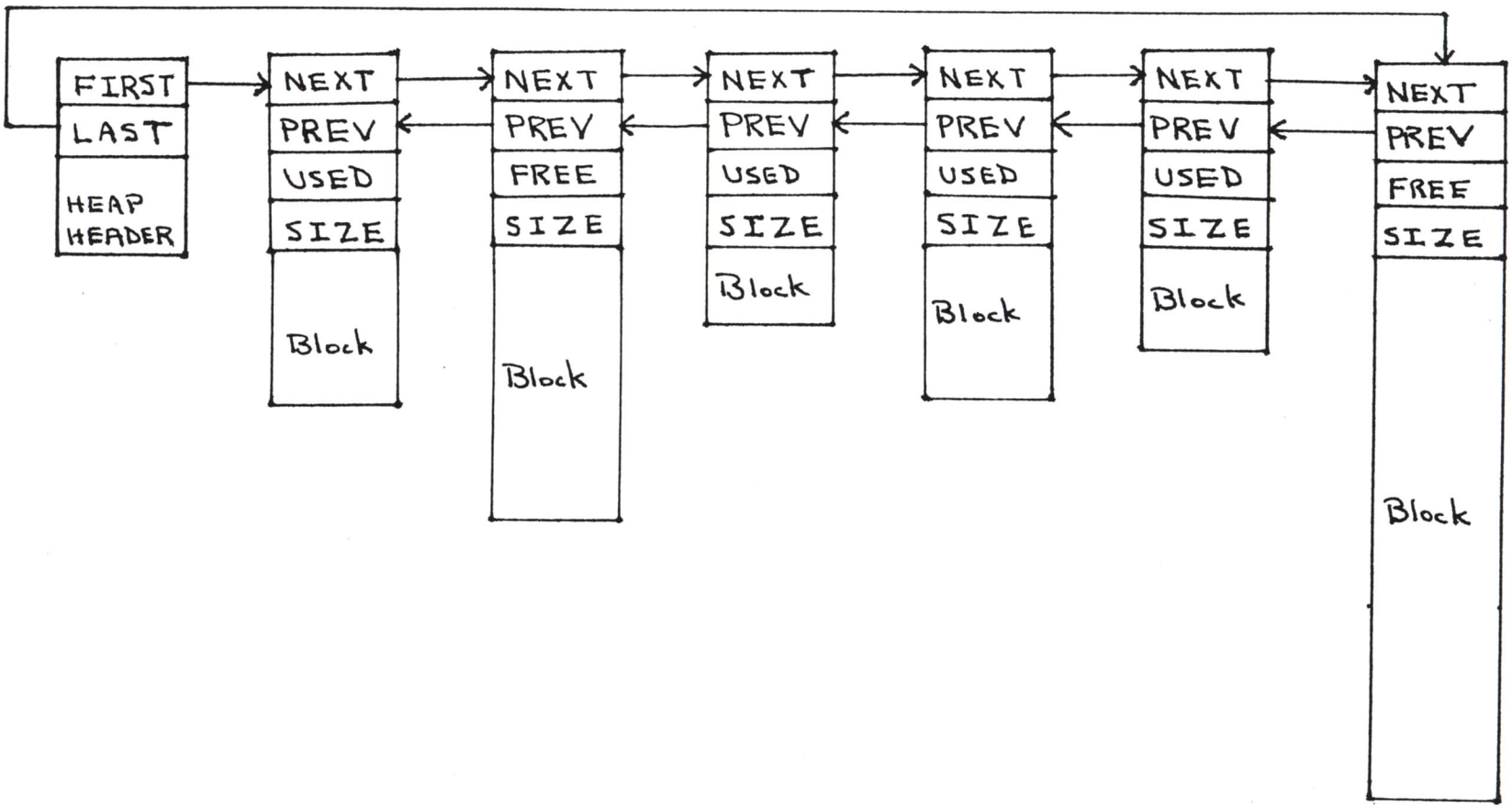
