Multitasking for Sensor Based Systems

by

Srinivas T. Reddy

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Electrical Engineering

APPROVED:

Dr. Charles E. Nunnally,
Chairman

Dr. Joseph G. Tront

Dr. Morton Nadler

July, 1985

Blacksburg, Virginia

Multitasking for Sensor Based Systems

by

Srinivas T. Reddy

Dr. Charles E. Nunnally, Chairman

Electrical Engineering

(ABSTRACT)

Multitasking systems are being used increasingly for real-time applications. Multitasking is suited very well for real-time systems since events in the real world do not occur in strict sequence but rather tend to overlap. Multitasking operating systems coordinate the activities of the different overlapping functions and give the user the appearance of concurrent activity. The coordination and scheduling is performed according to a user defined order of importance or priority. There are many multitasking operating systems available for all the popular microprocessors. One such multitasking executive is VRTX/86 for the 8086 microprocessor. This executive comes in a PROM and is independent of any specific hardware configuration. Using this executive the IBM PC has been converted into a multitasking environment and multitasking test programs have been executed on the PC.

A general methodology for defining tasks and assigning priorities to these tasks has been defined. Using this methodology a typical real-time application called a Vehicle Instrumentation System was developed.

ACKNOWLEDGEMENTS

I would like to thank Dr. Nunnally for his constant support and encouragement in doing this interesting project. He was instrumental in helping obtain the software from Hunter and Ready and providing help whenever I needed it. I would also like to thank Dr. Tront and Dr. Nadler for being on the committee I would like to thank them for reviewing the thesis and giving useful suggestions.

It is also a pleasure to acknowledge the support of Hunter and Ready in providing the software free of cost and the constant help of their technical support group.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

1.0 INTRODUCTION

A new class of operating systems called multitasking operating systems or real-time executives are available from almost all companies that manufacture 16 and 32 microprocessors. These systems are applicable in any system that uses the particular microprocessor for which it has been designed. Intel has introduced additions to its microprocessor line, with Operating System Firmware components like 80130 that provide hardware support for functions previously performed by software [1]. With increasing sophistication in microprocessor systems the cost of developing system software for system operations such as timing, I/O, interrupt services and data management has also increased [2]. It is no longer economically feasible to have a custom designed operating system for every application.

The operating systems available provide general calls and techniques using which the user can solve any specific problem. These operating systems are independent of any particular board environment and the only hardware assumption made is the presence of a microprocessor for which the operating system is designed.

Real-time systems have to respond to events within the limits imposed by the environment in which they work. Multitasking operating systems support the design of real-time

systems, since any real-time situation requires a number of computations to occur not necessarily in strict sequence. By splitting the system into tasks, the scheduling and switching of the CPU among the different tasks can be performed by the multitasking operating system. The design and development of real-time systems is generally complex due to the non-deterministic behavior of the external environment with which the system has to interface [3]. But the design of these systems has been simplified considerably by the availability of standard off-the-shelf operating systems like VRTX/86, etc. which can perform in any board environment.

Sensor based systems are classified as real-time systems, and they require immediate processing of the data they acquire. The software designed for a sensor based application has to meet the same set of requirements as any other real-time system. If multitasking is used a task has to be defined at the lowest level which involves the output from the sensor. We will look at a typical example that uses sensors and interfaces with the tasks at the sensor level in a later chapter.

This thesis deals with VRTX/86, a multitasking operating system from Hunter and Ready, and a general methodology to design multitasking systems which is backed up with a working example on the IBM PC and implementation techniques for a sensor based application. The VRTX operating system is supplied by the vendor in PROMs and these PROMS are plugged into

the IBM PC and the PC is converted into a multitasking environment, under which the user can develop multitasking programs which can then be transferred to any other board level computer.

This thesis is organized as follows.

- Chapter 2 deals with the discussion of multitasking operating systems and some rules involved in defining tasks for multitasking systems.
- Chapter 3 has an overview of VRTX/86, the multitasking executive from Hunter and Ready.
- Chapter 4 discusses the operation of the multitasking operating system on the IBM PC, with an example.
- Chapter 5 talks about the development of a typical example of multitasking called the 'Vehicle Instrumentation System' using the rules defined in chapter 2. The pseudo code for the multitasking software for this particular application is also discussed and developed.

2.0 MULTITASKING OPERATING SYSTEMS

An operating system of a computer is a large program that controls the execution of other programs and sharing of computer resources among the various programs. It is a program that is developed and used by a changing group of people [4] and is a complicated application of general programming techniques. An operating system enforces a set of rules for behavior on any user written program.

There are different classifications of operating systems like

1. Stand alone systems
2. Disk operating systems
3. Real-time systems
4. Time-sharing systems

Of these classifications real-time systems are designed so that they can respond to real-time events. Real-time operating systems support multitasking. Recent versions of time-sharing operating systems such as UNIX support multitasking.

2.1 PRINCIPLES OF OPERATING SYSTEMS

Operating systems perform several functions like initiating external activities, coordinating internal activities, responding to external events and allocating resources. This does not mean that all these functions have to be carried out by a single monolithic program. Instead a set of concurrent programs are used to perform these functions simultaneously. The term concurrent programs is normally used when defining an operating system with multiple number of processors but any single processor operating system has concurrency built into it.

The basis for an operating system is a sequential process. A sequential process defines a computation which follows a certain order of precedence so that the results of an operation can be used by others. A more formal definition of a process by Bowen [5] is as follows. In any program that has concurrency processing resources have to be shared. A sequential program then has to be augmented by a named set of state information which allows the program to be restarted at any point. This generalization of a sequential program is called a process. A process can be thought of as an abstraction at a level above program code.

A sequential process and a set of concurrent processes can be represented by means of graph shown in Figure 1 on page 6. In the sequential process the computations proceed one

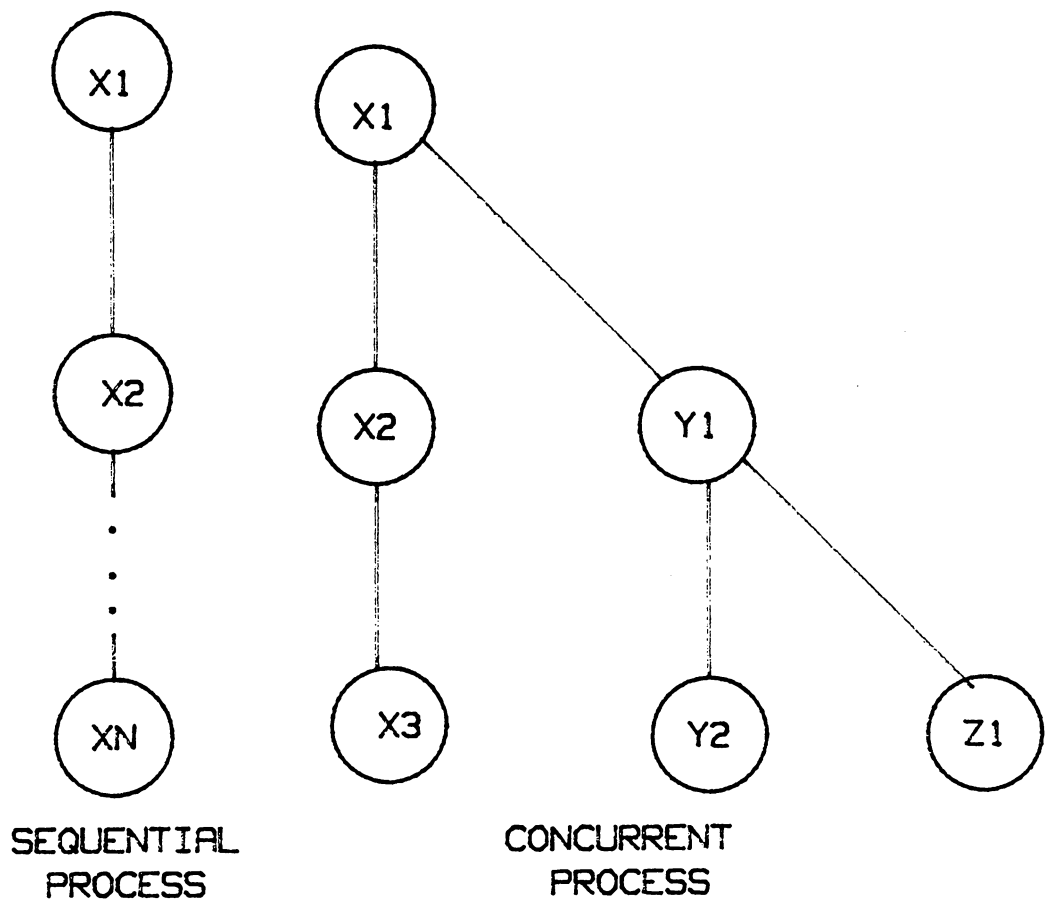


Figure 1. Flow graph of Sequential and Concurrent Processes [3].

after another. To compute x_j, x_{j-1} has to be computed. A set of concurrent processes on the other hand can proceed simultaneously. From the graph x_2 and y_1 can proceed concurrently, x_3, y_2 and z_1 can proceed concurrently. But x_2 and y_1 can proceed only if x_1 is complete. It is thus possible to have the operations x_2 and y_1 overlap in time one could be using an I/O device while the other uses the CPU.

2.2 RESOURCE SHARING

The number of resources available on any computer system is invariably less than the number of processes requiring access to them. So when more than one process tries to access the resource they share simultaneously, a set of rules have to be followed. If there are two processes X and Y sharing a common resource A, both processes should not try to access A at the same time. This mutual exclusion of the resource is achieved by making a process go through a cycle where it first reserves the resource, uses it and then releases it. Before the resource can be reserved the process that requires it waits. When both processes try to acquire the resource at the same time the decision to grant it to either of them should be made in a finite amount of time to prevent an infinite wait.

2.3 CRITICAL REGIONS

In the case of sharing data among processes there are areas defined as critical regions. A process accesses a variable and modifies it. If immediately after it has modified the variable but before it has completely used it, another process changes it back to its original value then the result of any computation involving the data will be incorrect. To prevent such failures operating systems use critical regions. Any common variable between processes can be changed only within critical regions. The concept of critical regions is due to Dijkstra[6]. Using Hansen's[4] terminology a critical region is denoted by

region v do S

This syntax associates a statement S with a common variable v. Critical regions that refer to the same variable v exclude one another in time. If a process is within a critical region then other processes trying to enter the same critical region will be delayed.

2.4 SEMAPHORES

Semaphores are used for mutual exclusion and signalling between processes. In the case of mutual exclusion the semaphores serve as a lock to prevent more than one process to access shared resources. A semaphore can also be used as

a signal between processes to indicate characters entering or leaving a buffer, a timeout indication etc.

The basic operations on a semaphore are wait and signal [6]. They are called P and V operations respectively. A semaphore consists of a counter and a queue of processes waiting for an event. When a number of processes try to enter a critical region the P and V operations can solve deadlocks and simultaneous entry into critical regions.

Since a semaphore is a common variable for the sending and receiving processes the wait and signal operations themselves should exclude each other in time (indicated by {}). They are critical regions with respect to the semaphore. The procedures for implementing a wait and signal operation are as shown below.

```
count     : int;
count     := 1;

proc (wait)
begin
     count = count - 1;
     { if count < 0
          then begin
               identify calling process;
               enter it into queue
               end
     endif}
```

```
end.

proc (signal)
begin
     count = count + 1;
     { if count <= 0
          then begin
               get next process from queue
               execute the process
               end
       endif}
end.
```

The semaphore count is set to 1 initially. When the critical region is busy this is set to 0. Any other process trying to enter this region performs a wait operation and adds itself to the queue. When a signal operation is performed the next process from the queue is removed to enter the critical region. Every time a wait is performed the variable count is decreased. The magnitude of the negative number gives the number of processes on the queue. The variable count is increased everytime a process performs a signal operation.

2.5 MULTITASKING OPERATING SYSTEMS

Multitasking operating systems follow the same principles as an operating system that can run concurrent processes.

Multitasking operating systems are intended for real-time applications and provide facilities for scheduling tasks, synchronizing tasks, passing messages between tasks and providing interrupt support for the system. A multitasking operating system can be provided for a system with multiple processors leading to a multiprocessor operating system. A large process control system involving real-time control is sure to involve multiple number of processors. In this situation multitasking with multiple processors is a better solution to the problem than just using one processor.

2.6 NEED FOR MULTITASKING

In any real world system events tend to overlap each other rather than occur in strict sequence [7]. In the field of real-time control it is necessary for several operations to be performed simultaneously or within a specified time limit. This sets a limit on the responsiveness of the system. If the system cannot deliver within a specified time limit then it has failed. Due to the serial nature of a microprocessor it cannot do more than one operation at a time. However it can be made to appear to do several operations simultaneously. This is done by the rapid switching of the CPU between various tasks. The philosophy behind this type of operation is the CPU is not always busy in a sequential program. For example when writing to an I/O device the CPU is idle for most

of the time. In a multitasking system the CPU would be executing another task. The job of the operating system is to make each program called a task appear to be running on its separate processor.

2.7 DEFINITION OF A TASK

A task is defined as an independent module of a program which in most cases can run independently. Any system has a number of needs to be serviced. The mechanism by which any one service is performed is assembled into a task.

2.8 COMPARISON OF MULTITASKING WITH OTHER SYSTEMS.

A multitasking system switches the CPU between several sequential tasks. The mechanism which causes the CPU to switch from one task to another is usually an event. The event could be external like an interrupt or a task which has been waiting for a message receiving it. The part of the operating system that manages scheduling of tasks, communication and memory management between tasks is called the kernel.

A multitasking system is different compared with a background-foreground system common in uniprocessor systems. This system always executes a background program and executes any foreground program only due to specific events but con-

trol always returns to the background program. In a multitasking system however there is no specific background task to which control always returns. The task to which control returns is controlled by the scheduler in the kernel. Control returns to the highest priority task that is ready to run.

A time-sharing system is a type of multitasking system but is not suitable for real-time applications [8]. A time-sharing system supports interactive users on terminals. The problem with time-sharing systems is the necessity to devote CPU time equally among the users or tasks. Real-time situations do not need equal allocation of CPU time but the only necessity is that a task should have access to the CPU when it needs it. A time-sharing system tries to isolate tasks from one another. However in a real-time system tasks interact to solve the different parts of a single big problem.

2.9 MULTIPROGRAMMING AND MULTITASKING

The principles of multiprogramming allow more than one independent program to execute on a computer system. Each program is completely independent of the other programs and operates as if it was the only program executing on the computer system. Multitasking is similar to multiprogramming except that the tasks present know about the existence of the other tasks and interact with them if necessary. The tasks in a multi-

tasking system are parts of the same problem, unlike multiprogramming where the programs are independent of one another.

2.10 MULTITASKING SYSTEM STRUCTURE

The general structure of a typical multitasking operating system is organized as layers and shown in Figure 2 on page 15. The kernel is the innermost layer of the operating system and is the layer which is the interface with the system hardware. The basic functions provided by the kernel are task switching, scheduling and synchronization. Kernels use queues to keep track of tasks which are ready to run, events that are to happen, timeouts etc. Linked lists are the most common data structures used within the kernel to maintain these queues [9]. The file system and I/O system can be modules which attach to the multitasking kernel to provide a complete system. The I/O and file system are normally designed to provide multitasking support.

The application program is the tasks that are provided by the user with the command processor providing the interface between the application and the system. The ISRs and device drivers provide the software support for the hardware components on the board. Device drivers provide support of display devices, printers, disks and other auxiliary support devices.

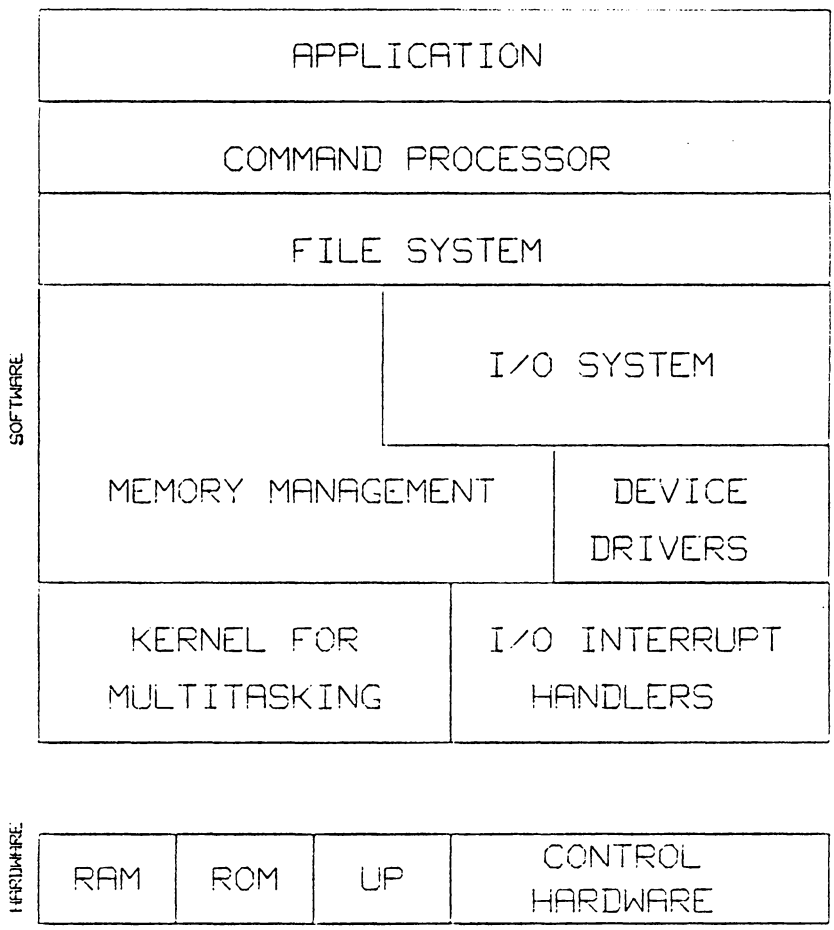


Figure 2. Organization of a Multitasking system [9].

2.11 TASK STATES

As mentioned before in a multitasking environment several tasks appear to be executing in parallel. At any given time however, the CPU is executing only one task. Every task in the system is assigned a priority which describes the relative importance of the task. The kernel always lets the highest priority task to execute. When the highest priority task requires a resource that is not immediately available, it is suspended and the next highest priority task starts executing. A task is in any one of the following states

1. Executing
2. Ready
3. Suspended
4. Dormant

The task state transition diagram is shown in Figure 3 on page 17. The executing state is the state of the task currently having control of the CPU. A task which is in the ready state is ready to execute but cannot until all higher priority tasks have finished executing. An executing task goes to the ready state when a higher priority task is ready to run and takes over the CPU. A task can be explicitly suspended by a system call, suspend waiting for an event to occur, or waiting for a specified interval of time to elapse.

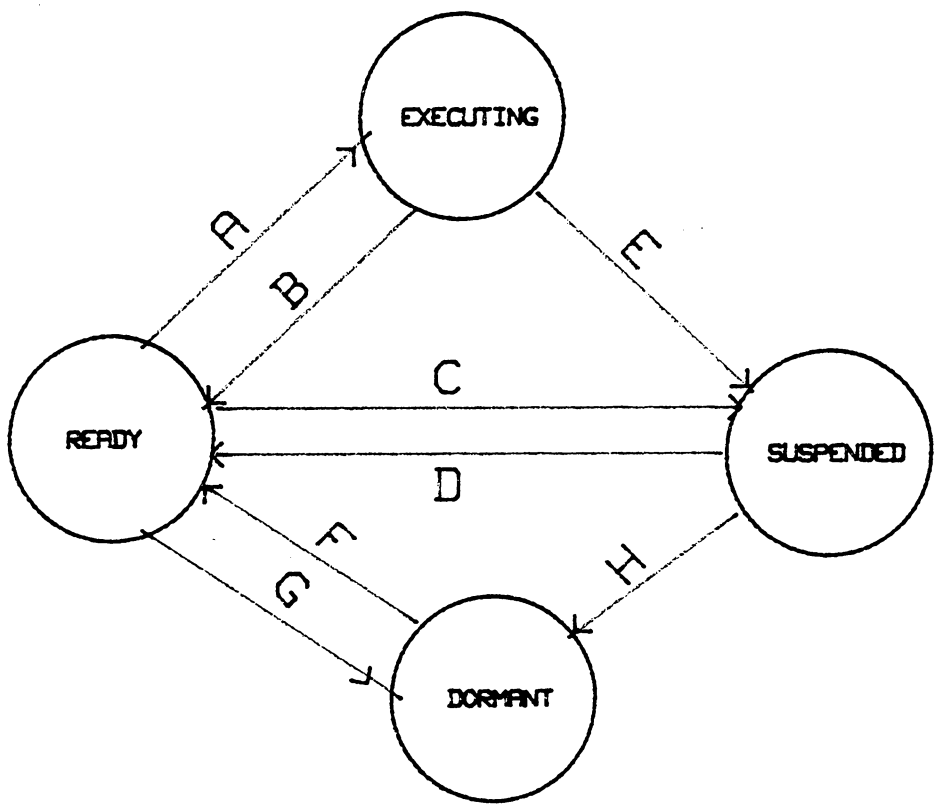


Figure 3. Task State diagram.

When a task is deleted it goes into the dormant state. A task can be deleted from any of the other states.

In an event driven operating system no special calls have to be made to perform a task switch. The operating system maintains the highest priority task in operation. The highest priority task executes until any of the following conditions occur

- the task terminates its own operation
- the task suspends
- a higher priority task is ready to execute

The kernel maintains a queue of ready to run tasks (RTRQ). It also maintains a queue of suspended and dormant tasks. The RTRQ is a dynamic queue. If a task has a suspended condition on it removed (i.e it resumes) it is immediately placed in the RTRQ since it is now in the ready state. Depending on the priority of the resumed task it is inserted in the correct position in the queue.

2.11.1 TASKS WITH SAME PRIORITY

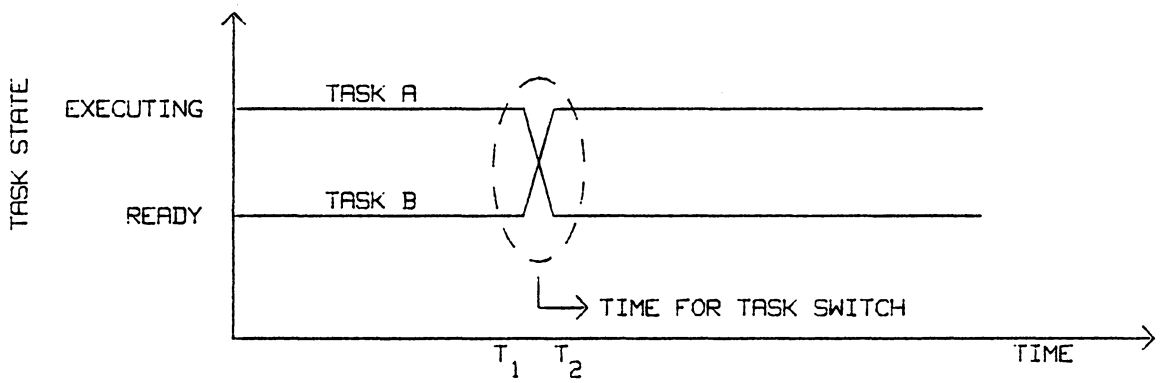
The position of a task in the RTRQ is determined by the task priority. 16 bit processors have 256 levels of priority. When more than one task has the same priority the CPU is allocated on a first-come first-serve basis. In some cases

round robin scheduling is done. In this technique all tasks are allocated the CPU for a fixed amount of time after which they are automatically suspended and the next task with the same priority is executed. Both these cases can are shown in Figure 4 on page 20.

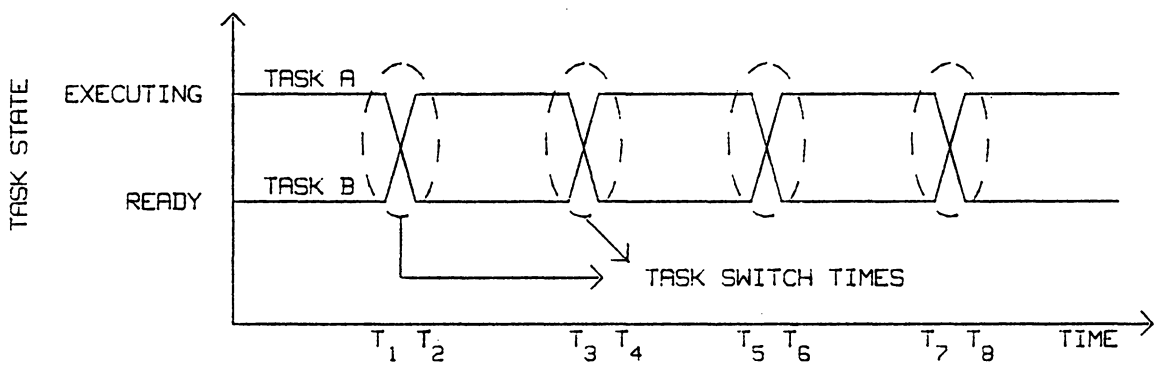
There are two tasks A and B which have the same priority. Task A takes a longer time to finish its computation than task B. Case 1 shows the CPU is allocated on a first-come first-served priority scheduling, assuming task A was ready to run before task B. Task A controls the CPU until it has finished with the computation. In case 2 the allocation of the CPU is split between tasks A and B for equal amounts of time. It can be noticed that in case 2, that some time is lost due to frequent task switching. There is only one task switch in case 1.

2.12 TASK CONTROL BLOCKS (TCBS)

Every task in the system has a task control block TCB associated with it. This data structure gives the current state of the task and provides pointers for saving the machine state of each task. It includes information about the priority, pointer to the memory where the task resides. Systems have separate stacks for system and user modes. Tasks usually run in the user mode with a stack for each task. Interrupts and kernel functions use a single system stack. If a system



CASE 1 FIRST-COME FIRST-SERVED SCHEDULING



CASE 2 ROUND ROBIN SCHEDULING

Figure 4. Example of CPU allocation using first-come first-servedand round robin scheduling techniques.

stack is allocated for each task then device drivers configured as tasks can run off this stack to perform I/O and other privileged operations useful for real-time systems. This however requires more memory for each task.

2.13 TASK COMMUNICATION

Task communication is necessary for proper operation of any multitasking system. Communication between tasks is established by means of mailboxes. Tasks do not communicate with each other directly. Instead they 'Post' a message to a memory location called a mailbox. Another task can receive a message from the mailbox.

Communication via mailboxes instead of direct communication makes it possible for tasks to communicate independent of their internal structure. If at a later date any task structure is changed, then the method of sending a message would also have to be changed. These problems can be avoided by using mailboxes.

