UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

INTERPROCESS COMMUNICATION MODEL BASED ON THE INVESTIGATION OF INTERRUPT MECHANISMS.

BY

Jan Polek

Submitted to the School for Graduate Studies in partial fulfillment for the requirements of the degree of Master of Applied Science.

Department of Electrical Engineering
Faculty of Science and Engineering
University of Ottawa
Ottawa, Ontario

August, 1975

VITA

NAME : Jan Polek

BORN : Stiavnik, Czechoslovakia
February 15th 1942

EDUCATION : DIPL.ENG., Technical University of Slovakia,
Bratislava, Czechoslovakia 1965

ACKNOWLEDGEMENT

I wish to thank Dr. S.I. Ahmad for his help, and also Dr. Birta and the Department of Computer Science for their support.

Abstract.

From a study of the computer systems operating in a multiprogramming environment, a model for the interrupt activity is derived. The basic components of the interrupt activity are identified and formulation is developed to express process switching in terms of these components. The systems studied include, in particular, the PDP 8/11/10, Sigma 7, Nova 800, Interdata 7/32, IBM 360, etc. Some important aspects of interrupt structure, such as, response time, overhead and saturation are examined and compared. The use of an interrupt mechanism in the scheduling of processes in multiprogramming systems is illustrated with examples. The problems of the Interprocess Control Communication are discussed. The design objectives for the general Interprocess Communication Facility are identified. The results of the aforementioned study are used to suggest a suitable mechanism for interprocess communication.

TABLE OF CONTENTS.

I INTRODUCTION.

1.1 Interrupt facilities.

The availability of computers capable of performing many concurrent operations has posed for the program designers the problem of how best to take advantage of this hardware. Hardware features, such as overlapping of input/output operations with computations by the central processing unit or multiprocessing (simultaneous computations on two or more processing units) are standard on many new computers. It is the function of the operating system to create a user interface which interacts efficiently with the hardware and also with other components of the software.

The structure of the first generation computers was such that all components , including basic peripherals, were controlled sequentially by the so called control program. The characteristic of such machines was that, if the program running on a central processing unit asked for some peripheral service, it issued a command to that device, and was halted until that operation was completed. This meant that the speed of the whole machine was limited to that of the slowest component in use at any moment. There was specially designed software used in conjuction with the system called the 'operating system'. This operating system was intended mainly to reduce the delays due to human

intervention by running whole sequences, or 'batches', of programs automatically.

Naturally, such an arrangement led to extremely low utilization of many parts of the system. With new advances in electronic technology, the difference of speed between the central processor operations and the peripheral operations became more pronounced. As a result there arose a strong economic pressure to arrange system control in such a way that two or more peripherals could be running simultaneously.

A fundamental advancement in computer design was made with the introduction of the 'interrupt'. The growth in the use of the interrupt mechanism in computer hardware technology has paralleled the growth in the sophistication of the software technology associated with interrupts. This concept was first used in some computers built in the late 1950's. In a machine fitted with this facility, each peripheral device can be started by the central processing unit but continues to work autonomously, while freeing the central processing unit for other computations. When the peripheral finishes its job, or needs attention for some other reason, it sends an 'interrupt', which usually causes the central processor to stop whatever it is doing, and start executing the interrupt program instead. The interrupt program takes appropriate actions based on the cause of the interrupt. This system makes it possible to arrange large overlapping of operations in a computer system.

The second stage in the use of interrupts brought up the class of monitored events broadened to include what were essentially software errors, e.g. division by zero or attempts to execute illegal operation codes. When the third generation of computers was introduced, with plans for still more comprehensive operating systems, a third stage in the use of interrupts was initiated. This stage added a new class of events which could cause interrupts to be generated, by the execution of a special instruction, the 'supervisor call'.

Present day computers consist of a large network of interacting machines. As was mentioned before, the peripheral devices operate at their own speed, independent of each other. The picture of what is going on inside the computer is very complicated if we try to think of it as one machine carrying out all sorts of tasks at a time. On the other hand, when several users share a computing facility, we wish that as many of the various tasks as the hardware allows are executed in parallel in order to minimize response time, or to reduce idle time of expensive machinery.

The task of designing software to organize all these activities proved tc be exceptionally difficult. It appears that the use of the interrupt in the above concept is not necessarily the best technique for the implementation of concurrent or parallel operations in the computer system (1,2,3). The basic software developed for the coordination

of these tasks, to implement sharing of the machinery and a satisfactory interface between other components of the software and hardware, namely, the operating system, tends to be expensive to write, difficult to analyze, inefficient and unreliable (1,2,3).

These considerations lead to a search for common principles in operating systems design and for solutions of problems of organizing and controlling the complexity of computer systems. The basic arguments for a new system can be summarized (2):

1. Software simplicity. It is required to have an architecture which allows us to write programs smaller, simpler and to contribute to the reliability and security of the system.
2. System hygiene. The design of the system should be based on sound principles.
3. System performance. The system should maximize efficiency in the utilization of resources (i.e percent of time not idle).

A method which has proved to be effective has been developed in the recent years(1,3,4,6,7). It is based on a system conceived as a set of sequential processes that interact as well-defined events. Instead of restricting the process to be associated with a hardware processor, it is regarded as an activity that executes one of the system functions, an activity which the programmer wishes to be executed conceptually in parallel with other activities. By

taking this point of view, a hardware processsor is considered as the resource which is needed by a process to carry its activity forward in real time and which could be shared among processes. 'Parallel (concurrent) activity' is then interpreted in the following sense: when a snapshot is taken of the system, several processes may be found somewhere between their starting point and their points of completion. Consistent with this point of view, a basic operating system is one that provides for the creation, removal, control, and intercommunication of the processes, all other operating system functions can be organized around this(5).