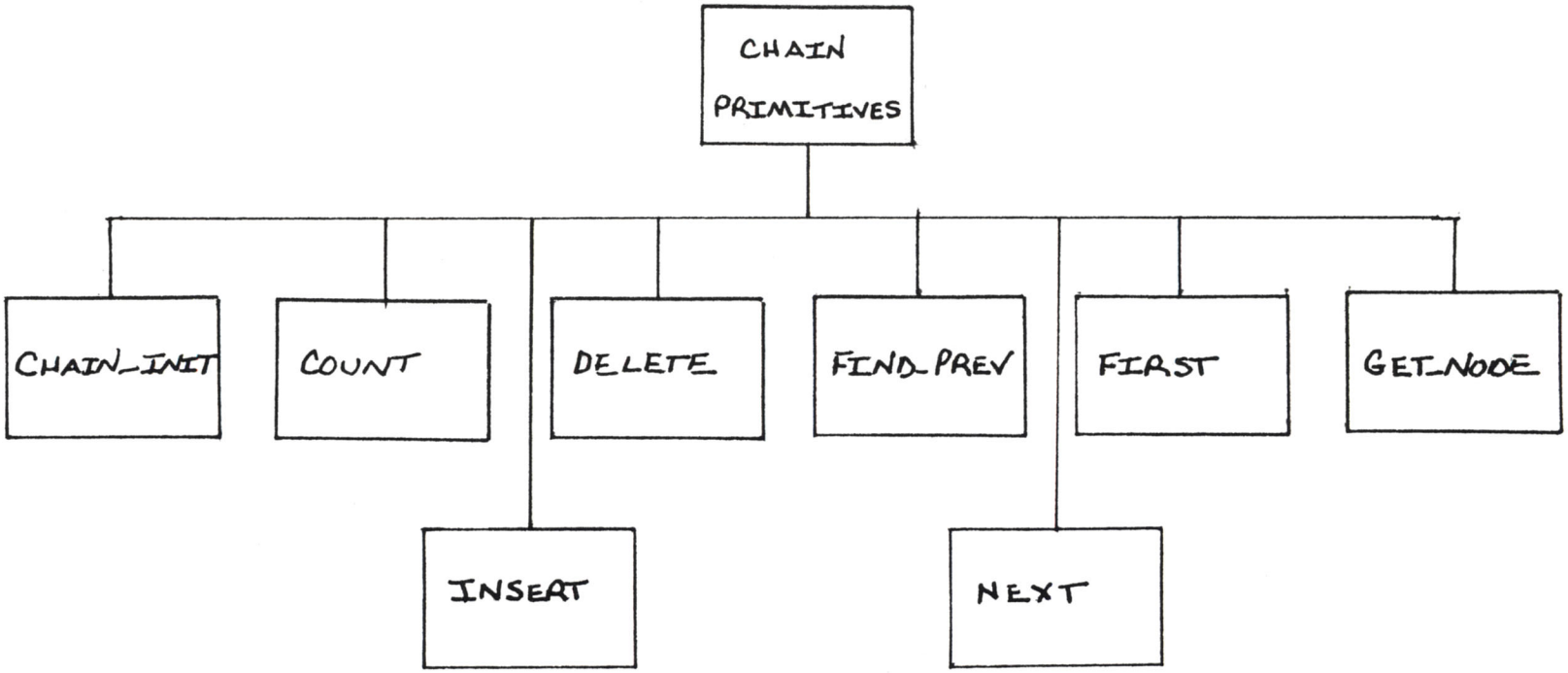