After a task posts a message to a mailbox, it can either continue or wait until the message is received. Alternately a task looking for a message in a mailbox can wait for the message, or continue if there is none. The term 'Pend' is normally used to indicate a task waiting to receive a message. Two tasks can be synchronized by means of mailboxes.

If there are two tasks A and B, mailboxes M1 and M2, the tasks A and B can be synchronized using the mailboxes as follows:

Task A POSTS to M1 and PENDS immediately at M2

Task B PENDS at M1 and POSTS immediately to M2

The assumption here is that a task pending at a mailbox does not continue until it has received a message.

2.14 REAL-TIME CLOCK

One of the most important interfaces of a real-time multitasking operating system with the hardware is the real-time clock. The control for scheduling, timeouts, delays are all based on the real-time clock. The simple case has a clock that interrupts at regular intervals (ticks). The ISR for the clock notifies the operating system through a system call that a tick has expired. The operating system keeps track of the time internally. Depending on the resolution of the clock required ticks could be spaced 1ms to 100ms apart. A 1ms resolution however interrupts the CPU too often and takes up a considerable part of the CPU time. If a large value of 100ms were chosen and a timeout of 10ms were required, then the timeout value could be anything between 1ms and 99ms since it takes that long to notice any change in the clock.

By having calls that account for delays it is possible to synchronize with a slow environment. If for example a processor gives a command to open a relay. It takes about 50ms typically before the relay action is complete. During this time another task could be executed. Delays are the easiest and sure ways to let tasks proceed concurrently.

One of the other functions that the real-time clock supports is scheduling of tasks. Different tasks have to be scheduled depending upon the time. The scheduling usually depends on a wakeup value that is specified i.e the task is woken up and executed after a specified delay. The scheduler maintains a queue of tasks based on timeout values.

A timeout value can be specified so that, if the required event occurs before the time specified in the timeout, then the task can execute. If the event does not occur within the timeout interval the task executes only after the interval has expired. If an event occurs before the timeout then the task has to be removed from the other queues like the clock queue of the scheduler.

2.15 EMBEDDED MICROPROCESSORS

In some systems a microprocessor is not visible to the user and most of the time the user has no interaction with it directly. It is buried deep inside a system like a communications system, inside an intelligent terminal or an in-

dustrial robot. Such a microprocessor is called an embedded microprocessor [10]. An embedded microprocessor is different from a stand alone microcomputer that is used in business systems, in that an embedded microprocessor needs to have real-time responsiveness. A multitasking operating system provides a powerful way to control these microprocessors, since these processors respond to unrelated events occurring asynchronously in time.

2.16 GRAPHICAL REPRESENTATION

A graphical representation of multitasking as shown in Figure 5 on page 25 can be used to get a better understanding. Suppose there are three tasks A, B, C requiring the services of the CPU, disk and terminal. If the three tasks were to run sequentially the total time taken would be as in case 1 of Figure 5 on page 25. If the three tasks run concurrently a different set of operations is possible as shown in case 2. The tasks A, B, C are assigned the priorities 1,2,3 respectively (1-indicating highest and 3-indicating lowest priorities). When task A reaches a pause condition, like waiting for a disk access, task B starts executing as long as it can, giving the appearance of simultaneously operation. The total multitasking time is less than the time in the sequential case.

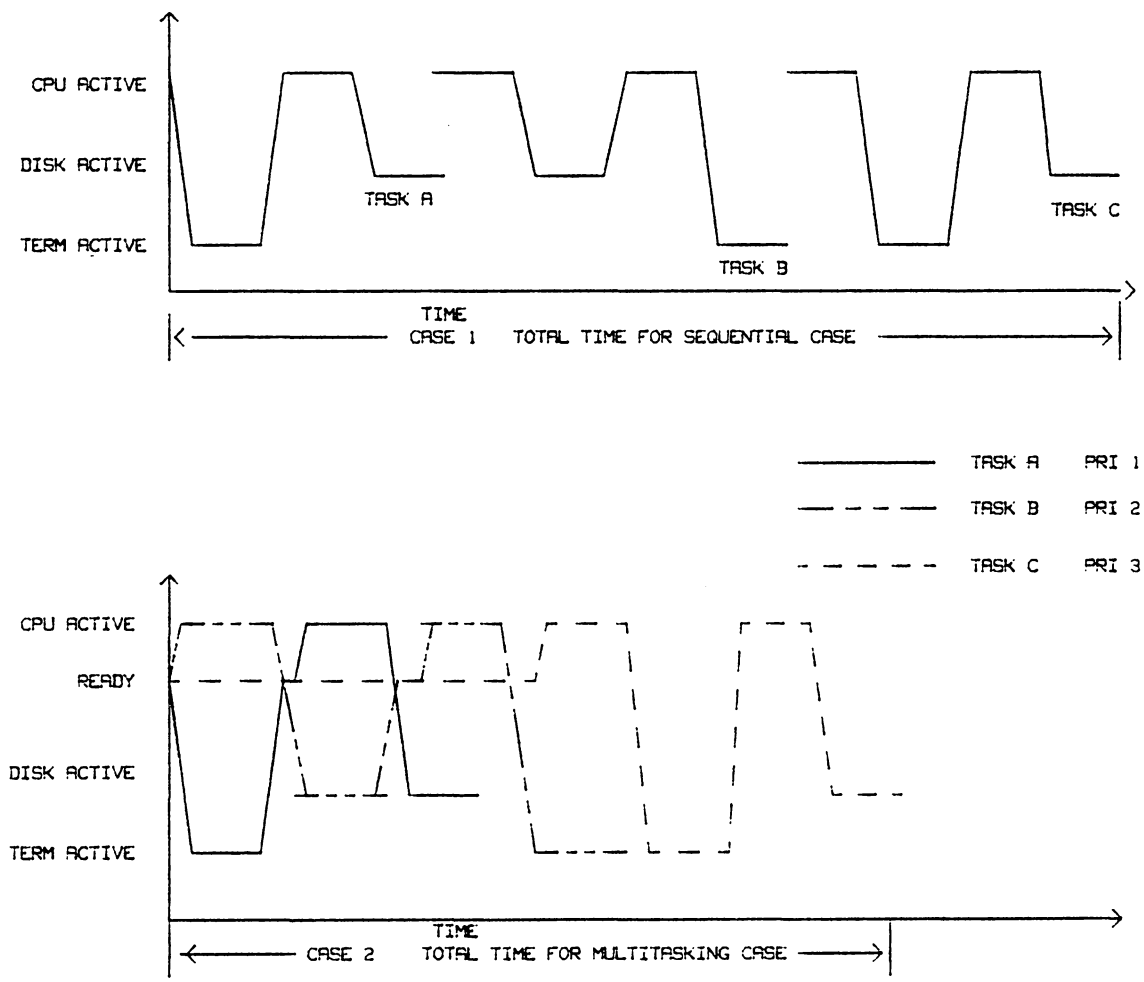


Figure 5. Graphical representation of Multitasking

2.17 DECOMPOSING A SYSTEM INTO TASKS.

For the proper and efficient operation of any given system, a method has to be found to split the various functions into tasks. A methodology for decomposing a system into tasks can be derived from the basic understanding of a task. By following a set of rules, a more structured approach can be arrived at in developing the program. The chief requirements of any multitasking system are

- Software should meet system' specifications, especially timing.
- Software should be easy to maintain.
- The software should be expandable, with very little modification to the original source code.

Before a set of rules can be defined, the difference between a condition and an event has to be mentioned.

2.17.1 CONDITIONS AND EVENTS.

A condition is a predicate that defines the state of the system [11]. For example x<y might be a condition. An event occurs when a condition changes from true to false or vice versa. Conditions are present for a finite amount of time

while an event occurs at a single point in time. When a variable changes from x to y it might indicate an event.

The rules to be followed in defining tasks are as follows. Each functionally distinct activity is assigned to a different task. Each function if it depends on a number of concurrent activities can be further split into tasks. By following these two rules, changes can be made very easily in the software and the tasks are highly independent. If all the subsystems had been assembled into one task any changes at a later date would require that the code be modified considerably. Tasks that depend on events or conditions need not know how these conditions or events are generated and how they are used by other tasks. If this rule is followed then the manner in which a condition or event is generated and used can be changed without modification to the program that uses it. Restricting information about the tasks that signal a condition to event, the other tasks need not be changed when there is a change in the condition to event task. The only tasks that need to be changed are those that detect the event.

A functional module can be split into different subfunctions. Subfunctions with different processing for different inputs are assigned to different tasks. This makes it easier for processing to proceed simultaneously as the results of one input are independent of other inputs. For example in a data logging system, if a signal has to be captured

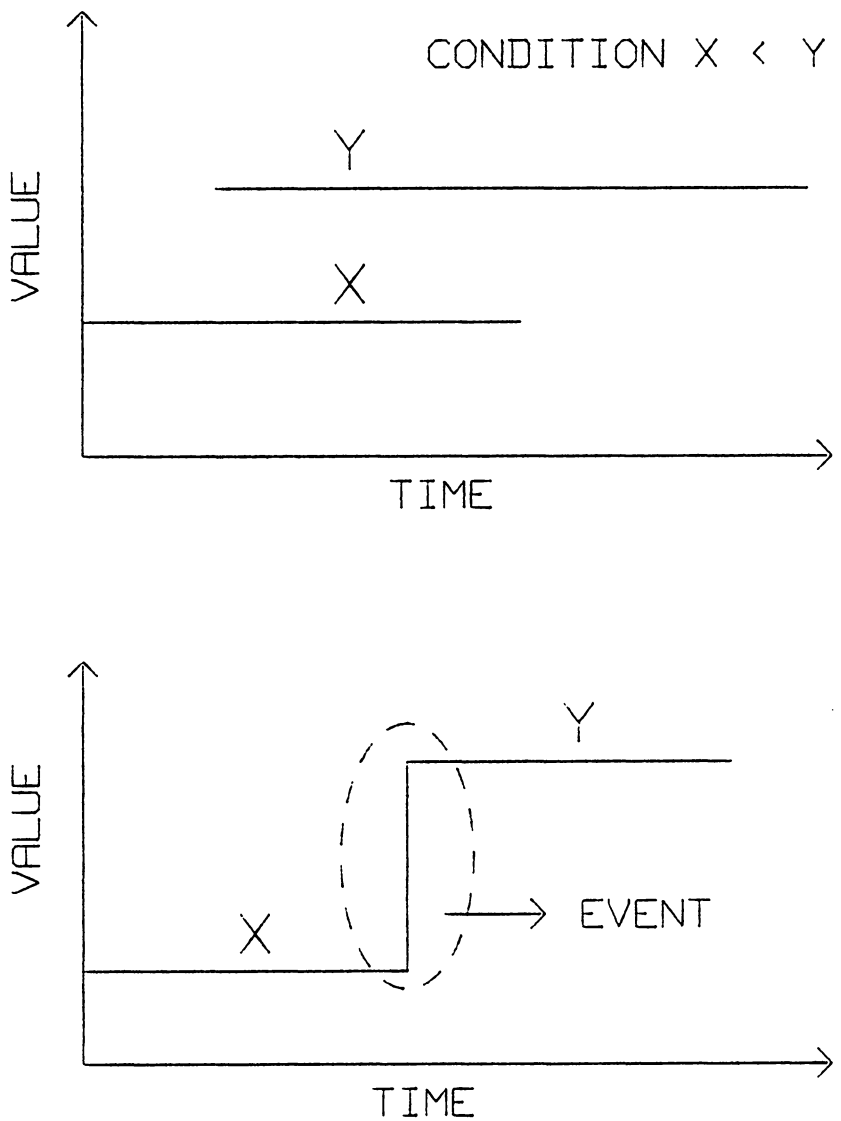


Figure 6. Difference between condition and event.

and analyzed this could be divided into two tasks instead of just one. Since the signals are transient and very quick, by the time the first signal is analyzed, the next few signals would be lost. It would be more efficient and reliable to have two tasks, one that would capture the signals and store them in a buffer temporarily and another task that analyzed the stored data when there are no signals to be captured. This decomposition of a system can proceed as shown in Figure 7 on page 30.

Subfunctions with different priorities are assigned to different tasks. The highest priority task is assigned to the functions which are critical and over which the user has no control. A good example would be a power failure protection task. This task would save the machine state at an impending power failure.

Functions activated by different interrupts are placed in different tasks. This lets less important tasks to be interrupted by higher priority tasks. When an ongoing processes is very fast it is better to use an interrupt server, since task switching takes time. For example if an operating system takes 't' microseconds to perform a task switch, any process that interrupts at a frequency less than 't' microseconds is better served with an Interrupt Service Routine (ISR).

The lowest priority tasks are assigned to terminal and other I/O devices which are inherently slow. For example if a disk read is required, the read head has to be positioned

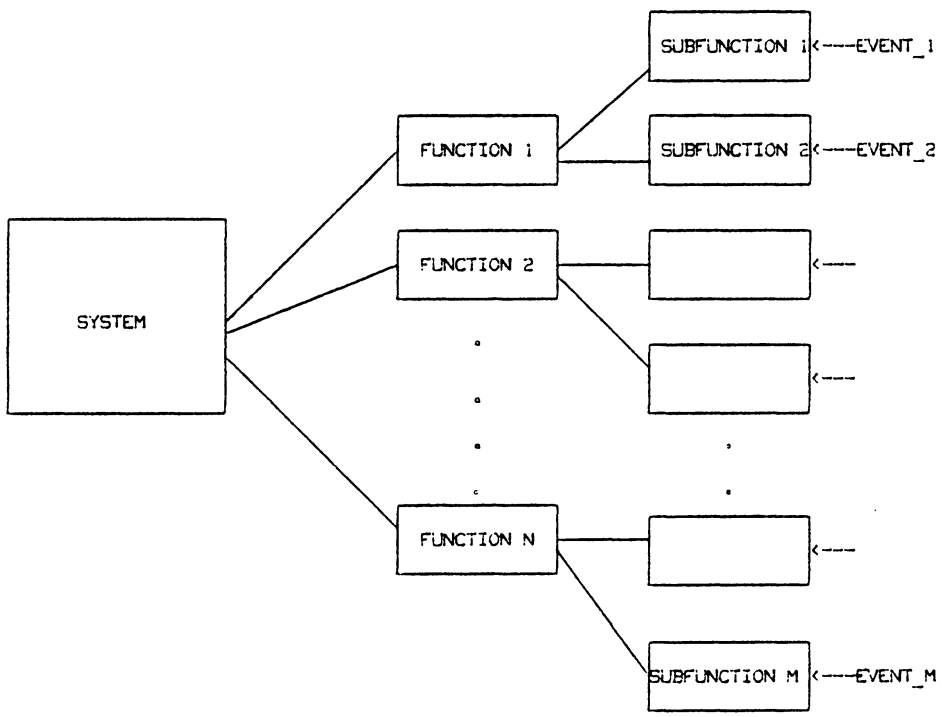


Figure 7. System Decomposition

at the correct sector and track. This operation involves mechanical components which are an order of magnitude slower than the processor. If these tasks are assigned high priority they could tie up the processor when it is required for another critical operation.

These are some the rules that need to be followed when designing a multitasking system. These rules have been followed in implementing a Vehicle Instrumentation System, that is shown in a later chapter.

3.0 AN OVERVIEW OF VRTX

There are a number of multitasking operating systems available from numerous vendors. The most commonly available multitasking systems are

- iRMX 86 - from Intel.
- AMX/86 - from Kadak Products.
- MTOS/86 - from Industrial Programming.
- VRTX/86 - from Hunter and Ready.

Due to the generous support of Hunter and Ready it has been possible to obtain VRTX/86 free of cost. This is the operating system that we have had access to and all discussions henceforth will be with respect to VRTX.

VRTX the versatile real-time executive is a multitasking operating system for 16 bit microprocessors [7]. It is a compact executive occupying less than 4K (0FA6H) bytes of code. It can perform a task switch in less than 100 microseconds. It is a silicon software component which is among a growing class of products available from numerous vendors [12]. VRTX features system calls for

- real-time clock
- task management

- memory allocation and management
- intertask communication and synchronization
- single channel I/O

Simple ISRs establish the connection between the VRTX kernel and the hardware.

3.1 SILICON SOFTWARE COMPONENTS

When a system has to be designed, cost and development time is minimized by using standard ICs [13]. The same cannot be said however, about software. Conventional software does not work on all microcomputers unless the source code is modified to fit the specific system. A silicon software component is an executable version of a microprocessor program that can operate on all board level microcomputers using the same microprocessors. Silicon software components can be interconnected amongst themselves like pieces of hardware. Operating systems are normally composed of separate modules which are linked together using system calls at run time. Each of the components can be coded in ROM and plugged into any board that uses the same microprocessor. Hunter and Ready has 4 components readily available providing a complete multitasking operating system for any microcomputer. They are VRTX which is the kernel, IOX - the I/O executive, FMX - the file manager and TRACER - which is a debugger.

3.2 VRTX ORGANIZATION

VRTX is written in position independent code which means that it can be placed anywhere in the user address space of the processor. It must however be placed starting at a paragraph address, i.e the offset into VRTX has to be zero. On the 8086/88 processors VRTX has to be located at an address of the form XXXX:0, where XXXX is the code segment. VRTX supports a wide range of processors including the 8086 and 8088. The packaging for the 8088 is slightly different from that of the 8086 since data is fetched in bytes instead of words. VRTX is completely independent of any assemblers, linkers or loaders and also independent of any specific target environment.

3.3 VRTX ARCHITECTURE

The VRTX architecture follows the layered approach to a multitasking operating system mentioned previously. A system based on VRTX is layered according to functions with each level making use of the functions provided by the level below it as shown in Figure 8 on page 35. It could be said that each level provides a virtual machine for the level above it. The interrupt handlers provide support for the interrupt driven peripherals. It is possible to have user defined calls to interface with the operating system and effectively extend

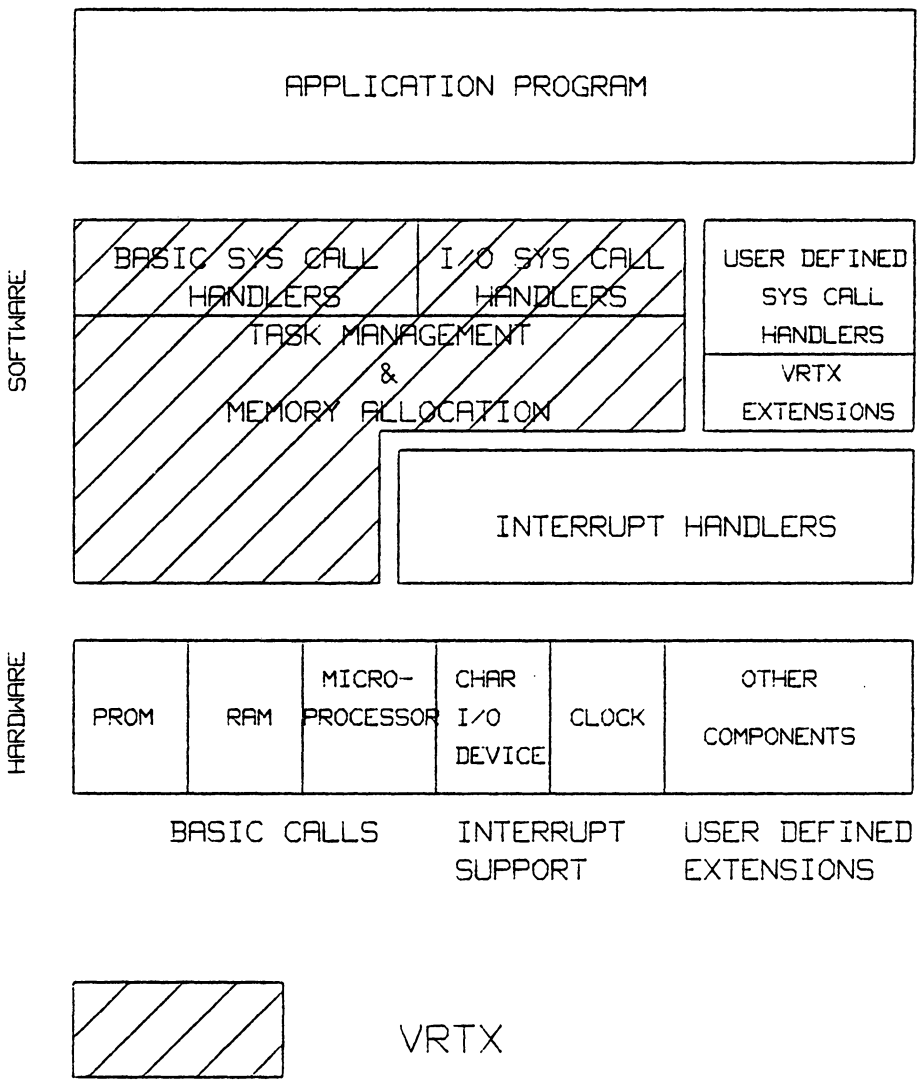


Figure 8. VRTX Architecture [7].

the capability of VRTX, so that it can be made specific to a particular application.

3.4 SYSTEM CONFIGURATION

VRTX is independent of any particular board environment. To run VRTX on any custom board the source code need not be modified. However to interface to the board some mechanism has to be found. VRTX does this using the interrupt vector table of the processor. The 8086 processor has an Interrupt Vector Table (IVT) to handle 256 interrupts. One interrupt vector points to the VRTX entry location. Another vector points to a configuration table (CT) which is used by VRTX for its operation. The configuration table defines addresses for VRTX workspace, the size of the workspace, interrupt support, multitasking controls and pointers to other silicon software components. The parameters in the CT indicate the number of tasks that can exist in the system at any one time, the size of each tasks' user stack, size of the interrupt stack, etc.

Any of the interrupt vectors can be used for entry into VRTX and the configuration table. There is a pointer to the CT within the VRTX to help VRTX to locate it. VRTX uses INT 80H as the interrupt for the CT by default. If any other interrupt is used to vector into the CT the default vector ad-

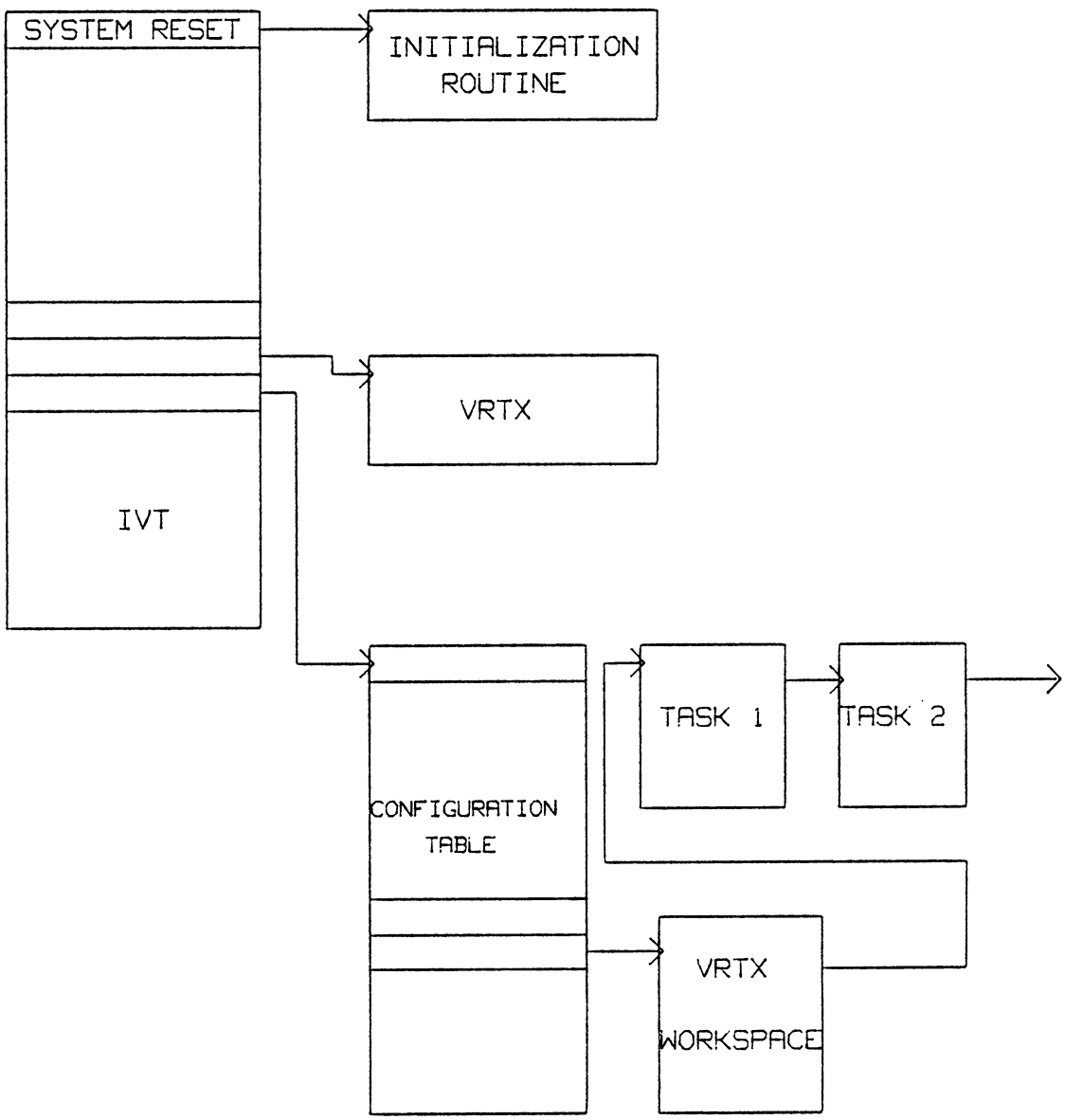


Figure 9. Connecting VRTX to the system board

dress has to be changed at offset 22H in the VRTX software. This can be done when copying the master PROM into backups.

3.5 VRTX CALLS

VRTX calls can be grouped as follows

- Task management
- Memory allocation
- Communication and synchronization
- Interrupt support
- Real-time clock
- Character I/O
- Initialization

Under each category there are several calls that accommodate user requested calls. The call is invoked by moving a function call number into the AX register and issuing an

```
INT  n
```

instruction, where n represents the interrupt number which vectors to the start of the VRTX code. Along with AX, other registers also contain parameters. A return code is passed back in AX. Depending on the return code different actions can be taken. The general format for making a call is

```
MOV  AX, function call number
INT  n
```

Before the call, any of the other parameters are passed into the appropriate registers.