One of the reasons why parallel programs(processes) need to interact with each other is because they need to share some limited computer resources. For example, several processes may need to communicate with an operator through a single console. In such cases it is important that no other process be permitted to access the resource while a given process is using it. This problem of interaction is called a mutual exclusion problem. A second reason for interactions of processes is when they cooperate on a common task. It is necessary then that they exchange information and interact through some common item of data. Certain actions must be stopped at a given point until some event under control of another process has occurred. This situation is known as a 'synchronization' problem. The interaction of processes could lead to interference. A typical interference problem

is 'the deadly embrace' between two processes. 'Deadly embrace' is a state in the computer when processes are stuck in 'circular wait', each waiting for another to release its claim on a certain resource. The anticipated and welcome interaction is called 'communication' (1,15). The main effort of the software system designer is to provide effective communication without interference.

A number of implementation methods have been proposed. One of the earliest suggestions was made by Dijkstra (8), and followed by many others (4,5,6).

The basic methodology in implemetation of the system in the above proposals is to structure the systems into small machines (processes), and organize them into a hierarchy of so called levels.

The main purpose of this thesis is to investigate implementation of the lower levels of the abstraction, i.e., the levels which implement processor allocation and communication between processes. The implementation techniques of this part of the operating system are built on machine hardware and affect the overall performance of the computer system. It is the desire of the system designer to achieve an efficient implementation of this mechanism for supporting concurrent (parallel) processes.

Although it was recognized that interrupts in the conventional form are not the best way for implementation of concurrent (parallel) activities, utilization of interrupts in different forms are still present. Interrupt mechanism

architecture has influence on some aspects of interprocess communication and multiplexing of processors.

This investigation is based on a survey of some computer systems. Implementation schemes of the interrupt systems in different computers exhibit variations. Developement of a general model of the interrupt mechanism and investigation of the particular implementation techniques that have been used to improve the performance of interprocess communication and processor multiplexing in the computer system are the main concerns of this thesis .

The work reported here is divided in terms of the following:

1. Analysis of the actual implementation of interrupt activities in some typical multiprogramming systems.
2. Development of a generalized model for Interrupt activities.
3. Suggestion of implementation strategies for the proposed model.
4. Investigation of the effect of interrupt mechanisms on general interprocess communication characteristics in a computer system.
5. Suggestion for a basic mechanism for generalized Interprocess communication.

II. BASIC ANALYSIS OF THE INTERRUPT ACTIVITY MECHANISMS.

2.1 Interrupt Signals and their types.

Interrupt signals are generated when certain conditions (events) arise in the computer system. When this happens they are directed to a specific processor to cause interruption of its activities. There are two basic sources of interrupt signals. The primary sources are hardware signals, while the other type of interrupts are generated internally by the software. For example, an input/output device may generate an interrupt signal on completion of an input/output operation, as shown in Figure 2-1. This is called hardware interrupt.

There are two major classes of interrupt signals:

1. Interrupts generated external to the central processing unit (CPU), for example, I/O interrupts, and
2. Interrupts generated within CPU, for example, arithmetic overflow or an illegal instruction execution.

These classes of interrupts are often called Interrupts and Traps (see Figure 2-2).

Interrupts can also be categorized, according to the types of conditions by which they are generated, into the following groups:

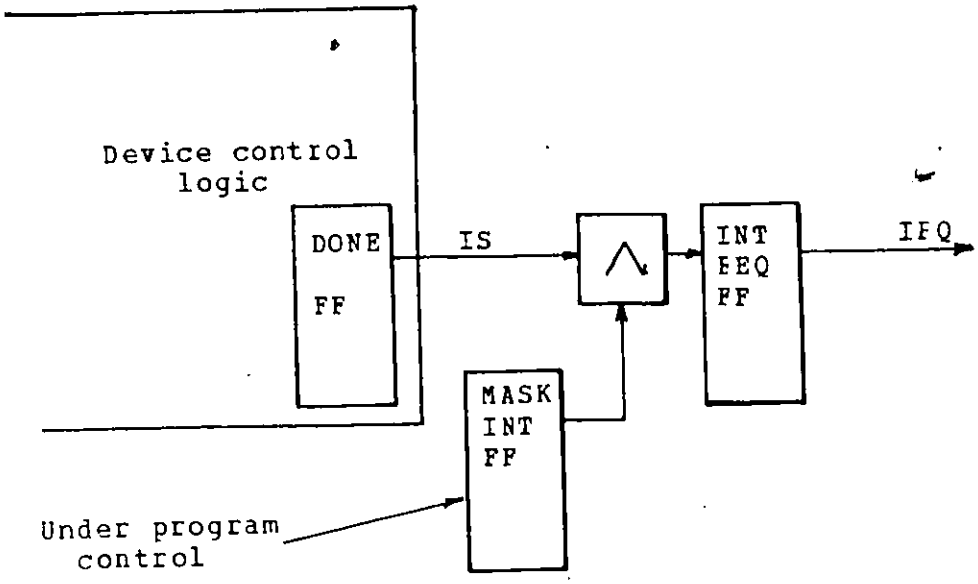


Figure 2-1. The Interrupt Signal is governed by a device status indicator 'Done' (operation is completed) and by a program controlled Interrupt Mask flag (MASK_INT). When a device completes its operation, it sets the 'DONE' flipflop, which in turn would set a flipflop in the interrupt request register(INT_REQ), provided this device is allowed to do so i.e its MASK_INT flipflop is set to 1. INT_REQ register usually contains one flipflop for each device capable of generating an independent interrupt signal. If any of the flipflops in the INT_REQ are on(i.e.set to one), an interrupt request(IRQ) line is energized. At the beginning of every machine cycle the processor synchronizes any requests that are pending, and initiates the appropriate actions.

