location TSTACK, is allocated for the task. Y-scratchpad space is determined by the calling task.

A reentrant example:

```
TDB:A  RBLOCK,RBEG,100,,90,,3
```

This macro generates a TDB for a reentrant task which may have up to three concurrent activities. Each is allowed 100 words of Y-scratchpad space, allocated by RTX4. A 90-word stack is allocated for each activity.

SECTION 4

ACTIVITIES

4.1 INTRODUCTION

An activity is an execution instance of a task. Each time a task begins , a new activity is created. When RTX4 is viewed as simulating multiple processors, each activity is equivalent to a separate CPU. The activity is the unit to which the real CPU's time is allocated. Only one activity can exist at a time for a serial task, as illustrated in Figure 4-1.

Figure 4-1. Task with One Activity

A reentrant task can have several concurrent activities, as illustrated in Figure 4-2.

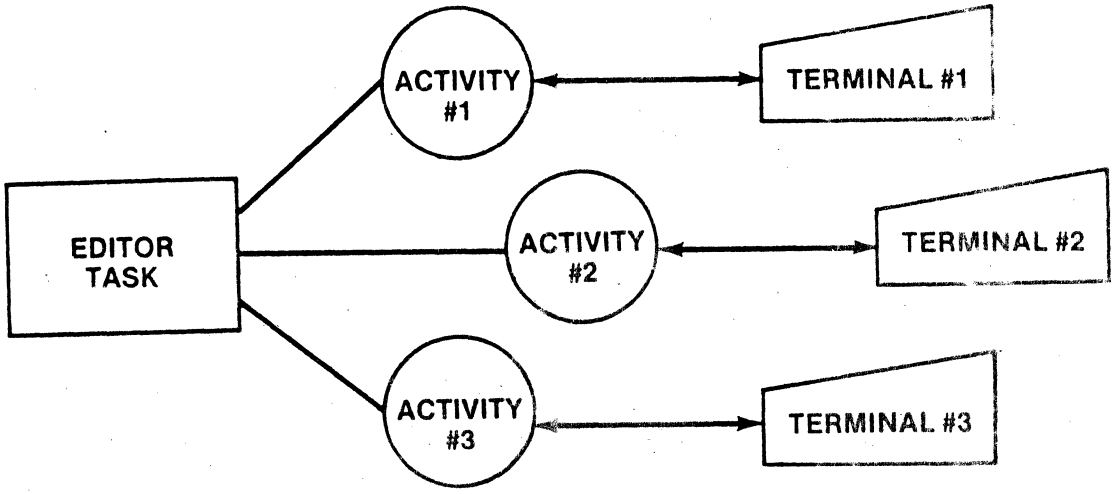


Figure 4-2. Task with Multiple Activities

4.2 ACTIVITY OPERATION

RTX4 creates an Activity Control Block when an activity is begun. The information derived from the Task Descriptor Block is placed in an available block obtained from the System Freepool.[1]

An Activity Control Block is always in a list, except for very short times while moving from one list to another. The nature of each list determines the state of activities that are in it. An activity must be in the system ready-to-run list (R:RDY) before it can execute. When an activity is waiting for an event to occur, it is usually in a semaphore wait list.[2]

The order in which activities are dispatched from the ready-to-run list (R:RDY) is determined by priority.

Priority is a means of assigning relative importance to activities. An activity of higher priority is always granted a requested resource before a lower priority activity. In RTX4 the priority of the first task is assigned by the INIT:A macro.[3] For other tasks, priority is established when the activity is begun.

Among activities of different priorities, the highest priority activity is always dispatched. If more than one activity is at the highest priority in the ready-to-run list, the first one inserted in the list is always dispatched. This dispatching algorithm is called "pure priority scheduling." It has some important ramifications in writing systems using RTX4.

It is possible to write systems in which some activities never receive CPU time. A simple example of this is a system consisting of two tasks: one task is executed at high priority and consists of one instruction, a jump to itself. The second task is executed at low priority and is intended to accomplish some useful function. In RTX4, after the first task begins, the second task never receives any CPU time. A second example is two activities that have the same priority. The first one to enter the ready-to-run list always executes first, and the second one only receives CPU time if the first makes a system call which suspends its activity.

RTX4 provides means for changing the task priority during activity execution,[4] so that other tasks may then supersede the first task in priority. It also provides a means for round-robin scheduling.[5]

---

[1]Subsection 5.3
[2]Section 6
[3]Topic 5.2.1
[4]Topic 4.3.3
[5]Subsection 7.4

Activities may be rescheduled when the following system services are called:

R:BGIN	Begin task (one extra block temporarily)
R:END	End task
R:SPRI	Set priority
R:SIG	Signal semaphore
R:WAIT	Wait on semaphore
R:SEND	Send message
R:RECV	Receive message
R:ITIC	Signal semaphore at a given time interval (for duration of time interval)
R:MTIC	Modify tick clock timer request
R:CTIC	Cancel tick clock timer request
R:AWAL	Signal semaphore at an absolute wall clock time
R:IWAL	Signal semaphore after a given time interval has elapsed
R:CWAL	Cancel wall clock timer request
R:ABUF	Allocate buffer
R:RBUF	Release buffer
R:CINT	Return to calling activity when console interrupt is pushed
R:PAUS	Drop seniority of the first activity of a given priority

4.3 ACTIVITY CONTROL

The programmer uses the R:BGIN macro to start task processing (i.e., to create an activity), the R:END macro to end a task, the R:GPRI and R:SPRI macros to get and set priorities during task processing.

4.3.1 R:BGIN Service

An activity is created by the system service R:BGIN. R:BGIN allocates stack space as specified in the Task Descriptor Block and creates an Activity Control Block for the activity which is then placed in the ready-to-run list. The register contents of the task that issues the request are the initial register contents of the activity created. This service requires 12 words of stack space. The format of the R:BGIN macro is:

```
R:BGIN    arg
```

where: arg — M4D12 pointer to the argument list.

The argument list can be generated via the BGIN:A macro, which has the format:

```
BGIN:A    arg,tdb,prdesc
```

where:

- arg — Must match the R:BGIN argument.
- tdb — Label of the Task Descriptor Block as specified in the TDB:A macro.
- prdesc — Priority descriptor defining the task's priority.

The priority descriptor is the effective address (not the contents of the effective address) of any valid M4D12 expression. The high-order bit of the priority descriptor determines whether the value is an absolute (bit 15=0) or a relative (bit 15=1) priority. If it is relative bit 14 determines whether to increase (bit 14=0) or decrease (bit 14=1) using the remaining value.

Only positive priorities are allowed in RTX4. A relative offset that results in a negative priority causes undefined results. RTX4 allows user priorities from 1 to :3FFF Higher priorities are reserved for system use.

4.3.2 R:END Service

When an activity completes its processing, resources are returned to the system by the R:END service routine. This routine terminates the activity by returning the Activity Control Block space to the System Freepool and any RTX4 allocated stack area to the Environment Memory Pool. This is the last request of any activity. The R:END macro has no parameters. This service requires 9 words of stack space.

4.3.3 R:GPRI and R:SPRI Services

The R:GPRI macro returns the activity's priority in the A Register. The macro has no parameters. Three (3) words of stack space are required for this service.

The R:SPRI request allows an activity to alter its priority. The new priority can be absolute or it can be set relative to the current priority. This service requires 10 words of stack space. The request format is:

```
R:SPRI    prdesc
```

where: prdesc — Priority descriptor expression; an M4D12 expression whose effective address is the priority descriptor.

The following is an example of using the priority service macros:

```
R:SPRI    100            Set priority to 100
R:SPRI    100 + :8000    Increase priority by 100
R:SPRI   -100            Decrease priority by 100
                         (negative value, always relative)
```

4.3.4 R:CINT Service

The activity making the R:CINT request returns when the console interrupt is pressed. If the console interrupt is never pressed, the activity never returns. Only one activity at a time can invoke this service. If this service is not requested, a console interrupt is ignored.

This service requires 5 words of stack space. The macro has no parameters.

4.4 ACTIVITY CONTEXT

A context is associated with each activity. The activity context is a set of task resources that is saved each time the execution of the task (the activity) must be suspended. The context is restored to the saved state when the task is resumed. The context of an activity includes the following items:

- An Activity Control Block (ACB) that contains pointers to the rest of the context.
- A priority that determines how real CPU time is allocated to activities.
- The task of which the activity is an execution instance.
- The environment[1] from which the activity's non-CPU resources are to be drawn.
- The stack allocated to the activity when the task was begun.
- The Y-scratchpad allocated to the activity when the task was begun.
- When the activity is executing, the contents of the registers are considered part of its context. When the activity is not executing, the register contents reside on the activity's stack.

The context of an activity provides the information necessary to simulate a dedicated CPU. Whenever an activity is dispatched (i.e., allowed to execute), RTX4 must be sure that the activity's environment is intact, then restore its register contents, including the P register, so that execution can continue. Whenever an interrupt occurs, RTX4 must save the activity's register contents on its stack so that they are not lost by further processing.

[1]Section 5

SECTION 5

SYSTEM INITIALIZATION AND ENVIRONMENT DEFINITION

5.1 INTRODUCTION

RTX4 execution starts at location :80, from which RTX4 goes to its initialization routine. The initialization routine needs certain information which is provided in the Initialization Block.

5.2 INITIALIZATION BLOCK

The Initialization Block is generated by the INIT:A macro. When system initialization is complete, the user task specified in the call is started. Only one activity can be initiated by the INIT:A routine. R:INIT must be declared as an entry point (NAM) by the user's program.

5.2.1 INIT:A Macro

The format of the INIT:A macro is:

INIT:A a,q,x,y,ecb,tdb,pri,amtfree,adrfree,topmem

where:

a,q,x,y	Initial values of the A, Q, X, and Y registers for initial user's task
ecb	Label of the Environment Control Block
tdb	Label of the Task Descriptor Block for initial user's task
pri	Activity priority for initial user's task
amtfree	Amount of System Freepool (words) (optional)
adrfree	Address of the freepool (optional)
topmem	Upper limit of memory available to RTX4 (optional)

Any addresses (adrfree, ecb, tdb, topmem) which are defined outside the module containing the INIT:A must be declared external.

When an optional parameter is omitted, a comma must be inserted to hold the position of later parameters.

If the upper limit of memory available (topmem parameter) is omitted, RTX4 searches for the end of memory and uses all that is available. This parameter is useful primarily for checkout; in this case, its use is necessary to prevent

RTX4 from allocating space for the Environment Memory Pool from the entire available memory. At checkout, other programs may be in upper memory. The parameter may be used also to check out an application on a computer that has more memory than the computer on which the program is to run for production.

5.2.2 Example

```
NAM       R:INIT          INITIALIZATION BLOCK NAME
EXTR      ECBNAME         ENVIRONMENT CONTROL BLOCK NAME
EXTR      TDBNAME         TASK CONTROL BLOCK NAME

INIT:A    0,0,0,0,ECBNAME,TDBNAME,700,100,0

          END
```

In this example:

- The A, Q, X, and Y registers are initially set to zero.
- The ECB and TDB names are in another module and are declared external.
- The activity priority is 700.
- The Freepool of 100 words is to be allocated by RTX4.
- RTX can use all available memory.

5.3 SYSTEM FREEPOOL

The System Freepool is a user-specified area that provides small buffers for RTX4 functions. The Freepool contains dynamically allocated areas such as Activity Control Blocks as well as areas used as temporary storage cells. It is organized as a linked list that can be dumped by DEBUG4[1] to provide a history of system execution.

The System Freepool consists of a region of memory provided to RTX4 at assembly time. The user must specify the amount of Freepool space he is allocating and may either allocate it himself or let the INIT:A macro allocate it.

During initialization, RTX4 breaks up the Freepool region into many fixed-length Freepool blocks. These blocks are used by RTX4 services to contain dynamically allocated blocks such as Activity Control and Clock Control blocks. Blocks are also used for short periods of time to contain temporary cells and for longer periods of time to contain information which controls a system resource.

The System Freepool is organized as a linked list to speed system processing and debugging. RTX4 keeps track of both the head and tail of the list. Blocks for RTX4 services are removed from the head of the list and are returned to either the head or tail of the list. Short-lived temporary cell blocks are returned to the head of the list to be recycled immediately. Blocks used for control information, such as Activity Control Blocks, are returned to the tail to provide a history of system and application program activity to aid in debugging. Freepool organization is illustrated in Figure 5-1.

Revised 12/78

[1] NAKED MINI 4 Debugging Monitor Reference Manual

Table 5-1. Freepool Blocks for RTX4 System Services

Service		Blocks Allocated (+) or Deallocated (-)
R:BGIN	Begin task (one extra block temporarily)	+1(+2)
R:END	End task	-1
R:GPRI	Get priority	0
R:SPRI	Set priority	0
R:SIG	Signal semaphore	0
R:WAIT	Wait on semaphore	0
R:SEND	Send message	0
R:RECV	Receive message	0
R:ITIC	Signal semaphore at a given time interval (for duration of time interval)	+1
R:MTIC	Modify tick clock timer request	0
R:CTIC	Cancel tick clock timer request	-1 (success) 0 (fail)
R:AWAL	Signal semaphore at an absolute wall clock time	+1
R:IWAL	Signal semaphore after a given time interval has elapsed	+1
R:CWAL	Cancel wall clock timer request	-1 (success) 0 (fail)
R:ABUF	Allocate buffer	0
R:RBUF	Release buffer	0
R:CINT	Return to calling activity when console interrupt is pushed	0
R:PAUS	Drop seniority of the first acitivity of a given priority	0
R:SATD	Set ASCII time and date	0
R:GATD	Read ASCII time and date	0
R:STOD	Set time of day	0
R:GTOD	Read time of day	0

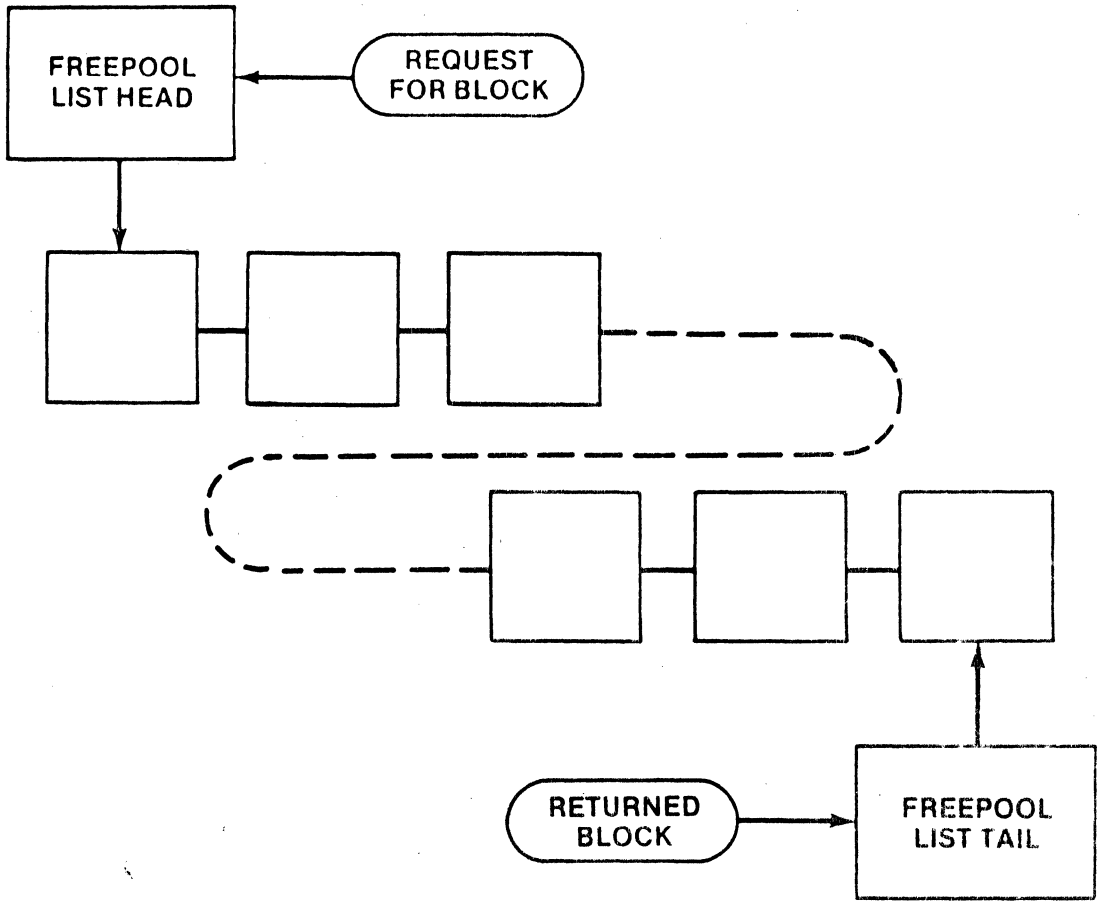


Figure 5-1. Functional Organization of System Freepool

5.3.1 Freepool Size

The Freepool is grouped into blocks of twelve words each. At least two of these blocks must be reserved.

The space needed for the System Freepool must be determined by the user. Freepool size is determined primarily by the amount required by each RTX4 system service request used in the application program. During debugging, a larger area may be desirable for system history.