3.6 TASK PRIORITY, SCHEDULING AND CREATION

VRTX supports upto 256 logically distinct and active tasks. In addition any number of dormant tasks can be present. All tasks are assigned unique identification numbers and priority levels. The priority levels are from 0 to 255 with 0 indicating the highest priority and 255 the lowest. A task with priority zero is created and this task creates the other tasks. The general format of creating tasks is

```
begin
  initialize system variables;
  create main task;
  start multitasking;
end.

proc ( main task);
begin
  create task 1;
  create task 2;
  ...
  ...
  create task n;
end.
```

Tasks are scheduled on the basis of priority. The highest priority task executes before a task of lower priority. At any given time the task that executes is the task that does not have any pending conditions imposed on it. If a task is waiting for an event like a message from another task or a timeout to occur, it cannot proceed. The task goes into a suspended state and the next highest priority task that has all its conditions satisfied starts executing. Assigning priorities to the different tasks determines the scheduling followed by the tasks. In most operating systems like VRTX the scheduling is transparent to the user.

3.7 INTERTASK COMMUNICATION AND SYNCHRONIZATION

In any large system with a number of tasks it is very common for one task to 'talk' to another task. This is usually the case even though the tasks operate asynchronously. VRTX uses mailboxes and queues for communication and synchronization. The mailbox in VRTX is a two word variable that the user allocates in read/write memory. It has to be created explicitly by the user before it can be used. Three very powerful commands in the form of system calls

```
SC_POST        ; Post a message
SC_PEND        ; Pend for a message
```

```
SC_ACCEPT       ; Accept a message
```

provide a means for intertask communication, mutual exclusion and resource locking.

A message is put in a mailbox using the SC_POST command. If a task needs to receive the message it issues a SC_PEND call. If there is a message the task receives it. If there is no message then the task waits for a message until it is posted. A timeout value can be given in the CX and DX registers, after which the task resumes operation if there is no message. If a task need not suspend operation if there is no message, then the SC_ACCEPT call can be used. This call returns an error code if there is no message present. When more than one task pends at the same mailbox the task having the highest priority receives the message. Resource locking using mailboxes can be done when all tasks that require the same resource pend at the same mailbox. After each task finishes with the resource it posts a message to the same mailbox to resume the next task.

Queues in VRTX are used to implement message queuing. They can also implement a generalized version of the SIGNAL and WAIT primitives mentioned before. It can also be used to implement a semaphore. The mailboxes as can be seen from above implement a binary semaphore. Using queues it is possible to implement semaphores for multiple resources of the same type. Consider a pool of N buffers that are to be used by tasks temporarily. When a task needs a buffer it performs a WAIT

operation and when it is finished with the buffer it performs a SIGNAL operation. These operations are performed not on a binary semaphore as before but on a queue. By creating a queue of size N it is possible to control access to the pool by every task. The whole pool of N buffers can be accessed at the same time, with one task acquiring one buffer. Every entry in the queue represents a semaphore for each of the buffers. We could say that we have here a general semaphore of size equal to N. If the length of the queue is one this case degenerates into a mailbox and thus implements a binary semaphore.

3.8 INTERRUPT SUPPORT

In any real-time system interrupts are generated to interface with the external environment. VRTX provides methods to handle interrupts asynchronously. ISRs are user supplied routines which control the scheduling of critical tasks. Interrupt service routines are coded as small as possible and data is passed on to the tasks for further processing. There has to be close cooperation between tasks and the ISRs. ISRs are executed directly without any intervention from VRTX. The user has to save any registers used and restore them eventually. Most of the VRTX calls can be made from the ISRs. The most commonly used calls are SC_POST,

SC_QPOST, SC_ACCEPT and SC_QACCEPT which allow communication of messages between tasks and ISRs.

The ISR is different from a task and the system has to know where a call is being made from - the task environment, or from an ISR. All multitaking suspends when there is an interrupt and control passes to the ISR. VRTX uses two calls, UI_ENTER and UI_EXIT to inform VRTX of control passing over to an ISR. UI_ENTER signals the start of the ISR and UI_EXIT signals the end of the ISR, and also reschedules the tasks, if necessary upon return to the task environment.

Using these two calls it is possible to make sure that proper communication is effected between the tasks and the ISRs, and on return to the task environment the rescheduling procedure is initiated which takes into account any calls made from the ISR. An ISR uses the stack of the interrupted task if it has not been allocated a separate stack. If a separate stack has been allocated, as defined by an entry in the configuration table, then the ISR uses this stack. Normally if interrupts occur with a very high frequency it is better to use the current task's stack, since it takes time to perform a switch between the task stack and the interrupt stack.

3.9 REAL-TIME CLOCK

VRTX keeps track of the real time with a 32 bit counter. It is possible to set the time and read the time. This timing support forms the basis for most of the scheduling by VRTX. The SC_PEND calls are given a timeout value, which indicates the maximum amount of time they can wait. These values are calculated with respect to this real-time clock. When round robin scheduling is enabled for equal priority tasks the time each task is active is determined by this clock. The SC_TDELAY call is used to delay the call for a specified amount of time.

The hardware clock on the system board generates an interrupt, at a frequency determined by the clock resolution. In the ISR the VRTX call UI_TIMER is issued which indicates to VRTX that a tick has expired. This increments the internal timer by one. Since the timer is 32 bits wide it rolls over after OFFFFFFFFH ticks. The ISR for the timer is a very small routine which is executed very fast.

3.10 CHARACTER I/O

VRTX supports character handling using a set of five calls. VRTX manages 64 byte FIFO buffers internally for character input and output. Any character handling is through these buffers. There is an assumption made by VRTX in re-

ceiving and transmitting characters. VRTX assumes that the character handling device (such as an USART) on the system board generates an interrupt whenever

- the device is ready to receive a character
- the device is ready to transmit a character

VRTX provides two calls UI_RXCHR and UI_TXRDY that can be used in the ISRs for the receiver and transmitter to inform VRTX about the status of the USART.

For example when the USART is ready to receive a character, it generates an interrupt. This character is read in and passed on to VRTX. This is done by using the call UI_RXCHR in the ISR. This passes the character into the input buffer of the VRTX. When any task requires a character it issues a call SC_GETC. The character in the buffer is passed onto the task. If there is no character in the buffer the requesting task is suspended and VRTX initiates the rescheduling procedure.

When any task wants to output characters it issues a SC_PUTC call. The characters are accumulated in the 64 byte output buffer. The UI_TXRDY call is used in the transmitter ready ISR. This call informs VRTX that the USART is ready to transmit. If there are any characters in the output buffer then the character is returned to the ISR by VRTX and it is output to the screen or any other I/O device. If there is no

character in the buffer an error message is returned. However the next time a task issues a SC_PUTC command the character is directly passed on to the ISR without going through the buffer.

The input-output executive IOX [14] provides more features for character control. It can handle multiple characters at the same time. When connected with VRTX it is possible to provide considerable amount of character handling support, more than is possible with VRTX alone.

3.11 MEMORY MANAGEMENT

The VRTX operating system is supplied in an EPROM. It could be loaded into dynamic memory for operation. VRTX requires some amount of memory for its own internal variables and working. VRTX has system calls to manage user memory if required. The VRTX workspace contains the system variables, a TCB for each task, a stack for each task in the system and an interrupt stack if necessary. The user memory that is managed by VRTX consists of partitions or blocks of memory.

Memory is another resource that has to be allocated like any other ordinary resource to competing tasks. There are two types of memory allocation techniques commonly used

- static allocation
- dynamic allocation

In static allocation each task is allocated a fixed block of memory for its use. This block of memory cannot be used by any other task. The problem with this type of allocation is that once allocated memory is tied up no matter it is used or not. In dynamic allocation the size of the blocks is not fixed and the system tries to match the demand of each task by allocating blocks of memory as and when requested. Blocks of memory are acquired and released by the tasks.

VRTX has both features included in its memory management scheme. Every task has its private stack which is fixed in size. The rest of the memory is allocated dynamically using partitions and blocks. Every partition is split up into blocks of fixed size. If any task requires a block of memory which is smaller than the amount of memory in a single partition block, the task uses only what it requires and leaves the rest of the partition free to other tasks. If on the other hand a task needs a block of memory greater than the largest partition available, by using the VRTX call SC_GBLOCK it is possible to organize the memory into appearing like a single contiguous block. The memory need not be contiguous physically but to the task it appears contiguous.

3.12 USER EXTENSIONS

An operating system provides almost all the functions required by the user. In some real-time systems it is necessary

to modify the software to suit the specific application [15]. It is possible for the user to extend the basic functions of the system calls. Three other vectors are also provided in the configuration table to customize the application. The three vectors are called sys-tcreate-addr, sys-tdelete-addr and sys-tswitch-addr. Whenever a task is created, and a sys-tcreate-addr is given by the user, control passes to that address. So in addition to normal processing by VRTX the user can perform special processing whenever a task is created or deleted. With the sys-tswitch-addr, every time there is a task switch special processing can be performed. For example if there is Floating Point processor on the board then every time there is a task switch the state of the Floating Point processor has to be saved. This section of the code can be pointed to by sys-tswitch-addr.

3.13 VRTX COMPONENT EXTENSIONS

As mentioned before there are three more silicon software components that can be connected to VRTX. These are the IOX, FMX and TRACER. All systems do not have these components but when they are present in the system they have to be connected to VRTX. VRTX knows about the existence of these components by means of a Component Vector table (CVT). An entry in the configuration table (CT) points to the CVT. If there is no CVT then the entry in the configuration table is set to zero.

The component vector table indicates the VRTX extensions available.

Every component has an id assigned to it. For example IOX has a component number of 2. Any IOX call has a field which indicates the component number. The kernel examines every system call sent to it, to determine where it has to be routed. The CVT contains information on the component's code start address and the workspace start address. The linkage between the IVT, CT and the CVT is shown in Figure 10 on page 50.

3.14 INITIALIZATION

Any program using VRTX has to initialize it for proper operation. Initialization is performed in three stages

- Pre-initialization
- VRTX-initialization
- Post-initialization

Pre-initialization sets up the interrupt vectors that point to the configuration table and the start of VRTX. The pointers to any user supplied ISRs are also set up during this time. After this the VRTX-initialization is performed by issuing the call VRTX_INIT. This call initializes all the internal variables used by VRTX. If enough memory is not

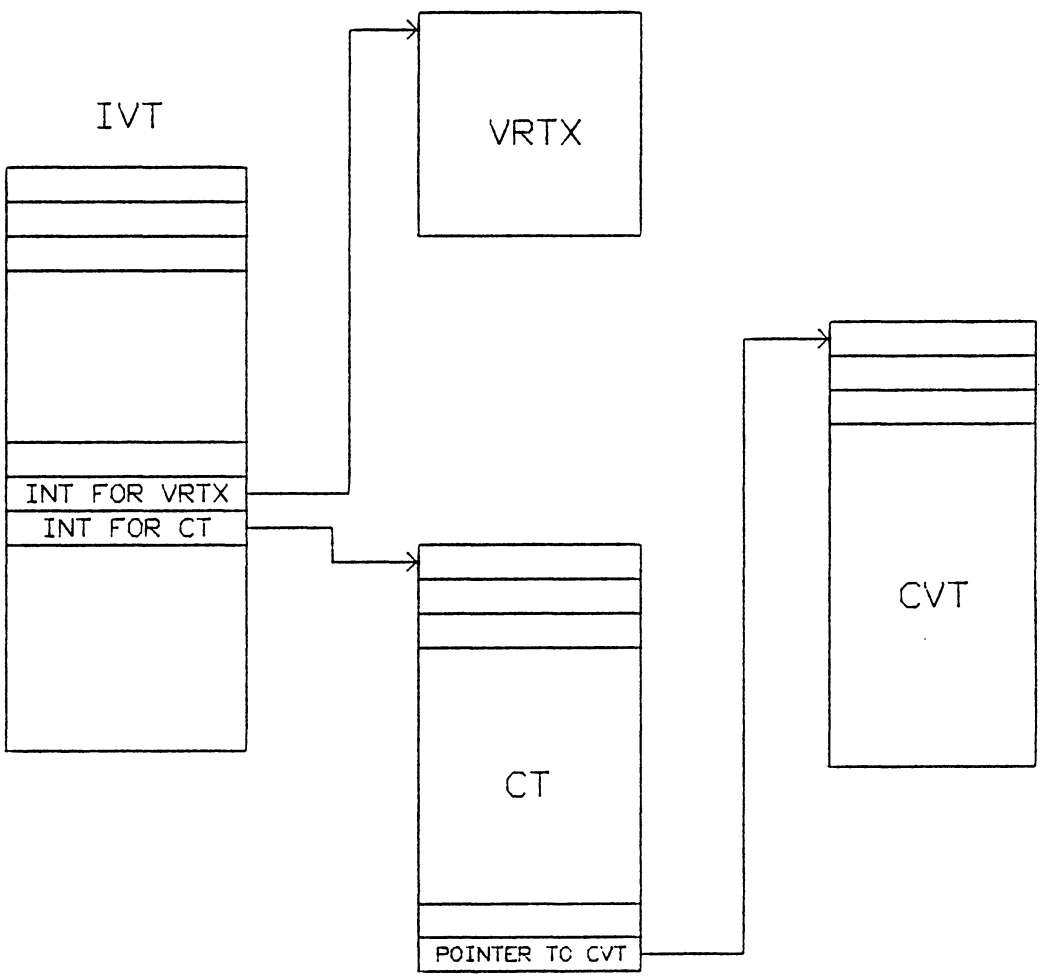


Figure 10. Linkage between IVT, CT and CVT [7].

provided for the given number of tasks an error code is returned to the user program.

The Post initialization routine consists of two important functions [16].

- Device initialization
- Initial task creation

The device initialization process consists of initializing the on board hardware components like the

- Interrupt controller
- Timer
- USART
- Any other devices present

The initial task is created in this section of code. This task is usually assigned a priority of zero (highest), and this task in turn creates the other tasks. Another possibility is, all the tasks and any other control structures like queues and partitions are created here. After device initialization and task creation are complete, multitasking is started using VRTX_GO. Once this call is issued the highest priority task starts executing and there is no return to user supplied initialization code.

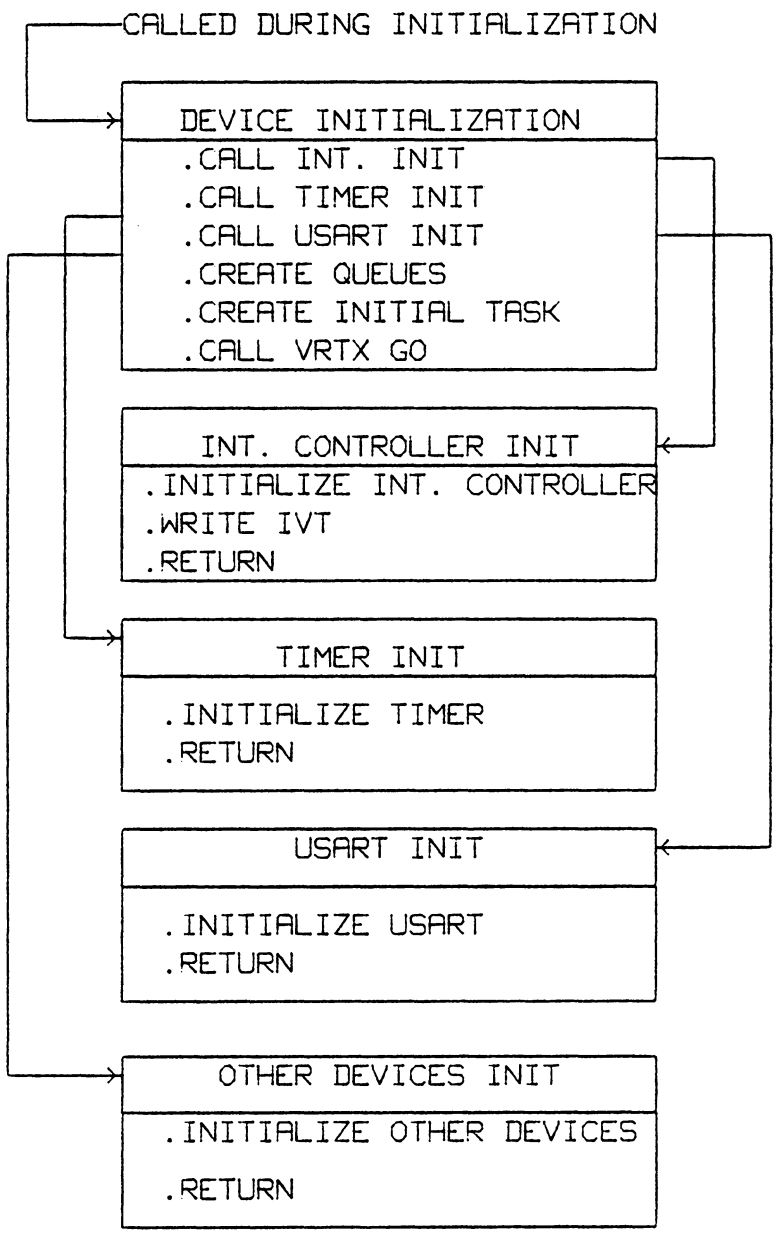


Figure 11. Initialization Sequence [16].

3.15 A COMPLETE SYSTEM

A complete multitasking system using VRTX,IOX and FMX can be configured as shown in Figure 12 on page 54. The complete system consists of not just the three components but also has interrupt service routines and device service routines and user defined extensions if any.

3.16 LANGUAGE SUPPORT

Any multitasking program for a real-time system, involves a considerable amount of code. Hunter & Ready provides high level language support for VRTX. Programs can be developed in Pascal, C or PL/M. Using standard interface libraries provided by Hunter & Ready [17] it is possible to interface with VRTX. All the calls available to the user in assembly language are also available to the high level language user.

For example VRTX calls are implemented as functions in C. By using these functions it is possible to incorporate multitasking into the C language without having to modify the compiler. Although the functions can be used in a high level language program, they are actually written in assembly language. When a user makes a high call in C it is similar to making it in assembly language. The parameters are taken off the stack, where the compiler puts them, and put into regis-

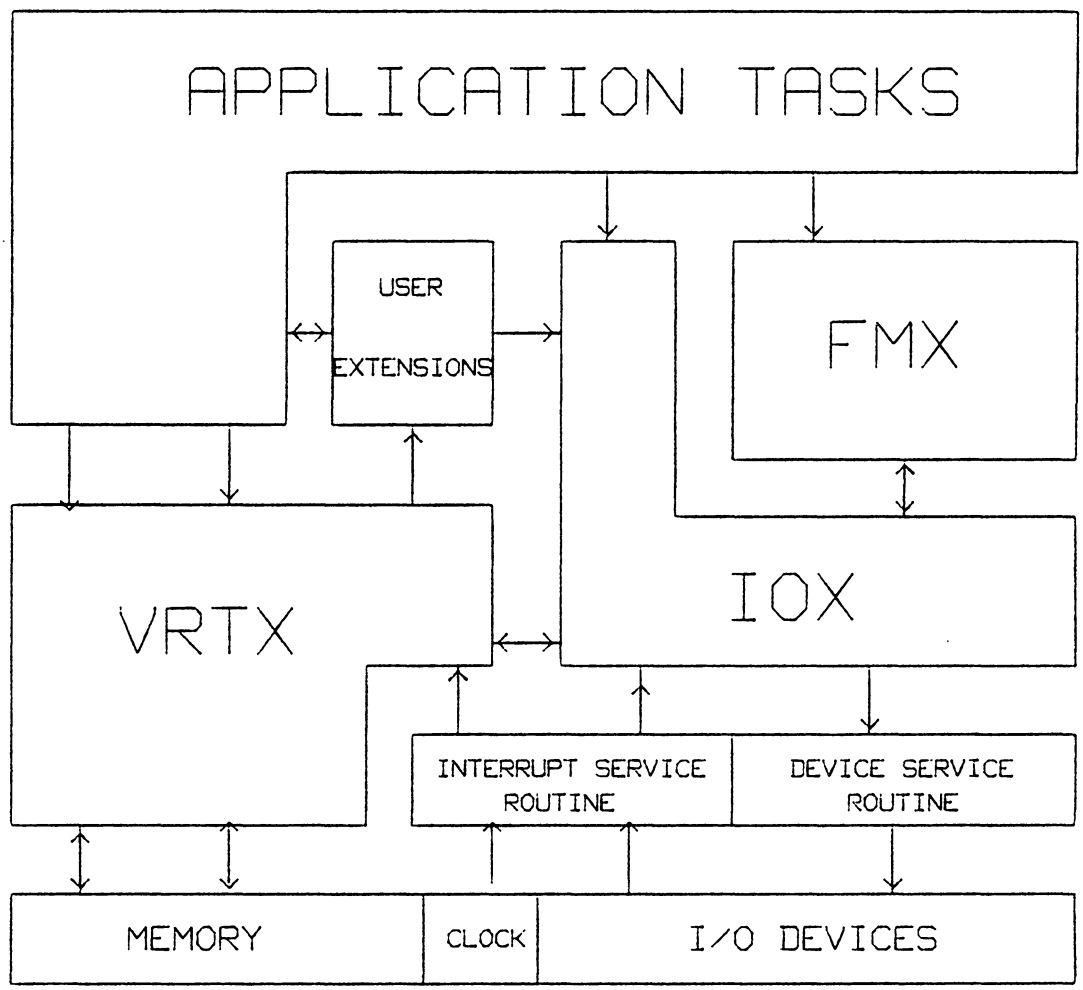


Figure 12. A Complete VRTX System [13].

ters where VRTX expects them and an interrupt instruction is executed which traps into the entry point of VRTX.

4.0 MULTITASKING ON THE IBM PC

VRTX is independent of any particular board level environment. It can be used on the IBM PC and operate the PC under a multitasking environment. Since the IBM PC uses an 8088 processor, which has an eight bit data bus, the data has to be fetched in bytes. VRTX is burnt into a single 4K PROM and placed anywhere in the 1 Megabyte address space of the processor. The VRTX operating system needs to be interfaced with the hardware on the PC. The user has to write the software to connect VRTX to the hardware. This piece of software is termed 'board support package' and typically involves about 100 lines of code.

4.1 SOFTWARE DEVELOPMENT CYCLE

The software development cycle for an application using multitasking is the division of activities among two computers as shown in Figure 13 on page 57. Since a multitasking system using an embedded processor is usually specific to the board on which it operates, a software development cycle which is different from standard microcomputers has to be followed [18].

There has to be a development computer on which the application is developed and then transferred to the target

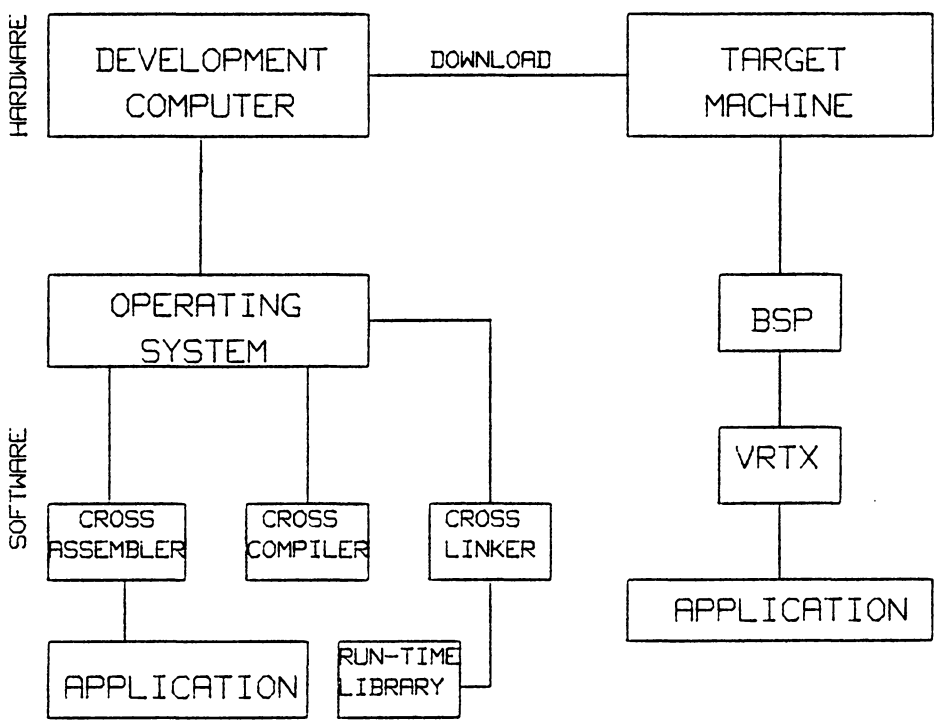


Figure 13. Elements involved in developing a multitasking program.

computer. The development computer is used for writing, assembling and linking program modules while the target computer is used for actually testing the software. The target computer runs under a multitasking environment. On the target board RAMs are used to store the programs being tested instead of PROMS. This allows the user to test the program, change it and reload it as the software is developed.

The development computer is very much different from the target computer. It is a machine designed for writing software and runs under a standard operating system like PC DOS. The development system also has a cross-assembler which generates code for the target computer. It also has a cross-linker which links together separately assembled modules to be down loaded on to the target machine. Programs can also be written in high level language like C or Pascal and then compiled on the development computer for transfer to the target machine.

The entire development cycle is an iterative process which can be as shown in the Figure 14 on page 60.

- Develop multitasking software on development computer as modules.
- Assemble the source modules on the development computer.
- Link the object modules and any run-time libraries needed.
- Download the executable module on to the target machine.

- Run the program on the target machine.
- Identify an errors and go through the whole process again.
- Repeat until desired performance is achieved.

4.2 BOARD SUPPORT PACKAGE FOR THE IBM PC

The board support package informs VRTX of a number of important parameters and user subroutines like

- Interrupt used to enter VRTX
- Configuration table pointer.
- Timer initialization routine.
- USART initialization routine.
- Keyboard initialization routine.
- Interrupt controller routine.