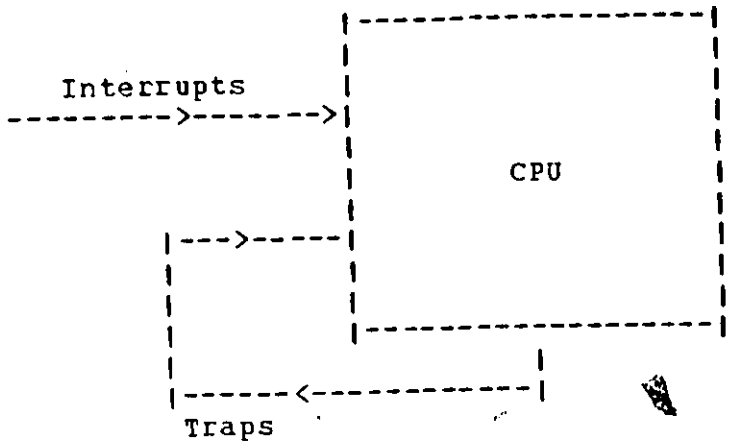


Figure 2-2. Interrupts as generated by sources external to the CPU. The so called traps are also interrupts generated by specific events within the CPU.

a. I/O interrupts. This group of interrupts is generated by the successful or unsuccessful completion of an I/O operation. Such an interrupt is generated when a peripherial processor(channel) has completed transmission of a block of data or a device has completed operation, such as tape rewind or positioning.

b. User program interrupts.

Two subclasses of interrupts could be generated in this group. One resulting from aritmetic operations such as divide check, arithmetic overflow, or from the attempts to execute illegal instructions or reference illegal memory locations. The other type is

the explicit call from the user program for some supervisor services (e.g. an SVC interrupt in IBM/360). The reason that the call to a supervisory service routine is treated as an interrupt is because it involves switching from the user mode to the supervisor mode of operation. A call to the supervisor may, therefore, be treated as a software generated interrupt signal.

c. Timer interrupts.

Computer systems have attached to them one or more hardware clocks which generate interrupts at fixed intervals. These timing marks are used for accounting purposes, as system protection and as input into scheduling routines. The system protection usually includes periodic checks of I/O operations to make sure that they are completed within the expected times, and that no hardware or software malfunction has occurred which could tie up an important piece of equipment (resource).

d. Hardware failure interrupts.

These interrupts result from hardware failures such as errors detected on data transmission or power failure.

In the computer system all interrupt signals are multiplexed (or-ed) onto one line which controls common interrupt flag in the CPU.

2.2 Priority Assignments.

A computer system may embody mechanisms to generate numerous interrupt conditions . It is possible for several of these conditions to appear simultaneously. The servicing of some of these interrupts may be more urgent than the program currently under execution. It is therefore necessary that some method be employed to specify and identify the urgency of an interrupt, and to arrange them in an appropriate order for receiving service when several interrupts occur at the same time.

Urgency is measured by the time allowed for the interrupt response and the cost of failing to respond to the interrupt in a timely manner. The cost is high if the interrupt is caused by an I/O device requesting memory for irreproducible input. The cost is low if the interrupt is caused by a low speed device during an output operation. An example would be, a typewriter signalling that it is ready to receive input. The urgency, however, is measured by a combination of factors and not just the speed with which the response must occur(12). With respect to the urgency of a particular interrupt signal, the priority is assigned to that interrupt line. Therfore, the higher the urgency of the signal, the higher the priority level will be assigned to that interrupt line and its service routine. Most systems preassign fixed priority levels for the various interrupt signals.

These interrupt request lines are connected at first to the priority control mechanisms, which will select one request among them, according to their preassigned priorities. But the assignment of fixed priority to the interrupt requests can be a complex task since the priority to interrupt depends not only on their urgency, but also upon their interaction (13). However, it is difficult to accurately forecast the exact nature of these interactions in advance, especially since the context tends to vary widely during system operation. The resulting assignment compromises can be avoided by supplying the freedom to dynamically reallocate priority levels. Such a tool enables the software to accurately establish the computer's response to changes in its environment. This capability is currently approached through some combination of arm/disarm ,enable/disable commands.

In most of the systems, interrupt hardware signals and associated service routines automatically get priority over software generated interrupt signals and other processes executing in the system.

III. ANALYSIS OF THE MULTIPROGRAMMING SYSTEM BASED ON THE INTERRUPT ACTIVITY MECHANISMS.

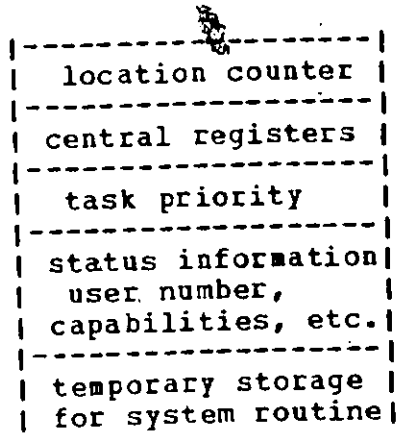
3.1 Interrupt Activities and the Structure of Multiprogramming Systems.