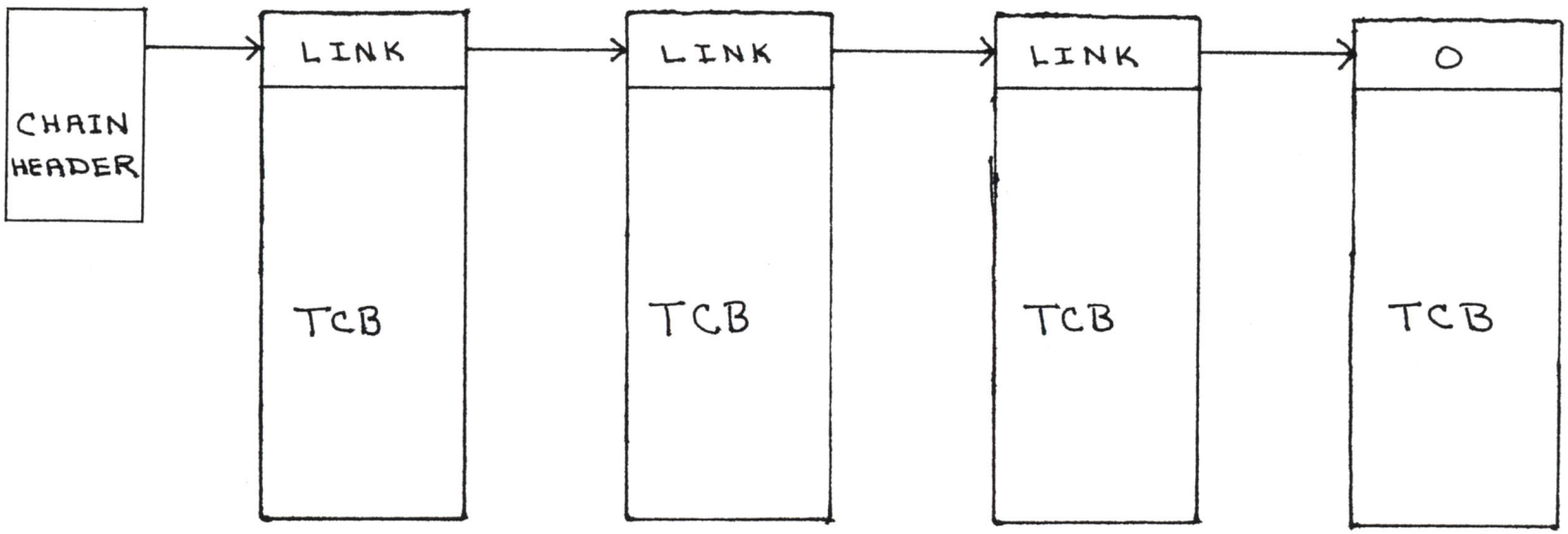