Any other devices like a MMU or other I/O devices also have to be initialized if necessary. The IBM PC on power up executes a ROM resident program which initializes all the devices present to a default setting. On the IBM PC, keyboard support is via a type 9 interrupt [19], i.e whenever a key is struck a type 9 interrupt is generated and the program vectors to the system keyboard routine. Using this routine however, the user cannot get control of the character and

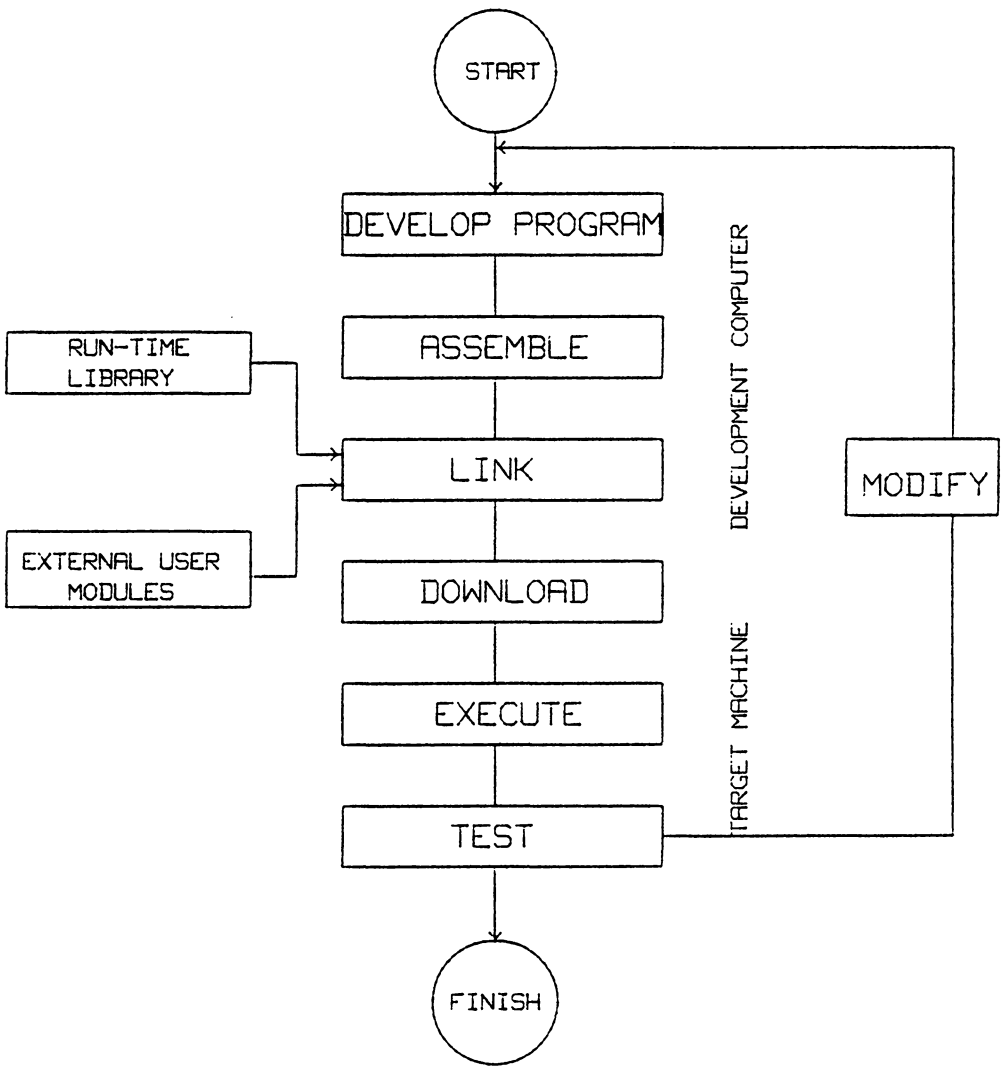


Figure 14. Development cycle

pass it to VRTX when required. In the board support package the keyboard interrupt vector is made to point to a user written keyboard routine. This routine performs the same function as the ROM resident code, in addition the user is able to get control of the character.

Depending on the application the board support package can be coded in RAM or ROM. For an industrial application which is custom built the board support code can be installed in ROM. This code can be executed whenever there is a system reset. On a general purpose microcomputer like the IBM PC the code can be loaded from a disk into RAM. It is operational only when multitasking is invoked. The connections between VRTX and the hardware are shown in Figure 15 on page 62.

A board support package is written in assembly language. It is not necessary that all systems execute the same type of code. If a system does not have an interrupt controller for example, the user need not write code for it. A board support package is tested by linking with a multitasking program [16] and executing them on the target machine. Once it works properly it can be linked with the application program.

4.3 PROGRAM DEVELOPMENT ON THE PC

To demonstrate the operation and capabilities of VRTX a multitasking program was developed on the IBM PC. The program

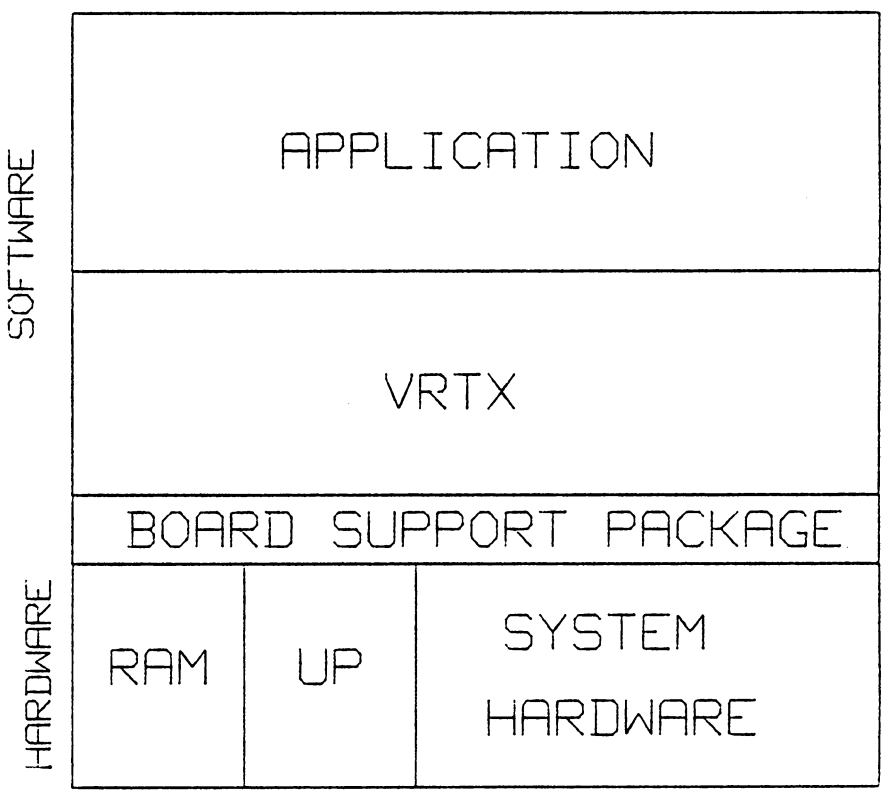


Figure 15. Interface between hardware and software [16].

is written in 8086 assembly language. A Microsoft Macro Assembler was used to assemble and link the program. The VRTX operating system was placed starting at address 4000:0H. The interrupt vector used was 60H and the interrupt vector for the configuration table was 61H. A board support package was written to initialize the pointers to VRTX, CT, keyboard ISR and timer ISR.

The first stage in developing the program was defining the problem. The program consisted of 5 tasks. The tasks are named

1. Task MAIN
2. Task TIME
3. Task WAITA
4. Task MSG_2
5. Task MSG_1

The program was developed so that it could demonstrate the basic operations using VRTX, like the interrupt handling using VRTX, character handling, the timer control provided, the message passing techniques used by VRTX and task switching. [See Appendix B for program listing.]

The task TIME keeps track of the real-time and displays it on the lower right hand corner of the screen. The task MSG_1 displays a message at half second intervals. Every 3 seconds task MSG_2 at a higher priority interrupts task MSG_1

and displays another message. Whenever the user inputs a character 'A' from the keyboard the displaying of messages stops and a prompt appears asking the user to input a character. The user has to input the character within 1 second. If the user does not input a character within 1 second the messages start displaying again. The next time a key is struck, the displaying of messages stops, the character input is displayed and the messages start displaying on the screen immediately after the character has been displayed. While all this is happening the time is continuously displayed on the screen at the lower right hand corner. A character can be input only after the character 'A' has been input. The keys are not recognized at other times. An escape causes the program to abort and the system is reset. Task MAIN at priority 0 is the only task that is explicitly created by the user and is the highest priority task. The other tasks are created by this task when multitasking starts.

When the command VRTX_GO is given VRTX puts the highest priority task in operation, which is task MAIN. This task then creates the other tasks and deletes itself leaving task TIME as the highest priority task.

4.4 PROGRAM EXECUTION

The program is assembled and linked using the Microsoft assembler and linker respectively. The program starts exe-

cuting from the symbolic address ENTRY. In this section the pointers to the various user routines are set up, VRTX is initialized, the main task is created and multitasking is started. After the program starts executing the procedure MAIN is executed. In this procedure the display is cleared and a sign on message is displayed. The other tasks are then created and the task MAIN is deleted.

VRTX now starts executing the highest priority task, which is the task TIME. This task calculates the time and displays the time at the lower right hand corner. Before the time is displayed in this position, the current cursor position is saved. The new cursor position is set and the time is displayed starting from this cursor position. The saving and setting the cursor is done using the video interrupt 10H from the ROM BIOS (Basic Input Output System) routines [20]. When the screen is full, the display is blanked except for the time and the cursor is reset to the top of the screen and messages are displayed once more starting from the top. The task is delayed for 1 second and it executes once more displaying the time again.

The task MSG_1 displays a message every half second. This task does not use the call UI_TXRDY, which is the VRTX call to output a character. This is because the output to the display on the IBM PC is not interrupt driven and as mentioned in a previous chapter, the UI_TXRDY call can be made only from a interrupt handler. To display a string of char-

acters the DOS function call 9 is used. MSG_1 waits for the display to be free before outputting the message. If the display is being used by another task, MSG_1 waits. This is an example of a number of tasks sharing the same resource. Whenever this task wants to use the display it pends at the mailbox SCREEN. If no other task is using the mailbox this task uses it. When the task receives the message the mailbox is reset to zero. So any other task trying to use the display will have to wait. When MSG_1 is finished with displaying the message it posts a message to the mailbox SCREEN enabling any task that is pending at it. The task is delayed for half second using the SC_TDELAY call and it starts executing again.

Task MSG_2 is similar to MSG_1 except it is at a higher priority then MSG_1. It displays a message every three seconds. When both MSG_1 and MSG_2 are ready to display a message, MSG_2 displays it first since it has a higher priority. The DOS function call 9 is used to display the messages. The pointer to the message is passed in the DX register.

Using the call SC_WAITC the task WAITA waits for the character 'A' to be input. The task is suspended until it receives the character 'A'. When the character 'A' has been input this task posts a message to a mailbox K1 and then displays a prompt asking the user to input a character. It then locks up the display for 1 second waiting for a character. If no character is input within one second the task

waits until the message in the mailbox K1 is received. This is done by trying to post a zero message to the mailbox K1. This message is posted only if the previously posted non-zero message has been received. If it has not been received the task waits until it has been before posting a new message.

4.5 TIMER ISR

The interrupt used for the timer is 1CH. The address of the user written ISR is loaded starting from 4*1CH = 70H. When there is a timer interrupt this ISR is executed. In this ISR the UI_TIMER call is used to inform VRTX that a timer tick has expired.

4.6 KEYBOARD ISR

The keyboard vector address starting from location 24H (interrupt 9) is changed and the address of the user written ISR is loaded. When a key is pressed an interrupt is generated by the keyboard. The keyboard port is read and the value returned is called a scancode. This code is a number which represents the key that has been pressed. A lookup table is written so that the character corresponding to each scancode is obtained. The character input is checked to see whether it is an escape character. If it is, the program is aborted and the system is reset. The routine then checks if the key

that is pressed is being done so after the character 'A'. This is done using the SC_ACCEPT on the mailbox K1. This call does not wait for a message at the mailbox. If there is a message it accepts it, otherwise it returns an error code. If there is a message in this mailbox and a key is pressed, it means that the user has input the character after an 'A' and it is displayed immediately. The call UI_RXCHR is used to pass the character into the VRTX internal buffer. VRTX internally checks if the character is an 'A' and if so resumes the task WAITA, making it ready to run. This checking for the character by VRTX is transparent to the user. The operation of the program is shown graphically in Figure 16 on page 69.

This multitasking program on the IBM PC demonstrates the general structure and methodology of a multitasking program. It should be mentioned that the PC is not the best of machines to perform multitasking for any specific application. It is however, a very good development system for multitask programs. A program developed on this machine can be transferred to any general purpose board on which the software is eventually going to be used.

From the structure of this multitask program a few interesting observations can be made.

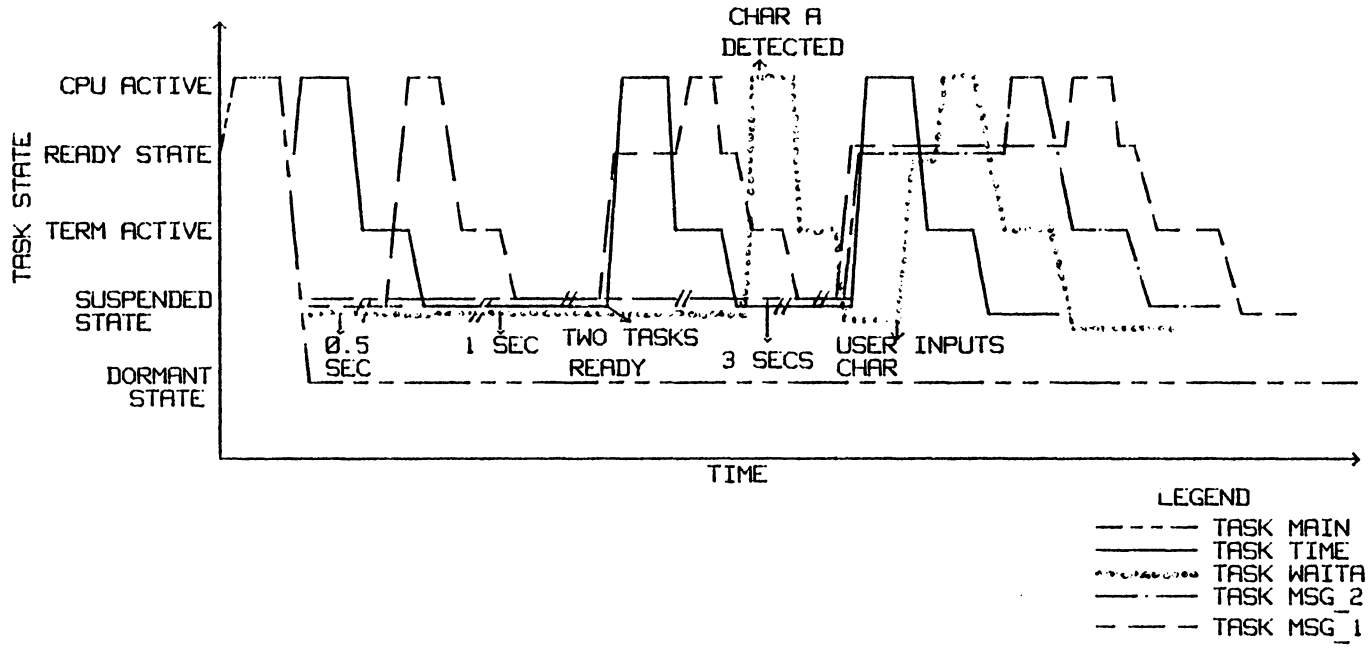


Figure 16. Graphical Representation of the Multitasking Program.

- The only loops in the program are the ones that enable the tasks to execute continuously, they are at the end of each task.
- There are no jumps within the program for testing of flags, like waiting for a flag to be set or reset. This is equivalent to waiting for an event and is implemented using mailboxes. The setting of a flag corresponds to posting a message to a mailbox.
- There are no loops that are executed to generate a delay. Time delays are implemented using system calls which perform the required function and at the same time allow the processor to perform other tasks.
- Each of the tasks can be written as independent modules making the program very structured. Within each task the system calls could be implemented using macros making the program highly readable.

5.0 A VEHICLE INSTRUMENTATION SYSTEM USING MULTITASKING

This chapter deals with the development of a typical application of multitasking to a real-time system. The system in question is a vehicle instrumentation system. Using the methodology provided in chapter 2 division of the system into various tasks is effected. The instrumentation system incorporates an automatic passenger counting system and other functions that indicate the performance of the system. This system is dependent on a number of inputs all of which have to be serviced as and when they occur. The system functions include accurate data on the number of passengers boarding and exiting the vehicle at every stop, the time spent at every stop and a location pointer indicating the current location of the vehicle.

To satisfy these requirements the system can be split up into four functional levels [20].

- Data Acquisition
- Data Recording
- Data Transfer
- Data Processing

The data acquisition function consists of the sensors for detecting passengers and detecting the distance travelled by

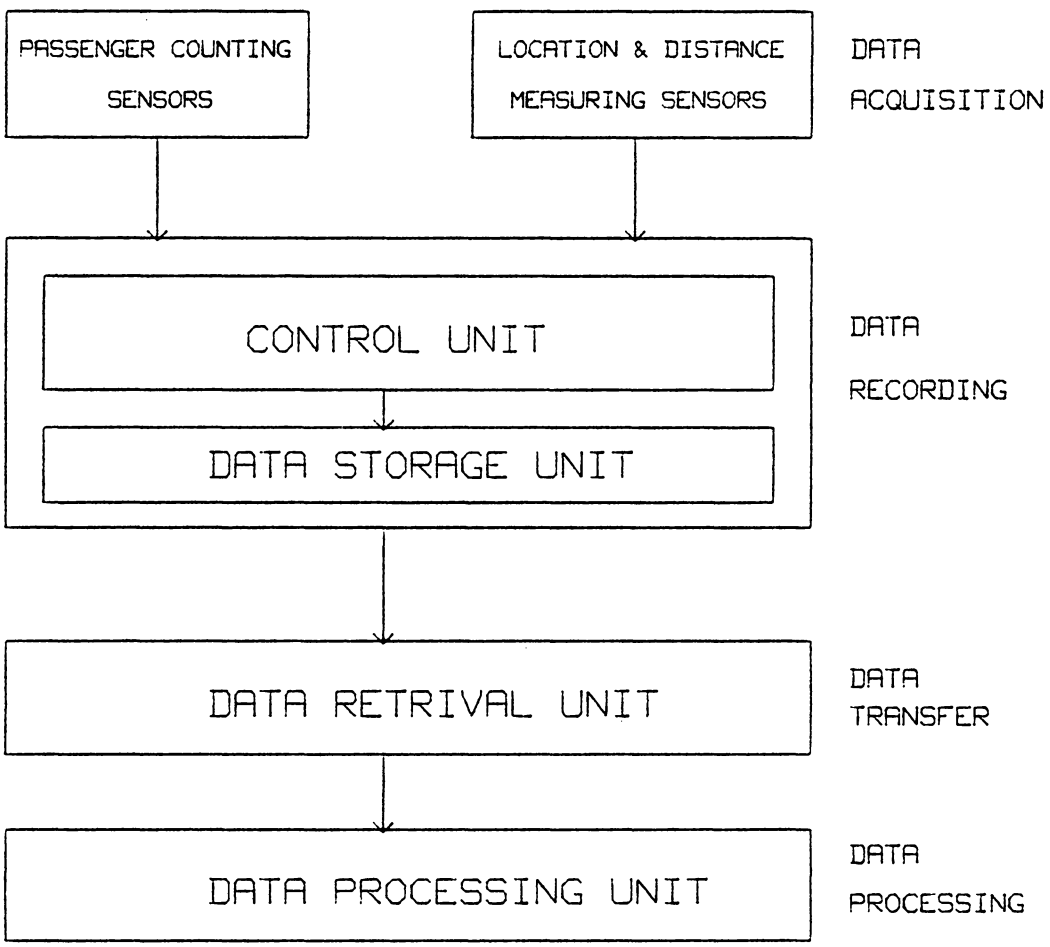


Figure 17. Organization of a vehicle instrumentation system

the vehicle. The data recording function in this system is by generating logs and writing these logs in user memory. Logs are generated that indicate the number of passengers entering the system, the number of times the vehicle has been stationary for 1 minute etc. The data transfer unit can read the logs from the memory whenever required. Analyzing this data provides a way for management to take decisions.

The system to be designed has to meet the following requirements.

1. Record the number of passengers entering and leaving the vehicle at every stop.
2. Record the number of passengers on the vehicle at any given time.
3. The number of times the vehicle has been stationary for 1 minute and the time at which it was stationary.
4. Indicate the number of times the vehicle was stationary for two or more minutes and the time at which it was stationary. Also indicate the number of minutes it was stationary if it is more than two minutes.
5. Record the number of hours the since the system has been powered up.
6. Provide a mechanism for the user to interact with the system and obtain information when required.

Based on these requirements we could have have the following tasks at a functional level organized as in Figure 18 on page 75.

- Power up
- Timer
- Door control
- Vehicle time control
- Passenger counting
- Log generation
- Log control
- Display control

5.1 PRIORITY ASSIGNMENT

We will now try to assign priorities using the rules defined in chapter 2. As can be seen from Figure 18 on page 75 7 levels of priority have been defined.

5.1.1 PRIORITY 0

Priority 0 is assigned to the main task which creates the other tasks. This guarantees that execution will start only after all the tasks are created.

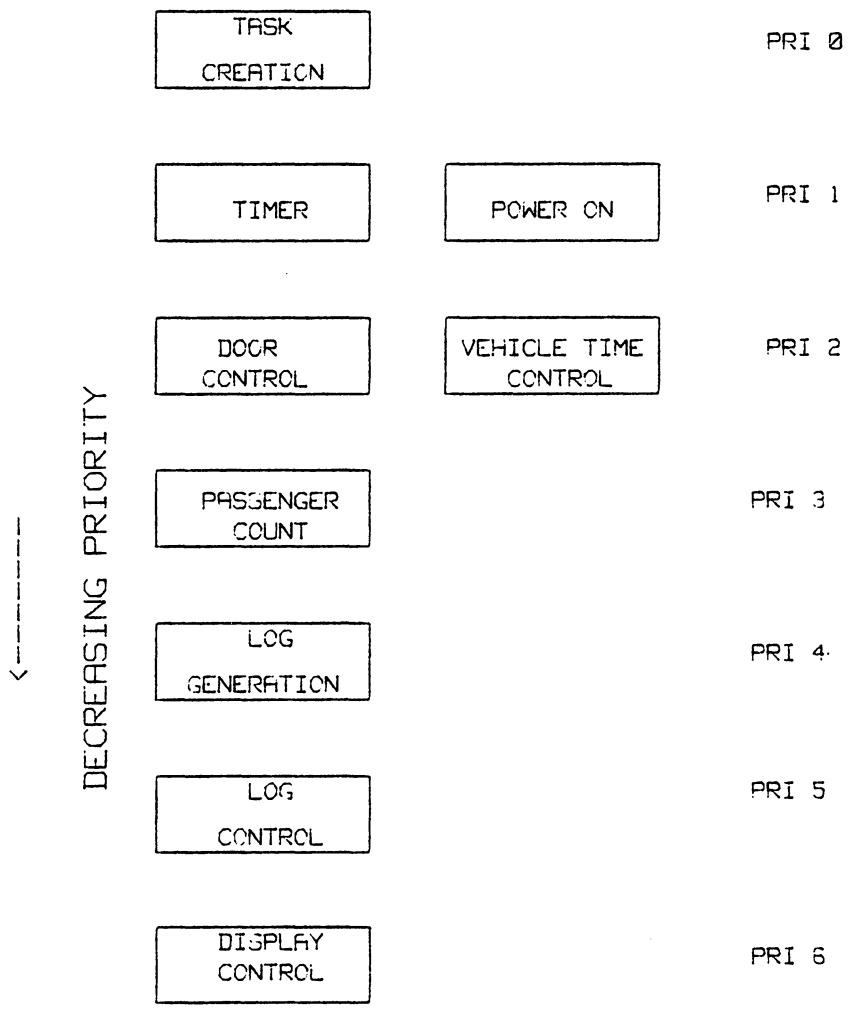


Figure 18. Task and Priority layout at a functional level.

5.1.2 PRIORITY 1

At priority 1, the highest priority executable tasks are present. These are the Timer and Power_on tasks. These are assigned priority 1 following the rule that the highest priority tasks are assigned to those which the user has no control. Irrespective of any other system activity the timer has to run, so it is assigned the highest priority. The Power_on task has to run immediately after the system has been powered up. It creates a log, which informs the user when the system was powered up, and deletes itself. This task executes only once and leaves the timer task as the highest priority task.

5.1.3 PRIORITY 2

The next event over which the user has no control is the entry and exit of passengers and the vehicle stopping and starting. To take care of these events two tasks at a functional level - Door control and Vehicle time control have been defined. The task door control has to respond immediately when a passenger enters or exits the vehicle and so is assigned the next available highest priority which is 2. The vehicle time control is also assigned the same priority since the system has to respond whenever the vehicle stops and starts over which the user has no control. Both these tasks are split

further as shown in Figure 19 on page 80 into different subfunctions. These tasks at a subfunctional level all have the same priority and they perform the required functions. If a task at this level is executing, then any other task at the same priority is put in the ready to run queue (RTRQ). Every task at this priority is executed before any task of a lower priority is executed. In the case of the Door control the passengers are detected and a count task at a lower priority level is enabled.

5.1.4 PRIORITY 3

The passenger counting task has to be at a lower priority than the the Door control task, because a passenger has to be detected first before being counted. The counting task does not need to execute every time a passenger enters or exits the vehicle. The Door control task can post a message to a queue every time a passenger enters or leaves the vehicle. The count task can pend on this queue and the number of messages it reads off the queue correspond to the number of passengers. As shown in Figure 19 on page 80 the passenger counting task is split into two levels of priority again. These two levels consist of counting the number of passengers that enter and exit through each door at every stop, and the total number of passengers who have used the vehicle. If the number of passengers at every stop are known, counting the

total number of passengers becomes easier. By letting the task that counts the number of passengers have a higher priority, the number of passengers at every stop are known before the total count is calculated.

5.1.5 PRIORITY 4

Referring to Figure 18 on page 75 priority level 4 is assigned to Log generation. Log generation consists of actually writing the logs. The logs should be written, without obstructing any of the computation going on. This would mean that this task would have to come after the passenger counting task. Assigning this task priority 4, the logs are only written when there is no computation being performed, like when the vehicle is moving. Following the rule that subfunctions with different processing for different inputs be assigned to different tasks, each type of log is assigned to different tasks as shown in Figure 19 on page 80, since the inputs to each of the tasks is different.

5.1.6 PRIORITY 5

Log control tasks are assigned this priority. These tasks include the commands given by the user to check the logs. If a user gives a command to view a log, the log has to be present, which would imply that it had to be generated be-

fore. This puts the priority of these tasks at a level lower than the log generation tasks.

5.1.7 PRIORITY 6

Following the rule that the lowest priority tasks are assigned to terminal and I/O devices, the display task which outputs the logs to the screen is assigned the lowest priority available. Any resource that is used by this task and a higher priority task is not tied up, the higher priority task can always pre-empt this task.

The organization of the system at the task level is shown in Figure 19 on page 80. It can be seen that there are a number of tasks at the same priority, and one of the functional levels of Figure 18 on page 75 has been split into two priority levels 3 and 4.