As mentioned in the introduction, the Interrupt Activity features provide some of the basic facilities for concurrent/parallel operations in multiprogramming - multiprocessing systems. A primary goal of multiprogramming is to share a central processor and its peripheral equipment among a number of programs loaded in the memory. A typical multiprogramming system is composed of two types of programs: so called Control programs and Processing programs. Processing programs operate under the control of the Control program. They are conveniently divided into three categories: language translators (such as the Fortran compiler), service programs (such as linkage editor), and user programs. The control program is usually constructed on the Interrupt Activity mechanism (known under such names as monitor, master control program, coordinator or supervisor,etc..). The basic functions of this program are as follows(5):

1. To keep records of the computations for which it is responsible, distinguishing between those that are free to run and those that are blocked for particular reasons. The control program maintains, for all the computations of which

it is responsible and those which are not actually running in a processor, all the information that is necessary for activating or reactivating them.

In current literature, a basic computation is called a process (1,3,5) or a task (10,11). In the CS/360 a task is defined as a fundamental unit of work. It may consist of a single program or a main program and a number of subroutines. Each process(task) is uniquely indentified by a data structure called process(task) descriptor. The information it contains,together with a minimal amount of other status information and temporary storage for essential system routines, is all that is needed to allow a process to run. All these data can be held in a block of memory locations which is called the Context block (6), or the Task Control Block (10,11). The address of this block is then sufficient information to give tc a processor when it starts to run the process. The operation of creating a process consists of creating a new context block. The typical layout of a context block is as follows:



2. To consult a scheduling algorithm to decide which, if any, of the tasks should be activated.

3. To provide intercommunication facilities between the processes as they may be required.

As described in the introduction, some of the basic aspects concerning Interprocess communication are mutual exclusion and synchronization problems. In the case of the mutual exclusion problem, there is a restriction on the interaction of processes, if the resource is used by one process, all other processes must be delayed (temporarily halted), until this resource becomes free.

In the case of the synchronization problem, one process must be delayed at a given point of execution until some event under control of another process has occurred.

The purpose of this section is to identify the main components of the control program, and to formulate the basic flow diagram of its execution with emphasis on the interrupt mechanism involved.

An example of the operations of the control program in the scheduling of tasks for achieving multiprogramming in a simple system (Interdata minicomputer) is as follows:

Assume that T1, T2 are two independent tasks, and T1 has higher priority than T2. The processor executing program T1 enters the Interrupt Activity Mechanism in response to an I/O request and performs the following, as directed by this system:

1. Initiate the I/O operation and record the name of the program which had made the request(i.e. T1). In the case of a busy device , enqueue the request.
2. Suspend program T1 (store registers,etc. into waiting list of tasks),and mark program T1 as awaiting an I/O operation.
3. Execute the scheduling program to find the task of highest priority that is capable of being executed, i.e. T2.
4. Load the relevant information from the task control block of T2 (registers,etc.)
5. Transfer control to task T2.

When the I/O operation for task T1 is completed, the processor will be interrupted and a control program will perform the following actions:

1. Basic Interrupt activity tasks will be executed (save Process Status Word, determine interrupting device, determine cause of interrupt - in this case completion of I/O in the proper way).
2. Determine the originator of the I/O request and report to this task (T1 in this case) that the operation has been completed.
3. Change the status of task T1 to ready state.
4. Execute the scheduling program to scan the task table for the highest priority job entry that is capable of being executed.

5. Load the relevant information from the task control block of T1 into the CPU registers.
6. Transfer control to task (T1).

A flow diagram of the operations as described above is shown in more general form, in Figure 3-1.