Table 5-1 lists the number of contiguous blocks required for each of the system requests. This number provides a rough guide to selecting an initial Freepool size. Then, determine how much additional space is required for a complete history during debugging. RTX4 keeps track of the maximum number of blocks used during an execution in location FPMAX:. This number helps determine final Freepool size. While asynchronous events and random chance may require more Freepool blocks for any one execution, the programmer can get a good estimate from several runs and compensate for possible increases by providing a somewhat larger Freepool area.

5.3.2 The Freepool and Debugging

If the user provides a much larger Freepool during debugging than he expects to use during production, a dump of the Freepool list can provide a significant history of what has happened in the system. The last item in the list is the most recently returned block, excluding those needed by the tick clock service. Preceding blocks mark historical events until the first one or two blocks are reached; at this point the history is lost. The head of the Freepool list is at location R:FPH; location R:FPT points to the last block on the list.

Computer Automation recommends that the initial development of an RTX4 application be performed on a computer system with more memory than the final program is expected to use. This procedure allows the programmer to ignore memory allocation problems such as Freepool while getting his application to work. Further, the history provided by the Freepool list is longer and more helpful for debugging.

Thus, the Freepool size may vary from debugging through running with test data to production runs. The original estimate based on Table 5-1 can be doubled for debugging to provide a complete history. Then, when the debugged program is run with test data several times, actual Freepool usage can be determined by examining location FPMAX:, keeping in mind that usage may vary from one run to the next. For production runs, the actual usage should be augmented by several words to provide a safety factor.

To summarize, the steps to determine Freepool size are:

1) Estimate size using Table 5-1.

2) Double or triple the estimate for debugging.

3) Exercise checked out program with test data.

4) Check actual Freepool usage by examining FPMAX: after running the program.

5) Add a safety factor to actual usage for production usage.

5.4 ENVIRONMENT CONTROL BLOCK

The Environment Control Block helps define user-occupied space to RTX4 and unit assignment to IOS4. The Environment Control Block is generated by the ECB:A macro. The only value that the user must provide is a pointer to the Unit Assignment Table; RTX4 automatically supplies all other required values to the ECB.

5.4.1 ECB:A Macro

The ECB:A macro call must be placed at the end of the last user module. The format of the ECB:A macro is:

```
ECB:A    label,uat
```

where:

label	Label to be assigned to start of ECB; referenced in the intialization call to RTX4 (INIT:A).
uat	Address of the Unit Assignment Table.

The Unit Assignment Table (UAT) must be constructed by the user.[1] If the INIT:A macro is not in the same module as the ECB:A macro, the user must declare "label" an entry point (NAM). If the Unit Assignment Table is not in the same module as the ECB:A macro, the user must declare "uat" to be external (EXTR). If the I/O subsystem is not required, the UAT address is zero.

5.4.2 Example

```
NAM      ECB1
EXTR     UAT
ECB:A    ECB1,UAT
END
```

This sequence generates an Environment Control Block starting at location ECB1. This ECB contains a pointer to the Unit Assignment Table located at UAT.

5.4.3 EDXVT:A Macro

The EDXVT:A macro is used to specify user written exceptions processor. A user may specify an exception processor for any exception needed for one and all exceptions. The user must make a call to the EDXVT:A macro for each exception to be processed. All calls to the EDXVT:A macro must follow the call to ECB:A macro.

Refer to Appendix C for a list of RTX4 Exceptions.

---

[1] Input/Output Subsystem (IOS4) User's Manual

EDXVT:A label, vector, address

where:

label — Label to be assigned at start of ECB

vector — Name of exception vector, (refer to Appendix C).

address — The address of the user's exception processor.

When control is received at the user's exception processor, the RTX4 has already executed a JSK and a PUSH giving the user registers and the return address, after the instruction which caused the trap. The two exceptions to this rule are listed below:

A. The first occurs if it was an unimplemented instruction trap, then RTX4 will transfer control to the emulator, if the user has previously linked the emulator with his application.

B. If it was a stack exception, then the JSK and PUSH instructions are not performed. Instead the user may specify a four word block where the A, Q, X, and Y registers can be saved. The block may be specified by calling the EDXVT:A macro using the XV:STKSV vector.

5.5 ENVIRONMENT MEMORY POOL

The Environment Memory Pool is the space used for Y-scratchpads and stacks and user-requested buffers. This space is whatever remains between the end of the user's program and the end of memory. The user is responsible for ensuring that the Environment Memory Pool is large enough provide for all allocations required of it. Allocation is performed by a standard first-fit algorithm, so some fragmentation may result, increasing the size requirement.

RTX4 does not use any space in the Environment Memory Pool for its own tables or control blocks. Space for these items is obtained from the System Freepool.

5.6 BUFFER ALLOCATION

RTX4 provides the R:ABUF and R:RBUF services for allocating and then releasing buffer space.

5.6.1 R:ABUF Service

The R:ABUF macro allocates a buffer for use by the program. The argument to this macro specifies the number of words to be allocated. This service requires 10 words of stack space. The format is:

```
R:ABUF    amount
```

where: amount — Number of words to be allocated.

The number of words which can be allocated via R:ABUF is limited only by the amount of contiguous space available in the Environment Memory Pool at the time the request is made.

The system returns the address of the allocated buffer in the X register.

For example:

```
R:ABUF    256
```

This request allocates a 256-word buffer. The buffer address is returned in register X.

5.6.2 R:RBUF Service

When the program has completed its work with a buffer allocated via a previous R:ABUF request, it can return that space to the system by executing an R:RBUF macro. This service requires 10 words of stack space. The format is:

```
R:RBUF    address
```

where: address — Address of the buffer to be returned.

The argument to this macro is the address of the buffer. Since this address is not known until execution, the argument typically specifies register-relative addressing mode.

For example:

```
R:RBUF    0(X)
```

This request releases previously-allocated buffer space. It assumes that the buffer address is still in register X.

SECTION 6

SEMAPHORES

6.1 INTRODUCTION

RTX4 provides system services that allow the efficient programming of intertask cooperation to synchronize the execution of tasks so that they occur in a specific time relationship to one another. The facilities that are provided are based on a concept called the "semaphore." Sempahores were first formalized in 1965 by E. W. Dijkstra.[1]

This section introduces semaphores by first discussing some alternative methods of intertask cooperation. For further information on the subject of semaphores, a book by Brinch-Hansen is recommended.[2]

6.2 ALTERNATIVE APPROACHES TO INTERTASK COOPERATION

Intertask cooperation may be used for passing data (intertask communication) or may be used simply because one task must be accomplished before another (intertask coordination). Intertask coordination includes "producer-consumer" problems and "resource sharing" problems.

6.2.1 Producer-Consumer Cooperation

Figure 6-1 illustrates an example of a producer-consumer problem: task A fills a buffer and task B wants to know when it is full. One solution is to designate a location (call it CELL) as a flag. Task A sets the flag to a one when the buffer is full. Task B tests the flag and knows that the buffer is full when the value of the flag becomes a one. The operation for task A is straightforward: fill the buffer, then set CELL to a one. Task B can process the buffer when CELL is found to be a one, but has no function while CELL is zero. One choice would be to test the CELL again immediately. This approach results in a very tight loop which has several bad side effects. If the task dispatching scheme is pure priority, task B uses all available CPU time so that task A never is able to change the value of CELL. In a time-slicing system, task B uses unnecessary amounts of time, perhaps keeping task A from filling the buffer as fast as it might. In terms of reasonable system throughput, the solution is unacceptable in either case.

---

[1] E. W. Dijkstra, "Cooperating Sequential Processes" (1965)
[2] Per Brinch-Hansen, Operating System Principles (Prentice-Hall 1973)

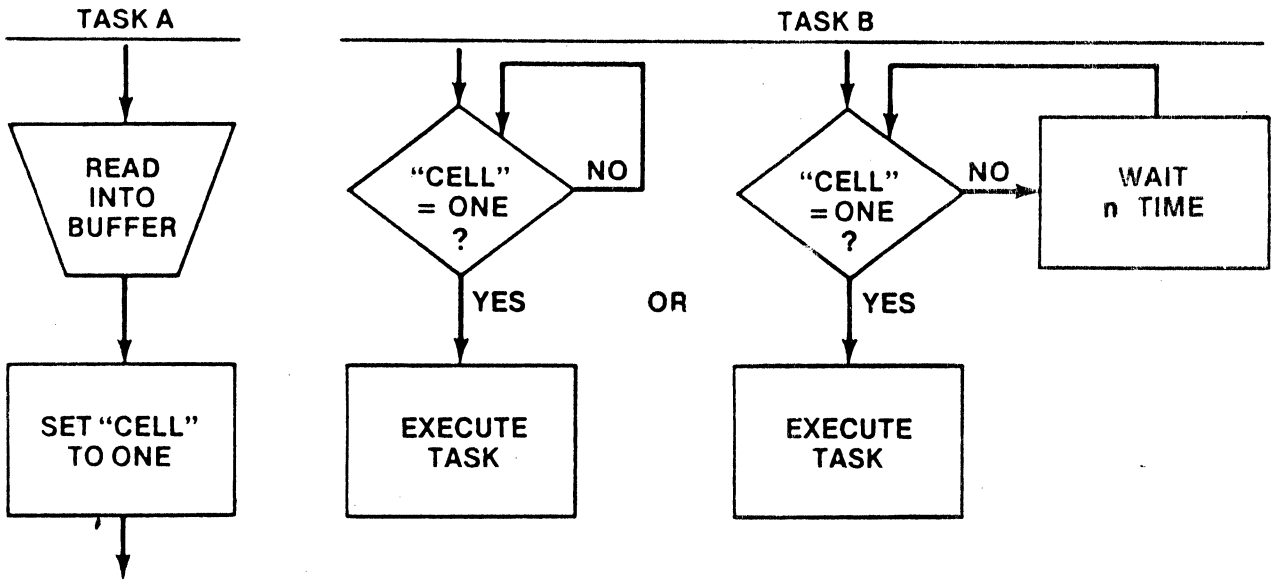


Figure 6-1. Producer-Consumer Problem Non-Semaphore Solution

An improvement would be for task B to suspend itself for a while, using the system clock, between each time it tests CELL. This solution reduces the load on the system and allows task A to complete filling the buffer in a finite time. Its main disadvantage is that task B may not find out that the buffer is full as soon as it should. There is a definite tradeoff: the less load task B places on the CPU (the longer the wait) the longer task B may be uninformed that the buffer is full.

This problem may be reduced if task A can cancel task B's wait on the clock, but there are several timing bugs associated with this operation. Suppose task A cancels the wait while task B is not waiting, for instance. Also, the system overhead and complexity is high (several waits on the clock and one cancel request). In some systems, task A sends a message to task B through a system message facility. Other systems require task A to "create" task B each time the buffer becomes full. Both solutions require more overhead than is needed to solve this problem.

The semaphore provides a simple solution to the producer-consumer problem.[1]

[1]Topic 6.3.1

6.2.2 Resource Sharing

The other general problem of concurrent task execution is resource sharing. Suppose several tasks wish to use a single resource, such as a disk. Although RTX4 can make it appear that several tasks are executing asynchronously in the computer, the disk must be used entirely sequentially, so a method must be devised to share the disk.

A location such as CELL may be used as in the previous example, but the same problems arise. The value of CELL may initially be one, meaning the resource is available. The first task that arrives examines the value of CELL. Since it is one, the task stores a zero in CELL and proceeds to use the resource. Since a zero value in CELL means that the resource is not available, tasks that come later can tell whether they may use it. Ignore for the moment their problems with doing anything reasonable when they find the resource unavailable. When the first task finishes using the disk, it must store a one in CELL, indicating the resource is now available. If only one task is waiting to use the resource, it may proceed. If more than one task is waiting, a new problem arises: which gets the resource next? If the waiting tasks are suspending themselves on the system clock, the choice of next task is made randomly: the task that completes a wait next (or a new task which arrives after all the rest) will get the resource. This is not fair. More fair would be first in, first out order. More preferable might be priority order.

Again, this problem is solved simply and with low system overhead using semaphores.[1]

6.3 SEMAPHORE SOLUTIONS TO INTERTASK COOPERATION PROBLEMS

Semaphore operation is described in a later subsection.[2] Briefly, a semaphore has two operations, signal and wait. The effect of the two operations depends on the current state of the semaphore, which is determined by its initial condition and previously performed operations.

The wait operation of a semaphore consists of determining whether the semaphore has been signalled. If it has been, the waiting activity proceeds without delay and the signal is cancelled. If the semaphore has not been signalled, the waiting activity is removed from the ready-to-run list and entered into a wait list associated with the semaphore. The signal operation first determines whether any activities are in the wait list associated with the semaphore. If so, one activity is removed from the list and placed in the ready-to-run list. If not, the state of the semaphore changes to indicate that the semaphore has been signalled.

Semaphores provide efficient solutions to the "producer-consumer" and "resource sharing" problems described previously.[3]

---

[1]Topic 6.3.2
[2]Subsection 6.5
[3]Subsection 6.2

6.3.1 Producer-Consumer Problems

In the producer-consumer problem, the initial value of the semaphore is zero, which indicates that it has not been signalled and that no activity is waiting on it. The solution is simple: the consumer "waits" on the semaphore when he wishes to know when the buffer is full; the producer "signals" the semaphore when the buffer is full. The coding is simple: each task requests a single system service, then continues processing. It does not matter which activity makes its system request first; synchronization occurs in any case.

The objections to the previous solutions do not apply to the semaphore solution. Each activity executes asynchronously. If the consumer waits for the buffer before it is available, it does not waste CPU time; it is simply removed from the ready-to-run list. Yet once the buffer is full and the producer signals the semaphore, the consumer becomes ready to run immediately. There is no delay, and the system overhead of using a semaphore is minimal.

6.3.2 Resource Sharing Problems

The resource sharing problem is solved similarly. The initial value of the semaphore is one, indicating that the resource is initially available. Each activity must wait on the semaphore before using the resource and signal the semaphore after using it. The first activity to wait on the semaphore proceeds to execute immediately because the resource is available. Any activities that request the resource later find the semaphore "unsignalled" and are removed from the ready-to-run list. When the first activity finishes using the resource, it signals the semaphore, allowing one and only one activity to use the resource next.

Again, using the semaphore has overcome the objections to the previously described solution. It has the same advantages as in the producer-consumer problem, plus an additional benefit: all of the waiting activities are in a single list, allowing great flexibility in choosing which one executes next. In most cases priority determines which waiting activity executes next. In some cases other criteria may be used. For instance, shortest seek time might be used if the semaphore is controlling a disk.

A further advantage of the semaphore is that if more than one of a particular type of resource is available (such as a limited set of buffers), the initial value of the semaphore can be changed to handle more than one. For instance, if two buffers are available, the initial value of the semaphore is two. The first two activities which request a buffer get them; all later requesting activities are suspended until one of the first two activities signals the semaphore.

6.4 SEMAPHORE DEFINITION BLOCK

A Semaphore Definition Block can be provided by the user to control the synchronization of tasks. This block is generated using the SDB:A macro.

6.4.1 SDB:A Macro

The SDB:A macro establishes the location and initial value of a semaphore in a user's program. The macro's format is:

```
SDB:A    label,value
```

where:

label	Address label of the semaphore.
value	Initial value of the semaphore.

6.4.2 Example

The following macro call generates a Semaphore Definition Block:

```
SDB:A    SEMA1,0
```

The address label of the semaphore is defined as SEMA1 and the initial value of the semaphore is defined as 0.

6.5 SEMAPHORE OPERATION

In RTX4 a semaphore consists of only one word. That word is used for a counter and a wait list head. The state of the semaphore is determined entirely by the value of this word.

In its initial condition, the wait list is always empty, and the counter may be either zero or positive, depending on the use of the semaphore (zero for producer-consumer problems, positive for resource sharing problems). The effects of the two operations on a semaphore are:

WAIT: If the counter is greater than zero, the counter is decremented and the waiting activity is re-scheduled according to priority. If the counter is zero, the waiting activity is removed from the ready-to-run list and placed in the semaphore wait list.

SIGNAL: If any activities are in the semaphore wait list, one of them is removed and placed in the ready-to-run list. If no activities are waiting, the counter is incremented.

Figure 6-2 illustrates the flow of these operations.