5.2 A DESCRIPTION OF TASK LEVEL LAYOUT.

The door control function can be split into two subfunctions for counting passengers. Each subfunction controls the counting for one door. At the subfunction level, splitting the task, we can interface directly with the conditions to be satisfied for a passenger to enter and leave the vehicle. Proceeding one more level down the tasks have to interface with the sensors which detect the entry and exit of passen-

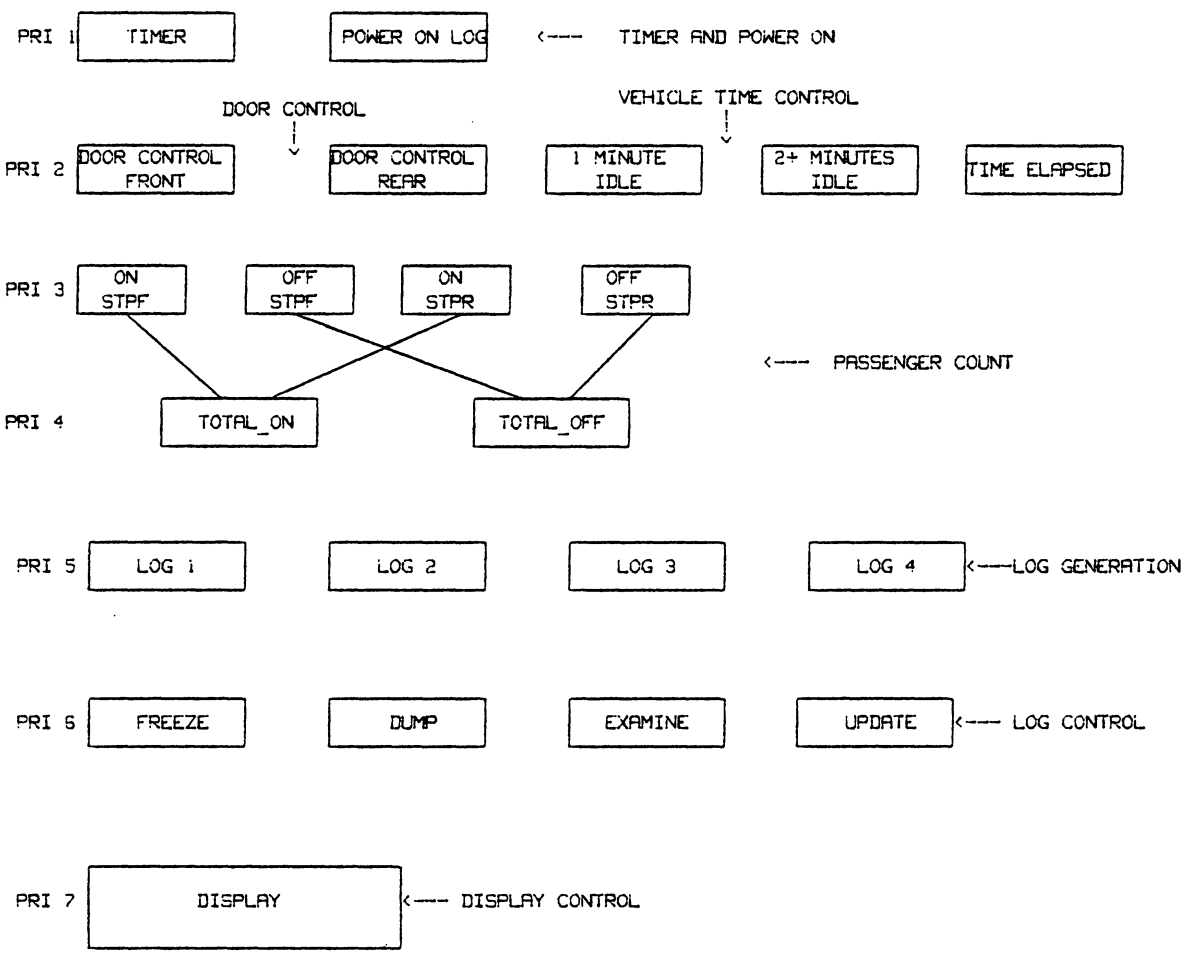


Figure 19. Task level layout of the system.

gers. The problem now has been defined as finding a set of tasks that interact with the sensors.

The door control unit consists of two sets of photocell detectors for each door to detect passengers entering and leaving the vehicle. Let us define two conditions possible on the beams

- break condition - the beam is cut
- make condition - the beam is complete after a break.

If the beams on the door are numbered as shown in Figure 20 on page 82, a sequence for proper entry and exit is

1. Passenger Entry

    - Break - 1
    - Make - 1
    - Break - 2
    - Make - 2

2. Passenger Exit

    - Break - 2
    - Make - 2
    - Break - 1
    - Make - 1

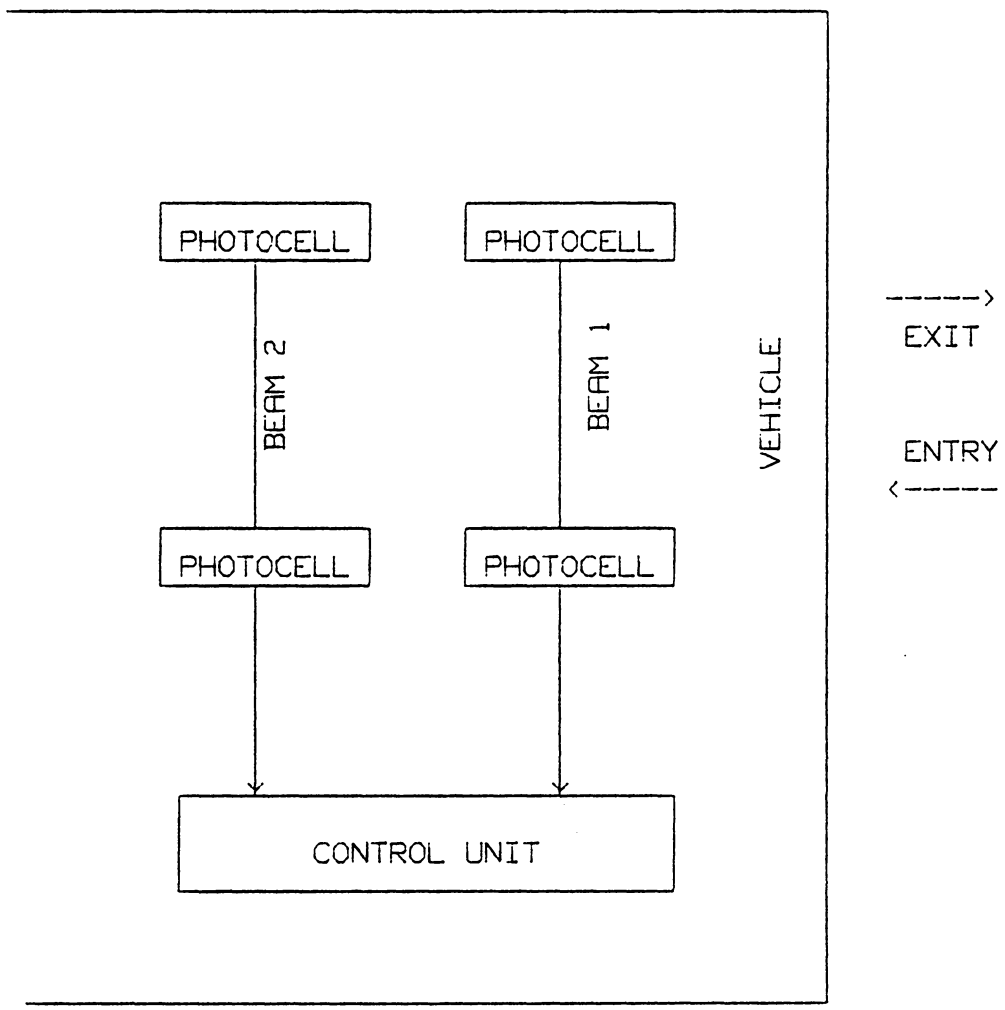


Figure 20. Beam numbering

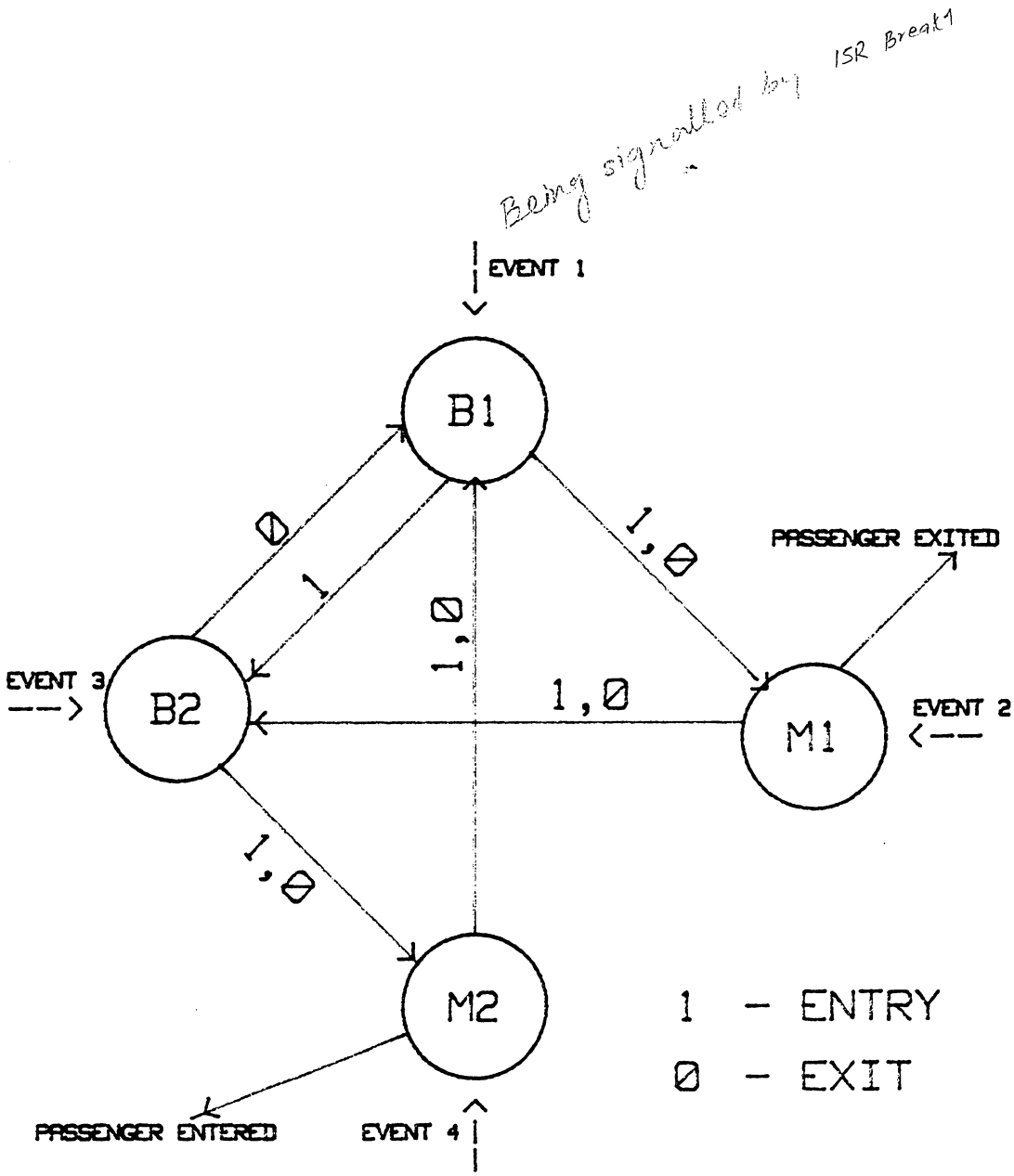


Figure 21. State diagram for the tasks

The make and break conditions can be realized by means of interrupts. When there is a break or make in the beam an interrupt is generated. A task can be defined for each of the break and make conditions. B1, M1, B2, M2 are tasks for the beams. A state diagram can be drawn as shown in Figure 21 on page 83 to check the sequence in which the beams are cut. The initial state for passenger entry is B1, and the initial state for passenger exit is B2. In this state diagram a task executes only if all the conditions represented by incoming arrows are satisfied. This coressponds to the pending conditions on a task being met. In the state diagram of Figure 21 on page 83 arrows into a node correspond to VRTX SC_PEND calls and arrows out of a node correspond to VRTX SC_POST calls.

The different levels in the design of the door control are as shown in the Figure 22 on page 85. Events 1, 2, 3, 4 are events that correspond to making or breaking a beam. Whenever a beam is broken or made an interrupt occurs. The ISR for this interrupt posts messages to events 1, 2, 3 or 4 correspondingly.

The vehicle time control function incorporates the 1 minute idle, 2 minutes or more idle and the number of hours since power up subfunctions. These are connected to the interrupts from the odometer. An odometer generates interrupts every time the vehicle moves a fixed distance. The vehicle is stationary if there are no interrupts. These subfunctions all

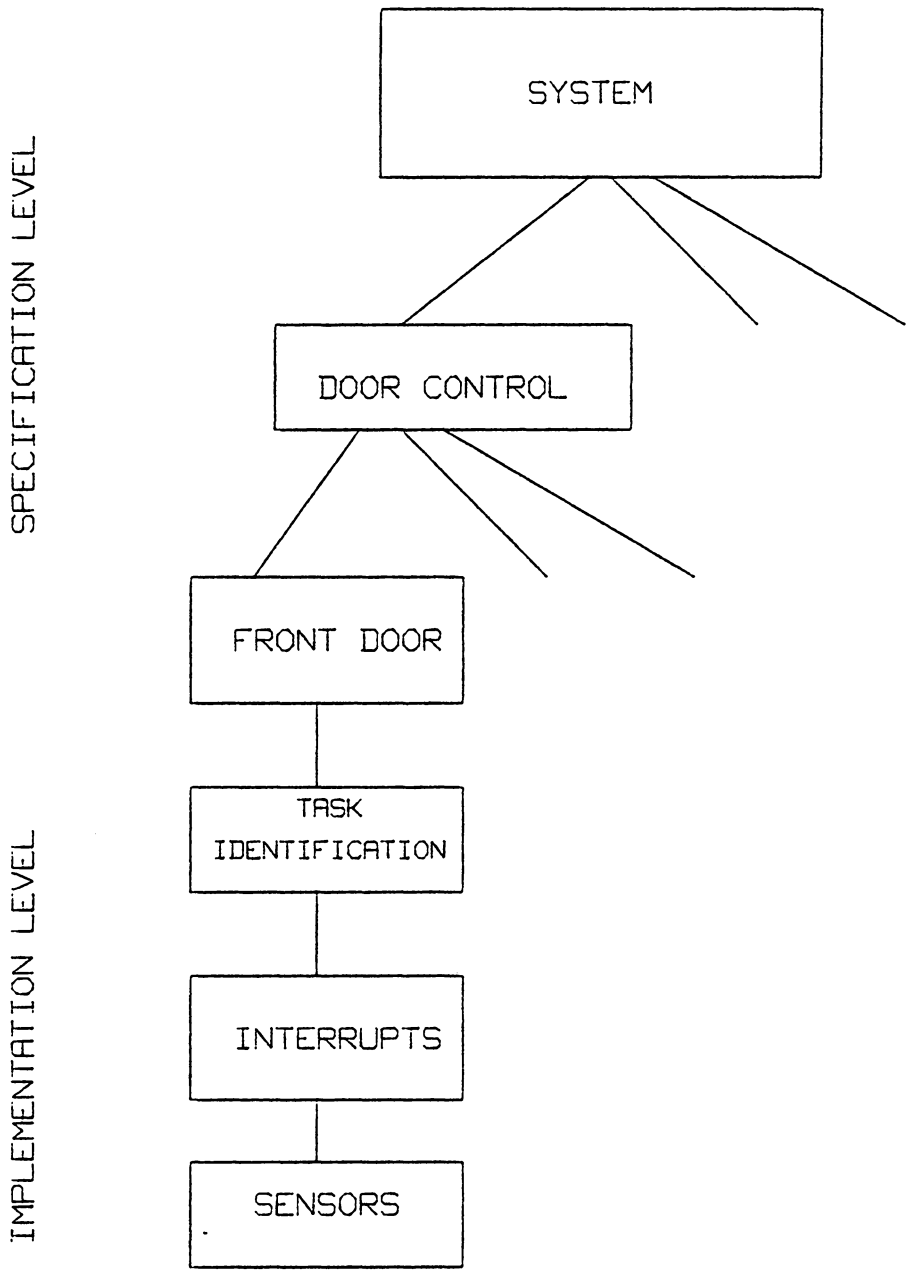


Figure 22. Heirarchy from Sensors to System level.

depend on one input - the odometer impulses, but the processing is different each time. So they are put in different tasks, each of which controls one function.

The passenger counting task can be split into two subfunctions at the first level. One subfunction for counting the number of passengers entering and leaving at every stop and the other for keeping the total count. Subfunction 1 which consists of four independent modules

- Passengers entering through front door
- Passengers exiting through front door
- Passengers entering through rear door
- Passengers exiting through rear door

can be assigned to four different tasks, with each task performing one of the modules. This corresponds to putting subfunctions with different processing for different inputs into different tasks. There are two other tasks that keep a total count of the number of passengers that enter and exit the vehicle since power up. The totalizing tasks are at a lower priority than the tasks that count the passengers at every stop, since it needs to be performed only after a passenger has entered or exited.

The log generation is the data recording section of the system. A log is a system generated message that gives information about various functions being executed. The log

generation function is split into four tasks. Each of the tasks writes a log into memory. Since each of the logs is different and the processing is also different they are assigned to different tasks.

The log control function takes commands from the operator and provides information on the status of the logs. Four commands have been defined to examine the logs. Each of the logs is assigned to a different task since the functions performed by each command is unique. Every command can be defined as subfunction which has different inputs and different processing for the inputs. The processing for each command is done by the coressponding task.

5.3 PSEUDO CODE TO SOLVE THE VEHICLE INSTRUMENTATION PROBLEM

This section of the chapter deals with a possible software solution to the vehicle instrumentation system described. The pseudo code has been written assuming the VRTX operating system to monitor the process. This is however also suitable for use with any operating system that has similar capabilities as VRTX.

5.4 POWER ON

When the system is powered up a type0 log is generated. This indicates when the system has been switched on. The rest of the log generation tasks are at a lower priority than this log. This log has to have a higher priority since it is activated only on powering up the system.

5.4.1 TASK TYPE0_LOG PRI = 1.

The format of this log is

log #, time

The task is activated whenever power is switched on. The activation can take place by generating an interrupt on power up. The ISR posts a message to a mailbox at which this task pends. The variable logptr points to the next free address to which the log can be written.

```
SC_PEND at power_up       /* wait at mailbox power_up */
logptr = start address ;  /* start address of memory */
[logptr] <= log0, time ;  /* write the log */
SC_TDELETE ;              /* delete task after system
                             has been powered up */
```

5.5 TIMER

This task allows the user to set the time and the real-time clock running. The VRTX real-time clock runs off the hardware timer on the computer. The hardware timer generates an interrupt every timer tick and this ISR informs VRTX that a tick has expired.

5.5.1 TIMER ISR

```
UI_ENTER ;          /* Enter ISR using VRTX */
UI_TIMER ;          /* Signal VRTX about timer tick */
UI_EXIT  ;          /* Exit ISR using VRTX */
```

5.5.2 TASK TIMER PRI = 1

```
    SC_STIME ;                /* Set the time*/
T1: SC_TDELAY for 1 sec ;     /* Delay for 1 sec */
    convert to hr:min:sec ;
    goto T1.
```

5.6 ISR FOR ODOMETER

To detect the movement of the vehicle a mechanism needs to be found. A commonly used technique is the generation of

pulses every time the vehicle moves a fixed distance. The pulses generated are connected to an interrupt controller which thus generates interrupts whenever the vehicle moves a fixed distance. The ISR for this keeps track of the distance travelled and other control functions. For example the passenger count log is generated only after the vehicle moves. This can be detected by making this ISR post to a mailbox, at which the log generating task is pended.

```
UI_ENTER ;                      /* enter ISR */
SC_POST to odpul_1 ;            /* post to the mailboxes */
SC_POST to odpul_2 ;
SC_POST to odpul_3 ;
SC_POST to odpul_4 ;
distance = distance + 1 ;       /* increment distance */
UI_EXIT ;                       /* exit ISR */
```

5.7 DOOR CONTROL

The door control section consists of the tasks that interact directly with the interrupts generated from the photocells. There is an interrupt corresponding to a break and make condition on each beam. The door control tasks check the sequence in which the beams are cut and determine whether a

passenger is entering or leaving the vehicle. Any invalid sequence is ignored.

5.7.1 ISR BREAK1

This is the ISR for the break interrupt of beam 1 on the front door. Initially the flags a1 and a2 are set to 0.

- If a1 = 0 and a2 = 0 and this interrupt occurs a passenger is trying to enter.
- If a1 = 0 and a2 = 1 and this interrupt occurs a passenger is leaving.

All other conditions involving a1, a2 and this interrupt are invalid. Any message posted to a mailbox is always nonzero unless specified otherwise.

```
UI_ENTER                    /* enter ISR */
if (a1 = 0 and a2 = 0)      /* is passenger entering */
    then  begin             /* yes ! */
          entry = 1;        /* set entry flag */
          SC_POST to event_1;
          a1 = 1                  /* set a1 */
          end;

    else  begin           /* passenger is not entering */
          if (a1 = 0 and a2 = 1)  /* check if leaving */
```

```
                then begin              /* yes ! */
                     SC_POST to event_1;
                     a2 = 0             /* reset a2 */
                     end;

                else begin              /* false alarm */
                     /* remove messages already posted
                        from mailboxes */
                     SC_POST zero message to b1_b2_on;
                     SC_POST zero message to m1_b2_on;
                     SC_POST to m2_b1_on;
                     a1 = 0
                     end;
           endif;
           end;
    endif;
    UI_EXIT.                            /* exit ISR */
```

5.7.2 ISR MAKE1

This ISR is for the make interrupt of beam 1 of the front door.

```
    UI_ENTER;               /* enter ISR */
    SC_POST to event_2;
    UI_EXIT.                /* exit ISR */
```

5.7.3 ISR BREAK2

This is the ISR for break interrupt of beam 2 of the front door.

- If a1 = 0 and a2 = 0 and this interrupt occurs a passenger is trying to leave
- If a1 = 1 and a2 = 0 and this interrupt occurs a passenger is trying to enter.

All other conditions are invalid.

```
UI_ENTER                        /* enter ISR */
  if (a2 = 0 and a1 = 0)        /* is passenger leaving */
     then  begin                /* yes! */
           exit = 1;            /* set flag */
           SC_POST to event_3;
           a2 = 1               /* set a2 */
           end;

     else  begin                /* passenger is not leaving */
                                /* check if entering */
           if (a2 = 0 and a1 = 1)
              then  begin       /* yes! */
                    SC_POST to event_3;
                    a1 = 0      /* reset a1 */
```

```
                    end;

                else  begin   /* false alarm */
                      /* remove messages from mailboxes */
                      SC_POST zero message to b2_b1_off;
                      SC_POST zero message to m2_b1_off;
                      SC_POST to m1_b2_off;
                      a2 = 0  /* reset a2 */
                      end;
            endif;
            end;
    endif;
    UI_EXIT.                    /* exit ISR */
```

5.7.4 ISR MAKE2

This is the ISR for the make interrupt on beam 2 of the front door.

```
    UI_ENTER;                   /* enter ISR */
    SC_POST to event_4;
    UI_EXIT.                    /* exit ISR */
```

5.8 TASK DESCRIPTION OF DOOR CONTROL

The tasks for the control of the front door have been shown in Figure 21 on page 83. These task can be further split functionally. There are two conditions for task transitions - PON and POFF. The four tasks present can be further split into the tasks corresponding to the PON condition and the POFF condition.

5.8.1 PON CONDITION

The task state diagram for a passenger getting on (PON) the vehicle is as shown in Figure 23 on page 96. The four tasks for the PON condition can be coded as follows. The valid sequence for getting on the vehicle is

- Break - 1
- Make - 1
- Break - 2
- Make _ 2

5.8.2 TASK B1ON PRI = 2.

This is the task for the break condition on beam 1 and activated when a passenger enters the vehicle.