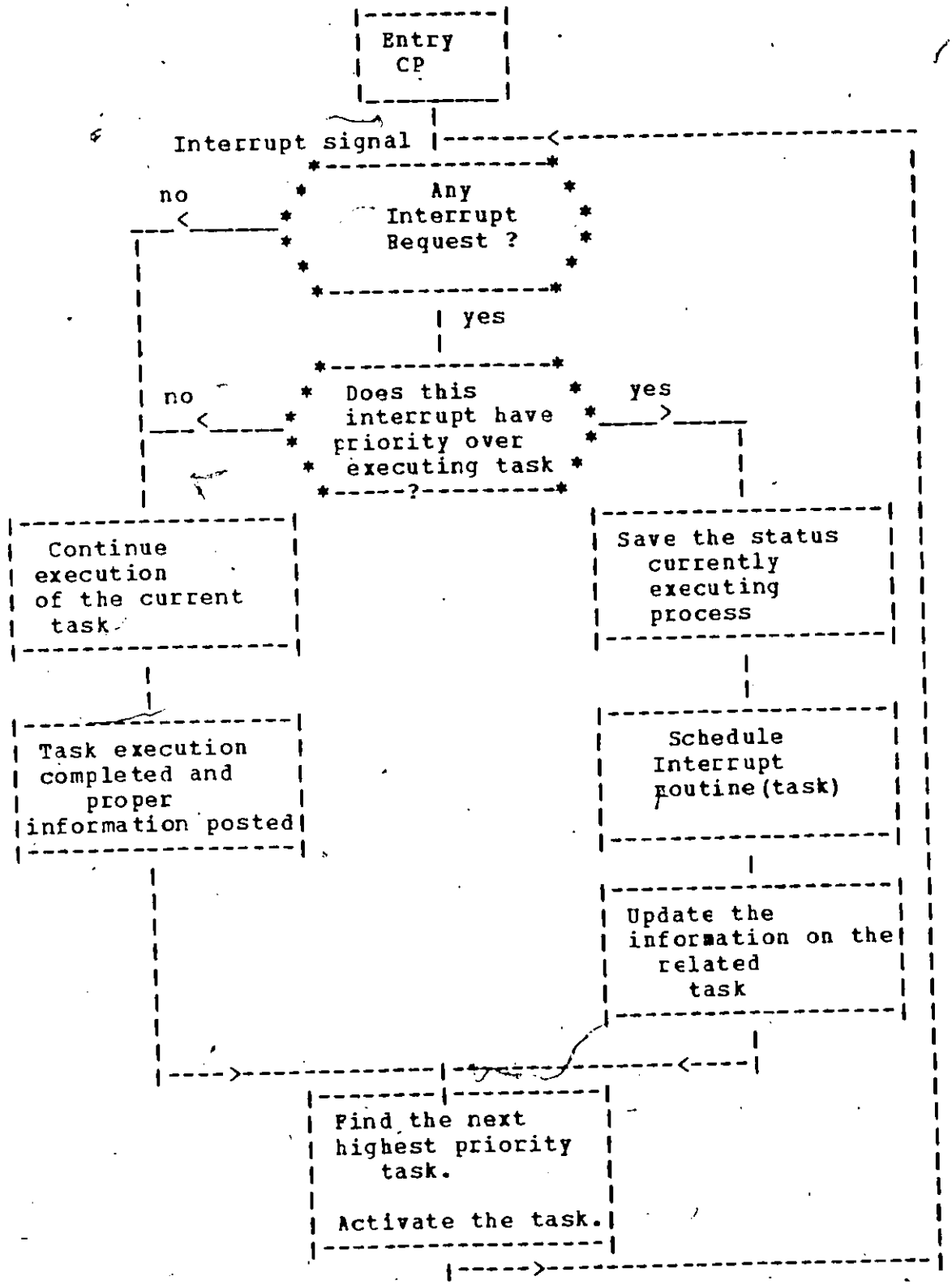


Figure 3-1. Main components of the control program in

the multiprogramming system, as related to peripherals interrupts. It is entered by the occurence of Interrupt Requests. Upon completion of interrupt processing the Scheduler program is entered, which selects the highest priority ready process. The control program then reloads the registers with the values from the selected process control block(stored there by the last interruption of the process) and transfers control of the CPU tc this process.

The scheduling part of this mechanism is built around the Task Description Tables. Each task can be in one of the following states:

- currently running
- ready (i.e awaiting a processor)
- blocked (i.e awaiting a signal from some other activity-task),

therefore, the necessary tables are a Ready Table and a Blocked Table.

In the case of blocked task, the task table may also indicate the reasons for the blocking, that is to say, the actions required to change its blocked status. The allocation of a processor is now the matter of selecting one out of the free (i.e ready status) tasks frcm the table. A typical state transition diagram of the scheduling policy of the user tasks is shown in Figure 3-2.

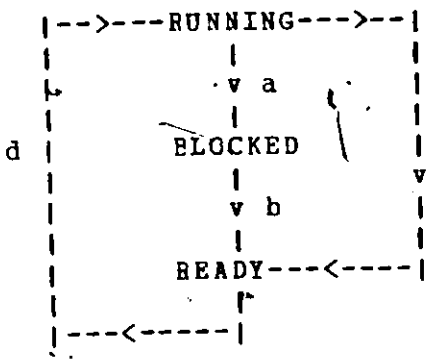


Figure 3-2. A typical state transition diagram for a process. An example of transition(a) of a process from 'RUN' to 'BLOCKED' state occurs when a process makes a request for I/O service (the process must wait for its completion). The transition(b) from 'BLOCKED' to 'READY' state occurs when an I/O request has been completed. When a process is running and is interrupted (e.g. higher priority process became 'ready'), the process is shifted to READY state(c). An example of transition(d) occurs when a process 'READY' state is scheduled to run.

The actual implementation mechanisms may differ in some systems. Real time systems often include an Interrupt Table. The purpose of this table is to keep track of all tasks which were started, but were interrupted by a higher priority Interrupt Request. The corresponding state transition diagram is shown in Figure 3-3.

The scheduling philosophy is achieved by specifying on

what occasions the table is to be scanned to reallocate the processor, and the algorithm to be used to select one activity to be run. A variety of processor scheduling schemes have been developed(9,12). They are based on different scheduling criteria such as priority, time frame,etc..

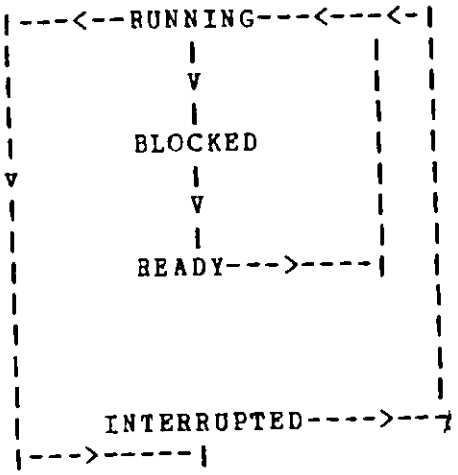


Figure 3-3. Example of modified state transition for a process. When running process is interrupted it is 'enqueued' into the so called 'interrupt table'. After servicing the interrupt the process is resumed.

3.2 General considerations in scheduling of tasks.

Most of the large computer systems divide the scheduling task into two parts, a so called 'Dispatcher' and a so called 'Scheduler'. The dispatcher is invclved after the handling of interrupts has been completed. It chooses the next process to be executed from the ready queue. Since the dispatcher is invoked very frequently(14), it should be short. For example, in the best case, it should only have to take the top process from the ready queue and assign the procesor to it(i.e., activate the process).

The major functions cf the scheduler program are:

1. keep track of the current condition, or state, of each task that is running in the system.
2. decide which task should be allowed to be executed next.

There are basically two approaches of keeping track of the condition of tasks.

The first approach is to identify all of the different states, or conditions, that the task could be in, with a 'state code'. Some examples of states code might be:

state 0 = runnable

state 1 = disk input wait

state 2 = teletype output wait

etc.