OAR

EXECUTIVE HEAP

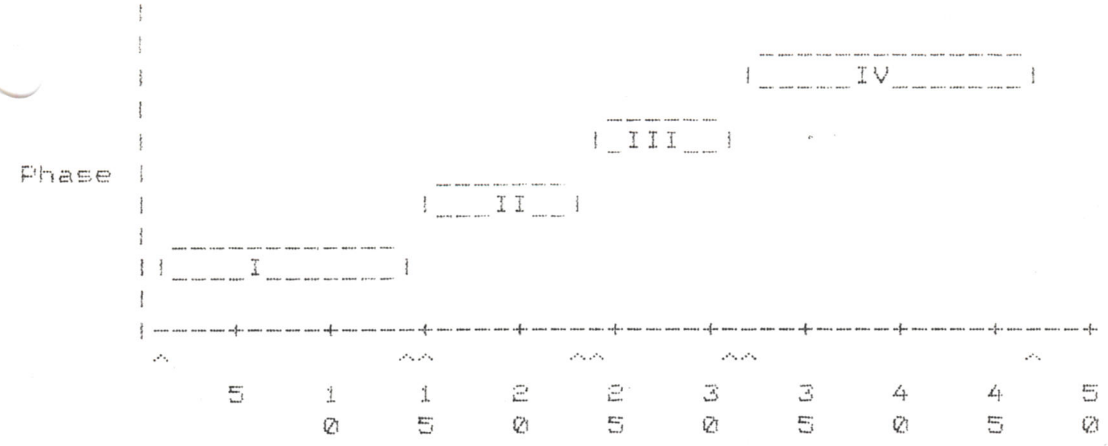

CHAIN
PRIMITIVES
CHAIN_INIT
COUNT
DELETE
FIND_PREV
FIRST
GET_NODE
INSERT
NEXT



OAR

TCB CHAIN

RTEID Project Schedule - 68020 Target

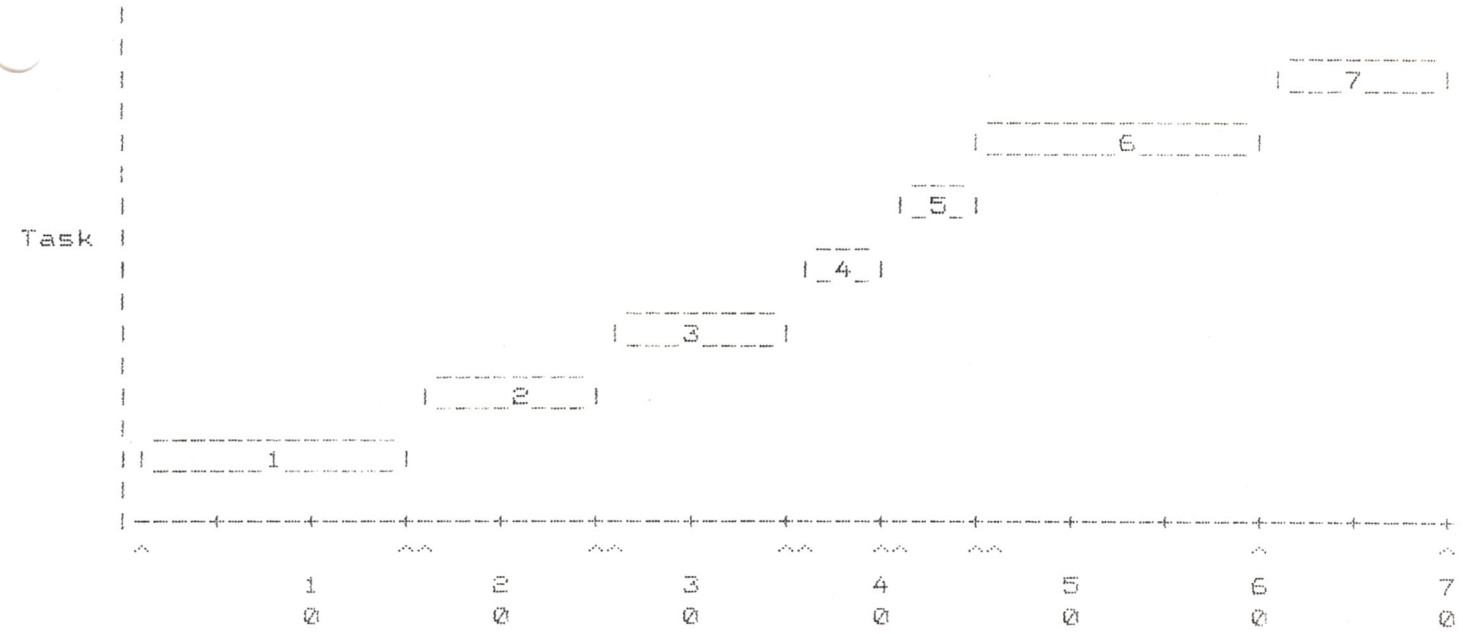


Calender Weeks

PHASE DESCRIPTIONS

Phase I			14 weeks
TASK 1	Boot software (EPROM)	3 weeks	
TASK 2	Board Support Package	2 weeks	
TASK 3	Kernel	2 weeks	
TASK 4	Interrupt Handler	1 weeks	
TASK 5	Fatal Error Handler	1 weeks	
TASK 6	Task Manager	3 weeks	
TASK 7	Time Manager	2 weeks	
Phase II			9 weeks
TASK 1	Message Manager	2 weeks	
TASK 2	Event Manager	1 weeks	
TASK 3	Signal Manager	1 weeks	
TASK 4	Semaphore Manager	2 weeks	
TASK 5	Memory Manager	3 weeks	
Phase III			8 weeks
TASK 1	I/O Device Manager	4 weeks	
TASK 2	MMU Manager	4 weeks	
Phase IV			16 weeks
TASK 1	File System Manager	16 weeks	
			47 weeks