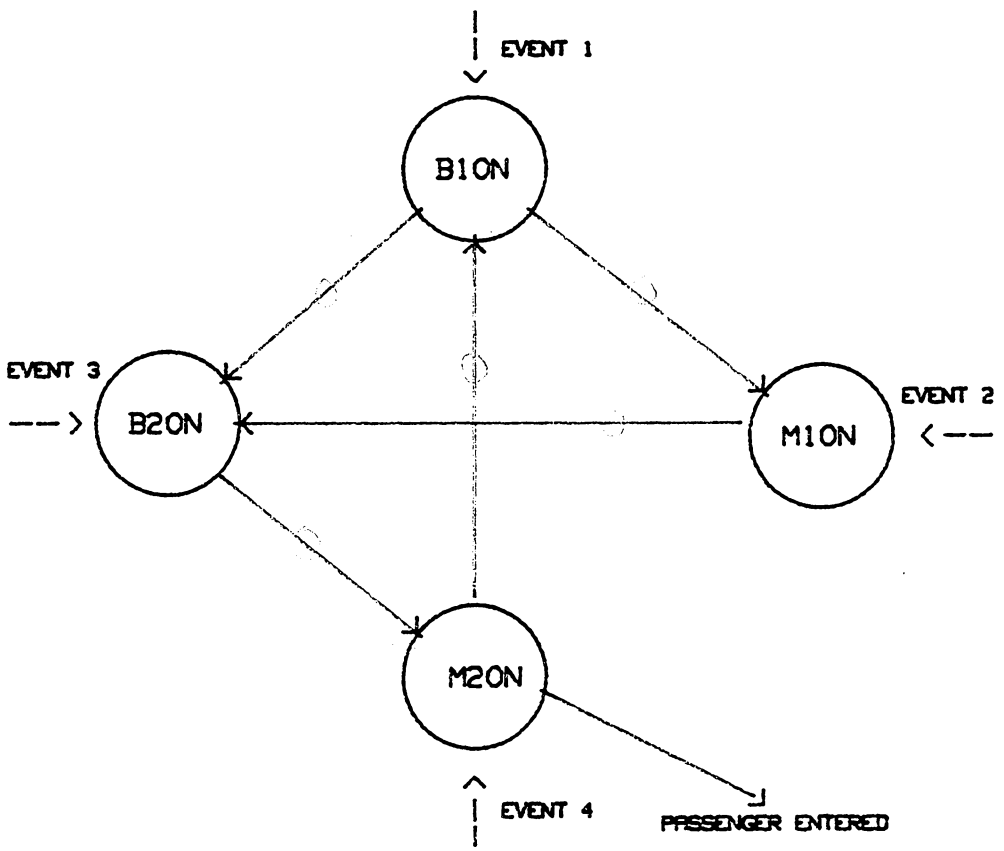


Figure 23. State diagram for PON

```
R1: SC_PEND at event_1;     /* wait for break on beam 1 */
    if entry = 1            /* check beam sequence */
       then  begin          /* correct sequence */
             SC_POST to b1_m1_on ;
             SC_POST to b1_b2_on ;
             SC_PEND at m2_b1_on ; /* wait for beam 2 to
             end ;                    be made */

       else  goto R1 /* beam cut in the wrong sequence */
    endif;
    entry = 0 ;
    goto R1.
```

5.8.3 TASK M1ON PRI = 2.

This tasks handles event_2, which is a make on beam 1.

```
R2: SC_PEND at event_2  ;    /* wait for make on beam 1 */
    SC_PEND at b1_m1_on ;    /* has beam 1 been broken */
    SC_POST to m1_b2_on ;
    goto R2.
```

5.8.4 TASK B2ON PRI = 2.

This task is for a break condition on beam 2.

```
R3: SC_PEND at event_3  ;   /* wait for break on beam 2 */
    if exit = 1 goto R3 ;   /* wrong sequence
                               ignore event */
    SC_PEND at b1_b2_on ;   /* was beam 1 broken ? */
    SC_PEND at m1_b2_on ;    /* was beam 1 made ? */
    SC_POST to b2_m2_on ;
    goto R3.
```

5.8.5 TASK M2ON PRI = 2.

This task checks the make condition on beam 2. It posts a message to increment the passenger count by 1.

```
R4: SC_PEND at event_4  ;    /* wait for make on beam 2 */
    SC_PEND at b2_m2_on ;    /* was beam 2 broken ? */
    SC_POST to m2_b1_on ;
    SC_QPOST to pass_on_f ;  /* passenger got into
                                the vehicle */
    goto R4.
```

5.8.6 POFF CONDITION

The task state diagram of Figure 24 on page 100corresponds to a passenger leaving the vehicle. The valid sequence for a passenger exiting the vehicle is

- Break - 2
- Make - 2
- Break - 1
- Make - 1

5.8.7 TASK B2OFF PRI = 2.

This task corresponds to the break condition on beam 2. It is activated when a passenger leaves the vehicle.

```
R5: SC_PEND at event_3 ;  /* break condition on beam 2 */
    if exit = 1           /* check beam sequence */
       then begin         /* passenger could be leaving */
            SC_POST to b2_m2_off ;
            SC_POST to b2_b1_off ;
            SC_PEND at m1_b2_off ; /* wait for make
                                      on beam 1 */
            end ;

         else goto R5 ;      /* wrong sequence,
```

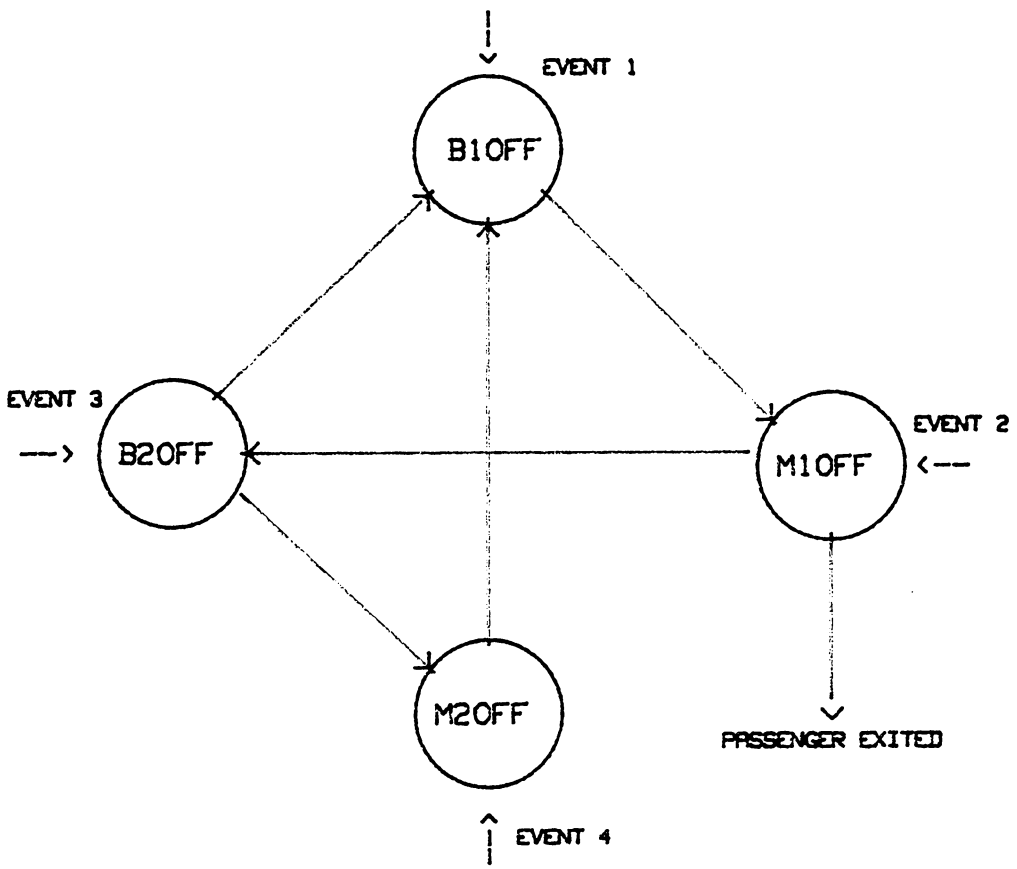


Figure 24. Task state diagram for POFF

```
                                    ignore the event */
    endif ;
    exit = 0 ;
    goto R5.
```

5.8.8 TASK M2OFF PRI = 2.

This task handles the make condition on beam 2. It checks whether the beam is being made after a break in beam 2.

```
R6: SC_PEND at event_4    ;   /* wait for make on beam 2 */
    SC_PEND at b2_m2_off ;   /* was there a break
                                in beam 2 */
    SC_POST at m2_b1_off ;
    goto R6.
```

5.8.9 TASK B1OFF PRI = 2.

This task checks the break condition on beam 1. It checks whether there was a break and make condition on beam 2.

```
R7: SC_PEND at event_1    ; /* wait for break on beam 1 */
    if entry = 1 goto R7 ; /* wrong sequence, so repeat */
    SC_PEND at b2_b1_off ; /* was there break in beam 2 */
```

```
SC_PEND at m2_b1_off ; /* was there make in beam 2 */
SC_POST to b1_m1_off ;
goto R7.
```

5.8.10 TASK M1OFF PRI = 2.

This task checks the make condition on beam 1. It posts a message to a mailbox indicating that a passenger has got off the vehicle.

```
R8: SC_PEND at event_2     ;  /* wait for make on beam 1 */
    SC_PEND at b1_m1_off   ;  /* check the sequence */
    SC_QPOST to pass_off_f ;  /* passenger exited
                                 the vehicle */
    goto R8.
```

5.8.11 REAR DOOR CONTROL

The control for passengers entering and leaving the vehicle through the rear door is similar to the front door. Eight more tasks are needed at the same priority. If passengers do not enter through the rear door - as is the case usually - then the PON condition does not exist. Hence the number of tasks for the rear door is now 4 bringing the total number of tasks for the control of front and rear doors to 12.

5.9 VEHICLE - IDLE TIME CONTROL

This control section generates logs depending on the amount of time the vehicle is idle. There are two tasks which account for the logs. A type 2 log is generated if the vehicle is idle for 1 minute. A type 3 log is generated if the vehicle is idle for 2 or more minutes.

5.9.1 TASK MIN1_IDLE PRI = 2.

This tasks posts to a mailbox on which is pended a type 2 log generating task.

```
X1: SC_PEND at odpul_1 for 1 minute;
    if timeout   /* vehicle has been stationary - 1 min */
       then begin
            SC_POST to log_2   ;
            SC_PEND at odpul_2    /* wait until
                                     vehicle moves */
            end;
    endif;
    goto X1.
```

5.9.2 TASK MIN2_IDLE PRI = 2.

If the vehicle is idle for 2 or more minutes then this task posts a message to log_3. This task also tells the time for which the vehicle was idle.

```
X2: SC_GTIME ;          /* time at which vehicle stops */
    SC_PEND at odpul_3 for 2 minutes ;
    if timeout          /* vehicle was stationary for 2 min */
       then begin
            SC_PEND at odpul_4 ;  /* wait until
                                     vehicle moves */
            SC_GTIME ;            /* time at which
                                     vehicle moves */
            time stationary = finish time - start time ;
            SC_POST to log_3
            end ;
    endif ;
    goto X2.
```

5.9.3 TASK HOUR_GONE PRI = 2.

This task indicates the number of hours since the system has been up.

```
X3: SC_DELAY for 60 minutes ; /* wait for an hour */
    hour = hour + 1 ;        /* count of number of hours */
    SC_POST to log_4;
    goto X3.
```

5.10 PASSENGER COUNTING TASKS

These tasks are tasks at priority 3 and count the number of passengers entering and leaving the vehicle every time it stops. The number of passengers entering and exiting the vehicle through each of the doors are counted.

5.10.1 TASK ON_STPF PRI = 3.

This task counts the number passengers entering the vehicle at every stop through the front door. Variable pass_oncnt_stpf gives a count of passengers entering the vehicle through the front door.

```
X4: SC_QPEND at pass_on_f ;  /* wait for passenger to get
                                in through front door */
    pass_oncnt_stpf = pass_oncnt_stpf + 1 ;
    if (door closed and vehicle moves) ;
       then begin
            v1 = pass_oncnt_stpf ;/* store the count in a
```

```
                                    temporary variable */
            pass_oncnt_stpf = 0 ; /* reset count to 0 */
            SC_POST to total_on   /* enable totalizing
                                    task */
            end;
      endif;
      goto X4.
```

5.10.2 TASK ON_STPR PRI = 3.

The number of passengers entering the vehicle through the rear door at every stop are counted by this task. Variable pass_oncnt_stpr keeps the count of the number of passengers entering through the rear door.

```
  X5: SC_QPEND at pass_on _r ;   /* wait for passenger
                                    through the rear door */
      pass_oncnt_stpr = pass_oncnt_stpr + 1 ;
      if (door closed and vehicle moves) ;
         then begin
              v3 = pass_oncnt_r ; /* store count in a
                                     temporary variable */
              pass_oncnt_r = 0 ;
              SC_POST to total_on
              end;
```

```
      endif;
      goto X5.
```

5.10.3 TASK OFF_STPF PRI = 3.

This task keeps a count of the number of passengers exiting the vehicle at every stop. The count is maintained in the variable pass_offcnt_stpf.

```
  X6: SC_QPEND at pass_off_f ; /* wait for passenger to exit
                                  through the front door */
      pass_offcnt_stpf = pass_offcnt_stpf + 1 ;
      if (door closed and vehicle moves)
         then begin
              v2 = pass_oncnt_stpf ;/* store count in a
                                       temporary variable */
              pass_oncnt_stpf = 0  ;     /* reset count */
              SC_POST to total_off
              end ;
      endif;
      goto X6.
```

5.10.4 TASK OFF_STPR PRI = 3.

The number of passengers exiting the vehicle through the rear door are accounted for by this task. The variable pass_offcnt_stpr maintains the count.

```
X7: SC_QPEND at pass_off_r ;  /* wait for passenger
                                 to exit */
    pass_offcnt_stpr = pass_off_stpr + 1 ;
    if (door closed and vehicle moves)
       then begin
            v4 = pass_offcnt_stpr ; /* store count */
            pass_offcnt_stpr = 0 ;  /* reset count */
            SC_POST to total_off
            end ;
    endif;
    goto X7.
```

5.11 PASSENGER TOTALIZING

There are two tasks in this category. One task counts the total number of passengers entering the vehicle and the other the total number of passengers leaving the vehicle. These two tasks enable the tasks that posts a type_1 log.

5.11.1 TASK TOTAL_COUNT_ON PRI = 4.

This task counts the total number of passengers who used the vehicle. The count is maintained in the variable passg_on. This task is enabled only when the vehicle moves after a stop and if passengers enter the vehicle.

```
X8: SC_PEND at total_on ;  /* wait until the vehicle moves
                              after a stop */
    passg_on = v1 + v3 ;   /* total count of passengers */
    SC_POST to log_1 ;
    goto X8.
```

5.11.2 TASK TOTAL_COUNT_OFF PRI = 4

The total number of passengers that exited the vehicle are checked by this task. This count is stored in the variable pass_off.

```
X9: SC_PEND at total_off ;   /* wait until vehicle moves
                                after a stop */
    passg_off = v2 + v4 ;    /* total count of passengers
                                leaving */
    SC_POST to log_1 ;
    goto X9.
```

5.12 LOG GENERATION

The log generation tasks generate the logs required. There are logs for power on, passenger count and vehicle time control. The log generation is controlled by a log pointer (logptr) which points to the address where the log is to be written. After every log has been written the pointer is incremented by a value depending on the log type. For example a type1_log occupies 4 bytes and a type3_log occupies 6 bytes.

The variable [logptr]n writes to n locations starting from address pointed to by logptr and increments the pointer to point to the location where the next log has to be written. For example in type1_log [logptr]4 indicates that the log occupies 4 bytes and is written starting from [logptr].

5.12.1 TASK TYPE1_LOG PRI = 5

This is the task that generates the passenger count log. The format of this log is

log #, passg_on, passg_off, stop #

The log is 4 bytes long with each parameter occupying one byte.

```
X10: SC_PEND at log_1 ;  /* wait until enabled
                            to generate log */
```

```
SC_LOCK ;       /* disable scheduling until
                   log is written */
                /* write the log */
[logptr]4 <= log1, passg_on, passg_off, stop # ;

SC_UNLOCK ;     /* enable scheduling */
goto X10.
```

5.12.2 TASK TYPE2_LOG PRI = 5.

A type 2 log, which indicates that the vehicle has been stationary for 1 minute, is generated by this task. The format of this task is

log #, time

where time indicates the time at which it is stationary. This log occupies 4 bytes, 1 byte for the log number and three bytes for the time.

```
X11: SC_PEND at log_2 ;  /* wait until log
                            has to be written */
     SC_LOCK ;           /* disable scheduling */
     SC_GTIME ;          /* get the time */
     [logptr]4 <= log2, time ;  /* write the log */
     SC_UNLOCK ;         /* enable scheduling */
     goto X11.
```

5.12.3 TASK TYPE3_LOG PRI = 5.

This task generates a log if the vehicle is stationary for two or more minutes. The format of this log is

log #, time, time stationary

The log is 6 bytes long with 1 byte for the log number, 3 bytes for the time and 2 bytes for the time stationary.

```
X12: SC_PEND at log_3 ;     /* wait until vehicle is
                               stationary for 2 minutes */
     SC_LOCK ;              /* disable scheduling */
     SC_GTIME ;             /* get the time */
                            /* write the log */
     [logptr]6 <= log3, time, time stationary ;
     SC_UNLOCK ;            /* enable scheduling */
     goto X12.
```

5.12.4 TASK TYPE4_LOG PRI = 5.

This log indicates an hour overflow or the number of hours that have elapsed since the system has been powered on. The format of this log is

log #, hour

where hour indicates the number of hours elapsed.

```
X13: SC_PEND at log_4 ;  /* wait for 1 hour */
     SC_LOCK ;           /* disable scheduling */
     [logptr]2 <= log4, hour;   /* write the log */
     SC_UNLOCK ;         /* disable scheduling */
     goto X13.
```

5.13 LOG CONTROL

This set of tasks control the generation of logs by the user. These tasks control the interaction of the user with the system. The user can enter commands from the keyboard for the following operations.

- Freeze - stops log generation at specified time.
- Dump - display all the logs beginning at address 1 upto address 2.
- Examine - examine a particular log.
- Update - display the last 100 logs.

5.13.1 TASK FREEZE PRI = 6.

This tasks stops log generation at the time specified by the user. This task is activated when the user enters the command

f < ftime >

where ftime is the user defined time at which all log generation stops. After this command is given the log pointer is reset and if necessary log generation starts again.

```
X14: SC_WAITC for 'f < ftime >'; /* wait for user to input
                                    the command & time */
     SC_TDELAY until ftime ;      /* wait until user
                                     specified time */
     SC_TSUSPEND all tasks at pri = 5 ;  /* stop all log
                              generation tasks at pri 5 */
     SC_POST to disp ;            /* enable display */
     temp_0 = logptr ;            /* save pointer */
     logptr = 0 ;                 /* reset log pointer */
     SC_WAITC for 'continue' ;    /* wait for user to
                                     continue again */
     /* resume all log generation tasks at pri 5 */
     SC_TRESUME all tasks at pri = 5 ;

     goto X14.
```

5.13.2 TASK DUMP PRI = 6.

This task reads in the starting and end address of the logs to be displayed. The format of the command to invoke this task is

d < addr1 >, < addr2 >

The parameters < addr1 > and < addr2 > are the start and end addresses respectively.

```
X15: SC_WAITC for 'd < addr1 >, < addr2 >' ; /* wait for
                                                command */
     temp_1 = addr1  ;      /* store the addresses in
                               temporary variables */
     temp_2 = addr2  ;
     SC_POST to disp ;      /* enable display */
     goto X15.
```

5.13.3 TASK EXAMINE PRI = 6.

Any single log can be examined by an user defined command. The format of the command to examine a log is

e < addr >

This command activates the task and lets the user examine any byte.

```
X16: SC_WAITC for 'e < addr >' ;  /* wait for command */
     temp_3 = addr ;               /* store address */
     SC_POST to disp ;            /* enable display task */
     goto X16.
```

5.13.4 TASK UPDATE PRI = 6.

The user can obtain the status of the last 100 logs using this command. The last 100 logs are displayed whenever this command is given. The format of this command is

u

```
X17: SC_WAITC for 'u' ;      /* wait for command */
     SC_POST to disp  ;      /* enable display */
     goto X17.
```

5.14 DISPLAY CONTROL

The display control section consists of a task which controls the displaying of logs and messages. All the logs are displayed only through this task. Since this task makes use of I/O devices it has the least priority.

5.14.1 TASK DISPLAY PRI = 7

This task controls the display of the logs when commands like the log control commands are given. A case statement branches control to the appropriate section depending on the command.

```
X18: SC_PEND at disp ;  /* wait until command complete */
     do case command of
          f:  begin     /* display message indicating
                           freeze command is complete */
              display command complete message ;
              display log status
              end ;

          d:  begin        /* command is dump */
              /* scrptr_d points to address
                 to be displayed */
              scrptr_d = temp_1 ;
              do while scrptr_d < temp_2
                    begin    /* repeat until end address */
                    display [scrptr_d] ;
                    scrptr_d = scrptr_d + 1
                    end;
              end ;

          e:  begin         /* command is examine */
```

```
                /* scrptr_e points to address
                   of log to be displayed.
                scrptr_e = temp_3 ;
                display [scrptr_e] ;
                end ;

        u:      begin           /* command is update */
                /* display last 100 logs */
                display 100 logs ;
                display command complete message ;
                end ;
    end case;
    goto X18.
```

This section has shown the development of a typical application using multitasking. As can be seen the code for this system is highly structured and using VRTX can be written in a high level language like C.

6.0 CONCLUSION

A multitasking executive called VRTX/86 was successfully interfaced with the IBM PC and the IBM PC was converted into a multitasking environment. A test program that executed successfully on the IBM PC was written. In this case both the development machine and the target machine were the same. Some basic rules for defining tasks and a general methodology for splitting a system into tasks and assigning priorities to the tasks was discussed. The advantage of using multitasking for real-time situations was also discussed.

Real-time systems relate to multitasking very easily and are easy to develop using multitasking. This has been shown for a typical real-time application called a Vehicle Instrumentation System. The methodology developed was used in designing the system. The system was designed with respect to the VRTX multitasking executive.

The VRTX executive has been studied and found to feature calls that reduce dependencies on any particular board. This executive is supplied in PROMs and can be transferred into user PROMs of any speed. VRTX has been found not to depend any specific board environment, it was easily interfaced with the IBM PC system board.

With a number of multitasking kernels, which are independent of any specific boards, available from commercial

vendors, real-time systems can be developed much more easily than was possible. Using VRTX on the IBM PC it is possible to convert the PC into a multitasking development system.

APPENDIX A. CONNECTING VRTX TO THE IBM PC

This appendix describes how VRTX is connected to the IBM PC. It also gives details about changing the position of VRTX in the address space of the processor and adding any new silicon software components.

A.1 CONNECTING VRTX

VRTX is supplied in two 2716 EPROMS from Hunter and Ready. These two EPROMs are merged into a single EPROM so that it can be attached to the IBM PC. To install any extra EPROMs on the IBM PC an extension board is needed. A Tecmar PC-Mate RAM/ROM board capable of taking in 2716, 2732 or 2764 type EPROMs was used as an extension board. This board was plugged into the expansion slot available on the IBM PC.

This extension board has 16 sockets to install the EPROMs. Jumpers are provided to select between the appropriate types [21]. Three switches are provided for address decoding on the board.

VRTX was transferred from the two 2716 EPROMs into a 2764 - 8K EPROM. The VRTX is placed starting from the physical address 40000H. The switch settings for this address are as follows.

Switch No.	Position	Value
* SW1	1	0
* SW1	2	1
* SW1	3	0
* SW1	4	1
* SW1	5	1
* SW1	6	1
* SW1	7	1
* SW1	8	1
* SW2	1	1
* SW2	2	0
* SW2	3	1
* SW2	4	1
* SW2	5	1
* SW2	6	1
* SW2	7	1
* SW2	8	0

To vector into VRTX, INT 60H has been used. This corresponds to a vector start address of 60H*4 = 180H. The interrupt vector for the configuration table is 61H. These are the first interrupts on the PC that are available to the user and they have been used. The vector start address for the configuration table, which is 61H*4 = 0:184H, is written start-

ing at offset 22H in the VRTX EPROM. The vector address for the configuration table has to be changed starting at offset 22H in the VRTX PROM if a different interrupt vector is used for the configuration table. The complete interconnection with all the addresses is shown in Figure 25 on page 124.

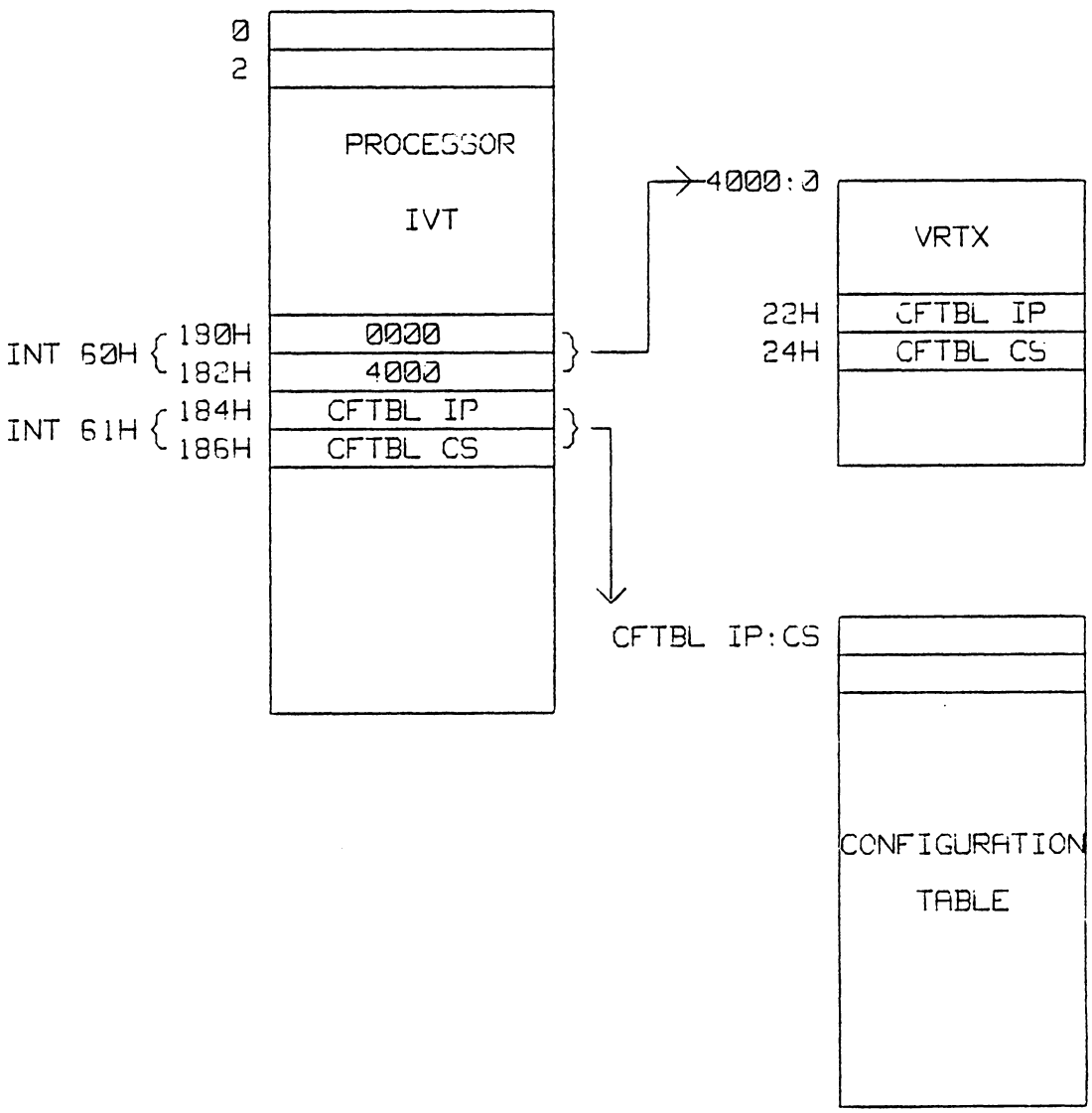


Figure 25. Interconnection between IVT, CT and VRTX.

APPENDIX B. A MULTITASKING PROGRAM ON THE IBM PC

This appendix gives a listing of the multitasking program described in chapter 4. A brief description of the program is given in this appendix.

There are five tasks called

- Main
- Time
- Waita
- Msg_2
- Msg_1

which accomplish the execution of the program. The program performs the following operations. There are two messages that are displayed on the screen. Message #1 is displayed at 0.5 second intervals and message #2 at 3 second intervals. The time is displayed on the lower right hand corner of the screen. Whenever the character 'A' is input from the keyboard, the displaying of messages stops for 1 second and the user is asked to input a character. If no character is input within 1 second the displaying of messages resumes. The user can however still input a character after the messages resume. A character is not recognized if a key is pressed without hitting the character 'A' first. After one character

is displayed, the next key is recognized only after the character 'A' is hit a again.

B.1 PROGRAM ELEMENTS

The program is organized into four segments. There are two data segments and two code segments. The CFTBL segment is a data segment and describes the configuration table of VRTX. The segment TEST_DATA is the data segment for the program, mailboxes and messages. The segment TEST is the code segment for all tasks and the other tasks. The INITIALIZE segment creates the task MAIN, initializes VRTX, sets up interrupt pointers and starts multitasking using the VRTX_GO command.