The second, status word method maintains a separate bit for each type of 'wait' condition in which the task might

find itself. This usually requires more memory space than the state oriented method, since a total of 15 states requires 15 bits instead of a number that ranges from 0 to 15 (could be represented by 4 bits). The advantage of this method is that the I/O routines can turn the appropriate bits on and off very easily. This feature is not beneficial if the computer lacks the kind of instruction set that makes it easy to set and reset individual bits. Moreover it is possible to turn on a contradictory combination of bits.

The two approaches are about equal in efficiency and convenience (12). The state code method is more compact and less error-prone, but the status word method seems to better suit the systems with many different wait conditions.

There are two different techniques of informing the scheduler of changes in a task's condition. The first approach involves explicitly calling the scheduler whenever anything happens to the task. Thus, whenever an I/O process recognizes that a task's I/O has been finished, it immediately calls the scheduler and indicates the new state. The advantage of this method is that the scheduler can take action immediately when it learns that a task has become runnable again. Thus, as soon as the I/O of a high priority task has finished, the scheduler will be able to interrupt the processing of any lower level priority task and resume the processing of high priority task.

The disadvantage of this method is that the scheduler is

forced to do a fair amount of processing at high priority levels. Thus, during the high-priority I/O processing, the scheduler may be called upon to juggle its scheduling queues and choose a new task.

The second method is an implicit approach. The I/O and other operating system processes change bits in the task's status word and then let the scheduler discover those changes during its regular scheduling time.

The advantage of this method is that the scheduler is processing very little at I/O priority levels. The actual scheduling is left for a lower scheduling priority level, usually triggered by a real-time clock.

The disadvantage of this method is that the system is less 'responsive', since the scheduler will not immediately start up a high-priority task when its I/O has finished. Also, it forces the scheduler to examine the status information of all the tasks to see which ones have changed, which could involve a fair amount of overhead.

IV. SURVEY OF HARDWARE IMPLEMENTATION TECHNIQUES OF THE INTERRUPT ACTIVITY .

4.1 The criteria for evalution and categorization of interrupt hardware structures.

Operations which are performed by the Interrupt Activity belong mostly tc the system overhead functions. Therefore the requirement on minimizing interrupt handling time and the memory space used by Interrupt Routines is important for system efficiency. During recent years many interrupt implementation techniques have been developed. In present computer systems, especially on-line systems the most common criterion used to evalute these techniques and, therefore to measure the effectiveness of priority-interrupt systems is the time spent on the Interrupt Activity executicn -T(INT). In the literature some intervals of the total interrupt processing time T(INT) are often indicated(13). These are:

RESPONSE TIME- T(RES) the time between occurrence of the interrupt signal and the start of execution of the first useful instruction requested by the signal.

OVERHEAD- T(OVH) the difference between the total time necessary to process the incoming request and the execution time of all useful instructicns.

Some of the other criteria for evaluation are as follows:

PRIORITY STRUCTURE-the feature of an interrupt structure that allows the CPU to react with the correct priority to incoming requests. If the most important interrupt, as determined by the current priority structure of request lines, has been requested but is not being serviced at a given time, then the priority structure of CPU is suboptimum.

SYSTEM SATURATION- a state reached when the CPU cannot respond quickly enough to all of the requests, and requests are essentially lost. The system is underdesigned if this state occurs.

The aim of the design of the Interrupt Activity structure is to minimize response and overhead times.

The interrupt mechanisms of some general purpose computers will be examined and discussed. It appears that there are many different types of interrupt systems, almost as many as there are computers. But from the detailed examination of these systems, it is evident that most interrupt structures can be divided into three fundamental groups. These are often called:

1. single level,
2. single line-multi level,
3. multi line-multi level,

An Interrupt line is a physical wire which has a direct connection to the processor, and is activated by one or more Interrupt Signal sources. An Interrupt level is defined as a hardware determined priority of an Interrupt Signal or group of Interrupt Signals.

This chapter describes each of these methods and examines in detail the interrupt mechanisms of some computers typically represented by that architecture. This is followed by a description of the features of some of the interrupt mechanisms under investigation summarized in a tabular form.

4.2 Single level structure.

The so called single level method is the simplest interrupt structure (Figure 4-1). To signal an interrupt, all devices use a single common line called the Interrupt bus. One or more devices raising the bus(changing the signal level of the line to which they are connected) will start the interrupt action. In this case, the computer, upon finishing the current instruction and storing the value of the program counter, will transfer control to the service task, which will poll each device's interrupt flag to determine which device or devices are seeking an interrupt. In the case of several simultaneous interrupts, the sequence in which devices are polled would serve as an implicit priority algorithm.