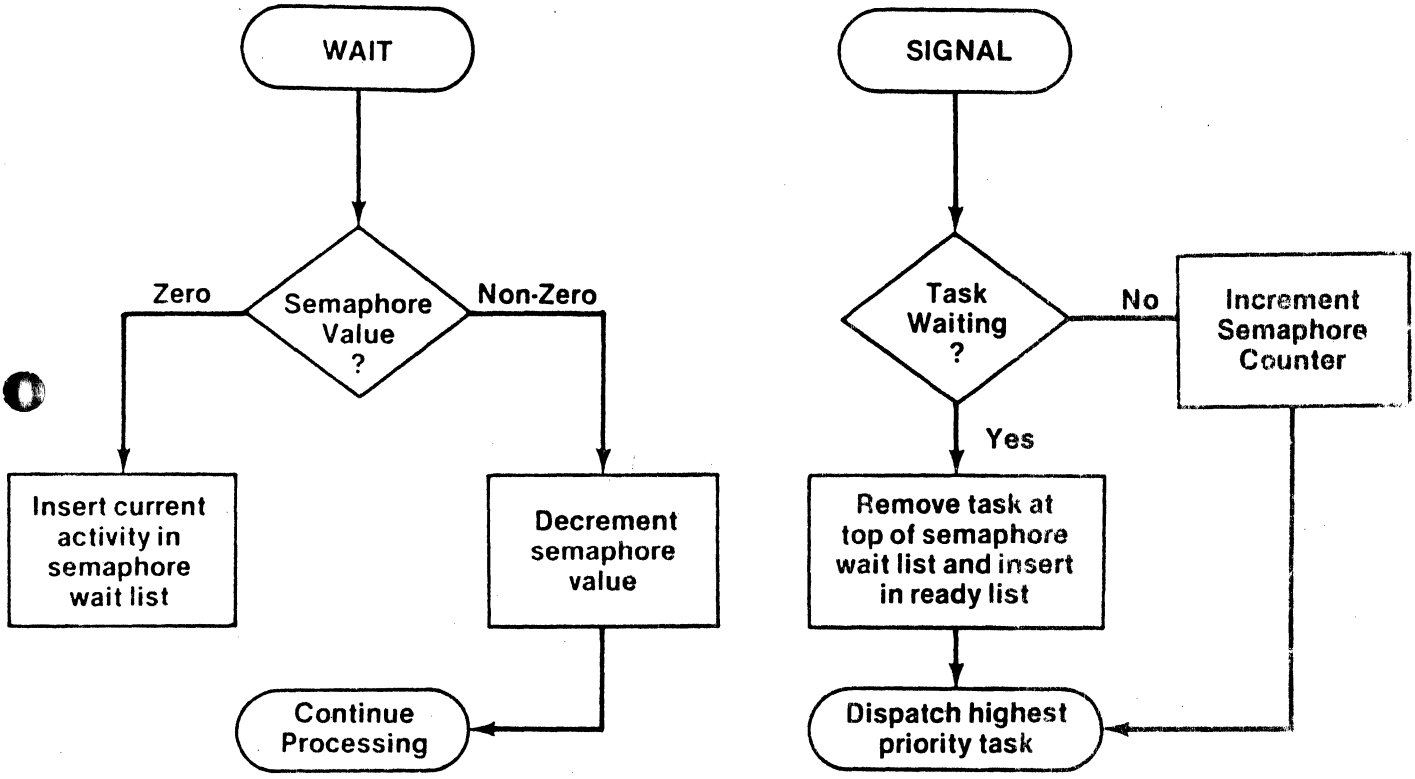


Figure 6-2. Flow of Semaphore Operations

The length of the wait list is the number of unsignalled wait requests, and the counter indicates the number of unwaited signal requests. Thus, the counter is never negative.

RTX4 contains a special form of semaphore. The main feature of an RTX4 semaphore is that an exception occurs if the counter reaches 127. An RTX4 semaphore consists of one word only that is used for both the counter and wait list head. If the value of the word lies in the range 0 through 255, then it is the semaphore counter value. If the value is larger than 255, it is the head of the list of Activity Control Blocks that are waiting on the semaphore. Thus, the state of an RTX4 semaphore is determined entirely by the value of its one word. The format of the semaphore word is shown in Figure 6-3.

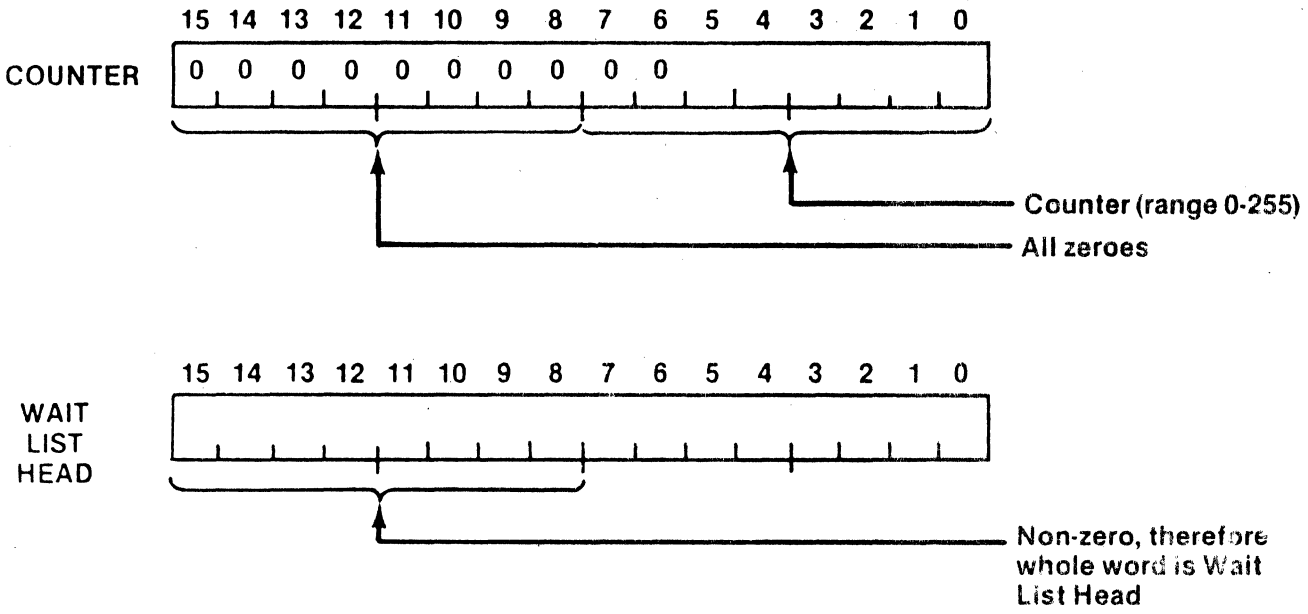


Figure 6-3. Formats of Semaphore Word

6.5.1 R:SIG Service

A sempahore is signalled by the R:SIG system request. This service causes the waiting activity that has the highest priority to be placed on the ready list. If no activity is waiting, the semaphore value is incremented. A semaphore can be signalled only 127 times without a wait. The amount of stack space required for this service is nine words. The format is:

```
R:SIG    sema4
```

where: sema4 Label of the Semaphore Descriptor Block to be signalled.

6.5.2 R:WAIT Service

An activity waits for a semaphore by using an R:WAIT system request. If the value of the semaphore is between 1 and 127, R:WAIT places the requesting activity on the ready list and decrements the semaphore. If the value is zero or greater than 127, the activity is inserted according to priority into the semaphore wait list. The amount of stack space required for this service is 9 words. The format is:

```
R:WAIT    sema4
```

where: sema4 Label of the Semaphore Descriptor Block to wait on.

6.5.3 Example

```
        .
        .                   FILL BUFFER
        .
R:SIG       X1              SIGNAL BUFFER FULL
R:WAIT      X2              WAIT FOR BUFFER EMPTY
        .
        .
        .
SDB:A       X1,0            BUFFER FULL SEMA4
SDB:A       X2,0            BUFFER EMPTY SEMA4
```

SECTION 7

SYSTEM CLOCKS

7.1 INTRODUCTION

RTX4 supports a high-resolution tick clock and a medium-resolution wall clock to simplify timekeeping functions for user applications. The time base for the two clocks is provided by the processor's Real-Time Clock (RTC).

The tick clock provides a high-resolution clock for measuring or determining relatively short time intervals. It simulates the existence of multiple hardware RTCs. The wall clock provides a medium-resolution clock that can cover all but the shortest time intervals. It provides a time-of-day and date facility. Figure 7-1 illustrates the interrelationships of the clocks.

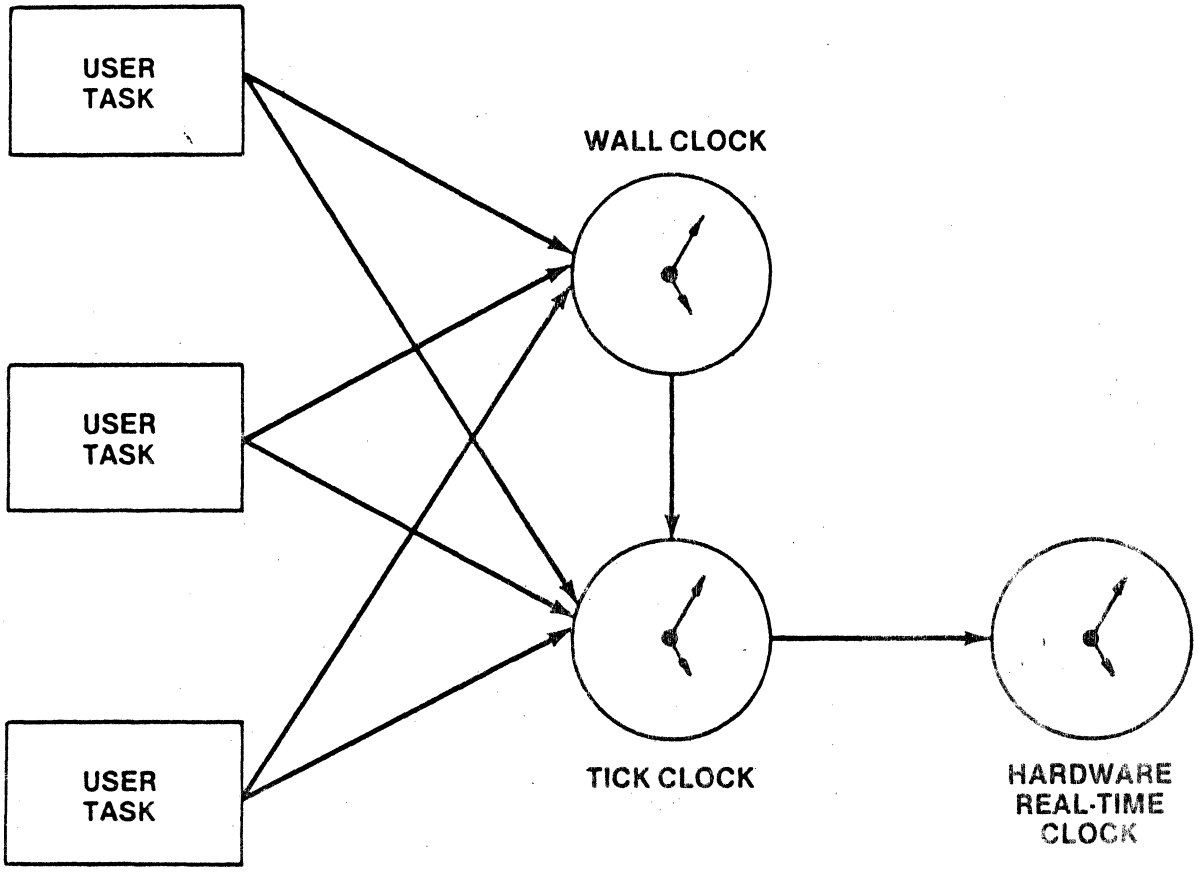


Figure 7-1. RTX4 Clocks

7.2 TICK CLOCK OPERATION

RTX4 supports a tick clock to provide measurement of very short time intervals. The rate of the tick clock is based on the rate of the processor's Real-Time Clock.

The resolution of the tick clock is dependent on the interval of the Real-Time Clock. The Real-Time Clock can be internally based on the AC line frequency of the computer. This frequency is almost exact and has excellent accuracy over longer time intervals. It is sufficient for most applications. When a shorter interval or greater precision is required, the clock can operate from an external source provided by the user. An external clock rate also allows clocks to be synchronized with other equipment. The user of the tick clock must be aware of its rate since programs may operate differently when the clock rate changes.

If the required time interval is large enough and the required accuracy is small enough, the wall clock should be used rather than the tick clock because the wall clock places a significantly smaller burden on CPU resources. If the application requires the tick clock, however, the additional overhead is well worth it.

The precision and maximum interval of the tick clock service are presented in Table 7-1.

Table 7-1. Real-Time Clock and Tick Clock Parameters

	Frequency Source		
	Internal		External
Line Frequency	60 Hz	50 Hz	
Clock Frequency	120 Hz	100 Hz	f
Interrupt Period ("Tick")	8.333 ms	10.000 ms	$t = \frac{1}{f}$
Tick Clock Service Precision	±8.333 ms	±10 ms	±t
Maximum Interval	273 seconds	327 seconds	$t \times 2^{15}$

7.3 TICK CLOCK TIMERS

RTX4 provides clock services (R:ITIC, R:MTIC, and R:CTIC) which enable the user to utilize the tick clock as if it were an alarm clock.

7.3.1 R:ITIC Service

The R:ITIC macro initiates a timer to cause a semaphore to be signalled after a specified number of ticks of the Real-Time Clock. This service requires 11 words of stack space.

The macro format is:

```
R:ITIC    arg
```

where: arg — M4D12 pointer to the argument list.

The argument list can be generated via the TICK:A macro, which has the format:

```
TICK:A    arg,id,sema4,count
```

where:

- arg — Must match the R:ITIC argument.
- id — 16-bit integer used to identify this timer.
- sema4 — Address of the sempahore to be signalled.
- count — Number of ticks that must elapse before the sempahore is signalled.

The timer identifier is a 16-bit integer. To allow for possible modification or cancellation of tick clock timer requests, all identifiers in concurrent requests within a common environment must be unique, with one exception. The programmer can specify any number of requests having identifiers with the value 0. This exception eliminates the need to create unique identifiers. However, a tick clock request with a 0 identifier cannot be modified or cancelled.

In using the R:ITIC macro, the programmer must recognize the possibility that the first tick may occur immediately after the request is made. A one-tick request may, therefore, result in the semaphore being signalled in much less time than a one-tick interval. For this reason, requests that demark a small time interval should be made for one more tick than the calculated number.

7.3.2 R:MTIC Service

If the programmer wishes to modify a previously-initiated tick clock timer request, he can use the R:MTIC request. However, the timer to be modified must have a unique (i.e., non-zero) identifier; R:MTIC cannot operate on a request having a 0 identifier. This service requires 15 words of stack space.

The format of the request macro is:

```
R:MTIC    arg
```

where: arg — M4D12 pointer to the argument list.

The argument list can be generated via the TICK:A macro, which has the format:

```
TICK:A    arg,id,sema4,count
```

where:

- arg — Must match the R:MTIC argument.
- id — Identifier of the timer to be modified; must be non-zero.
- sema4 — Address of the semaphore to be signalled.
- count — Number of ticks that must elapse before the semaphore is signalled.

Register A receives the return status of the R:MTIC requests. A -1 in register A indicates either that the specified timer identifier does not exist or that the timer has expired, i.e., the semaphore has been signalled. A 0 in register A indicates that the timer request has been successfully modified.

7.3.3 R:CTIC Service

The R:CTIC macro allows the programmer to cancel a tick clock timer request. This service requires 15 words of stack space.

The macro format is:

```
R:CTIC    arg
```

where: arg — M4D12 pointer to the argument list.

The argument list can be generated via the TICK:A macro, which has the format:

```
TICK:A    arg,id,dmy,dmy
```

where:

- arg — Must match the R:CTIC argument.
- id — Identifier of the timer to be cancelled: must be non-zero.
- dmy,dmy — Dummy arguments; may have any defined value.

The success/fail status of an R:CTIC macro is returned in register A. A -1 in register A indicates either that the specified timer identifier does not exist or that the timer has expired, i.e., the semaphore has been signalled. A 0 in register A indicates that the request has been successfully cancelled.

7.4 ROUND ROBIN SCHEDULING

If several activities at a given priority level are sharing all available processor time, no activity at a lower priority is able to execute. However, any time that all activities at the higher level are inactive, round robining at a lower priority level may take place.

To support round robin scheduling, the user writes a task which alternately waits on the clock and invokes the R:PAUS service. One activity of this task must be started for each level of round robining. The activity for each level of the round robining must be of a higher priority than the round robin it is controlling.

7.4.1 R:PAUS Service

A call to the R:PAUS macro causes the removal of the first activity of a given priority from the ready list and the reentry of that activity into the ready list. This has the effect of dropping the seniority of the activity so that another activity at the same priority is allowed to execute. The service does not change the priority scheduling rules of RTX4; it only changes the seniority rules.

This service requires one word of stack space. The format of the macro is:

```
R:PAUS    prdesc
```

where: prdesc Priority descriptor.

7.4.2 R:PAUS Example

In this example R:PAUS is used to cause two activities of the same task and priority to share processing time. Each activity gives up control every 12 ticks.

```
STACKS          EQU       7+8+7+11        STACK SPACE FOR ROBIN
PRIORITY        EQU       :300            PRIORITY OF ROUND ROBIN LEVEL
INTERVAL        EQU       12              ROUND ROBIN INTERVAL IN TICKS
*
*INITIALIZATION
*
                .
                R:BGIN    ABC             FIRST LEVEL
                .
                R:BGIN    ABC             SECOND LEVEL
                .
                BGIN:A    ABC,ROBIN,PRIORITY
                .
                R:END
*
*TASK ROBIN
*
                TDB:A     ROBIN,START,0,0,STACKS
START           R:ITIC    TIMER
                R:WAIT    SEMA4
                R:PAUS    PRIORITY
                JMP       START
                TICK:A    TIMER,0,SEMA4,INTERVAL
                SDB:A     SEMA4,0
```

Refer to Section 3.4.1.

7.5 WALL CLOCK OPERATION

RTX4 supports a wall clock to provide time-of-day and date for user programs. The wall clock uses the tick clock to keep its time. It handles both relative and absolute time intervals.

The wall clock provides the following characteristics:

- The interval of the wall clock is an absolute interval (.25 seconds) rather than a function of the Real-Time Clock.
- The interval of the wall clock is sufficiently small for many human oriented operations.
- The wall clock provides unique times and dates for a period of over 17 years.
- The processing overhead of wall clock services is much less than that of tick clock requests.