The VRTX workspace starts at address 2000H. The workspace address can be changed to any other location by changing the corresponding entry in the configuration table. The ROM BIOS routine [19] for the keyboard is not used. Instead a user written routine for the keyboard, which performs the same functions as the original routine and in addition gives the user control of any character that is input, is used. The character set supported by the user written routine is the upper case alphabets and numerals. The function keys, special keys like control, alternate, etc., are not recognized. The video routine using INT 10H in the ROM BIOS is used to control the displaying of characters on the screen and setting the cursor position to display the time.

The software interrupt 1CH is used for the timer. The address of a user timer interrupt service routine is loaded starting from 1C*4 = 70H. The timer interrupts the processor at the rate of 18.2 interrupts per second giving a clock resolution of approximately 55 milliseconds.

B.2 EXECUTING THE PROGRAM

This program can be run on the IBM PC by using the Microsoft assembler and linker. The procedure is as follows.

- Set the default drive to b by typing in b: < cr >
- Place source diskette in the b drive and assembler diskette in the 'a' drive.
- Assemble the source file using the command a:masm < fn >;
- Link the file using the command a:link < fn >;
- To execute the program type < fn >

B.3 PROGRAM LISTING

A listing of the program described is shown in the following pages.

```
Microsoft MACRO Assembler  Version 3.00              PAGE1-1
    07-06-85
Page ,132
;*******************************************************************************
;
;       A MULTITASKING PROGRAM USING VRTX ON THE IBM PC
;       AUTHOR : SRINIVAS REDDY
;       SYSTEM : IBM PC
;
;       THIS PROGRAM DEMONSTRATES THE MULTITASKING CAPABILITIES
;       OF VRTX ON THE IBM PC. THERE ARE 4 TASKS DEVELOPED BY THE
;       USER. THERE ARE TWO MESSAGES THAT ARE DISPLAYED
;       ON THE SCREEN. MESSAGE #1 IS DISPLAYED AT 0.5 SEC INTERVALS
;       AND MESSAGE #2 AT 3 SEC INTERVALS. THE TIME IS DISPLAYED
;       ON THE LOWER RIGHT HAND CORNER OF THE SCREEN. WHENEVER
;       THE CHARACTER 'A' IS HIT ON THE KEYBOARD THE DISPLAYING
;       OF MESSAGES STOPS AND THE PROGRAM ASKS THE USER TO
;       INPUT A CHARACTER. THE PROGRAM WAITS FOR 1 SECOND FOR THE
;       USER TO INPUT A CHARACTER. IF NO CHARACTER IS INPUT
;       WITHIN 1 SECOND THE ORGINAL MESSAGES START DISPLAYING
;       AGAIN. IF ANY KEY IS HIT, AFTER THE KEY 'A' IS PRESSED
;       THE CHARACTER IS DISPLAYED. THE CHARACTER IS NOT
;       RECOGNIZED IF A KEY IS HIT WITHOUT PRESSING 'A'.
;       AFTER ONE CHARACTER IS DISPLAYED, THE NEXT KEY IS
;       RECOGNIZED ONLY AFTER THE CHARACTER 'A' IS HIT AGAIN.
;
;       THE SCREEN IS BLANKED OUT AFTER IT IS FULL AND THE
;       MESSAGES START DISPLAYING FROM THE TOP OF THE
;       SCREEN.
;
;       TASK0 - TASK 'MAIN' CREATES ALL THE OTHER TASKS AND
;       DELETES ITSELF AFTER IT HAS CREATED THE OTHER TASKS.
;
;       TASK1 - TASK 'TIME' AT PRIORITY 1.
;       THIS TASK KEEPS TRACK OF THE REAL-TIME AND DISPLAYS
;       IT ON THE LOWER RIGHT HAND CORNER OF THE SCREEN
;       IN THE FORM HRS:MIN:SEC. THIS TASK ALSO BLANKS THE
;       SCREEN ONCE IT IS FULL AND RESETS THE CURSOR SO
;       THAT THE MESSAGES ARE DISPLAYED STARTING FROM THE TOP.
;
;       TASK2 - TASK 'WAITA' AT PRIORITY 2
;       THIS TASK WAITS FOR THE CHARACTER 'A' TO BE INPUT.
;       ONCE THE CHARACTER 'A' IS INPUT THIS TASK DISPLAYS
;       A MESSAGE ASKING THE USER TO INPUT A CHARACTER.
;       THE USER HAS 1 SEC TO INPUT A CHARACTER.
;
;       TASK3 - TASK 'MSG_2' AT PRIORITY 3
;       THIS TASK DISPLAYS A MESSAGE ON THE SCREEN EVERY 3 SECS.
;
;       TASK4 - TASK 'MSG_1' AT PRIORITY 4
;       THIS TASK DISPLAYS A MESSAGE EVERY 0.5 SEC.
;
;*******************************************************************************
```

```
Microsoft MACRO Assembler  Version 3.00              PAGE1-2
     07-06-85
    ;*************************************************************************
    ;       CONFIGURATION TABLE
    ;


0000        CFTBL SEGMENT 'DATA'
    ;
                PUBLIC TBL
0000      TBL   LABEL BYTE
    ;
0000  2000            DW  2000H             ; VRTX WORKSPACE START ADDRESS
0002  0100            DW  0100H             ; VRTX WORKSPACE SIZE
0004  0000            DW  0                 ; RESERVED MUST BE 0
0006  0000            DW  0                 ; NO INTERRUPT STACK
0008  0000            DW  0                 ; RESERVED MUST BE 0
000A  0000            DW  0                 ; RESERVED MUST BE 0
000C  0000            DW  0                 ; RESERVED MUST BE 0
000E  0010            DW  0010H             ; USER STACK SIZE - PARAS.
0010  00 00 00 00     DD  0                 ; RESERVED MUST BE 0
0014  0005            DW  05H               ; NUMBER OF TASKS
0016  0000            DW  0                 ; RESERVED MUST BE 0
0018  00 00 00 00     DD  0                 ; NO TXRDY DRIVER ROUTINE
001C  00 00 00 00     DD  0                 ; NO TCREATE ROUTINE
0020  00 00 00 00     DD  0                 ; NO TDELETE ROUTINE
0024  00 00 00 00     DD  0                 ; NO TSWITCH ROUTINE
0028  00 00 00 00     DD  0                 ; NO COMPONENT VECTOR ROUTINE
    ;
002C  56 52 54 58 20 49        MSG DB 'VRTX INITIALIZATION ERROR$'
      4E 49 54 49 41 4C
      49 5A 41 54 49 4F
      4E 20 45 52 52 4F
      52 24
    ;
0046      CFTBL ENDS
    ;*************************************************************************
    ;
    ;               DATA FOR PROGRAM, MAILBOXES ,MESSAGES
    ;
0000        TEST_DATA  SEGMENT  'DATA'
    ;
0000  01 00 00 00               SCREEN   DD  1      ; MAILBOX FOR LOCKING SCREEN
0004  54 48 49 53 20 49         MSG1     DB  'THIS IS THE START OF A TEST PROGRAM$'
      53 20 54 48 45 20
      53 54 41 52 54 20
      4F 46 20 41 20 54
      45 53 54 20 50 52
      4F 47 52 41 4D 24
0028  54 48 49 53 20 49         MSG2     DB  'THIS IS MESSAGE #1 AT PRI 4$'
      53 20 4D 45 53 53
      41 47 45 20 23 31
      20 41 54 20 50 52
      49 20 34 24
0044  49 4E 54 45 52 52         MSG3     DB  'INTERRUPTING MESSAGE #1 AT PRI 3$'
      55 50 54 49 4E 47
      20 4D 45 53 53 41
      47 45 20 23 31 20
      41 54 20 50 52 49
      20 33 24
```

```
0065  41 57 41 49 54 49          MSG4    DB  'AWAITING CONSOLE INPUT$'
      41 47 20 43 4F 4E
      53 4F 4C 45 20 49
      4E 50 55 54 24
007C  3D 3E 24                    MSG5    DB  '=>$'
                ;
007F  00 00 00 00                 K1      DD  0       ; MAILBOX FOR RECEIVING CHARACTER
0083      0A                      BUFFER  DB  10 DUP(0)  ; BUFFERSIZE FOR KEY BOARD
                00

008D  0000                        BUFPTR1 DW  0           ; POINTER TO START OF BUFFER
008F  0000                        BUFPTR2 DW  0           ; POINTER TO END OF BUFFER
                ;
                ;               LOOK UP TABLE FOR KEYBOARD
                ;
0091  00 1B 31 32 33 34                 SCANTABLE DB 0,1BH,'1234567890-=',8,0
      35 36 37 38 39 30
      2D 3D 08 00
00A1  51 57 45 52 54 59                   DB 'QWERTYUIOP[]',0DH,0
      55 49 4F 50 5B 5D
      0D 00
00AF  41 53 44 46 47 48                   DB 'ASDFGHJKL;',0,0,0,0
      4A 4B 4C 3B 00 00
      00 00
00BD  5A 58 43 56 42 4E                   DB 'ZXCVBNM,./',0,0,0
      4D 2C 2E 2F 00 00
      00
00CA  20 00 00 00 00 00                   DB ' ',0,0,0,0,0,0,0,0,0,0,0,0,0,0
      00 00 00 00 00 00
      00 00
00D8  37 38 39 20 34 35                   DB '789 456 1230'
      36 20 31 32 33 30
                ;
00E4  5B 10 00 10           BOOT      DD  0F000E05BH; RESET VECTOR FOR SYSTEM BOOT
                ;
00E8  30                    HR_HIG    DB  30H       ; HIGH AND LOW ORDER DIGITS
00E9  30                    HR_LOW    DB  30H       ; FOR HOUR.
00EA  3A                    COL1      DB  3AH       ; ASCII FOR COLON.
00EB  30                    MIN_HIG   DB  30H       ; HIGH AND LOW ORDER DIGITS
00EC  30                    MIN_LOW   DB  30H       ; FOR MINUTES.
00ED  3A                    COL2      DB  3AH       ; ASCII FOR COLON.
00EE  30                    SEC_HIG   DB  30H       ; HIGH AND LOW ORDER DIGITS
00EF  30                    SEC_LOW   DB  30H       ; FOR SECONDS.
                ;
00F0          TEST_DATA ENDS
                ;
                ;*****************************************************************************
                ;
                ;
                ;       THIS SEGMENT IS THE CODE SEGMENT FOR THE TASKS AND THE
                ;       REST OF THE PROGRAM.
                ;
```

```
Microsoft MACRO Assembler  Version 3.00              Page1-4
      07-06-85
0000                TEST SEGMENT 'CODE'

                    ASSUME CS:TEST,DS:TEST_DATA
                ;
                ;*****************************************************************************
                ;
                ;       SUBROUTINE DISP DISPLAYS A STRING USING THE DOS FUNCTION
                ;       CALL NO. 9
                ;
                ;       INPUT  : DX POINTS TO THE STRING
                ;       OUTPUT : MESSAGE IS DISPLAYED
                ;       DESTROYS AX
                ;
                                PUBLIC  DISP
0000                    DISP    PROC    NEAR
0000  B4 09                     MOV     AH,9
0002  CD 21                     INT     21H
0004  C3                       RET
0005                    DISP    ENDP
                ;
                ;*****************************************************************************
                ;
                ;       SUBROUTINE CRLF GIVES A CARRIAGE RETURN AND LINE FEED
                ;
                ;       INPUT  : NOTHING
                ;       OUTPUT : CR, LF
                ;       DESTROYS AX, DL
                ;
                                PUBLIC  CRLF
0005                    CRLF    PROC    NEAR
0005  B2 0A                     MOV     DL,0AH
0007  B4 02                     MOV     AH,2
0009  CD 21                     INT     21H
000B  B2 0D                     MOV     DL,0DH
000D  CD 21                     INT     21H
000F  C3                       RET
0010                    CRLF    ENDP
                ;
                ;*****************************************************************************
                ;
                ;       THIS PROCEDURE IS THE CODE FOR THE MAIN TASK. THIS
                ;       PROCEDURE IS EXECUTED WHEN THE VRTX_GO CALL IS GIVEN.
                ;       THIS TASK BLANKS THE SCREEN AND DISPLAYS A SIGN ON
                ;       MESSAGE. IT THEN CREATES THE REST OF THE TASKS AND DELETES
                ;       ITSELF.
                ;
                ;             CODE FOR TASK MAIN
                ;
                                PUBLIC  MAIN
0010                    MAIN    PROC    FAR
0010  B8 ---- R                MOV     AX,TEST_DATA    ; SET UP CORRECT DATA SEGMENT
0013  8E D8                     MOV     DS,AX           ;
                ;
                ;             CLEAR THE SCREEN
                ;
0015  B9 07D0                   MOV     CX,2000         ; NO. OF CHARACTER POSITIONS
```

```
                                                    ; TO BE FILLED WITH BLANKS.
0018  B7 00           MOV     BH,0                  ; ACTIVE PAGE
001A  B0 20           MOV     AL,' '                ; ASCII FOR BLANK
001C  B4 0A           MOV     AH,10                 ; FUNCTION CALL NO.
001E  CD 10           INT     10H                   ; BIOS VIDEO INT.
    ;
    ;               DISPLAY SIGN ON MESSAGE
    ;
0020  BA 0004 R       MOV     DX,OFFSET MSG1 ; POINT TO MESSAGE1
0023  E8 0000 R       CALL    DISP           ; DISPLAY IT
0026  E8 0005 R       CALL    CRLF           ; GIVEN A CR, LF
    ;
    ;               DELAY FOR 1 SEC BEFORE CREATING THE TASKS
    ;
0029  B9 0012         MOV     CX,18          ; NO. OF TIMER TICKS
002C  BA 0000         MOV     DX,0           ;
002F  B8 000C         MOV     AX,0CH         ; FUNCTION CALL FOR SC_TDELAY
0032  CD 60           INT     VRTX           ; USING VRTX
    ;
    ;               CREATE TASK TO PRINT "THIS IS TEST MESSAGE #1"
    ;
0034  BB 0080 R       MOV     BX,OFFSET MSG_1    ; OFFSET OF TASK
0037  B8 ---- R       MOV     AX,SEG MSG_1       ; SEG OF TASK
003A  8E C0           MOV     ES,AX              ; INTO ES
003C  B1 01           MOV     CL,1               ; ID = 1
003E  B5 04           MOV     CH,4               ; PRI = 4
0040  B8 0000         MOV     AX,0               ; SC_TCREATE
0043  CD 60           INT     VRTX               ; USING VRTX
    ;
    ;               CREATE TASK TO PRINT "THIS IS TEST MESSAGE #2"
    ;
0045  BB 0086 R       MOV     BX,OFFSET MSG_2    ; OFFSET OF TASK
0048  B8 ---- R       MOV     AX,SEG MSG_2       ; SEG OF TASK
004B  8E C0           MOV     ES,AX              ; INTO ES
004D  B1 02           MOV     CL,2               ; ID = 2
004F  B5 03           MOV     CH,3               ; PRI = 3
0051  B8 0000         MOV     AX,0               ; SC_TCREATE
0054  CD 60           INT     VRTX               ;
    ;
    ;               CREATE TASK TO WAIT FOR CHARACTER 'A'
    ;
0056  BB 001C R       MOV     BX,OFFSET WAITA    ; OFFSET OF TASK
0059  B8 ---- R       MOV     AX,SEG WAITA       ; SEG OF TASK
005C  8E C0           MOV     ES,AX              ; INTO ES
005E  B1 03           MOV     CL,3               ; ID = 3
0060  B5 02           MOV     CH,2               ; PRI =2
0062  B8 0000         MOV     AX,0               ; SC_TCREATE
0065  CD 60           INT     VRTX               ;
    ;
    ;               CREATE TASK TO  HANDLE TIME REAL-TIME
    ;
0067  BB 0159 R       MOV     BX,OFFSET TIME     ; OFFSET OF TASK
006A  B8 ---- R       MOV     AX,SEG TIME        ; SEG OF TASK
006D  8E C0           MOV     ES,AX              ; INTO ES
```

```
Microsoft MACRO Assembler  Version 3.00              Page1-6
      07-06-85


006F  B1 04             MOV       CL,4                ; ID = 4
0071  B5 01             MOV       CH,1                ; PRI =1
0073  B8 0000           MOV       AX,0                ; SC_TCREATE
0076  CD 60             INT       VRTX
    ;
    ;                   DELETE SELF LEAVING TASK TIME AS THE HIGHEST PRIORITY
    ;                   TASK
0078  B9 0000               MOV       CX,0            ; SELF
007B  B8 0001               MOV       AX,1            ; SC_TDELETE
007E  CD 60                 INT       VRTX            ;
0080          MAIN ENDP
    ;
    ;*************************************************************************
    ;
    ;                   CODE FOR TASK MSG_1
    ;                   THIS TASK DISPLAYS A MESSAGE EVERY HALF SECOND
    ;
                            PUBLIC    MSG_1
0080        MSG_1           PROC      NEAR
    ;
    ;                   WAIT FOR DISPLAY TO BE FREE BEFORE DISPLAYING
    ;                   THE MESSAGE.
    ;
0080  BB 0000 R     PRMSG:  MOV       BX,OFFSET SCREEN ; OFFSET OF MAILBOX
0083  8B ---- R             MOV       AX,SEG SCREEN    ; SEG OF MAILBOX
0086  8E C0                 MOV       ES,AX            ; INTO ES
0088  2B C9                 SUB       CX,CX            ; NO TIMEOUT, WAIT UNTIL
008A  2B D2                 SUB       DX,DX            ; SCREEN IS FREE
008C  B8 0009               MOV       AX,09H           ; SC_PEND
008F  CD 60                 INT       VRTX             ;
    ;
    ;                   PRINT "THIS IS TEST MESSAGE #1"
    ;
0091  BA 00B8 R             MOV       DX,OFFSET MSG2   ; OFFSET OF MESSAGE INTO DX
0094  E8 0000 R             CALL      DISP             ; DISPLAY IT
0097  E8 0005 R             CALL      CRLF             ; GIVE A CR, LF
    ;
    ;                   RELEASE DISPLAY FOR OTHER TASKS TO USE.
    ;
009A  BB 0000 R             MOV       BX,OFFSET SCREEN ; OFFSET OF SCREEN
009D  8B ---- R             MOV       AX,SEG SCREEN    ; SEG. OF SCREEN
00A0  8E C0                 MOV       ES,AX            ; INTO ES
00A2  B9 0001               MOV       CX,1             ; POST NON ZERO MESSAGE
00A5  B8 0008               MOV       AX,08            ; SC_POST
00A8  CD 60                 INT       VRTX             ;
    ;
00AA  B9 0009               MOV       CX,09            ; DELAY FOR HALF SEC.
00AD  2B D2                 SUB       DX,DX            ;
00AF  B8 000C               MOV       AX,0CH           ; SC_TDELAY
00B2  CD 60                 INT       VRTX             ;
00B4  EB CA                 JMP       PRMSG            ; PRINT MESSAGE AGAIN
00B6        MSG_1           ENDP
    ;*************************************************************************
    ;
```

```
                ;               CODE FOR TASK MSG_2
                ;               THIS TASK INTERRUPTS MSG_1 EVERY 3 SECONDS
                ;
                                PUBLIC  MSG_2
00B6            MSG_2           PROC    NEAR
                ;
                ;               WAIT FOR DISPLAY TO BE FREE BEFORE USING IT
                ;
00B6  BB 0000 R         M1:     MOV     BX,OFFSET SCREEN ; OFFSET OF MAILBOX
00B9  B8 ---- R                 MOV     AX,SEG SCREEN    ; SEG. OF MAILBOX
00BC  8E C0                     MOV     ES,AX            ; INTO ES
00BE  2B C9                     SUB     CX,CX            ; NO TIMEOUT
00C0  2B D2                     SUB     DX,DX            ;
00C2  B8 0009                   MOV     AX,09H           ; SC_PEND
00C5  CD 60                     INT     VRTX             ;
                ;
                ;               PRINT "INTERRUPTING MESSAGE #1 AT PRI 3"
                ;
00C7  BA 0044 R                 MOV     DX,OFFSET MSG3   ; OFFSET OF MESSAGE INTO DX
00CA  E8 0000 R                 CALL    DISP             ; DISPLAY THE MESSAGE
00CD  E8 0005 R                 CALL    CRLF             ; ISSUE A CR, LF
                ;
                ;               RELEASE DISPLAY FOR OTHER TASKS TO USE
                ;
00D0  BB 0000 R                 MOV     BX,OFFSET SCREEN ; OFFSET OF SCREEN INTO BX
00D3  B8 ---- R                 MOV     AX,SEG SCREEN    ; SEG. OF SCREEN
00D6  8E C0                     MOV     ES,AX            ; INTO ES
00D8  B9 0001                   MOV     CX,1             ; POST NON ZERO MESSAGE
00DB  B8 0008                   MOV     AX,08            ; SC_POST
00DE  CD 60                     INT     VRTX             ;
                ;
                ;               WAIT FOR 3 SECONDS BEFORE INTERRUPTING TASK MSG_1 AGAIN
                ;
00E0  B9 0036                   MOV     CX,54            ; DELAY FOR 3 SECONDS
00E3  2B D2                     SUB     DX,DX            ;
00E5  B8 000C                   MOV     AX,0CH           ; SC_TDELAY
00E8  CD 60                     INT     VRTX             ;
00EA  EB CA                     JMP     M1               ; PRINT MESSAGE AGAIN
00EC            MSG_2           ENDP
                ;
                ;*******************************************************************************
                ;
                ;               CODE FOR TASK WAITA
                ;               THIS TASK WAITS FOR THE CHARACTER 'A' AND THEN WAITS
                ;               FOR A CHARACTER TO BE INPUT.
                ;
                                PUBLIC  WAITA
00EC            WAITA           PROC    NEAR
                ;
                ;               THE NEXT THREE INSTRUCTIONS SUSPEND THIS TASK UNTIL
                ;               THE CHARACTER 'A' IS PRESSED
                ;
00EC  B5 41             W1:     MOV     CH,'A'           ; ASCII FOR A
00EE  B8 000F                   MOV     AX,0FH           ; SC_WAITC
```

```
00F1  CD 60                   INT       VRTX              ;
    ;
    ;                 CHARACTER 'A' HAS BEEN ENCOUNTERED; WAIT FOR DISPLAY
    ;                 TO BE FREE BEFORE DISPLAYING USER PROMPT.
    ;
00F3  BB 0000 R               MOV       BX,OFFSET SCREEN ; OFFSET OF SCREEN
00F6  8B ---- R               MOV       AX,SEG SCREEN    ; SEG. OF SCREEN
00F9  8E C0                   MOV       ES,AX            ; INTO ES
00FB  2B C9                   SUB       CX,CX            ; NO TIMEOUT
00FD  2B D2                   SUB       DX,DX            ;
00FF  B8 0009                 MOV       AX,09H           ; SC_PEND
0102  CD 60                   INT       VRTX             ;
    ;
    ;                 PRINT "CONSOLE AWAITING INPUT","=>"
    ;
0104  BA 0065 R               MOV       DX,OFFSET MSG4   ; OFFSET OF MESSAGE INTO DX
0107  E8 0000 R               CALL      DISP             ; DISPLAY IT
010A  E8 0005 R               CALL      CRLF             ; DISPLAY CR, LF
010D  BA 007C R               MOV       DX,OFFSET MSG5   ; DISPLAY "=>"
0110  E8 0000 R               CALL      DISP             ;
    ;
    ;                 POST A NON ZERO MESSAGE TO THE MAILBOX K1. THIS
    ;                 MAILBOX INDICATES THAT A CHARACTER IS BEING INPUT
    ;                 AFTER THE CHARACTER 'A' HAS BEEN ENCOUNTERED.
    ;
0113  BB 007F R               MOV       BX,OFFSET K1     ; OFFSET OF K1 INTO BX
0116  8B ---- R               MOV       AX,SEG K1        ; SEG. OF K1
0119  8E C0                   MOV       ES,AX            ; INTO ES
011B  B9 0001                 MOV       CX,1             ; NON ZERO MESSAGE
011E  2B D2                   SUB       DX,DX            ;
0120  B8 0008                 MOV       AX,08H           ; SC_POST
0123  CD 60                   INT       VRTX             ;
    ;
    ;                 WAIT FOR 1 SEC BEFORE ISSUING A TIMEOUT. CHARACTER
    ;                 HAS TO BE INPUT WITHIN 1 SECOND.
    ;
0125  B9 0012                 MOV       CX,18            ; DELAY VALUE OF 1 SEC
0128  BA 0000                 MOV       DX,0             ;
012B  B8 000C                 MOV       AX,0CH           ; SC_TDELAY
012E  CD 60                   INT       VRTX             ;
    ;
    ;                 RELEASE DISPLAY SO THAT OTHER TASKS CAN USE IT.
    ;
0130  E8 0005 R               CALL      CRLF             ; DISPLAY CR, LF
0133  BB 0000 R               MOV       BX,OFFSET SCREEN ; OFFSET OF SCREEN INTO BX
0136  8B ---- R               MOV       AX,SEG SCREEN    ; SEG. OF SCREEN
0139  8E C0                   MOV       ES,AX            ; INTO ES
013B  B9 0001                 MOV       CX,1             ; NON ZERO MESSAGE
013E  2B D2                   SUB       DX,DX            ;
0140  B8 0008                 MOV       AX,08H           ; SC_POST
0143  CD 60                   INT       VRTX             ;
    ;
    ;                 POST A ZERO MESSAGE TO K1 AFTER IT HAS RECEIVED
    ;                 THE FIRST MESSAGE.
```

```
                ;
0145  BB 007F R           MOV     BX,OFFSET K1    ; OFFSET OF K1 INTO BX
0148  B8 ---- R           MOV     AX,SEG K1       ; SEG. OF TASK
014B  8E C0                MOV     ES,AX           ; INTO ES
014D  B9 0000              MOV     CX,0            ; ZERO MESSAGE
0150  2B D2                SUB     DX,DX           ;
0152  B8 0008              MOV     AX,08H          ; SC_POST
0155  CD 60                INT     VRTX            ;
                ;
0157  EB 93                JMP     W1              ; WAIT FOR 'A'
0159        WAITA     ENDP
                ;
                ;*************************************************************************
                ;
                ;       TASK TIME DISPLAYS THE TIME AND BLANKS THE
                ;       SCREEN AFTER 23 LINES ARE DISPLAYED.
                ;       AND DISPLAYS THE REAL TIME.
                ;
                                PUBLIC  TIME
0159             TIME           PROC    NEAR
                ;
0159  FE 06 00EF R        T1:   INC     SEC_LOW         ; INCREMENT LOW DIGIT SECONDS
015D  80 3E 00EF R 3A           CMP     SEC_LOW,3AH     ; LAST DIGIT > 9
0162  75 55                     JNE     Z1              ; NO ! THEN QUIT
0164  C6 06 00EF R 30           MOV     BYTE PTR SEC_LOW,30H  ; YES RESET LOCATION TO 0
                ;
0169  FE 06 00EE R              INC     SEC_HIG         ; INCREMENT HIGH DIGIT SECONDS
016D  80 3E 00EE R 36           CMP     SEC_HIG,36H     ; SECONDS > 59
0172  75 45                     JNE     Z1              ; NO ! THEN QUIT
0174  C6 06 00EE R 30           MOV     BYTE PTR SEC_HIG,30H  ; YES RESET SEC COUNT TO 0
                ;
0179  FE 06 00EC R              INC     MIN_LOW         ; INCREMENT MIN COUNT
017D  80 3E 00EC R 3A           CMP     MIN_LOW,3AH     ; LAST DIGIT > 9
0182  75 35                     JNE     Z1              ; NOT THEN QUIT
0184  C6 06 00EC R 3A           MOV     BYTE PTR MIN_LOW,3AH  ; RESET LOW DIGIT TO 0
                ;
0189  FE 06 00EB R              INC     MIN_HIG         ; INCREMENT MINUTES
018D  80 3E 00EB R 36           CMP     MIN_HIG,36H     ; MINUTES > 59
0192  75 25                     JNE     Z1              ; NO ! QUIT
0194  C6 06 00EB R 30           MOV     BYTE PTR MIN_HIG,30H  ; RESET MINUTES TO 0
                ;
0199  FE 06 00E9 R              INC     HR_LOW          ; INCREMENT HOUR
019D  80 3E 00E9 R 3A           CMP     HR_LOW,3AH      ; LAST DIGIT > 9
01A2  75 15                     JNE     Z1              ; NO ! QUIT
01A4  C6 06 00E9 R 30           MOV     BYTE PTR HR_LOW,30H   ; RESET LOW DIGIT TO 0
                ;
01A9  FE 06 00E8 R              INC     HR_HIG          ; INCREMENT HOUR
01AD  80 3E 00E8 R 33                   CMP     HR_HIG,33H      ; HOURS > 24
01B2  75 05                     JNE     Z1              ; NO ! QUIT
01B4  C6 06 00E8 R 30           MOV     BYTE PTR HR_HIG,30H   ; RESET HOURS TO 0
                ;
                ;       WAIT UNTIL DISPLAY IS FREE BEFORE OUTPUTTING
                ;       THE TIME.
                ;
```

```
01B9  BB 0000 R     Z1:     MOV     BX,OFFSET SCREEN ; OFFSET OF SCREEN INTO BX
01BC  B8 ---- R             MOV     AX,SEG SCREEN    ; SEG. OF SCREEN
01BF  8E C0                 MOV     ES,AX            ; INTO ES
01C1  2B C9                 SUB     CX,CX            ; NO TIMEOUT
01C3  2B D2                 SUB     DX,DX            ;
01C5  B8 0009               MOV     AX,09H           ; SC_PEND
01C8  CD 60                 INT     VRTX             ;
   ;
   ;                THE FOLLOWING SECTION OF CODE SAVES THE CURRENT CURSOR
   ;                VALUE AND SETS A NEW CURSOR POSITION TO DISPLAY THE
   ;                TIME. THE SCREEN IS ALSO BLANKED OUT AFTER 23 LINES
   ;                HAVE BEEN DISPLAYED.

01CA  B4 05                 MOV     AH,5             ; SET ACTIVE PAGE
01CC  B0 00                 MOV     AL,0             ; TO 0
01CE  CD 10                 INT     10H              ; USING VIDEO_IO
   ;
01D0  B7 00                 MOV     BH,0             ; ACTIVE PAGE NUMBER
01D2  B4 03                 MOV     AH,3H            ; READ CURRENT CURSOR POSITION
01D4  CD 10                 INT     10H              ;
   ;
01D6  80 FE 16              CMP     DH,22            ; 23 LINES OVER
01D9  7E 1F                 JLE     C2               ; NO ! THEN JUMP
01DB  BA 0000               MOV     DX,0             ; SCREEN IS FULL
01DE  52                    PUSH    DX               ;
   ;
01DF  B7 00                 MOV     BH,0             ; SET CURSOR TO TOP OF
01E1  B4 02                 MOV     AH,2H            ; SCREEN
01E3  CD 10                 INT     10H              ;
   ;
01E5  B9 0780               MOV     CX,1920          ; BLANK THE SCREEN
01E8  B7 00                 MOV     BH,0             ; ACTIVE PAGE NO.
01EA  B0 20                 MOV     AL,' '           ; ASCII FOR BLANK
01EC  B4 0A                 MOV     AH,10            ;
01EE  CD 10                 INT     10H              ;
   ;
01F0  BA 0000               MOV     DX,0             ; SET CURSOR TO TOP
01F3  B7 00                 MOV     BH,0             ; OF SCREEN
01F5  B4 02                 MOV     AH,2H            ;
01F7  CD 10                 INT     10H              ;
01F9  5A                    POP     DX               ;
   ;
01FA  52            C2:     PUSH    DX               ; SAVE CURRENT CURSOR POSITION
   ;
01FB  B8 ---- R             MOV     AX,SEG HR_HIG    ; SEG OF HR_HIG
01FE  8E C0                 MOV     ES,AX            ; INTO ES
0200  BF 00E8 R             MOV     DI,OFFSET HR_HIG ; OFFSET INTO DI
   ;
0203  B6 18                 MOV     DH,24            ; NEW ROW 24 INTO DH
0205  B2 40                 MOV     DL,40H           ; NEW COLUMN 64 INTO DL
   ;
0207  57            C1:     PUSH    DI               ; SAVE DI
0208  B7 00                 MOV     BH,0             ; ACTIVE PAGE
020A  B4 02                 MOV     AH,2H            ; SET THE CURSOR TO 24,64
```

```
020C  CD 10                   INT       10H                 ;
020E  5F                      POP       DI                  ; RESTORE DI
                              ;
020F  57                      PUSH      DI                  ; SAVE DI
0210  8A 05                   MOV       AL,"DI"             ; DISPLAY CHARACTER POINTED
0212  B9 0001                 MOV       CX,1                ; TO BY DI
0215  B7 00                   MOV       BH,0                ;
0217  B4 0A                   MOV       AH,10               ;
0219  CD 10                   INT       10H                 ;
                              ;
021B  5F                      POP       DI                  ; RESTORE DI
021C  47                      INC       DI                  ; INCREMENT POINTER
021D  FE C2                   INC       DL                  ; INCREMENT COLUMN
021F  80 FA 48                CMP       DL,48H              ; ALL NUMBERS DISPLAYED?
0222  75 E3                   JNE       C1                  ; NO ! DISPLAY ONE MORE
                              ;
0224  5A                      POP       DX                  ; RESTORE PREVIOUS
0225  B7 00                   MOV       BH,0                ; CURSOR POSITION
0227  B4 02                   MOV       AH,2                ;
0229  CD 10                   INT       10H                 ;
                              ;
                              ;               RELEASE THE DISPLAY FOR OTHER TASKS TO USE.
                              ;
022B  BB 0000 R               MOV       BX,OFFSET SCREEN    ; OFFSET OF SCREEN INTO BX
022E  B8 ---- R               MOV       AX,SEG SCREEN       ; SEGMENT OF SC   REEN
0231  8E C0                   MOV       ES,AX               ; INTO ES
0233  B9 0001                 MOV       CX,1                ; NON ZERO MESSAGE
0236  2B D2                   SUB       DX,DX               ;
0238  B8 0008                 MOV       AX,08H              ; SC_POST
023B  CD 60                   INT       VRTX                ;
                              ;
023D  B9 0012                 MOV       CX,18               ; DELAY FOR 1 SEC
0240  2B D2                   SUB       DX,DX               ;
0242  B8 000C                 MOV       AX,0CH              ; SC_IDELAY
0245  CD 60                   INT       VRTX                ;
0247  E9 0159 R               JMP       T1                  ; DISPLAY TIME AGAIN
                              ;
024A                          TIME      ENDP
                              ;
                              ;*******************************************************************************
                              ;
                              ;               TIMER INTERRUPT HANDLER
                              ;               THIS INTERRUPT ROUTINE IS SERVICED WHENEVER THERE
                              ;               IS A TIMER INTERRUPT.
                              ;
024A                          TINT      PROC      FAR
024A  50                      PUSH      AX                  ; SAVE AX
024B  B8 0016                 MOV       AX,16H              ; UI_ENTER
024E  CD 60                   INT       VRTX                ;
0250  B8 0012                 MOV       AX,12H              ; UI_TIMER
0253  CD 60                   INT       VRTX                ;
0255  B8 0011                 MOV       AX,11H              ; UI_EXIT
0258  CD 60                   INT       VRTX                ;
025A                          TINT      ENDP
```

```
;*****************************************************************************
;
;               KEYBOARD INTERRUPT SERVICE ROUTINE
;               A KEYBOARD INTERRUPT TYPE 9 CAUSES THIS ROUTINE TO
;               BE SERVICED. ONLY UPPER CASE LETTTERS ARE RECOGNIZED.
;               TO ABORT THE PROGRAM PRESS ESCAPE.
;
025A                    KEYINT  PROC    FAR
025A  FB                        STI                     ; ENABLE INTERRUPTS
025B  50                        PUSH    AX              ; SAVE AX
025C  B8 0016                   MOV     AX,16H          ; UI_ENTER
025F  CD 60                     INT     VRTX            ;
0261  53                        PUSH    BX              ; SAVE ALL REGISTERS USED
0262  51                        PUSH    CX              ;
0263  52                        PUSH    DX              ;
0264  1E                        PUSH    DS              ;
;
0265  E4 60                     IN      AL,60H          ; READ KEYBOARD PORT
0267  50                        PUSH    AX              ; SAVE THE VALUE
0268  E4 61                     IN      AL,61H          ; READ CONTROL PORT OF 8255
026A  0C 80                     OR      AL,80H          ; SET UP ACKNOWLEDGE SIGNAL
026C  E6 61                     OUT     61H,AL          ; OUTPUT IT
026E  24 7F                     AND     AL,7FH          ; RESET ACKNOWLEDGE SIGNAL
0270  E6 61                     OUT     61H,AL          ; RESTORE ORGINAL CONTROL
0272  58                        POP     AX              ; RESTORE SCAN CODE
0273  A8 80                     TEST    AL,80H          ; KEY RELEASE ?
0275  75 6D                     JNZ     KB1             ; BRANCH IF YES
;
0277  50                        PUSH    AX              ; SAVE SCAN CODE
0278  B8 ---- R                 MOV     AX,TEST_DATA    ; SET UP THE SCAN TABLE
027B  8E D8                     MOV     DS,AX           ; FOR TABLE LOOKUP.
027D  BB 0091 R                 MOV     BX,OFFSET SCANTABLE ;
0280  58                        POP     AX              ; RESTORE SCAN CODE
0281  D7                        XLATB                   ; TRANSLATE THE CHAR. TO ASCII
0282  3C 1B                     CMP     AL,1BH          ; IS IT AN ESCAPE
0284  75 12                     JNE     P1              ; BRANCH IF NOT
;
0286  1E                        PUSH    DS              ; SAVE DATE SEGMENT
0287  B8 0040                   MOV     AX,40H          ; SET DATA SEGMENT TO 40H
028A  8E D8                     MOV     DS,AX           ;
028C  BB 0072                   MOV     BX,72H          ; SET FLAG FOR RESET FUNCTION
028F  C7 07 1234                MOV     [BX],1234H      ;
0293  1F                        POP     DS              ; RESTORE ORGINAL DS
0294  FF 2E 00E4 R             JMP     BOOT            ; RESET SYSTEM
;
0298  3C 00             P1:     CMP     AL,0            ; IS IT A VALID CHAR
029A  74 48                     JZ      KB1             ; BRANCH IF NOT
029C  50                        PUSH    AX              ; SAVE THE CHARACTER
;
;               CHECK WHETHER THE KEY IS BEING PRESSED AFTER
;               'A' HAS BEEN PRESSED.
;
029D  BB 007F R                 MOV     BX,OFFSET K1    ; OFFSET OF K1      INTO BX
```

```
02A0  B8 ---- R          MOV     AX,SEG K1          ; SEGMENT OF K1
02A3  8E C0              MOV     ES,AX              ; INTO ES
02A5  B8 0025            MOV     AX,25H             ; SC_ACCEPT
02A8  CD 60              INT     VRTX               ;
02AA  3D 0000            CMP     AX,0               ; IS THE KEY AFTER AN 'A'
02AD  75 10              JNE     P2                 ; BRANCH IF ORDINARY CHARACTER
02AF  58                 POP     AX                 ; RESTORE THE CHARCTER
   ;
02B0  50                 PUSH    AX                 ; SAVE THE CHARACTER
02B1  8A D0              MOV     DL,AL              ; DISPLAY THE CHAR   ACTER
02B3  B4 02              MOV     AH,2               ; USING DOS FUNCTION CALL 2
02B5  CD 21              INT     21H                ;
02B7  B2 0D              MOV     DL,0DH             ; GIVE A CARRIAGE RETURN
02B9  CD 21              INT     21H                ;
02BB  B2 0A              MOV     DL,0AH             ; GIVE A LINE FEED
02BD  CD 21              INT     21H                ;
02BF  58          P2:    POP     AX                 ; RESTORE THE CHARCTER
   ;
02C0  50                 PUSH    AX                 ; SAVE THE CHARACTER
02C1  8A E8              MOV     CH,AL              ; MOVE CHARACTER TO CH
02C3  B8 0013            MOV     AX,13H             ; UI_RXCHR; PASS CHARACTER
02C6  CD 60              INT     VRTX               ; TO VRTX BUFFER
   ;
02C8  58                 POP     AX                 ; RESTORE THE CHARACTER
02C9  8B 1E 0081 R       MOV     BX,BUFPTR2         ; POINTER TO END OF BUFFER
02CD  88 87 0083 R       MOV     BUFFER+BX[BX],AL   ; PLACE CHAR. AT END OF BUFFER
02D1  43                 INC     BX                 ; INCREMENT BUFFER POINTER
02D2  83 FB 0A           CMP     BX,10              ; WRAP AROUND ?
02D5  72 03              JC      KB2                ; NO ! THEN JUMP
02D7  BB 0000            MOV     BX,0               ; YES BUFFER WRAPPED AROUND
02DA  3B 1E 008D R KB2:  CMP     BX,BUFPTR1         ; IS THE BUFFER FULL
02DE  74 04              JZ      KB1                ; BRANCH IF YES
02E0  89 1E 008F R       MOV     BUFPTR2,BX         ; SET NEW END OF BUFFER
   ;
02E4  B0 20         KB1: MOV     AL,20H             ; END OF INTERRUPT COMMAND
02E6  E6 20              OUT     20H,AL             ; TO 8259 COMMAND REGISTER
02E8  1F                 POP     DS                 ; RESTORE ALL REGISTERS
02E9  5A                 POP     DX                 ;
02EA  59                 POP     CX                 ;
02EB  5B                 POP     BX                 ;
02EC  B8 0011            MOV     AX,11H             ;
02EF  CD 60              INT     VRTX               ;
   ;
02F1          KEYINT  ENDP
02F1          TEST    ENDS
   ;
   ;*****************************************************************************
   ;
0000             INITIALIZE SEGMENT 'CODE'
   ;
   ;        THIS SECTION OF CODE INITIALIZES ALL THE VECTORS AND
   ;        SETS UP POINTERS TO THE CONFIGURATION TABLE, VRTX,
   ;        THE KEYBOARD ISR AND THE TIMER ISR.
   ;
```

```
                        ASSUME CS:INITIALIZE
                ;
= 0184                  CFTBL_PTR_OFF   EQU WORD PTR 184H ;CFTBL POINTER OFFSET
= 0186                  CFTBL_PTR_SEG   EQU WORD PTR 186H ;CFTBL POINTER SEGMENT
= 0000                  VIP             EQU 0H            ;VRTX ENTRY POINT IP
= 4000                  VCS             EQU 4000H         ;VRTX ENTRY POINT CS
= 0060                  VRTX            EQU 60H           ;VRTX INTERRUPT
= 0180                  INT60_VCT       EQU 180H          ;VRTX INT VECTOR
= 0070                  TIMER           EQU 70H           ;TIMER INT VECTOR
= 0024                  KEY             EQU 24H           ;KEYBOARD INT VECTOR
                ;
                ;       LOAD POINTER TO CFT
                ;
0000  B8 0000           ENTRY: MOV      AX,0              ; SET DATA SEGMENT TO 0
0003  8E D8                    MOV      DS,AX             ;
0005  C7 06 0186 ---- R MOV    DS:CFTBL_PTR_SEG,SEG TBL ; CFTBL SEGMENT
000B  C7 06 0184 0000 R MOV    DS:CFTBL_PTR_OFF,OFFSET TBL ;CFTBL OFFSET
                ;
                ;       LOAD INTERRUPT VECTOR 60H FOR VRTX
                ;
0011  B8 0000                  MOV      AX,VIP            ; VRTX ENTRY IP
0014  A3 0180                  MOV      DS:INT60_VCT,AX   ;
0017  B8 4000                  MOV      AX,VCS            ; VRTX ENTRY CS
001A  A3 0182                  MOV      DS:INT60_VCT+2,AX;
                ;
                ;       LOAD TIMER INTERRUPT VECTOR 70H
                ;
001D  B8 024A R                MOV      AX,OFFSET TINT    ; OFFSET OF TIMER
0020  A3 0070                  MOV      DS:TIMER,AX       ; SERVICE ROUTINE
0023  B8 ---- R                MOV      AX,SEG TINT       ; SEGMENT OF TIMER
0026  A3 0072                  MOV      DS:TIMER+2,AX     ; SERVICE ROUTINE
                ;
                ;       LOAD INTERRUPT VECTOR FOR KEYBOARD ROUTINE
                ;
0029  FA                       CLI                        ; DISABLE INTERRUPTS
002A  B8 025A R                MOV      AX,OFFSET KEYINT  ; OFFSET OF KEYBOARD
002D  A3 0024                  MOV      DS:KEY,AX         ; SERVICE ROUTINE
0030  B8 ---- R                MOV      AX,SEG KEYINT     ; SEGMENT OF KEYBOARD
0033  A3 0026                  MOV      DS:KEY+2,AX       ; SERVICE ROUTINE
0036  B0 FC                    MOV      AL,0FCH           ; ENABLE KEYBOARD AND
0038  E6 21                    OUT      21H,AL            ; TIMER INTERRUPTS
003A  FB                       STI                        ; ENABLE PROCESSOR INTERRUPTS
                ;
                ;       VRTX INITIALIZATION
                ;
003B  B8 0030                  MOV      AX,30H            ; VRTX INIT
003E  CD 60                    INT      VRTX              ;
0040  3D 0000                  CMP      AX,0              ; ERROR IN INITIALIZATION
0043  74 0A                    JZ       NOERR             ; BRANCH IF NOT
0045  BA 0020 R                MOV      DX, OFFSET MSG    ; DISPLAY ERROR MESSAGE
0048  B4 09                    MOV      AH,9              ;
004A  CD 21                    INT      21H               ;
004C  CD 20                    INT      20H               ; RETURN TO DOS
004E  CC                       INT      3H                ;
```

```
                ;
                ;               THE FOLLOWING SECTION OF CODE CREATES THE FIRST TASK
                ;               WHICH IN TURN CREATES THE OTHER TASKS
                ;
004F  BB 0010 R  NOERR: MOV     BX,OFFSET MAIN  ; OFFSET OF TAS       K INTO BX
0052  B8 ---- R         MOV     AX,SEG MAIN     ; SEGMENT OF TASK
0055  8E C0              MOV     ES,AX           ; INTO ES
0057  B1 00              MOV     CL,0            ; ID=0
0059  B5 00              MOV     CH,0            ; PRI=0
005B  B8 0000            MOV     AX,0            ; SC_TCREATE
005E  CD 60              INT     VRTX            ;
                ;
                ;               START MULTITASKING USING VRTX_GO
                ;
0060  B8 0031            MOV     AX,31H          ; VRTX_GO
0063  CD 60              INT    VRTX             ;
                ;
0065            INITIALIZE ENDS
                END     ENTRY
```

```
Microsoft MACRO Assembler  Version 3.00              PageSymbols-1
      07-06-85
Segments and Groups:
N a m e                         Size Align Combine Class

CFTBL. . . . . .0046 PARA  NONE     'DATA'
INITIALIZE . . . . . .0065 PARA  NONE     'CODE'
TEST. . . . . .02F1 PARA  NONE     'CODE'
TEST_DATA. . . . . .00F0 PARA  NONE     'DATA'

Symbols:
N a m e                         Type    Value Attr

BOOT . . . . . .L DWORD 00E4  TEST_DATA
BUFFER . . . . . .L BYTE  0083  TEST_DATA Length =000A
BUFPTR1. . . . . .L WORD  008D  TEST_DATA
BUFPTR2. . . . . .L WORD  008F  TEST_DATA
C1 . . . . . .L NEAR  0207  TEST
C2 . . . . . .L NEAR  01FA  TEST
CFTBL_PTR OFF. . . . . .E WORD  0184
CFTBL_PTR SEG. . . . . .E WORD  0186
COL1 . . . . . .L BYTE  00EA  TEST_DATA
COL2 . . . . . .L BYTE  00ED  TEST_DATA
CRLF . . . . . .N PROC  0005  TEST Global Length =000B
DISP . . . . . .N PROC  0000  TEST Global Length =0005
ENTRY. . . . . .L NEAR  0000  INITIALIZE
HR_HIG . . . . . .L BYTE  00E8  TEST_DATA
HR_LOW . . . . . .L BYTE  00E9  TEST_DATA
INT60_VCT. . . . . .Number  0180
K1 . . . . . .L DWORD 007F  TEST_DATA
KB1. . . . . .L NEAR  02E4  TEST
KB2. . . . . .L NEAR  02DA  TEST
KEY. . . . . .Number  0024
KEYINT . . . . . .F PROC  025A  TEST Length =0097
M1 . . . . . .L NEAR  00B6  TEST
MAIN . . . . . .F PROC  0010  TEST Global Length =0070
MIN_HIG. . . . . .L BYTE  00EB  TEST_DATA
MIN_LOW. . . . . .L BYTE  00EC  TEST_DATA
MSG. . . . . .L BYTE  002C  CFTBL
MSG1 . . . . . .L BYTE  0004  TEST_DATA
MSG2 . . . . . .L BYTE  0028  TEST_DATA
MSG3 . . . . . .L BYTE  0044  TEST_DATA
MSG4 . . . . . .L BYTE  0065  TEST_DATA
MSG5 . . . . . .L BYTE  007C  TEST_DATA
MSG_1. . . . . .N PROC  0080  TEST Global Length =0036
MSG_2. . . . . .N PROC  00B6  TEST Global Length =0036
NOERR. . . . . .L NEAR  004F  INITIALIZE
P1 . . . . . .L NEAR  0298  TEST
P2 . . . . . .L NEAR  02BF  TEST
PRMSG. . . . . .L NEAR  0080  TEST
SCANTABLE. . . . . .L BYTE  0091  TEST_DATA
SCREEN . . . . . .L DWORD 0000  TEST_DATA
SEC_HIG. . . . . .L BYTE  00EE  TEST_DATA
SEC_LOW. . . . . .L BYTE  00EF  TEST_DATA
```

```
Microsoft MACRO Assembler  Version 3.00              PageSymbols-2
      07-06-85
T1 . . . . . . . . . . . . . . .L NEAR  0159  TEST
TBL. . . . . . . . . . . . . . .L BYTE  0000  CFTBL Global
TIME . . . . . . . . . . . . . .N PROC  0159  TEST Global Length =00F1
TIMER. . . . . . . . . . . . . .Number  0070
TINT . . . . . . . . . . . . . .F PROC  024A  TEST Length =10
VCS. . . . . . . . . . . . . . .Number  4000
VIP. . . . . . . . . . . . . . .Number  0000
VRTX . . . . . . . . . . . . . .Number  0060
W1 . . . . . . . . . . . . . . .L NEAR  00EC  TEST
WAITA. . . . . . . . . . . . . .N PROC  00EC  TEST GlobalLength =006D
Z1 . . . . . . . . . . . . . . .L NEAR  01B9  TEST

48312 Bytes free
Warning Severe
Errors  Errors
0       0
```