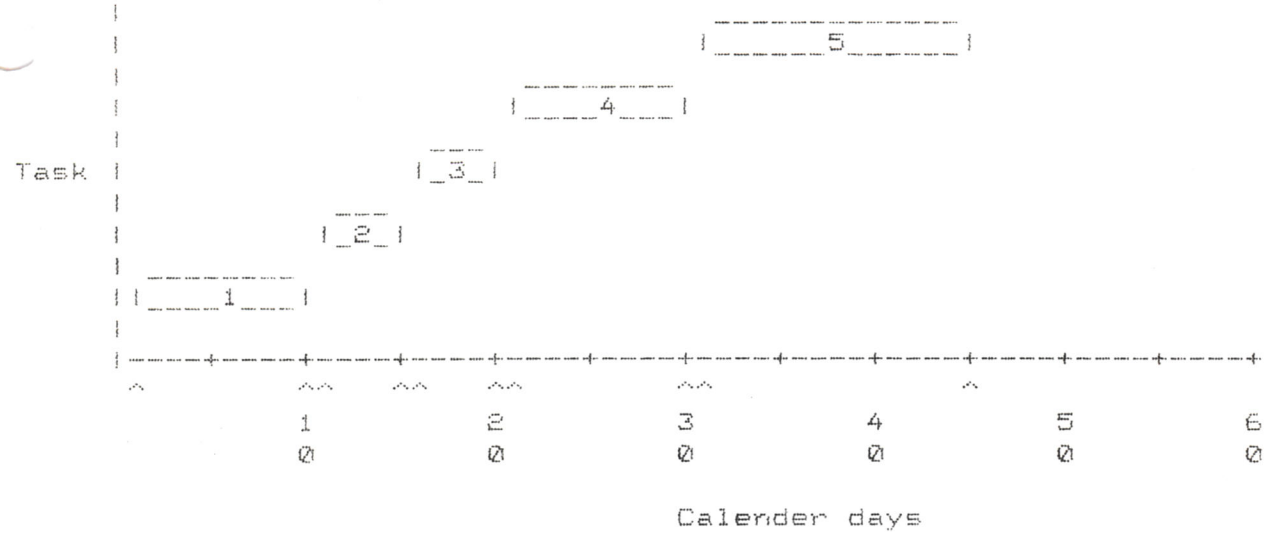
RTEID Phase I Schedule - 68020 Target



TASK DESCRIPTIONS

TASK 1	Boot software (EPROM)	3 weeks
TASK 2	Board Support Package	2 weeks
TASK 3	Kernel	2 weeks
TASK 4	Interrupt Handler	1 weeks
TASK 5	Fatal Error Handler	1 weeks
TASK 6	Task Manager	3 weeks
TASK 7	Time Manager	2 weeks
Phase I total		14 weeks

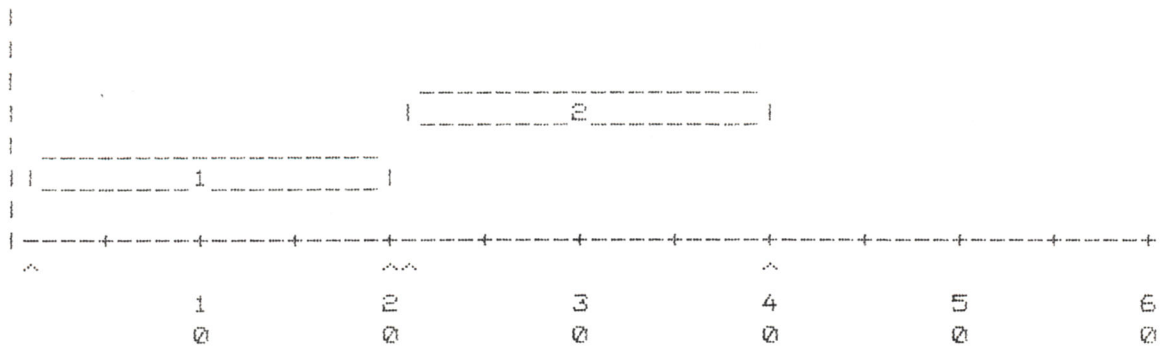
RTEID Phase II Schedule – 68020 Target



TASK DESCRIPTIONS

TASK 1	Message Manager	2 weeks
TASK 2	Event Manager	1 weeks
TASK 3	Signal Manager	1 weeks
TASK 4	Semaphore Manager	2 weeks
TASK 5	Memory Manager	3 weeks
	Phase II total	9 weeks