The wall clock keeps the time and date as a double-precision integer which counts the number of quarter seconds which have elapsed since 1 March 1976. The double-precision integer format allows the time and date to be kept until 1 March 1993, which should be sufficient for most applications.

The wall clock uses the tick clock to keep its time, so it is as accurate as the hardware Real-Time Clock frequency source allows it to be. RTX4 is delivered with a parameter which relates the wall clock period (.25 seconds) to the 60 Hz TTLF (Twice The Line Frequency) source. This parameter must be modified if some other frequency source is used.[1]

The wall clock can be used to handle relative time intervals as well as absolute times. For instance, it may be useful to perform some functions on a daily basis. The programmer may add the correct number to the current wall clock value and request that RTX4 notify him when that absolute time is reached; or he may request that RTX4 notify him when a certain interval has elapsed. Double-precision integer values for common time intervals are:

Interval	Decimal	Hexadecimal
Quarter Second	1	:1
Second	4	:4
Minute	240	:F0
Hour	14400	:3840
Day	345600	:54600
28 Day Month	9676800	:93A800
29 Day Month	10022400	:98EE00
30 Day Month	10368000	:9E3400
31 Day Month	10713600	:A37A00
Year	126144000	:784CE00
Leap Year	126489600	:78A1400

If the user has no need for the wall clock, he can omit it from his configuration to reduce memory space and CPU usage.

7.6 WALL CLOCK VALUE DEFINITION/ACCESS

When an RTX4 application program begins -- i.e., when it is AutoLoaded -- the wall clock has an initial value of 0. That is, the wall clock starts at time 00:00:00 of March 1, 1976.

The program R:STOD and R:GTOD services allow the program to modify the wall clock value and to access that value, respectively. If the programmer prefers to deal with the wall clock value in ASCII, he can use the R:SATD and R:GATD services.

[1] Appendix D

7.6.1 R:STOD and R:GTOD Services

The R:STOD macro sets the time of day to the value specified in the AQ register pair. (The Q register contains the least significant bits.) The value represents the number of quarter seconds that have elapsed since March 1, 1976. The macro takes no parameters. This service requires one word of stack space.

The R:GTOD macro allows the program to obtain the time of day at a particular moment during execution. The time of day (i.e., the number of quarter seconds that have elapsed since March 1, 1976) is returned in the AQ register pair with the Q register containing the least significant bits. The macro takes no parameters. This service requires one word of stack space.

7.6.2 R:SATD and R:GATD Services

The R:SATD (set time and date in ASCII) and the R:GATD (get time and date in ASCII) macros enable the time and date to be passed in ASCII either way between a task and RTX4. These services each require one word of stack space.

The R:SATD request format is:

```
R:SATD    arg
```

where: arg — M4D12 pointer to the argument list.

The argument list is a seven-word block containing the date and time values to be set in the order: year, month, day, hour, minute, second, and hundredths of a second. (The time resolution is to a quarter second.) Any illegal values entered are converted to zeros. The base date is March 1, 1976; any earlier date is invalid.

If R:GATD is called before the time and date are set, the values received are meaningless. The request format is:

```
R:GATD    arg
```

where: arg — M4D12 pointer to a seven-word block to receive the date and time values.

7.7 WALL CLOCK TIMERS

RTX4 provides three services (R:AWAL, R:IWAL, and R:CWAL) which enable the programmer to use the wall clock in activity control functions.

7.7.1 R:AWAL Service

The R:AWAL macro initiates a timer to cause a semaphore to be signalled at an absolute wall clock time. Normally, the programmer specifies a time in the future; a time in the past causes the semaphore to be signalled immediately. This service requires 11 words of stack space.

The format is:

```
R:AWAL    arg
```

where: arg — M4D12 pointer to the argument list.

The argument list can be generated via the WALL:A macro, which has the format:

```
WALL:A    arg,id,sema4,upper,lower
```

where:

- arg — Must match the R:AWAL argument.
- id — 16-bit integer used to identify this timer.
- sema4 — Address of the semaphore to be signalled.
- upper — Upper word (most significant bits) of the 32-bit integer specifying the absolute wall clock time, represented as the number of 1/4 seconds that have elapsed since March 1, 1976, at which the semaphore is to be signalled.
- lower — Lower word (least significant bits) of the 32-bit integer specifying the absolute wall clock time at which the semaphore is to be signalled.

The time is specified as a 32-bit integer representing the number of quarter seconds that have elapsed since March 1, 1976. The identifier is a 16-bit integer. To allow for possible subsequent cancellation of wall clock timer requests, all identifiers in concurrent requests within a common environment must be unique, with one exception. The programmer can specify any number of requests having identifiers with the value 0. This exception eliminates the need to create unique identifiers. However, a wall clock request with a 0 identifier cannot be cancelled as can a request with a unique identifier.

7.7.2 R:IWAL Service

The R:IWAL macro initiates a timer to cause a semaphore to be signalled after a specified time interval has elapsed. This service requires 11 words of stack space.

The format is:

```
R:IWAL    arg
```

where: arg — M4D12 pointer to the argument list.

The argument list can be generated via the WALL:A macro, which has the format:

```
WALL:A    arg,id,sema4,upper,lower
```

where:

arg — Must match the R:IWAL argument.

id — 16-bit integer used to identify this timer.

sema4 — Address of the semaphore to be signalled.

upper — Upper word (most significant bits) of the 32-bit integer specifying the number of wall clock intervals (1/4 second) that must elapse before the semaphore is signalled.

lower — Lower word (least significant bits) of the 32-bit integer specifying the number of wall clock intervals that must elapse before the semaphore is signalled.

7.7.3 R:CWAL Service

The R:CWAL macro allows the programmer to cancel a wall clock timer request. R:CWAL cannot cancel a timer having a 0 identifier. This service requires 15 words of stack space. The format is:

```
R:CWAL    arg
```

where: arg — M4D12 pointer to the argument list.

The argument list can be generated via the WALL:A macro, which has the format:

```
WALL:A    arg,id,dmy,dmy,dmy
```

where:

arg — Must match the R:CTIC argument.

id — Identifier of the timer to be cancelled; must be non-zero.

dmy,dmy,dmy — Dummy arguments; may have any defined values.

The success/fail status of a R:CWAL request is returned in register A. A -1 in register A indicates either that the specified identifier does not exist or that the timer has already expired, i.e., the semaphore has already been signalled. A 0 in register A indicates that the request has been successfully cancelled.

revised 12/

SECTION 8

MAILBOX

8.1 INTRODUCTION

The mailbox is a facility for communicating between activities in RTX4. It enables 32-bit messages to be passed from one activity to another. A program can have any number of mailboxes.

A mailbox is created with the MDB:A macro. RTX4 provides two mailbox services (R:SEND and R:RECV) for communicating 32-bit messages from one task to another through the Q and A registers.

Figure 8-1 shows the general flow of processing messages. Only one message can be held in a mailbox at a time, but messages cannot be lost. The first activity to send a message deposits its message immediately and continues execution. A subsequent sender is suspended until the first message is received. The second sender then resumes execution, deposits its message, and continues processing.

Each message can be received only once. If all previous messages have been received and an activity tries to receive a message, the activity is suspended until the next message is sent. Thus, no messages are received twice.

Messages are sent and received in priority order. If several activities are waiting to receive a message, the highest priority activity receives the next message. Alternatively, if no activities are waiting, or if all activities are of the same priority, messages are processed on a first-come, first-served basis.

The mailbox transmits an unformatted 32-bit message consisting of two computer words of the programmer's choice. Typically, the mailbox contains a 16-bit address and a 16-bit word describing what the address contains.

Mailboxes and semaphores have some similarities. Semaphores should be used where only synchronization is necessary. Mailboxes can be used where data must be transferred between the synchronizing tasks. In such usage, a mailbox may replace the use of two or more semaphores and aid in simplifying the problem. However, a mailbox takes more space and consumes more CPU time.

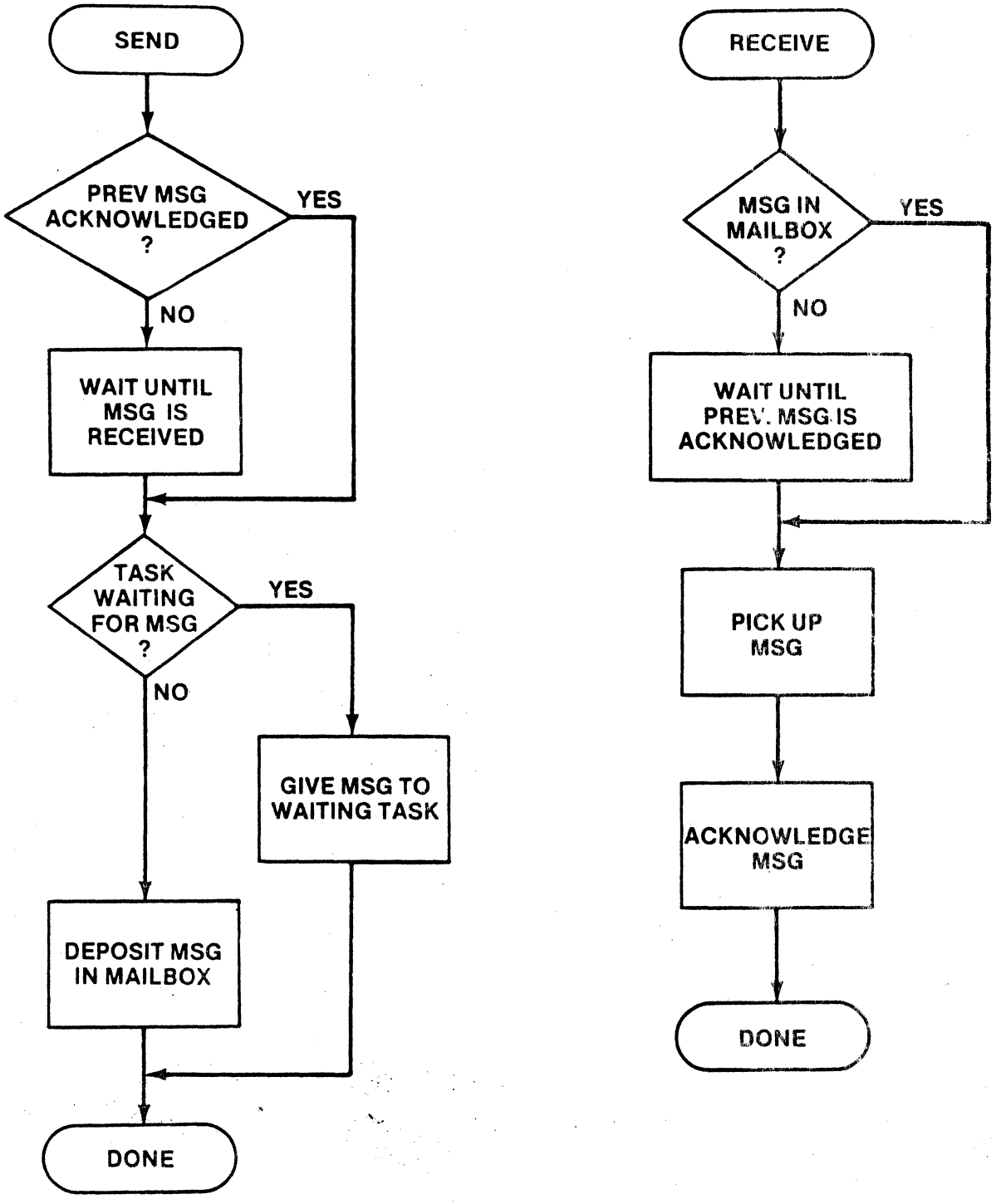


Figure 8-1. Processing Messages in the Mailbox

8.2 MAILBOX DEFINITION

To define a mailbox facility, the programmer calls the MDB:A macro. In addition, he provides two words of storage for the mailbox. The user can define any number of mailboxes in his program.

8.2.1 MDB:A Macro

The MDB:A macro defines a mailbox facility. A Mailbox Definition Block is provided by the user program via an MDB:A macro call. The format of the macro is:

```
        MDB:A     mail
```

where:	mail	Two character identifier to be assigned to the mailbox.

8.3 MAILBOX OPERATION

RTX4 provides the R:SEND and R:RECV services for communicating messages from one task to another via a mailbox.

8.3.1 R:SEND Service

This request causes a message to be sent from one task to another. The message is contained in the Q and A registers. This service requires 15 words of stack space.

The format of the R:SEND macro is:

```
    R:SEND    arg
```

where:	arg	M4D12 pointer to the argument list.

The argument list can be generated via the MAIL:A macro, which has the format:

```
    MAIL:A    arg,mail
```

where:	arg	Must match the R:SEND argument.
	mail	Two character identifier of the appropriate mailbox as defined in the MDB:A macro.

8.3.2 R:RECV Service

The R:RECV system request is used by one task to receive a message sent by another. The message is received into the Q and A registers. This service requires 15 words of stack space.

The format is:

```
R:RECV    arg
```

where: arg M4D12 pointer to the argument list.

The argument list can be generated via the MAIL:A macro as in the R:SEND service above.

8.4 SAMPLE SEQUENCE

In the following sequence, the user defines a mailbox named 'B1' and sends a message to that mailbox.

```
MDB:A     'B1'
  .
  .
MAIL:A    LABEL1, 'B1'
  .
  .
R:SEND    LABEL
```

APPENDIX A

GLOSSARY

activity	An execution instance of a task. Every time a task is begun, a new activity is created.
activity context	The activity resources that are maintained throughout the life of the activity. Context includes the L and K registers, priority, task identification, and environment.
environment	The set of all the physical resources required by the activity, except CPU time. This includes memory, I/O devices, and hardware exception traps.
environment memory pool	The area between the end of the user's program and the end of memory that is used by RTX4 to allocate Y-scratchpads and stacks for tasks as they are begun.
interrupt latency	The time that passes from when the highest priority interrupt is asserted by the hardware to the time it is acknowledged by the CPU. It is usually caused by interrupt lockouts within RTX4 which are necessary for the internal operation of RTX4.
list	An ordered or unordered collection of blocks.
mailbox	A facility for communicating 32-bit messages from one task to another.
priority descriptor	A word whose high-order bit indicates whether the priority is absolute (bit 15=0) or relative (bit 15=1) to the calling task. If the word is an M4012 expression, the effective address <u>is</u> the priority.
reentrant	Used to describe a task that can have two or more activities executing concurrently.
register context	The contents of registers of an activity. While the activity is executing, the register context is defined by the hardware register contents. While an activity is inactive, the register context is stored on the stack in the order P Y X Q A S L, and the pointer to the register context is stored in the Activity Control Block.

round robin	Insuring that two or more activities share CPU time rather than using it on a pure priority or seniority basis. The user of RTX4 can implement round robin scheduling at a single priority level using the R:PAUS and clock services.
semaphore	A common data mechanism for transmitting timing signals between concurrently executing tasks. A semaphore has two operations: signal and wait. In RTX4, semaphores provide the primary mechanism for resource control and timing conflict resolution.
serial	Used to describe a task in which the execution of one activity must be completed before another activity in that task can start.
system freepool	A user-specified area that provides small buffers for RTX4 functions such as Activity Control Blocks and Clock Control Blocks.
task	An ordered collection of machine instructions that perform a particular function within the real-time application.
tick	The real-time clock increment, typically 8.33 ms for 60 Hz line frequency.

APPENDIX B

RTX4 TABLES

B.1 INTRODUCTION

This appendix describes the following tables and the macros which generate them:

- Task Descriptor Block (TDB:A)
- Environment Control Block (ECB:A)
- Mailbox Definition Block (MDB:A)
- Activity Control Block
- Clock Control Block
- Semaphore Definition Block (SDB:A)
- Initialization Block (INIT:A)
- Parameter Blocks (BGIN:A, R:SATD, R:GATD, TICK:A, WALL:A, MAIL:A)

B.2 TASK DESCRIPTOR BLOCK

The Task Descriptor Block is provided by the user to describe the attributes of a task to RTX4. This table can be generated using the TDB:A macro. The address of a Task Descriptor Block is the task ID.

TDB:A label,start,yscratch,stackad,stackamt,flags,usage

where:

label	Label to be assigned to start of TDB.
start	Starting address of task.
yscratch	Amount of Y-scratchpad used by task (for reentrant program only); if zero, the register value of the calling task is used.
stackad	Address of preallocated stack (for serial program only); if zero, stack space is allocated by RTX4.
flags	None currently defined.
usage	Number of concurrent activities of this task (optional).

0	TD:PER	:0
1	TD:FLG	:1
2	TD:USA	:2
3	TD:NOX	:3
4	TD:Y	:4
5	TD:AD	:5
6	TD:AMT	:6
7	TD:P	:7
8	TD:CSA	:8
9		:9
10		:A
11	TD:CKW	:B

Field	Word	Bits	Description
TD:PER	0	0-15	Link pointer to the list of other Task Descriptor Blocks, the head of which is in ECBTLH (word 16) of the Environment Control Block. This list is linked as part of the loading process.
TD:FLG	1	0-15	TDB flags; none currently defined.
TD:USA	2	0-15	Either: controls the number of concurrent executions of a reentrant task, or queues requests for execution of a serially reusable task.
TD:NOX	3	0-15	Maximum permitted number of concurrent executions of a reentrant task.
TD:Y	4	0-15	Length of the Y-scratchpad to be dynamically allocated to each execution of this task. If the length is zero, the Y value of the task that called this task for execution is retained.
TD:AD	5	0-15	Either: the address of the area to be used as the stack for each execution of this task or 0, if the stack is to be dynamically allocated.
TD:AMT	6	0-15	Either end limit address of the stack if TD:AD ≠ 0; or if TD:AD = 0, it is then the stack length.
TD:P	7	0-15	Address of the start of task execution.
TD:CSA	8	0-15	Concurrency semaphore.
	9-10		Reserved.
TD:CKW	11	0-15	TDB checkword; contains :F01E.