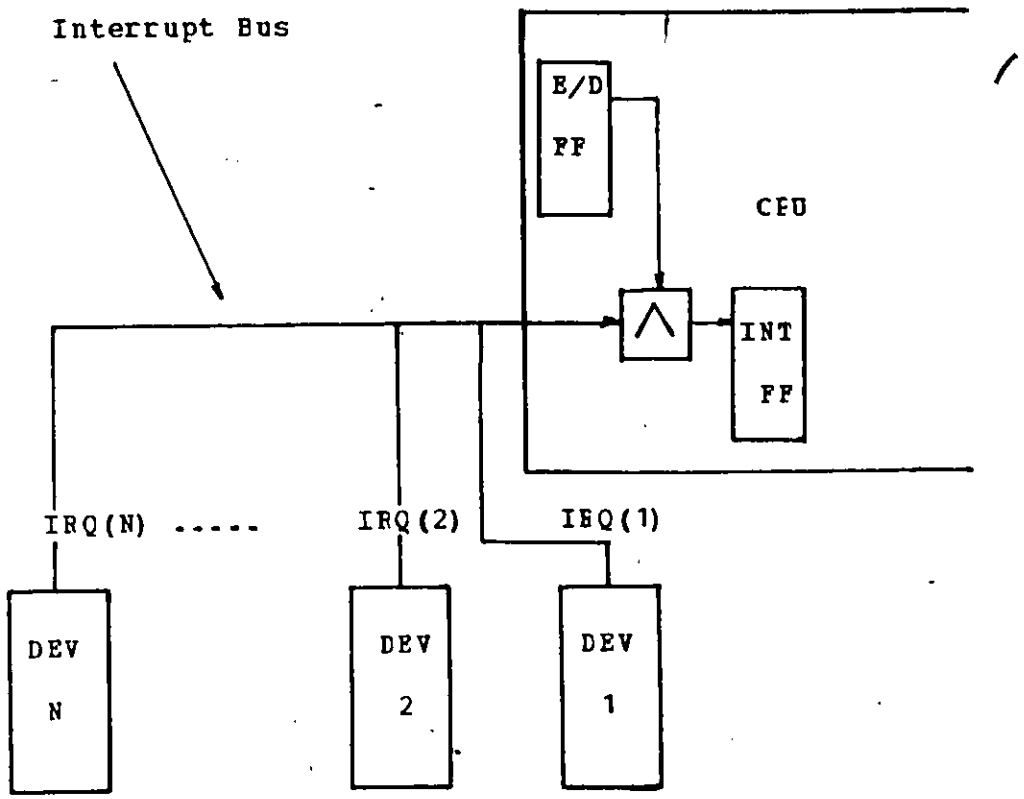


Figure 4-1. Single level interrupt structure. Any active Interrupt flag IRQ(I) will cause an interrupt request to be sent to the CPU. This request will intitiate a program to start a task which scans each Interrupt Request flag to identify the active one(s). E/D is the master enable/disable flipflop. When E/D=0 the processor ignores all interrupts.

EXAMPLE: PDP-8 Structure.

In this case all interrupt lines develop an 'or-ed' output, which controls a common interrupt flag(INT) in the CPU(15). The completion of the current instruction causes the program counter (PC) to be stored in location 0, and control progresses to location 1. At this location the CPU will execute jump to service task (JMP I 2). The service task starts by saving CPU registers (acumulator-AC, link-L). The next part of the program (often called a scan skip chain) tests each interrupt signal flag to determine which flag caused the interrupt. The order of the sequence of testing interrupt flags gives one interrupt signal priority over another. If an interrupt signal line is to be given a higher priority, its test-skip instruction is placed higher in the skip chain. This arrangement does nct permit equal priorities to be given to interrupt lines. The service routine, upon recognition of a specific interrupt, transfers control to the corresponding device service routine(task). The service routine first resets the recognized interrupt request, then executes the service program, and terminates by transferring control to the common routine 'Exit'. Exit restores the registers of the interrupted program and transfers control to it.

An example of the Interrupt service program in the

PDP-8 is as follows:

```
*0
/ first instruction after an interrupt
        0               /STORES RETURN ADDRESS
        JMP I 2         /JUMP TO SERVICE ROUTINE
        SERV            /CONTAINS ADDRESS OF SERVICE ROUTINE

/SERVICE ROUTINE
SERV,   DCA AC          /SAVE ACCUMULATOR IN A LOCATION AC
        RAL             /ROTATE LINK REGISTER IN ACCUMULATOR
        DCA L           /SAVE LINK IN A LOCATION CALLED L

/SKIP CHAIN ROUTINE
        KSF             /KEYBOARD INTERRUPT ?
        SKP             /NO;CHECK PRINTER
        JMP KB          /YES;SERVICE KEYBOARD
        TSF             /PRINTER INTERRUPT ?
        SKP             /NO;SKIP JUMP TO THE PRINT ROUT
        JMP TP          /YES, SERVICE THE PRINTER
        HLT             /FATAL HALT IF NO FLAG SET

/KEYBOARD INPUT SERVICE ROUTINE
KB,     KCC             /CLEAR KEYBOARD FLAG
        KRB             /READ THE CHARACTER
        TSL             /ACKNOWLEDGE ON THE PRINTER
        DCA I BUFFER    /STORE CHARACTER IN AN AREA CALLED BUFFER
        ISZ BUFFER      /INCREMENT BUFFER POINTER
        JMP EXIT        /RETURN TO THE INTERR PROGR
        ......
        ......

/ROUTINE FOR RETURNING TO THE MAIN PROGRAM
EXIT,   TAD I           /RESTORE LINK
        CLL RAR         /CLEAR LINK AND ROTATE ACCUMULATOR
        TAD AC          /RESTORE ACCUMULATOR
        ION             /RE-ENABLE INTERRUPT MECHANISMS
        JMP I 0         /LOC 0 CONTAINS RETURN ADRESS
```

4.3 Single line-multi level structure.

In this structure a single interrupt line (Interrupt bus) is again used (Figure 4-2). Also a single go ahead line(scmetimes called acknowledging or grant line) is used. Devices are connected in a daisy-chain arrangement on the go ahead line, with priority assigned by the order of the physical connection to the line. Higher priority units obtain access first, and when not bidding, relay the go ahead signal to the next lower level priority device. The device which is succssesful in obtaining an acknowledgement for the interrupt sends its identity (ex.device number) on the address lines.

EXAMPLE: NOVA 800 (17).