RTEID Phase III Schedule - 68020 Target

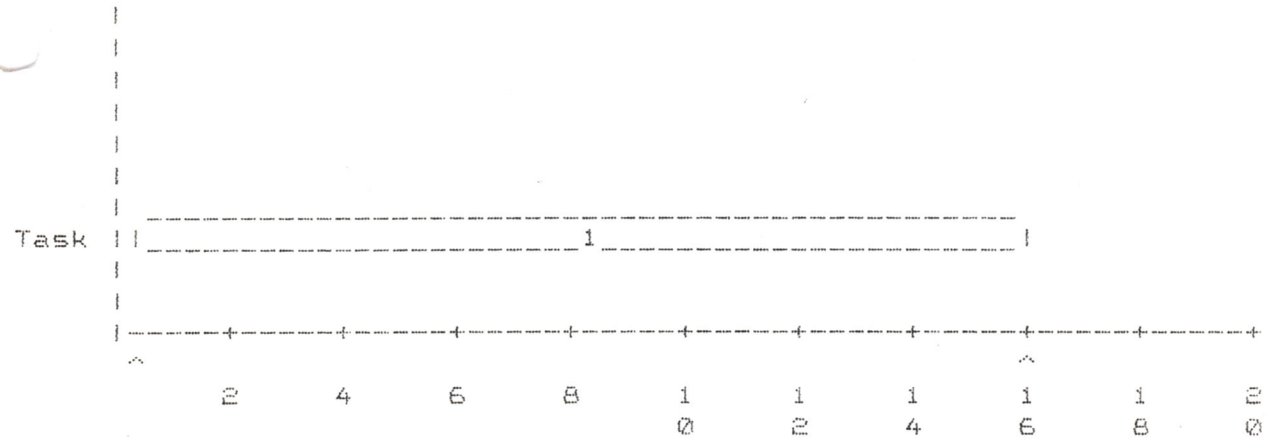


Calender days

TASK DESCRIPTIONS

TASK 1	I/O Device Manager	4 weeks
TASK 2	MMU Manager	4 weeks
Phase III		8 weeks

RTEID Phase IV Schedule - 68020 Target



TASK DESCRIPTIONS

TASK 1	File System Manager	16 weeks
Phase IV		16 weeks

OAR Sept 26, 1988

The questions below are releated to the RTEID Specification Draft 21.
s prepared by Motorola Microcomputer Division and Software Components Group. If a more recent version of this document has been released, we need the information necessary to acquire an updated copy.

General Questions

1. How do SYSTEM and USER sets work ?
   eg. registers, events, and signals
   Who can access the SYSTEM set and who can access the USER set ?

2. What scheduling algorithm(s) is the RTEID executive supposed to use? If all implementations of RTEID do not use the same scheduling algorithm, then the run-time behavior for different implementations will be different.

3. Added t_create, t_delete, t_switch, and TCB extension block capability.

Message Manager Questions

1. Why is message.h specified to be included in the message primitives and none of the other managers specify a .h file. Is there a message.h file defined that we have to follow. We have defined a complete set of .h files to be included as needed for our executive.

2. q_create - If LIMIT was not set and RESVD was set, then we cleared the RESVD flag and continued processing without an error indication being generated. Is this a correct assupmtion?

3. q_create - What is the TYPE flag value?

4. q_send, q_urgent, q_broadcast - The phrase "the task's message buffer may be reused immediately" is used in the specification of these primitives. Is this just a comment to require the implementor to copy the buffers instead of storing addresses?

5. q_receive - Specification requires the executive to return a -1 and the NO MESSAGE AT QUEUE ERROR number if an error is detected. The way the primitive is defined it cannot return 2 things simultaneously.

Signal Manager Questions

1. 3 ways to enable/disable asynchronous routines:
   a. t_mode - NOASR flag
   b. as_catch - DISASR flag
   c. as_catch - asraddr = 0
   Why three ways? What is the difference between each way.