B.3 ENVIRONMENT CONTROL BLOCK

The Environment Control Block is created by the user to describe resources to RTX4. The address of the ECB is its ID. This block can be generated using the ECB:A macro.

```
ECB:A     label,uat
```

where:

label	Label to be assigned to start of ECB; referenced in INIT:A.
uat	Address of the Unit Assignment Table.[1]

0	EC:PER	0	Peer link.
1	EC:FLG	:1	ECB flags - none defined.
2	EC:EDB	:2	Environmental descriptor block pointer.
3	EC:LUS	:3	Logical unit semaphore.
4		:4	Reserved
5		:5	
6	EC:CNT	:6	Number of task activities.
7	EC:ALH	:7	ACB list head.
8	EC:SUB	:8	Subordinate list head.
9	EC:MST	:9	Master environment.
10	EC:NEC	:A	Necessary environment.
11	EC:CKW	:B	ECB checkword; contains :F06E.
12	ED:EVO	:C	Exception vector offset (=16).
13	ED:MRO	:D	Map register offset (=48).
14		:E	Reserved
15		:F	
16	ED:EVT	:10	Exception Vector Table
.	.	.	
.	.	.	
.	.	.	
31	ED:EVT	:1F	
.	.	.	
.	.	.	
.	.	.	

(table continues on next page)

[1] Input/Output Subsystem IOS4 User's Manual (90-93430-00)

(continued from previous page)

.
.
.

32	ED:UAT	:20	Unit Assignment Table address
33	ED:LMA	:21	Low memory address
34	ED:MPA	:22	Environment pool address
35	ED:HMA	:23	High memory address
36	ED:EUS	:24	Environmental usage semaphore
37	ED:PRI	:25	Maximum priority
38	ED:TLH	:26	Task list head
39	ED:SLH	:27	Semaphore list head
40	ED:MLH	:28	Mailbox list head
41		:29	Reserved
42		:2A	
43		:2B	
44	Undefined	:2C	Number of task activity
45	Undefined	:2D	TDB of initial task
46	Undefined	:2E	Priority of initial activity
47	Undefined	:2F	ID of initial activity
48	Undefined	:30	Map register table

B.4 MAILBOX DEFINITION BLOCK

A Mailbox Definition Block is provided by the user to provide a mechanism for communication between activities. An MDB can be generated via the MDB:A macro:

```
MDB:A     mail
```

where: mail — Identifier to be assigned to the mailbox.

Two words of staorage must be provided for the mailbox.

0	MD:PER	:0	Peer link
1	MD:FLG	:1	MDB flags
2	MD:MBX	:2	Mailbox usage semaphore (initially = 1)
3	MD:MSG	:3	Message signalling semaphore (initially = 0)
4	MD:A	:4	A register of message
5	MD:Q	:5	Q register of message
6		:6	Reserved
7		:7	
8		:8	
9	MD:ECB	:9	Master environment
10	MD:ID	:A	ID
11	MD:CKW	:B	MDB checkword; contains :F07E

B.5 ACTIVITY CONTROL BLOCK

RTX4 creates the Activity Control Block when an activity is begun.

0	AC:PER	:0	Peer link
1	AC:FLG	:1	ACB flags
2	AC:PRI	:2	Priority
3	AC:K	:3	K register
4	AC:Y	:4	Y register initial value
5	AC:L	:5	L register initial value
6		:6	Reserved
7	AC:LST	:7	Environment activity peer link
8	AC:TDB	:8	Master TDB
9	AC:ECB	:9	Master ECB
10	AC:ID	:A	ACB identifier
11	AC:CKW	:B	ACB checkword; contains :F02F

B.6 CLOCK CONTROL BLOCK

The Clock Control Block is created by RTX4.

0	CC:PER	:0	Peer link
1	CC:FLG	:1	CCB flags
2	CC:TIC	:2	Tick clock expiration
3		:3	Not applicable to tick clock
4	CC:STS	:4	Semaphore or address of semaphore
5		:5	Reserved
6		:6	
7		:7	
8		:8	
9	CC:ECB	:9	Master environment
10	CC:ID	:A	CCB identifier
11	CC:CKW	:B	CCB checkword; contains :F04E

B.7 SEMAPHORE DEFINITION BLOCK

The semaphore definition block is provided by the user to control the synchronization of tasks. This table can be generated using the SDB:A macro.

SDB:A label,value

where: label Address label of the semaphore.

value Initial value of the semaphore.

0	SD:PER	:0	Peer link
1	SD:FLG	:1	SDB flags; none defined (bits 8-15) Semaphore initial value (bits 0-7)
2	SD:SEM	:2	Semaphore word for waiting task
3	SD:CKW	:3	SDB checkword; contains :F03E.

B.8 INITIALIZATION BLOCK

The Initialization Block is generated by the INIT:A macro and provides the information needed to initialize the entry point. In order to use this macro, the user must declare the label R:INIT as an external in his program (NAM R:INIT). This macro must preceed any other code in the program.

```
INIT:A    a,q,x,y,ecb,tdb,pri
INIT:A    a,q,x,y,ecb,tdb,pri,amtfree
INIT:A    a,q,x,y,ecb,tdb,pri,amtfree,adrfree
INIT:A    a,q,x,y,ecb,tdb,pri,amtfree,adrfree,topmem
INIT:A    a,q,x,y,ecb,tdb,pri,,,topmem
INIT:A    a,q,x,y,ecb,tdb,pri,amtfree,,topmem
```

where:

a,q,x,y	Initial values of the A, Q, X, and Y registers
ecb	Address of the Environment Control Block
tdb	Address of the Task Descriptor Block
pri	Activity priority
amtfree	Amount of freepool space (optional)
adrfree	Address of the freepool (optional)
topmem	Upper limit of memory available to RTX4 (optional)

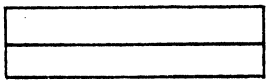
When an optional parameter is omitted, a comma must be inserted to hold the position of later parameters.

0	IN:A	:0	Initial contents of A register
1	IN:Q	:1	Initial contents of Q register
2	IN:X	:2	Initial contents of X register
3	IN:Y	:3	Initial contents of Y register
4	IN:ECB	:4	Address of Environment Control Block
5	IN:TDB	:5	Address of Task Descriptor Block
6	IN:PRI	:6	Initial activity priority
7	IN:ID	:7	Block identifier
8	IN:FPL	:8	Freepool length (words)
9	IN:FPA	:9	Freepool address
10	IN:EOM	:A	End-of-Memory address

B.9 RTX4 SERVICE PARAMETER BLOCKS

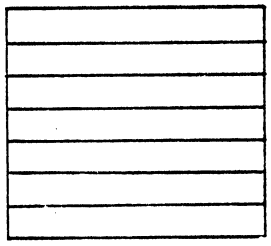
The BGIN:A, R:SATD, R:GATD, TICK:A, WALL:A, and MAIL:A macros involve parameter blocks, as follows.

BGIN:A



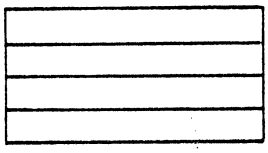
0	Address of TDB
1	Priority

ASCII Time of Day Block (R:SATD,R:GATD)



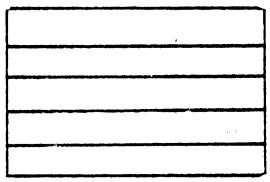
0	Year
1	Month
2	Day
3	Hours
4	Minutes
5	Seconds
6	100ths of second (only on R:GATD, resolution is ¼ sec.)

TICK:A



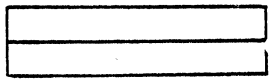
0	Reserved
1	16-bit identifier
2	Address of semaphore to signal
3	Number of ticks

WALL:A



0	Reserved
1	16-bit identifier
2	Address of semaphore to signal
3	High order 16 bits of time
4	Low order 16 bits of time

MAIL:A



0	Address of Mailbox Definition Block
1	Reserved

APPENDIX C

RTX4 EXCEPTIONS

When an error is discovered during the execution of RTX4, an exception condition is generated. Refer to EDB:A Macro to see how to link in a user exception processor. The names of the form XV:xxxxx, listed in the table below are used to define exception vectors for processing exceptions.

Table C-1. RTX4 Exceptions

EXCEPTION TYPE	If exception processor specified JMP to routine specified in EDB.	If no exeption, processor CPU HALTS, with following register contents.
[illegible] instruction trap	A = Note 2. X = Note 2. Q = 0 = XV:UINTP	A = Note 6. X = Note 6. Q = 0000 P = 80
Memory exception trap	A = Note 3. X = Note 4. Q = 1 = XV:MEMTP	A = 208 X = Undefined Q = 0001 P = 80
Character/ mnemonic exception trap	A = Note 1. X = Note 2. Q = 2 = XV:CNMTP	A = 208 X = Note 6. Q = 0002 P = 80
User trap	A = Note 1. X = Note 2. Q = 3 = XV:USRTP	A = 208 X = Note 6. Q = 0003 P = 80
Arithmetic exception trap	A = Note 1. X = Note 2. Q = 4 = XV:AERTP	A = 208 X = Note 6. Q = 0004 P = 80
Stack exception trap	[illegible]	[illegible]

Table C-1. RTX4 Exceptions (Continued)

EXCEPTION TYPE	If exception processor specified JMP to routine specified in EDB.	If no exeption, processor CPU HALTS, with following register contents.
Unimplemented system service	A = Undefined X = Contents of Strap trap location (next P) Q = 8 = XV:USTEX	A = 208 X = Address where exception occurred Q = 0008 P = 80
Strap 0	A = Undefined X = Undefined Q = 9 = XV:STOEX	A = Undefined X = Undefined Q = 000A P = 80
RTX door service exception	A = Negative error code X = Undefined Q = :A = XV:DOREX	A = Error code on door exit (positive value), see Table C-2 X = Undefined Q = 000A P = 80
RTX system error	A = Negative error code X = Location where error occurred +1. Q = :B = XV:RTXEX	A = Error code (positive value) X = Location where exception occurred +1, see Table C-2 Q = 000B P = 80

NOTES:

1. Contents of trap location +1 (instructions causing trap).
2. Contents of trap location (next P).
3. Contents of trap location +1 (undefined).
4. Contents of trap location (undefined).
5. Contents of trap location +1.
6. Address where exception occurred +1.

Table C-2. Error Code Indicators

CODE	EXCEPTION	DEFINITION
201	EX:SEM	SEMAPHORE EXCEPTION
202	EX:STP	STRAP OUT OF RANGE
203	EX:STK	INSUFFICIENT STACK SPECIFICATION
204	EX:EMP	UNABLE TO FILL E.M.P. REQUEST
205	EX:SEP	UNABLE TO FILL SYSTEM FREEPOOL
206	EX:PRI	NEGATIVE ACTIVITY PRIORITY
207	EX:CCB	CCB EXCEPTION, TICK CLOCK
208	EX:TRP	HARDWARE TRAP EXCEPTION (X register contains address of hardware trap)
209	EX:TBL	DEBUG VERSION, TABLE ID CHECK FAILURE
20A	EX:SYS	DEBUG VERSION, SYSTEM ACTIVITY VIOLATION
20B	EX:WBC	CCB EXCEPTION, WALL CLOCK
20C	EX:MBC	MAILBOX ID CHECK (INVALID ID)

APPENDIX D

CONFIGURATION OPTIONS

D.1 INTRODUCTION

Configuration options allow the user to tailor his system to meet his needs.

D.2 NONSTANDARD LINE FREQUENCIES

The RTX4 wall clock provides accurate time to ±.25 second precision. RTX4 is delivered with a parameter which relates the wall clock period to the 60 Hz TTLF (Twice the Line Frequency) source. If some other frequency source is used, the programmer must set the contents of location TTLF4: to half the value of the Line Frequency of the Real-Time Clock used. (The absolute location of TTLF4: can be found in the load map.) The default TTLF4: value, for 60 Hz, is 30 (:1E).

The RTX4 source module CLK50: is provided for setting TTLF4: to the correct value for a European system with a Real-Time Clock using 50 Hz Line Frequency. A user on this type of system needs only include the statement:

```
LOAD    CLK50:
```

in his program to set TTLF4: to the appropriate value, 25 (:19).

D.3 PROGRAM RESTARTS WITHOUT RELOADING

Normally, the programmer must load a fresh copy of his program in order to restart. Many variables and pointers are initialized during loading, reducing the size of RTX4 initialization code. Reloading is simple when disk storage is used. However, in paper tape development and some other circumstances, the programmer must be able to restart an RTX4 program without reloading.

In these cases, the programmer can invoke the REINI: option via a LOAD directive in any user module:

```
LOAD    REINI:
```

By invoking the REINI: option, the user assures that all RTX4 variables and pointers are reinitialized whenever RTX4 is restarted.

D.4 DEBUGGING FACILITIES

The Debugging Monitor allows debugging of RTX4 applications. One of the options described below can be included to start execution in DEBUG rather than in RTX4. The programmer can then start RTX4 by jumping or breakpointing to location :80, the normal start location. If an unresolved error condition occurs, a pseudo-breakpoint occurs at location :7E rather than a halt at :7F.

See the NAKED MINI 4 Debugging Monitor Reference Manual for complete information on the Debugging Monitor.

D.4.1 The DEBUG4 Option

This option includes the standard version of the Debugging Monitor. This version provides the following commands:

A	Assign list output device.
J	Re-enter user program or set temporary breakpoints.
B	Set temporary breakpoints or re-enter user program.
R	Display or change user program registers.
G	Display or change general register.
I	Inspect memory.
L	List memory.
F	Fill memory.
S	Search memory.
Z	Print chain.
D	Dump memory.
V	Verify memory.
W	Load binary tape.

This option is invoked via the following LOAD directive in any module:

```
LOAD      DEBUG4
```

D.4.2 The MDBUG4 Option

This option causes the MDBUG4 version of the Debugging Monitor to be included. MDBUG4 provides all of the DEBUG4 commands except:

Z	Print chain.
W	Load binary tape.
D	Dump memory.
V	Verify memory.
A	Assign list output device.

This option invoked via the following LOAD directive in any module:

```
LOAD      MDBUG4
```

D.4.3 The XDBUG4 Option

This option causes the XDBUG4 version of the Debugging Monitor to be included. XDBUG4 provides all of the DEBUG4 commands plus:

M Monitor computer bus.
T Intercept traps.

This option is invoked via the following LOAD directive in any module:

```
LOAD XDBUG4
```

D.5 WALL CLOCK OMISSION

The NOWAL: option allows the programmer to omit the wall clock from his configration. If the programmer has no need for the wall clock services, he can save the time and space normally used for these services by invoking this option via a LOAD directive in any user module:

```
LOAD      NOWAL:
```

APPENDIX E

RTX4/IOS4 APPLICATION DEVELOPMENT SYSTEM GENERATION USING OS4

E.1 INTRODUCTION

This appendix outlines a suggested procedure for creating a system for developing RTX4/IOS4 application programs using the OS4 system. This discussion provides specific, step-by-step instructions for generating such a system on a floppy diskette. The user can modify this procedure as necessary to suit his particular needs.

A sample RTX4/IOS4 application appears at the end of the appendix.

E.2 RECOMMENDED PROCEDURE

To generate an RTX4/IOS4 application development system on a floppy diskette, the programmer can take the following steps:

1. AutoLoad the OS4 system diskette. (The AutoLoad procedure is described in the OS4 System User's Manual and the NAKED MINI 4 AutoLoad manual.

2. Install a new, formatted diskette in Unit 1.

3. Label the new diskette as described in the OS4 user's manual or in the IOS4 user's manual.

4. Remove the OS4 system diskette from Unit 0; install the RTX4 product diskette (F41001).

5. Enter the command:

```
/COPY DF1=DF.RTX.LIB[1]
```

6. Remove the RTX4 product diskette; install the RTX4 macros diskette (F42501).

7. Enter the command:

```
/COPY DF1=DF.GEN.MAC
/COPY DF1=DF.RTX.MAC
/COPY DF1=DF.IOS.MAC
/COPY DF1=DF.SFM.MAC[1]
```

---

[1]If any problem arises during this step, the user must re-install and AutoLoad the OS4 system diskette and then retry this step.

8. Remove the RTX4 macros diskette; install the IOS4 product diskette (F43001).

9. Enter the command:

```
/COPY DF1=DF.IOS.LIB
```

10. Remove the IOS4 product diskette; install the SFM product diskette (F44001).

11. Enter the command:

```
/COPY DF1=DF.SFM.LIB[1]
```

12. Remove the SFM product diskette; install the OS4 system diskette.

13. Enter the command:

```
/COPY DF1.PROGRAM.ASM=CI
```

and respond to the > prompt by entering the symbolic text of the application program; enter a /* command to signal the end of the text.

14. Build a JCL file by entering the command:

```
/COPY DF1.PROGRAM.JCL=CI
```

and respond to the > prompt with the lines:

```
/MACRO PROGRAM(DEFINITION=GEN+RTX+IOS+SFM)
/LINK PROGRAM(AB=100)+SFM+IOS+RTX
```

Enter a /* command to signal the end of the text.