REFERENCES.

1. Using Operating System Firmware Components to Simplify Hardware and Software Design, Intel application note, Santa Clara, California, March 1982.

2. Microprocessor Operating Systems, Microcomputer Applications, Suisun City, California, 1981.

3. Vasudevan, R., "Issues in the Design and Development of Real-Time Data Acquisition and Control Systems," IEEE 1981 Real-Time Systems Symposium.

4. Operating System Principles, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1973.

5. The Logical Design of Multiple-Microprocessor Systems, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1980.

6. Dijkstra, E. W.,"The structure of THE Multiprogramming system," Communications of the ACM, May 1968.

7. VRTX/86 User's Guide, version 3, Hunter & Ready, Palo Alto, California, 1984.

8. Multiple Processor Systems for Real-Time Applications, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1985.

9. Real-Time Microprocessor Systems, Van Nostrand Reinhold Company, New York, 1985.

10. Yelvington, Patricia., "Embedded Microprocessor Applications Require a Real-Time Operating System," Digital Design, January 1983.

11. Faulk, Stuart. R. and Parnas, David. L., "On the uses of Synchronization in Hard Real-Time Systems," IEEE 1983 Real-Time Systems Symposium.

12. Keller, Erik. L., "Real-Time becomes Commercial Issue," Systems and Software, February 1985.

13. Ready, James. and Funck, Gary., "Software Components Create Plug-in OS," Electronic Design, April 19, 1984.

14. IOX/86 User's Guide version 3 Hunter & Ready, Palo Alto, California, 1984.

15. Bunce, Phil., "Silicon Operating System Modules Aid Realtime Comtrol," Computer Design, November 1982.

16. How to Write a Board Support Package for VRTX, Hunter & Ready, Palo Alto, California, 1984.

17. VRTX C Interface Library User's Guide Version 3, Hunter & Ready, Palo Alto, California, 1984.

18. Getting Started with Silicon Software Components Version 3, Hunter & Ready, Palo Alto, California, 1984.

19. IBM Personal Computer Hardware Reference Library, Technical Reference, IBM Corp. 1983, Boca Raton, Florida.

20. APC Uniform Functiona Requirements Definition by the Canadian Urban Transit Association, November 1983.

21. Tecmar PC-Mate RAM/ROM Installation Manual Users Guide, Tecmar Inc., 1984.

The vita has been removed from the scanned document