2. Can a task send signals to itself? If yes then does ASR run immediately or the next time the task runs?

3. Why not have seperate ASR for each signal capability? I.E. Have each task have a 32 slot ASR vector table so it can define a seperate ASR routine for each type of signal.

4. How does the ASR know which signals were sent to it? If ASR uses the signal stack frame to get this information, then the specification should define the exact format of the "signal stack frame" .

5. Can an ASR get blocked and swapped out for another task? If yes, then then hat if some signals are sent to the task while he is swapped out, when the ASR is swapped in again will he recognize the new signals or will he have to re-execute the ASR to pick up the new signals?

Event Manager Questions

1. ev_send - Can a task send an event to itself ?
   Does tid = 0 apply here ?

2. ev_receive - We assumed that calls to ev_receive do not build upon one another.

3. ev_receive - We assumed that eventout is set according to eventin. only the events that were sent that match up with eventin are reported in eventout. For example, an event that was not specified in eventin could have be a pending event, but it would not be reported in eventout.

4. ev_receive - "returns immediately with -1 and the no event available error number" ?

5. ev_receive - Why can't it be called from an ISR and the NOWAIT option be enforced like the q_receive and sm_p directives ?

Semaphore Manager Questions

1. sm_v - We added an error message for no sempahore acquired. This occurs when sm_v is called but no semaphores have been obtained with a previous call to the sm_p directive.

2. If a task is deleted that owns some semaphores, should these semaphores be released or are they lost forever ?

Fatal Error Handler Questions

1. We generated a very simple fatal error handler in our board support package. In our configuration table we allow the user to configure his own fatal error handler or default to our simple one.

Task Manager Questions

1. t_create - CMASK?

2. t_init, t_go?

3. t_start - NOASR flag versus as_catch's DIS flag.

4. For directives where tid = 0 means requesting task, do we treat the actual requesting task's tid in the same manner?

5. t_setpri - Should a task be preempted if it sets its priority to the same as its current priority and there are no other ready tasks with a higher priority?
   t_restart - restarting self has this same problem.

6. t_setpri - A minimum of 32 priorities is specified, but how is range, total number of supported priorities and whether a priority of 2 is of higher or lower priority than 3 specified?

7. t_mode, t_start - LEVEL - Is this a 3 bit field?

8. t_mode - Changing from SUPV to USER or USER to SUPV causes stack problems involving the addresses of local variables being on the stack.

9. t_create, t_start primitives

   Can an event be sent to task that has been created but not started (dormant)? If it can, will the event that was sent be recognized by the task when it is started (made ready)?

Time Manager Questions

1. tm_evafter, tm_evwhen - Why can't these be directed to tasks other than the calling task?

2. Why are there no tm_asafter and tm_aswhen directives for signals?

3. tm_evafter - Says it always succeeds, but it has an error message?

4. Number of timers must be a configuration parameter since there is a "too many timers" error message in tm_evafter and tm_evwhen.

5. tm_cancel - When the timer event not set error occurs while trying to cancel a timer that has expired, we don't see anything the caller needs to do to clear the event condition.

6. tm_cancel - We assume that timer associated with the tmid is owned by the requesting task.