15. Enter the command:

```
/DO PROGRAM
```

to execute the JCL file, which assembles and links the program.

16. Enter the command:

```
/AUTOLOAD DF1.PROGRAM.BIN
```

to execute the program.

17. Debug the program (assuming DEBUG4 was loaded with the program).

Revised 8/78

[1]If any problem arises during this step, the user must re-install and AutoLoad the OS4 system diskette and then retry this step.

18. If corrections to the symbolic version of the program are required, take the following steps:
    a. AutoLoad the system diskette.
    b. Edit the file PROGRAM.ASM by entering the command:
       /EDIT PROGRAM
    c. Perform steps 15, 16, and 17.
    d. If necessary, perform step 18.

19. If the completed application is to reside on the development diskette, rename PROGRAM.BIN or copy it to another file. (To preserve the source code of the program, copy PROGRAM.ASM to another file. To preserve the object version, either rename PROGRAM.OBJ or copy it to another file.) Alternatively, copy PROGRAM.BIN (and optionally PROGRAM.ASM and PROGRAM.OBJ) to another medium (e.g., paper tape or another disk).

This procedure produces a binary file which can be loaded into any NAKED MINI 4® Family computer. If the file resides on a disk, it can be loaded via the /AUTOLOAD command. If the file has been copied to paper tape, it can be loaded via the hardware AutoLoad procedure. The system diskettes are left intact and can be stored in a safe place for backup. The following files are created on the development diskette:

RTX.LIB
GEN.MAC
RTX.MAC
IOS.MAC
SFM.MAC
IOS.LIB
SFM.LIB
PROGRAM.ASM
PROGRAM.JCL
PROGRAM.OBJ (unless renamed in step 19)
PROGRAM.BAK (created when PROGRAM.ASM is edited in step 18)
PROGRAM.BIN (unless renamed in step 18)

and any files created by copying PROGRAM.ASM, PROGRAM.OBJ, and/or PROGRAM.BIN in step 19.

To develop another application, the user needs only edit the PROGRAM.ASM file to contain the new source text and then perform steps 15-19 outlined above.

NOTE

This procedure assumes the standard OS4 configuration, in which the UF logical unit is assigned to the DF01 physical unit. If UF has some other assignment in the user's configuration, the user must include the device specification in the file identifiers specified in steps 14 and 15.

E.3 SAMPLE APPLICATION PROGRAM

The following pages present the assembly listing, link map, and diskette view of a sample RTX4/IOS4 application program.

```
PAGE  0001  MACRO (C1) RTX4/IOS4 EXAMPLE APPLICATION PROGRAM
1979/01/18 20:41:21.25 INITIALIZATION

      0000        0003              NAM       R:INIT          INITIALIZATION BLOCK NAME
                  0004              EXTR      ECB             ENVIRONMENT CONTROL BLOCK
                  0005              EXTR      TDB             TASK DESCRIPTOR BLOCK
      00000000    0006  AR          EQU       0               INITIAL A REGISTER,
      00000000    0007  QR          EQU       0               Q REGISTER,
      00000000    0008  XR          EQU       0               X REGISTER,
      00000000    0009  YR          EQU       0               AND Y REGISTER
      0000000A    0010  PRIORITY    EQU       10              ACTIVITY PRIORITY
      00000100    0011  FREEPOOL    EQU       :100            FREEPOOL SIZE
0000  0000        0012              INIT:A    AR,QR,XR,YR,ECB,TDB,PRIORITY,FREEPOOL
0001  0000
0002  0000
0003  0000
0004  0003
0005  0002
0006  000A
0007  F00E        0012+
0008  0100        0012+
0009  000B        0012+
000A  0000        0012+
000B  8080        0012+
                  0013              END

0000   ERRORS     (0000)
0000   WARNINGS   (0000)
```

Figure E-1. RTX4/IOS4 Example Application Program, Page 1 of 7

```
PAGE  0002  MACRO (C1) RTX4/IOS4 EXAMPLE APPLICATION PROGRAM
1979/01/18 20:41:23.25 UNIT ASSIGNMENT TABLE

      0001          0016              NAM       UAT             TABLE NAME
0000  F09E          0017              UAT:AA    UAT
0001  0001          0017+
0002  0000          0017+
0005  5456          0018              UAT:EE    'TV',0:TV00
0006  0000          0018+
      00000007      0019              UAT:ZZ
                    0020              END

0000   ERRORS     (0000)
0000   WARNINGS   (0000)
```

Figure E-1. RTX4/IOS4 Example Application Program, Page 2 of 7

```
                    0023  * TASK DESCRIPTOR BLOCK
     0000           0024          NAM       TDB              TDB NAME
     00000000       0025  STACKAD EQU       0                HAVE RTX ALLOCATE STACK
     00000050       0026  STACKAM EQU       :50              STACK SIZE
     00000000       0027  FLAGS   EQU       0                NO FLAGS
     00000001       0028  USAGE   EQU       1                ONE CONCURRENT ACTIVITY
0000                0029          TDB:A     TDB,START,0,STACKAD,STACKAM,FLAGS,USAGE
0001 0040           0029+
0002 0008           0029+
0003 0001           0029+
0004 0000           0029+
0005 0000           0029+
0006 0050           0029+
0007 000C           0029+
0008 0001           0029+
0009 0000           0029+
000B F01E           0029+
                    0030  * THE ACTIVITY
     0000000C       0031  START   EQU       $                START ADDRESS
000C 3A07           0032          I:IO      CRTIOB           OPEN THE CRT
000D 0017           0032+
000E 0E0D           0033          HLT                        ABNORMAL RETURN
     0000000F       0034  LOOP    EQU       $
000F 3A07           0035          I:IO      MSGIOB           WRITE MESSAGE TO CRT
0010 001F           0035+
0011 0E0D           0036          HLT                        ABNORMAL RETURN
0012 3A0B           0037          R:ITIC    TIMER            START TIMER
0013 002F           0037+
0014 3A02           0038          R:WAIT    SEMA4            WAIT FOR TIME TO EXPIRE
0015 0035           0038+
0016 9E78 000F      0039          JMP       LOOP             GO DISPLAY MESSAGE AGAIN
                    0040  * I/O BLOCK TO OPEN CRT
0017 5456           0041          IOB:A     CRTIOB,'TV',FU:,UP:,0,0,0
0018 0000           0041+
0019 0000           0041+
001A 0030           0041+
```

Figure E-1. RTX4/IOS4 Example Application Program, Page 3 of 7

```
001B 0000          0041+
001C 0000          0041+
001D 0000          0041+
001E 0000          0041+
                   0042   * I/O BLOCK TO WRITE TO CRT
001F 5456          0043             IOB:A     MSGIOB,'TV',WR:,FA:,COUNT,BUFFER,0
0020 0000          0043+
0021 0000          0043+
0022 001B          0043+
0023 0010          0043+
0024 0027          0043+
0025 0000          0043+
0026 0000          0043+
                   0044   * MESSAGE TEXT BUFFER
     0000000D      0045   CR        EQU       :0D              ASCII CARRIAGE RETURN
     0000000A      0046   LF        EQU       :0A              ASCII LINE FEED
0027 204D          0047   BUFFER    BYTE      ' MESSAGE TEXT ',CR,LF
0028 4553
0029 5341
002A 4745
002B 2054
002C 4558
002D 5420
002E 0D0A
     00000010      0048   COUNT     EQU       $-BUFFER*2       BYTE COUNT
                   0049   * TIMER CONTROL BLOCK
     0000002F      0050   ID        EQU       $                UNIQUE 16 BIT TIMER ID
     00000258      0051   TICKS     EQU       120*5            TIME = 5 SECONDS
002F 0000          0052             TICK:A    TIMER,ID,SEMA4,TICKS
0030 002F          0052+
0031 0035          0052+
0032 0258          0052+
                   0053   * SEMAPHORE DESCRIPTOR BLOCK
     00000000      0054   VALUE     EQU       0                INITIAL VALUE
0033               0055             SDB:A     SEMA4,VALUE
0034 0000          0055+
```

Figure E-1. RTX4/IOS4 Example Application Program, Page 4 of 7

```
PAGE  0005  MACRO (C1) RTX4/IOS4 EXAMPLE APPLICATION PROGRAM
1979/01/18 20:41:30.00 THE TASK

0035 0000          0055+
0036 F03E          0055+
                   0056                END

0000   ERRORS      (0000)
0000   WARNINGS    (0000)
```

Figure E-1. RTX4/IOS4 Example Application Program, Page 5 of 7

```
PAGE  0006  MACRO (C1) RTX4/IOS4 EXAMPLE APPLICATION PROGRAM
1979/01/18 20:41:55.00 ENVIRONMENT CONTROL BLOCK

      0000          0059              NAM       ECB             ENVIRONMENT CONTROL BLOCK NAME
                    0060              EXTR      UAT             UNIT ASSIGNMENT TABLE
0000                0061              ECB:A     ECB,UAT
0001 0000           0061+
0002 0000           0061+
0003 0001           0061+
0004 0000           0061+
0006 0000           0061+
0007 0000           0061+
0008 0000           0061+
0009 0000           0061+
000A 0000           0061+
000B F06E           0061+
000C 0010           0061+
000D 0030           0061+
000E 0000           0061+
0010 0000           0061+
0020 0000           0061+
0021 0000           0061+
0022 0000           0061+
0023 0000           0061+
0024 0000           0061+
0025 7FFF           0061+
0026                0061+
0027                0061+
0028                0061+
0029 0000           0061+
                    0062              END

0000  ERRORS     (0000)
0000  WARNINGS   (0000)
```

Figure E-1. RTX4/IOS4 Example Application Program, Page 6 of 7

```
PAGE  0007  MACRO (C1)
1979/01/18 20:41:33.75

**** GRAND-TOTALS ****

0000  ERRORS    (0000)
0000  WARNINGS  (0000)
```

Figure E-1. RTX4/IOS4 Example Application Program, Page 7 of 7

```
PAGE   1          79/01/18  20:42:55        LINK   (A4)


SO FILE                    = PROGRAM     BIN
SI FILE                    = PROGRAM     OBJ
SA FILE(S)                 = IOS         LIB
                             RTX         LIB

STATUS                     = RELOCATABLE
                             WITHIN MEMORY LIMITS
                             UNRESOLVED SECONDARIES
LOAD OFFSET                = 0000
TRANSFER ADDRESS           = 0080
MAIN MEMORY LIMITS         = 0000-FFFF


(ABSOLUTE SYMBOLS)
  0066....R:CNTR     0068....R:CNSM     006A....R:WLKS     006C....R:PFLG
  006D....R:PFK      006E....R:SREG     006F....R:SNSW     0070....R:CDRG
  0071....R:BTC1     0072....R:MPM1     0073....R:MPM2     0078....R:RTCI
  007E....R:FATL     007F....R:FAT1     0080....RTX:

(REL AREA   1)    BLANK    (0000-1002R   - RAM)
  0000....R:INIT     010C....DAT        0112....IDB        0149....ECB
  017A....I:IO       017A....R:LOW      01F5....I:SLO      021C....I:DOIO
  0238....I:RET      0242....I:STOP     0245....I:STOO     0268....I:STOR
  027C....I:WAIT     0284....I:FINI     02A0....I:NEOB     02AC....I:POWR
  02B9....I:WDT      02EE....I:DOGY     02F0....I:LWDT     0300....I:KWDT
  0307....I:WDAC     0313,(M)I:INIT     0320....I:ERTB     0332....I:ERS1
  0333....I:ERS2     0334....I:ERS3     0335....I:DOSV     0335....I:ABRT
  0335....I:DOER     0336....D:TV00     0352....C:TY0      036A....TYEOL:
  036C....TYTOF:     036F....TYELI:     0371....TYBUF:     0374....C:HEAD
  0375....LP:WRT     0375....TY:WRT     0375....PP:WRT     04AF....PR:RD
  04AF....TY:RD      0603....TY:FUN     0603....LP:FUN     0603....PP:FUN
  06DB....FBR:       0764....FBW:       07E9....I:TRLL     07F0....I:TRTB
  0800....I:DIO      0801....I:SDIO     080B....I:SIV      0829....I:RPF
  082E....I:COMP     0832....I:STAT     0840....I:RST      0846....I:EMER
  085F....I:EOB      0863....I:ECTT     0872....I:VCNT     0877....I:OPEN
  0884....I:OPCL     0899....R:IDUR     08E5....R:UDUR     08E8....R:SERR
```

Figure E-2. Memory Map of Linked RTX4/IOS4 Example Application Program, Page 1 of 2

```
  0906....R:SERL     0907....R:BGIN     092A....R:BEGI     09BE....R:END
  09F0....R:PRID     09FC....R:ISIG     09FC....R:SIG      0A05....R:SSIG
  0A1C....R:IWAI     0A1C....R:WAIT     0A24....R:SWAI     0A4B....R:ITIC
  0A9D....R:TICI     0AA3....R:TICP     0ACA....R:LTIC     0ADA....R:KTIC
  0AE1....R:TKAC     0AED....R:CTIC     0B1C....R:MTIC     0B37....R:RWAL
  0B66....R:WLAC     0BB3....TTLF4:     0BBA....R:STOD     0BC2....R:GTOD
  0BCA....R:AWAL     0BE9....R:IWAL     0C33....R:CWAL     0C67....STILF:
  0C68....R:GPRI     0C6C....R:GPR      0C70....R:SPRI     0C84....R:CINT
  0C9E....R:CNSL     0CAF....R:SEND     0CBE....R:ISND     0CE6....R:RECV
  0CF8....R:IRCV     0D2D....R:GETM     0D55....R:GIVM     0D8A....R:ABUF
  0D95....R:RBUF     0DA1....R:RE12     0DA1....G:Z        0DA4....R:DAWX
  0DA8....R:DA       0DAC....R:DISP     0DE2....R:SACT     0DE8....R:IACB
  0E12....R:LSTK     0E52....R:KSTK     0E52....R:STRT     0E60....R:INIT
  0E7F....R:PWRF     0EA0....R:GTS      0EAB....R:GVST     0EB5....R:GVSH
  0EBE....R:PAUS     0ED7....R:AETH     0EDF....R:CNTH     0EE5....R:USTH
  0EEC....R:UMTH     0EF3....R:UITH     0EF5....R:UIRTN    0F08....R:USTREX
  0F1D....R:SETH     0F38....R:STROEX   0F3A....R:DOORKEX  0F48....R:RTXEX
  0F48....R:XPTE     0F4C....R:TABL     0F51....R:SYSX     0F69....R:DCHK
  0F7C....R:RINT     0F7C....R:UINI     0F7E.(S)DEBUG4     0F80....R:EMUL
  0F81....R:PATC     0F9C....R:NOPF     1002....R:HIGH

(REL AREA    2)    R:HIGHER(1003-1003R   - RAM)
  1003....R:HIGHER



* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

MISSING                        = F:MONT       F:CREA       R:SAID
                                 F:CFNO       R:GAID       F:DMNT
                                 F:DELE       F:CONN
```

Figure E-2. Memory Map of Linked RTX4/IOS4 Example Application Program, Page 2 of 2

APPENDIX F

RTX4 DEMONSTRATION PROGRAM

The RTX4 Demonstration Program is designed to exercise the system services of RTX4. The services exercised are:

- R:BGIN — Begin on activity
- R:SIG — Signal a semaphore
- R:WAIT — Wait on a semaphore
- R:CINT — Wait for console interrupt
- R:ITIC — Tick clock timer

When the program is executed it transfers immediately to DEBUG4. To begin execution of the program, jump to location :80. From here the program goes through the initialization, then begin an activity of the Master Task. This activity waits for a console interrupt. At this time the system is in the dispatcher idle loop, which is indicated by blinking the byte mode and overflow indicator lights. The user should now press console interrupt which causes an activity of the Timer Task to be started by the Master Task. Each of the next 15 console interrupts causes another timer activity to start until 16 timer activities exist. All subsequent console interrupts are ignored.

Each timer activity causes a different bit of the console data register to blink. The location and frequency of these bits is determined by a table. The frequency is based on tick clock intervals.

At any time a power fail may be caused and the system recovers completely when power is restored, provided the memories are properly powered.