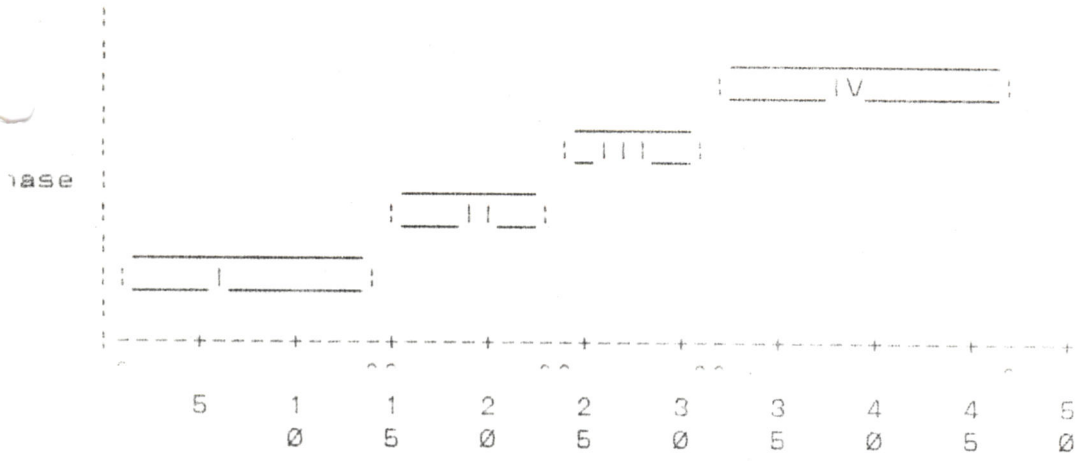
Why RTEID

- The Problem
  - Real-time applications require a basic core of capabilities
    - acquiring memory segments
    - creating and managing tasks
    - creating and managing message queues
    - manipulating semaphores and events
    - sending and receiving messages between tasks
  - Real-time applications need a real-time executive
    - real-time executive provides these services to applications programs
  - Real-time executive development is relatively new industry with most popular executives introduced in last 6 years
  - Current industry real-time executives present different user interfaces
    - different scheduling algorithms
    - different names for basic services
    - different calling sequences for basic services
    - different processor support
    - different I/O device interfacing mechanisms
    - different high level language support
    - conflicts in multi-processing support
    - different file system support
    - different operational behavior
  - Have open systems architecture but closed system software
    - porting to another real-time executive requires extensive application software rewrites

Why RTEID

- The Solution: Define a real-time executive interface standard
  - Limited de facto standardization already exists
  - Facilitate writing of real-time applications that are directly portable across multiple real-time executives
  - Allows flexibility by specifying comprehensive set of real-time capabilities
  - Guarantee identical operational behavior of an application across multiple real-time executives
  - Allows programmers to concentrate on hardware dependencies of a real-time system
  - Reduced programmer learning curves and improved productivity
  - Allows software for using the basic real-time services to be developed and tested once and re-used many times
  - Industry moving toward operating system standardization such as MOSI, POSIX, and SVID
  - Software industry is moving toward a standard software project development environment (CASE) so it seems logical to extend it to operating systems
  - Provides insurance against software obsolescence
  - Software can be developed then hosted on the most efficient or cost effective hardware
  - Executive design easily lends itself to both tightly-coupled and loosely-coupled multiprocessor configurations
  - Standard UNIX development tools and languages can be used to generate real-time applications which are later hosted on target hardware

RTEID Project Schedule - MVME136 (68020) Target



PHASE DESCRIPTIONS

Phase I			14 weeks
TASK 1	Boot software (EPROM)	3 weeks	
TASK 2	Board Support Package	2 weeks	
TASK 3	Kernel	2 weeks	
TASK 4	Interrupt Handler	1 weeks	
TASK 5	Fatal Error Handler	1 weeks	
TASK 6	Task Manager	3 weeks	
TASK 7	Time Manager	2 weeks	
Phase II			9 weeks
TASK 1	Message Manager	2 weeks	
TASK 2	Event Manager	1 weeks	
TASK 3	Signal Manager	1 weeks	
TASK 4	Semaphore Manager	2 weeks	
TASK 5	Memory Manager	3 weeks	
Phase III			8 weeks
TASK 1	I/O Device Manager	4 weeks	
TASK 2	MMU Manager	4 weeks	
Phase IV			16 weeks
TASK 1	File System Manager	16 weeks	
			47 weeks

STATUS: 9/15/88
used 64% of resources allocated for phases I and II
completed 82% of work required for phases I and II

RTEID Project Extensions

- Phase V
  - debug manager primitives
- Phase VI
  - homogeneous multiprocessing
  - heterogeneous multiprocessing
- Phase VII
  - homogeneous distributed processing
  - hetrogeneous distributed processing
- Phase VIII
  - UNIX System V Interface Library
- Phase IX
  - convert executive to ROMABLE relocatable executive
- Phase X
  - multiple language interface capability
    - assembly language
    - C
    - PASCAL
    - ADA