NOTE

The missing external R:DBUG which shows on the link map will not affect the execution of RTX4.

```
PAGE  0001  MACRO (C1) RTX4 DEMO PROGRAM NO. 1                    93410-10 B3
1979/01/17 02:59:34.25

      0000        0002              NAM       R:INIT
                  0003              LOAD      REIN1:
                  0004              LOAD      DEBUG4
                  0005  *
                  0006  *       THIS PROGRAM TESTS THE TICK CLOCK SERVICES.
                  0007  *
                  0008  *       THE TEST CONSISTS OF TWO TASKS, THE MASTER
                  0009  *       AND THE TIMER.  THE MASTER IS THE INITIAL
                  0010  *       TASK AND THERE IS ONLY ONE ACTIVITY OF
                  0011  *       IT.  THIS ACTIVITY BEGINS AN ACTIVITY OF
                  0012  *       THE TIMER TASK EACH TIME CONSOLE INTERRUPT
                  0013  *       IS PRESSED, UNTIL 16 ACTIVITIES OF THE TIMER
                  0014  *       TASK ARE BEGUN.  FURTHER CONSOLE INTERRUPTS
                  0015  *       ARE IGNORED.
                  0016  *
                  0017  *       THE TIMER TASK MAKES A TICK CLOCK REQUEST
                  0018  *       ACCORDING TO THE PARAMETERS IN ITS Y SCRATCHPAD.
                  0019  *       WHEN THE TIMER EXPIRES IT COMPLEMENTS A
                  0020  *       BIT IN THE CONSOLE WORD REGISTER AND REPEATS.
                  0021  *
      00000008    0022  TBLSIZ   EQU       8                 NUMBER OF WORD/TABLE ENTRY
                  0023  *
                  0024  *       BEGIN MASTER TASK WITH:   A=0
                  0025  *                                 Q=0
                  0026  *                                 X=0
                  0027  *                                 Y=TABLE
                  0028  *       AT PRIORITY :100 WITH :200 WORDS OF FREEPOOL
                  0029  *
0000 0000         0030              INIT:A    0,0,0,TABLE,ECB1,MSTRTD,:100,:200
0001 0000
0002 0000
0003 024A
0004 030A
0005 020B
0006 0100
0007 F00E         0030+
```

Figure F-1. RTX4 Demonstration Program, Page 1 of 13

```
                                                        93410-10 R3

PAGE  0002  MACRO (C1) RTX4 DEMO PROGRAM NO. 1
1979/01/17 02:59:35.15

0008 0200          0050+
0009 000B          0050+
000A 0000          0050+
000B 8080          0050+
```

Figure F-1. RTX4 Demonstration Program, Page 2 of 13

```
                    0032  *
                    0033  *       THIS TASK BEGINS THE TIMER TASK.
                    0034  *         EACH TIME CONSOLE INTERRUPT IS PRESSED
                    0035  *         A NEW TIMER ACTIVITY IS CREATED.
                    0036  *
                    0037  *
                    0038  *       MASTER TASK WILL HAVE Y EQUAL TO TABLE
                    0039  *                                    :35 WORDS OF STACK, SYSTEM ALLOCATED
                    0040  *                                      1 POSSIBLE CONCURRENT EXECUTIONS
                    0041  *
0208                0042              TDB:A     MSTRTD,MASTER,0,0,:35,0,1
020C 0040           0042+
020D 0213           0042+
020E 0001           0042+
020F 0000           0042+
0210 0000           0042+
0211 0035           0042+
0212 0217           0042+
0213 0001           0042+
0214 0000           0042+
0216 F01E           0042+
     00000217       0043  MASTER    EQU       $
0217 4910           0044              COPY      =16,0           ALLOW ONLY 16 TIMERS
     00000218       0045  LOOP      EQU       $
0218 5546 021F      0046              JEQD      0,NOMORE        IF 16 TIMERS ALREADY
0219 1A08           0047              R:CINT
021A 0000           0047+
021B 1A03           0048              R:BGIN    TDB(Y)          BEGIN TIMER ACTIVITY
021C 1000 0000      0048+
021D 6B08           0049              ADD       =TBLSIZ,Y       MOVE TABLE POINTER TO NEXT ENTRY
021E 9E79 0218      0050              JMP       LOOP            REPEAT
                    0051  * 16 TIMERS ARE RUNNING, IGNORE FURTHER CONSOLE INTERRUPTS
     0000021F       0052  NOMORE    EQU       $
021F 1A08           0053              R:CINT
0220 0000           0053+
0221 9E7D 021F      0054              JMP       NOMORE
```

```
                                                        93410-10 B5

PAGE 0004  MACRO (C1) RTX4 DEMO PROGRAM NO. 1
1979/01/17 02:59:37.75 MASTER TASK

0222  0001      0055      LPOOL
```

Figure F-1. RTX4 Demonstration Program, Page 4 of 13

```
                    0057  *
                    0058  *     *  TIMER  TASK  *
                    0059  *
                    0060  *       TIMER TASK WILL HAVE:  Y OF ACTIVITY DOING BEGIN
                    0061  *                                :20 WORDS OF STACK SPACE, SYSTEM ALLO
                    0062  *                                 16 POSSIBLE CONCURRENT EXECUTIONS
                    0063  *
0223                0064              TDB:A     TMRTD,TIMER,0,0,:20,0,16
0224 0040           0064+
0225 022B           0064+
0226 0010           0064+
0227 0000           0064+
0228 0000           0064+
0229 0020           0064+
022A 022F           0064+
022B 0010           0064+
022C 0000           0064+
022E F01E           0064+
                    0065  *
                    0066  *       DELAYS N TICKS THEN TOGGLES CONSOLE WORD REGISTER BIT
                    0067  *       BEGAN WITH  Y= TABLE ENTRY
                    0068  *
     0000022F       0069  TIMER     EQU       $
022F 1A0B           0070              R:ITIC    ID(Y)           INITIATE TIMER REQUEST
0230 1002 0002      0070+
0231 DC46 0006      0071              IMS       DELAYS(Y)       BUMP DELAY COUNTER
0232 0000           0072              NOP
0233 FE83 0237      0073              JSK       OUT             TOGGLE CONSOLE WORD REGISTER BIT
0234 1A02           0074              R:WAIT    *SEMADR(Y)      WAIT FOR TIMER TO EXPIRE
0235 5004 0004      0074+
0236 9E78 022F      0075              JMP       TIMER           REPEAT
```

Figure F-1. RTX4 Demonstration Program, Page 5 of 13

```
                    0077  *       COMPLEMENTS THE ADDRESSED BIT
                    0078  *       CALLED WITH Y= TABLE ENTRY
                    0079  *
     00000237       0080  OUT       EQU       $
0237 C047 0007      0081            COPY      BIT(Y),0        GET BIT TO TOGGLE
0238 4E31           0082            SHIFT     0,LO,4          POSITION TO K4 FIELD OF CBIT INSTR
0239 3A02           0083            R:WAIT    CDR             REQUEST USE OF CONSOLE WORD REGIST
023A 0244           0083+
023B 0104           0084            IN        CONDA:+CDR:,A   READ CONSOLE WORD REGISTER
023C 430A           0085            XNX       0               INDEX CBIT INSTRUCTION WITH BIT AI
023D 000E           0086            CBIT      0,A             COMPLEMENT ADDRESSED BIT
023E 0404           0087            SELP      A,CONDA:+CDR:   OUTPUT TO CONSOLE WORD REGISTER
023F 3A01           0088            R:SIG     CDR             GIVE UP USE OF CONSOLE WORD REGIS
0240 0244           0088+
0241 2309           0089            RSK                       RETURN
                    0090  *
                    0091  *    * SEMAPHORE TO CONTROL CONSOLE WORD REGISTER ACCESS *
                    0092  *
0242                0093            SUB:A     CDR,1
0243 0001           0093+
0244 0001           0093+
0245 F03E           0093+
     0004           0094            LPOOL
0246
0247
0248
0249
```

Figure F-1. RTX4 Demonstration Program, Page 6 of 13

```
                    0096  *
                    0097  *    THIS TABLE DEFINES THE PARAMATERS FOR EACH ACTIVITY
                    0098  *         OF THE TIMER TASK.
                    0099  *    FORMAT IS:
                    0100  *
    00000000        0101  TDB       EQU     0               TASK DESCRIPTOR BLOCK ADDRESS
    00000001        0102  PRI       EQU     1               PRIORITY TO BEGIN ACTIVITY AT
    00000002        0103  ID        EQU     2               ID TO USE IN TIMER REQUESTS
    00000004        0104  SEMADR    EQU     4               ADDRESS OF TICK REQUEST SEMAPHORE
    00000005        0105  TICKS     EQU     5               NUMBER OF TICKS TO REQUEST
    00000006        0106  DELAYS    EQU     6               NUMBER OF REQUEST MADE SO FAR BY
    00000007        0107  BIT       EQU     7               BIT TO TOGGLE
                    0108  *
                    0109  *
                    0110  *
    0000024A        0111  TABLE     EQU     $
024A 0223           0112            WORD    TMRTD,:100,0,:3004,SEM3,4,0,3
024B 0100
024C 0000
024D 3004
024E 02D8
024F 0004
0250 0000
0251 0003
0252 0223           0113            WORD    TMRTD,:100,0,:2003,SEM2,3,0,2
0253 0100
0254 0000
0255 2003
0256 02D4
0257 0003
0258 0000
0259 0002
025A 0223           0114            WORD    TMRTD,:100,0,:C00D,SEMC,:D,0,:C
025B 0100
025C 0000
025D C00D
```

Figure F-1. RTX4 Demonstration Program, Page 7 of 13

```
025E 02FC
025F 000D
0260 0000
0261 000C
0262 0223       0115            WORD    TMRTD,:100,0,:6007,SEM6,7,0,6
0263 0100
0264 0000
0265 6007
0266 02E4
0267 0007
0268 0000
0269 0006
026A 0223       0116            WORD    TMRTD,:100,0,:E00F,SEME,:F,0,:E
026B 0100
026C 0000
026D E00F
026E 0304
026F 000F
0270 0000
0271 000E
0272 0223       0117            WORD    TMRTD,:100,0,:4005,SEM4,5,0,4
0273 0100
0274 0000
0275 4005
0276 02DC
0277 0005
0278 0000
0279 0004
027A 0223       0118            WORD    TMRTD,:100,0,:8009,SEM8,9,0,8
027B 0100
027C 0000
027D 8009
027E 02EC
027F 0009
0280 0000
0281 0008
```

```
0282 0223        0119            WORD    TMRTD,:100,0,:0001,SEM0,:1,0,0
0283 0100
0284 0000
0285 0001
0286 02CC
0287 0001
0288 0000
0289 0000
028A 0223        0120            WORD    TMRTD,:100,0,:900A,SEM9,:A,0,9
028B 0100
028C 0000
028D 900A
028E 02F0
028F 000A
0290 0000
0291 0009
0292 0223        0121            WORD    TMRTD,:100,0,:F010,SEMF,:10,0,:F
0293 0100
0294 0000
0295 F010
0296 0308
0297 0010
0298 0000
0299 000F
029A 0223        0122            WORD    TMRTD,:100,0,:7008,SEM7,8,0,7
029B 0100
029C 0000
029D 7008
029E 02E8
029F 0008
02A0 0000
02A1 0007
02A2 0223        0123            WORD    TMRTD,:100,0,:D00E,SEMD,:E,0,:D
02A3 0100
02A4 0000
02A5 D00E
```

Figure F-1. RTX4 Demonstration Program, Page 9 of 13

```
02A6 0300
02A7 000E
02A8 0000
02A9 000D
02AA 0223       0124            WORD    TMRTD,:100,0,:A00B,SEMA,:B,0,:A
02AB 0100
02AC 0000
02AD A00B
02AE 02F4
02AF 000B
02B0 0000
02B1 000A
02B2 0223       0125            WORD    TMRTD,:100,0,:5006,SEM5,6,0,5
02B3 0100
02B4 0000
02B5 5006
02B6 02E0
02B7 0006
02B8 0000
02B9 0005
02BA 0223       0126            WORD    TMRTD,:100,0,:B00C,SEMB,:C,0,:B
02BB 0100
02BC 0000
02BD B00C
02BE 02F8
02BF 000C
02C0 0000
02C1 000B
02C2 0223       0127            WORD    TMRTD,:100,0,:1002,SEM1,2,0,1
02C3 0100
02C4 0000
02C5 1002
02C6 02D0
02C7 0002
02C8 0000
02C9 0001
```

Figure F-1. RTX4 Demonstration Program, Page 10 of 13

```
02CA                0129            SDB:A     SEM0,0
02CB 0000           0129+
02CC 0000           0129+
02CD F03E           0129+
02CE                0130            SDB:A     SEM1,0
02CF 0000           0130+
02D0 0000           0130+
02D1 F03E           0130+
02D2                0131            SDB:A     SEM2,0
02D3 0000           0131+
02D4 0000           0131+
02D5 F03E           0131+
02D6                0132            SDB:A     SEM3,0
02D7 0000           0132+
02D8 0000           0132+
02D9 F03E           0132+
02DA                0133            SDB:A     SEM4,0
02DB 0000           0133+
02DC 0000           0133+
02DD F03E           0133+
02DE                0134            SDB:A     SEM5,0
02DF 0000           0134+
02E0 0000           0134+
02E1 F03E           0134+
02E2                0135            SDB:A     SEM6,0
02E3 0000           0135+
02E4 0000           0135+
02E5 F03E           0135+
02E6                0136            SDB:A     SEM7,0
02E7 0000           0136+
02E8 0000           0136+
02E9 F03E           0136+
02EA                0137            SDB:A     SEM8,0
02EB 0000           0137+
02EC 0000           0137+
02ED F03E           0137+
```

Figure F-1. RTX4 Demonstration Program, Page 11 of 13

```
02EE             0138          SDB:A     SEM9,0
02EF 0000        0138+
02F0 0000        0138+
02F1 F03E        0138+
02F2             0139          SDB:A     SEMA,0
02F3 0000        0139+
02F4 0000        0139+
02F5 F03E        0139+
02F6             0140          SDB:A     SEMB,0
02F7 0000        0140+
02F8 0000        0140+
02F9 F03E        0140+
02FA             0141          SDB:A     SEMC,0
02FB 0000        0141+
02FC 0000        0141+
02FD F03E        0141+
02FE             0142          SDB:A     SEMD,0
02FF 0000        0142+
0300 0000        0142+
0301 F03E        0142+
0302             0143          SDB:A     SEME,0
0303 0000        0143+
0304 0000        0143+
0305 F03E        0143+
0306             0144          SDB:A     SEMF,0
0307 0000        0144+
0308 0000        0144+
0309 F03E        0144+
```

Figure F-1. RTX4 Demonstration Program, Page 12 of 13

```
PAGE  0013  MACRO (C1) RTX4 DEMO PROGRAM NO. 1                    93410-10 B3
1979/01/17 02:59:47.25 ENVIRONMENT INFORMATION

                    0146   *
                    0147   *    DEFINE ENVIRONMENT CONTROL BLOCK
                    0148   *
030A                0149            ECB:A     ECB1,0
030B  0000          0149+
030C  030A          0149+
030D  0001          0149+
030E  0000          0149+
0310  0000          0149+
0311  0000          0149+
0312  0000          0149+
0313  0000          0149+
0314  0000          0149+
0315  F06E          0149+
0316  0010          0149+
0317  0030          0149+
0318  0000          0149+
031A  0000          0149+
032A  0000          0149+
032B  0000          0149+
032C  0000          0149+
032D  0000          0149+
032E  0000          0149+
032F  7FFF          0149+
0330                0149+
0331                0149+
0332                0149+
0333  0000          0149+
                    0150            END

0000  ERRORS     (0000)
0000  WARNINGS   (0000)
```

Figure F-1. RTX4 Demonstration Program, Page 13 of 13

```
PAGE   1          79/01/17  03:00:04        LINK   (A4)


SO FILE                    = RTXDEMO     BIN
SI FILE                    = RTXDEMO     OBJ
SA FILE(S)                 = RTX         LIB

STATUS                     = RELOCATABLE
                             WITHIN MEMORY LIMITS
                             UNRESOLVED SECONDARIES
LOAD OFFSET                = 0100
TRANSFER ADDRESS           = 043B
MAIN MEMORY LIMITS         = 0000-FFFF


(ABSOLUTE SYMBOLS)
  0066....R:CNTK    0068....R:CNSM    006A....R:WLKS    006C....R:PFLG
  006D....R:PFK     006E....R:SREG    006F....R:SNSW    0070....R:CDRG
  0071....R:BTC1    0072....R:MPM1    0073....R:MPM2    007E....R:FATL
  007F....R:FAT1    0080....RTX:

(REL AREA   1)   BLANK   (0100-134FR  - RAM)
  0100....R:INIT    0100....R:LOW     043B.(M)DEBUG4    0B8F....R:IDOR
  0BD9....R:ODOR    0BDE....R:SERB    0BFC....R:SERL    0BFD....R:BGIN
  0C20....R:BEGI    0CB4....R:END     0CE6....R:PRID    0CF2....R:ISIG
  0CF2....R:SIG     0CFB....R:SSIG    0D12....R:IWAI    0D12....R:WAIT
  0D1A....R:SWAI    0D41....R:ITIC    0D93....R:TICI    0D99....R:TICP
  0DC0....R:LTIC    0DD0....R:KTIC    0DD7....R:TKAC    0DE3....R:CTIC
  0E12....R:MTIC    0E2D....R:RWAL    0E5C....R:WLAC    0EA9....TTLF4:
  0EB0....R:STOD    0EB8....R:GTOD    0EC0....R:AWAL    0FDF....R:IWAL
  0F29....R:CWAL    0F5D....STTLF:    0F5E....R:GPRI    0F62....R:GPR
  0F66....R:SPRI    0F7A....R:CINT    0F94....R:CNSL    0FA5....R:SEND
  0FB4....R:ISND    0FDC....R:RECV    0FFE....R:IRCV    1023....R:GETM
  104B....R:GIVM    1080....R:ABUF    108B....R:RBUF    1097....R:RE12
  1097....G:Z       109A....R:DAWX    109E....R:DA      10A2....R:DISP
  10D8....R:SACT    10DE....R:IACB    1108....R:LSTK    1148....R:KSTK
  1148....R:STRT    1156....R:INIT    1175....R:PWRF    1196....R:GTS
  11A1....R:GVST    11AB....R:GVSH    11B4....R:PAUS    11CD....R:AETH
  11D4....R:CNTH    11DB....R:USTH    11E2....R:UWTH    11E9....R:UITH
```

Figure F-2. Memory Map of Linked RTX4 Demonstration Program, Page 1 of 2

```
11EB....R:UIRTN    11FE....R:USTREX   1213....R:SETH     122E....R:STROEX
1230....R:DODREX   123E....R:RTXEX    123F....R:XPTE     1242....R:TABL
1247....R:SYSX     125F....REINI:     125F.(M)R:RINT     12AE....R:SECB
12B3....R:NECB     12B6....R:DECH     12B7....R:DCHK     12CA....I:INIT
12CA....R:UINI     12CE....R:EMUL     12CF....R:PATC     12EA....R:NOPF
1350....R:HIGH

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

MISSING                 = G:4          G:E          F:MONT
                          G:V          G:0          G:L
                          G:6          G:G          I:IO
                          G:X          G:1          G:B
                          G:S          F:CREA       G:N
                          R:SATD       G:8          F:CFNO
                          G:I          G:3          G:D
                          G:U          G:P          R:GATD
                          F:DMNT       G:K          G:5
                          G:F          G:W          G:0
                          G:A          F:DELE       G:R
                          G:M          G:7          G:H
                          G:Y          G:2          G:C
                          G:T          G:0          G:9
                          F:CONN       G:J
```

Figure F-2. Memory Map of Linked RTX4 Demonstration Program, Page 2 of 2

APPENDIX G

RTX4 MACRO SUMMARY

Page			
4-3	BGIN:A	arg,tdb,prdesc	
		where: arg	Must match the argument specified in the R:BGIN macro.
		tdb	Address of the Task Descriptor Block as specified in the TDB:A macro.
		prdesc	Priority descriptor defining the task's priority.
		This macro generates an argument list for an R:BGIN macro.	
5-6	ECB:A	label,uat	
		where: label	Label to be assigned to the start of the Environment Control Block; referenced in the INIT:A macro.
		uat	Address of the Unit Assignment Table.
		This macro generates the Environment Control Block. Must immediately precede the END statement of the last user's program module.	
5-1	INIT:A	a,q,x,y,ecb,tdb,pri,amtfree,adrfree,topmem	
		where: a,q,x,y	Initial values of the A, Q, X, and Y registers for initial user's task.
		ecb	Address of the Environment Control Block.
		tdb	Address of the Task Descriptor Block for initial user's task.
		pri	Activity priority for initial user's task.
		amtfree	Amount of freepool space in words (optional).
		adrfree	Address of the freepool (optional).
		topmem	Upper limit of memory available to RTX4 (optional).
		This macro generates the Initialization Block.	

Page

8-4 MAIL:A arg,mail

where: arg — Must match the argument of an R:SEND macro or R:RECV macro.

mail — Label of the appropriate mailbox as defined by the MDB:A macro.

This macro generates an argument list for an R:SEND macro or R:RECV macro.

8-3 MDB:A mail

where: mail — Identifier to be assigned to the mailbox.

This macro defines a mailbox facility.

5-7 R:ABUF amount

where: amount — Number of words to be allocated.

This macro submits a request for the system buffer allocation service. This service allocates a buffer for the program's use.

7-8 R:AWAL arg

where: arg — M4D12 pointer to the argument list, generated via the WALL:A macro.

This macro submits a request for the system wall clock absolute timer service. This service initiates a timer to cause a semaphore to be signalled at an absolute wall clock time.

4-3 R:BGIN arg

where: arg — M4D12 pointer to the argument list, generated via the BGIN:A macro

This macro submits a request for the system task-processing service. This service initiates task execution; i.e., it creates an activity.

4-4 R:CINT

This macro submits a request for the system console interrupt control service. This service causes the activity to return if the console interrupt is pressed.

Page

7-4 R:CTIC arg

where: arg M4D12 pointer to the argument list generated by the TICK:A macro.

This macro submits a request for the system tick clock timer request cancellation service. This service cancels a previous R:ITIC request.

7-9 R:CWAL arg

where: arg M4D12 pointer to the argument list, generated via the WALL:A macro.

This macro submits a request for the system wall clock timer request cancellation service. This service cancels a previous R:IWAL or R:AWAL request.

4-3 R:END

This macro submits a request for the system activity termination service. This service terminates the activity requesting the service.

7-7 R:GATD

This macro submits a request for the system time and date access service. This service reads the time and date in ASCII.

4-4 R:GPRI

This macro submits a request for the system activity priority access service. This service returns the calling activity's priority in the A register.

7-7 R:GTOD

This macro submits a request for the system time of day access service. This service returns the time of day in the AQ register pair.

7-3 R:ITIC arg

where: arg M4D12 pointer to the argument list, generated via the TICK:A macro.

This macro submits a request for the system tick clock timer service. This service initiates a timer to cause a semaphore to be signalled after a specified number of ticks of the Real-Time Clock.

Page

7-8 R:IWAL arg

where: arg M4D12 pointer to the argument list, generated via the WALL:A macro.

This macro generates a request for the system wall clock interval timer service. This service initiates a timer to cause a semaphore to be signalled after a specified time interval has elapsed.

7-3 R:MTIC arg

where: arg M4D12 pointer to the argument list, generated via the TICK:A macro.

This macro submits a request for the system tick clock timer request modification service. This service modifies a previous R:ITIC request.

7-5 R:PAUS prdesc

where: prdesc Priority descriptor.

This macro submits a request for the system round robin scheduling service. This service removes the first activity of a given priority from the ready list and reenters that activity into the ready list.

5-7 R:RBUF address

where: address Address of the buffer to be released.

This macro submits a request for the system buffer release service. This service releases space previously allocated for a specified buffer.

8-4 R:RECV arg

where: arg M4D12 pointer to the argument list, generated via the MAIL:A macro.

This macro submits a request for the system message receipt service. This service receives a message from a specified mailbox.

3-10 R:SATD arg

where: arg Address of argument block...a sevenword block containing date and time values in the order: year, month, day, hour, minute, second, and hundredths of a second.

This macro submits a request for the system date and time definition service. This service sets the date and time in ASCII.

8-4 R:SEND arg

where: arg M4D12 pointer to the argument list, generated via the MAIL:A macro.

This macro submits a request for the system message transmission service. This service sends a message to a specified mailbox.

6-7 R:SIG sema4

where: sema4 Address of the Semaphore Definition Block to be signalled.

This macro submits a request for the system semaphore signal service. This service causes a specified semaphore to be signalled.

4-4 R:SPRI prdesc

where: prdesc Priority descriptor.

This macro submits a request for the system activity priority definition service. This service alters the calling activity's priority according to the supplied priority descriptor.

7-7 R:STOD

This macro submits a request for the system time of day definition service. This service sets the time of day to the value specified in the AQ register pair.

6-7 R:WAIT sema4

where: sema4 Address of the Semaphore Descriptor Block to wait on.

This macro submits a request for the system semaphore wait service. This service causes the activity to wait on a specified semaphore.

Page

6-5 SDB:A label,value

where: label — Address label of the semaphore.

value — Initial value of the semaphore.

This macro generates a Semaphore Definition Block.

7-3 7-4 TICK:A arg,id,sema4,count

where: arg — Must match the argument of an R:ITIC, R:MTIC, or R:CTIC macro.

Identifier of a tick clock timer.

sema4 — Address of the semaphore to be signalled (R:ITIC or R:MTIC), or dummy argument with any defined value (R:CTIC).

count — Number of ticks that must elapse before the semaphore is signalled (R:ITIC or R:MTIC), or dummy argument with any defined value (R:CTIC).

This macro generates an argument list for an R:ITIC, R:MTIC or R:CTIC macro.

3-7 TDB:A label,start,yscratch,stackad,stackamt,flags,usage

where: label — Label to be assigned to start of TDB.

start — Starting address of task.

yscratch — Amount of Y-scratchpad to be used by the task. If zero, the Y register of the calling task is used. Usually is zero (or omitted) for a serial task.

stackad — Address of preallocated stack. If zero, stack space is allocated by RTX4. Must be zero (or omitted) for a reentrant task.

stackamt — Amount of stack space used by the task.

flags — None currently defined. (Optional.)

usage — Maximum number of concurrent activities of this task. (Optional.)

This macro generates a Task Descriptor Block.

Page

7-8
7-9 WALL:A arg,id,sema4,upper,lower

where:

arg	Must match the argument specified in an R:AWAL, R:IWAL, R:CWAL macro.
id	Identifier of a wall clock timer.
sema4	Address of the semaphore to be signalled (R:AWAL or R:IWAL), or dummy argument with any defined value (R:CWAL).
upper	Upper word of the 32-bit integer specifying the number of wall clock intervals that must elapse before the semaphore is signalled (R:AWAL or R:IWAL), or dummy argument with any defined value (R:CWAL).
lower	Lower word of the 32-bit integer specifying the number of wall clock intervals that must elapse before the semaphore is signalled (R:AWAL or R:IWAL), or dummy argument with any defined value (R:CWAL).

This macro generates an argument list for an R:AWAL, R:IWAL or R:CWAL request.

APPENDIX H

MACRO FILE CONTENTS

The RTX4 user accesses macro definitions, table definitions, or other code via certain macro files. These macro files and their contents are listed below.

GEN.MAC (General Macro File)

- Macro definitions:
  - EXCH:M
  - COPY:M
- Fixed memory address assignments
- Non-printing ASCII characters
- S register bit definitions
- SYMATT directive bit definitions
- Hardware stack definitions
- Distributed I/O device and interrupt addresses

RTX.MAC (RTX4 User Macro File)

- Macro definitions
  - INIT:A
  - SINGL:A
  - TDB:A
  - ECB:A
  - EDXVT:A
  - SDB:A
  - MDB:A
  - BGIN:A
  - TICK:A
  - MAIL:A
  - WALL:A

RTXD.MAC (RTX4 Development Macro File)

- Macro definitions
  - PUSH:
  - POP:
  - COPY:
  - EXCH:
  - ASTAR:
  - RSTAK:
  - CHK:
  - SYS:A
- Table definitions
  - TDB - Task Descriptor Block
  - ACB - Activity Control Block
  - ECB - Environment Control Block
  - EDB - Environment Descriptor Block
  - CCB - Clock Control Block
  - SDB - Semaphore Descriptor Block
  - INIT - RTX Initialization Block
- RTX exception codes
- RTX block check values
- RTX base page definition
- Environment memory pool definition
- Miscellaneous RTX system equates

IOS.MAC (IOS4 User's Macro File)

- Macro definitions
  - BUF:R
  - IOB:A
  - UAT:AA
  - UAT:EE
  - UAT:ZZ
- Table definitions
  - IOB - Input/Output Block
- Operation, position, and function codes
- Error codes
- Status codes

IOSD.MAC (IOS4 Development Macro File)

- Macro definitions
  - PATCH:
  - I:EOB
  - CIB:DM
  - CIB:DH
  - DIB:DM
  - DIB:DH
  - DIB:LP
  - DIB:ST
- Table definitions
  - CIB - Controller Information Block
  - DIB - Device Information Block
  - TIB - Temporary Information Block
- Distributed I/O equates
- I/O error block definitions

SFM.MAC (SFM User's Macro File)

- Macro definitions
  - CONN:A
  - DELE:A
  - CREA:A
  - MONT:A
  - FCB:SA
- Table definitions
  - FCB - File Control Block
  - FDB - File Descriptor Block
- Parameter list equates

SFMD.MAC (SFM Development Macro File)

- Table definitions
  - VCB - Volume Control Block
- F list entry definition
- Buffer control and flag word definitions

NOTES ON ITEMS ISSUED WITH RTX4 (C1)

1. RTX User's Manual (C0)

Appendix H describes the macro files (supplied with OS4 and RTX4 and their contents. The contents described for GEN.MAC should include all RTX4/IOS4/SFM service call macros.

2. IOS4 User's Manual (C0)

2.1 Similar comment as given in 1, except that it is Appendix G. Also page 8.1 refers to Appendix I instead of G and the Contents List has omitted the Appendix altogether.

2.2 Appendix B

The Introduction B.1 should include reference to the Volume Control Block and FLIST described later in the Appendix.

3. IOS4 (C1)

3.1 The IOS.HLP

This file includes description of the IOSDEMO program files. This demo is now called SFMDEMO.

3.2 The Line Printer DIB (Standard)

This is configured for 80 characters per line and 57 lines per page. The DIB:LP macro also defaults to these values and not 133 and 39 as described.

3.3 IOSD.MAC

Note that this file equates the CRT DIO channel address to 2 instead of 4 as one might expect.

NOTES ON ITEMS ISSUED WITH RTX4 (C1) (Cont.)

3.4 Write Direct Stream I/O

There is a fault connected with this. If a program attemps to do Write Direct Stream to a file in order to overwrite the exact number of bytes remaining in the file, SFM ignores the request and indicates an end of a block error (:4E). This fault may be overcome by patching as follows:

Location	Old Contents	New Contents
F:CEOF+:A	:9E82	:0000

The address of F:CEOF may be determined by examining the link-map produced by linking the user program with RTX/IOS/SFM.

3.5 TV/TK/TY End-of-Input Action

Currently, when carriage-return is required to terminate an input I/O request, IOS4 responds by repeating just that character, which means that it is possible for subsequent output to overprint the previously typed line. (In the case of OS4 message output, no overprinting occurs because a line-feed is output first, before the message.)

To ensure that no overprinting occurs, users may modify the location identified on link maps by the symbol TYELI:. Normally this location contains 1, but 2 should be put in its place to ensure that carriage-return is followed by a line-feed after every input line is terminated.

4. RTX (C1)

4.1 The fault described in connection with the previous version of RTX4 namely R:IWAL still exists and the same patch applies. For the benefit of those users new to RTX4, a copy of the EN issued just before this C1 release is attached to these notes.

/Cont.

NOTES ON ITEMS ISSUED WITH RTX4 (C1) (Cont.)

4.2 R:PAUS

This service should allow an activity to de-schedule itself so that it is placed at the end of the queued activities of the same prioity as itself. However, R:PAUS de-schedules the next activity in the queue. The following patch cures the fault:

Location	Old Contents	New Contents
R:PAUS+:8	:A022	:2922

4.3 MAILBOX

MDB:A macro is wrong. It allocates word containing 0 for Mailbox Usage Semaphore and it should contain 1.

Change source line 319 from "Word 0 - Mailbox Usage Semaphore" to "Word 1 - Mailbox Usage Semaphore".

4.4 RTX MACROS

TICK:A, WALL:A, MAIL:A,SDB:A,MDB:A Macros contain invalid constructions for testing number of parameters supplied with the call, e.g. 0<#?<3.

There are no simple changes that can be made and users are advised to ensure that they provide the correct number of parameters since the macro definitions do not check correctly.

ComputerAutomation
18651 Von Karman, Irvine, Calif.

ENGINEERING NOTICE

NO 71691S

DOCUMENT NO.	REV. IS	REV. WAS	TITLE	INCORP. DATE
093410-XX	B2	B1	RTX4 - R:IWAL	

TYPE

AEN	☐
STOP ORDER	☐
DEVIATION	☐
RELEASE	☐
STANDARD	☒

CLASS

A-MAND/FUNC	☒
B-NON-MAND/FUNC	☐
C-RECORD CHG	☐

AFFECTED ITEMS

	PRIM.	SEC.
HARDWARE CHG.	☐	☐
SOFTWARE CHG.	☒	☐
PUBL. CHG.	☐	☐
CAPABLE CHG.	☐	☐
DOC. CHG.	☐	☐
CONFIGURATIONS	☐	☐
PROCEDURES	☐	☐
TOOLING	☐	☐
TEST EQUIP.	☐	☐

EFFECTIVITY

ACTIVITY	U	P	S	P
NOTIFY VEND				
IN STOCK				
KITTING				
ASSEMBLY				
TOUCH UP				
IPT				
FIN GOODS				
CUST. RET.				

REWK TEST REQ'D

CONTINUITY	☐
CABLE SCAN	☐
CAPABLE	☐
MEMORY	☐
CARD	☐
FINAL	☐
NO TEST REQ'D	☒

APPROVALS

ENGR.	
SOFTWARE	
Q.A.	
CAP. TEST	
MAST SCHED	
MATERIALS	
TEST ENGR.	
TECH SERV	
CUST SERV	
MFG. ENGR	
PUBLICATIONS	
DR. BY: A. DUZICH	
CHKD. BY: D. EDBECK	
REL. BY: A Iannicco	
DATE: 1-4-79	

SHEET 1 OF 1

EFFECTIVITY NOTES:

REASON FOR CHANGE:

REA NO. 04447

CO-ORD WITH:

CERTAIN COMBINATIONS OF R:TODL AND USER-SPECIFIED INTERVAL VALUES PRODUCE INCORRECT SHORT TIME INTERVALS BECAUSE THE CODE ASSUMES THAT THE ADD INSTRUCTION AFFECTS THE CARRY STATUS BIT, WHICH IT DOESN'T!

DESCRIPTION OF CHANGE:

A. PATCH AS FOLLOWS:

LOCATION	OLD CONTENTS	NEW CONTENTS
R:IWAL+:22	:C844	:0E07 RBIT 0,S
+:23	:C483	:4712) ADDC TL(Y),Q
+:24	:8843	:0004)
+:25	:56C1	:C483 COPY Q,CC:TL(X)
+:26	:0801	:0712) ADDC_TU(Y),A
+:27	:8482	:0003)
+:28	:9E80	:8482 COPY A,CC:TU(X)

(NOTE: JMP POST AT R:IWAL +:28 IS REDUNDANT BECAUSE POST IS THE NEXT LOCATION.)