A description of the activities of the interrupt system in the Nova series of computers could be divided into the following routines:

1. Interrupt Hardware Routine (IHARD) - the activities of the hardware of the interrupt system.
2. Interrupt Handler Routine (IHR) - the first part of the software activities; for example identification of interrupting device, saving registers,etc..
3. Service Routine (SER) - activities for servicing of the identified interrupt.
4. Exit Routine (EXIT) -execution of activities required

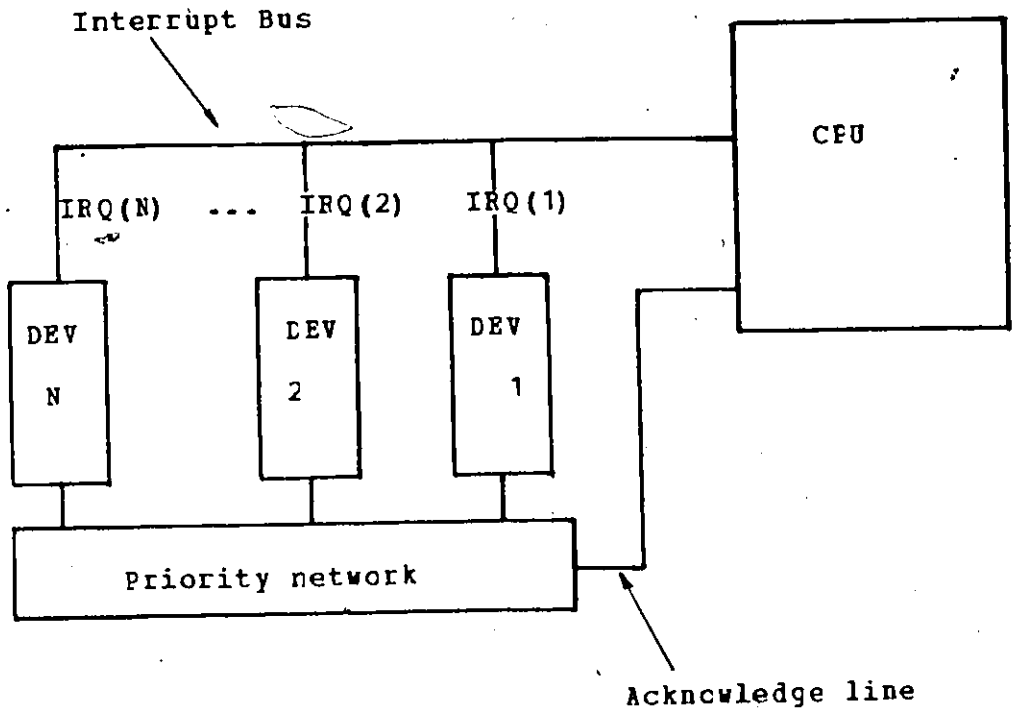


Figure 4-2. Single line - multi level interrupt structure. All devices present their Interrupt requests IRQ(N) on the common line(Interrupt bus). The processor answers with an Interrupt Acknowledge pulse on the so called acknowledge line. The first active interrupt request indicator (as defined by the physical position on the acknowledge line to the CPU) will trap this pulse and enable the device to present its address to the CPU on the appropriate lines. The Interrupt Acknowledge pulse also resets the identified active interrupt request indicator.

to return to the interrupted program.

A description of these activities is given below:

Interrupt hardware routine (IHARD):

1. Sense all interrupt signals anywhere in the system and set flags in the interrupt register (IR) for those not masked.

2. Sum logically all outputs of IR, to determine whether an interrupt has been requested and, if so, set the interrupt flag (INT) in CPU if CPU is enabled for interrupts (i.e., EN=1).

3. If INT=0 continue monitoring Interrupt signals.

4. If INT=1 disable CPU from receiving further interrupts (EN=0).

5. Store program counter (PC) into memory location 0.

6. Execute instruction from location 1 (usually branching to interrupt handler routine).

Interrupt handler routine (IHR):

1. Save contents of the CPU registers and the Interrupt Mask Register.

2. Save location 0

3. Read device code of the highest priority interrupting device (single instruction).

4. Branch to service routine for that device.

Service routine (SER).

1. Mask out same level and lower level priority interrupts.
2. Enable CPU to receive further interrupts.
3. Perform service function.

Exit routine (EXIT).

1. Disable CPU from receiving further interrupts.
2. Restore mask frcm save area and mask out interrupts according to this mask.
3. Issue acknowledge instruction to see if any interrupts are waiting.
4. If yes, go directly to service routine for the device.
5. If no, restore registers and the location 0
6. Enable CPU to receive futher interrupts.
7. Return to the interrupted program

EXAMPLE: INTERDATA 7/32.

A diagram of the organization of the interrupt hardware mechanism for the Interdata 7/32 computer is shown in Figure 4-3.

All interrupt requests from I/O devices are connected to one line(Interrupt bus). There are additicnal hardware facilities to respond to more critical internal interrupts(illegal instruction, arithmetic faults, machine malfunction,etc.) and the so called Supervisor call interrupt. The Supervisor call interrupt occurs as the result of the execution of a Supervisor call instruction. This instruction provides a means for user programs to communicate with system tasks. Some interrupts(I/O and arithmetic faults) are controlled by bits in the Program Status Word(hardware register which defines the operating state of the processor). That is, they can be enabled or disabled by setting or resetting a bit in the PSW under program control. Other interrupts are not controlled by PSW bits and are always enabled(illegal instruction interrupt,SVC). The interrupt processing is based on the concept of old,current, and new PSW. The current PSW defines the operating state of the prccessor. The new PSW becomes the current PSW. The current PSW now contains the operating status and the locaticn counter for the interrupt service task.

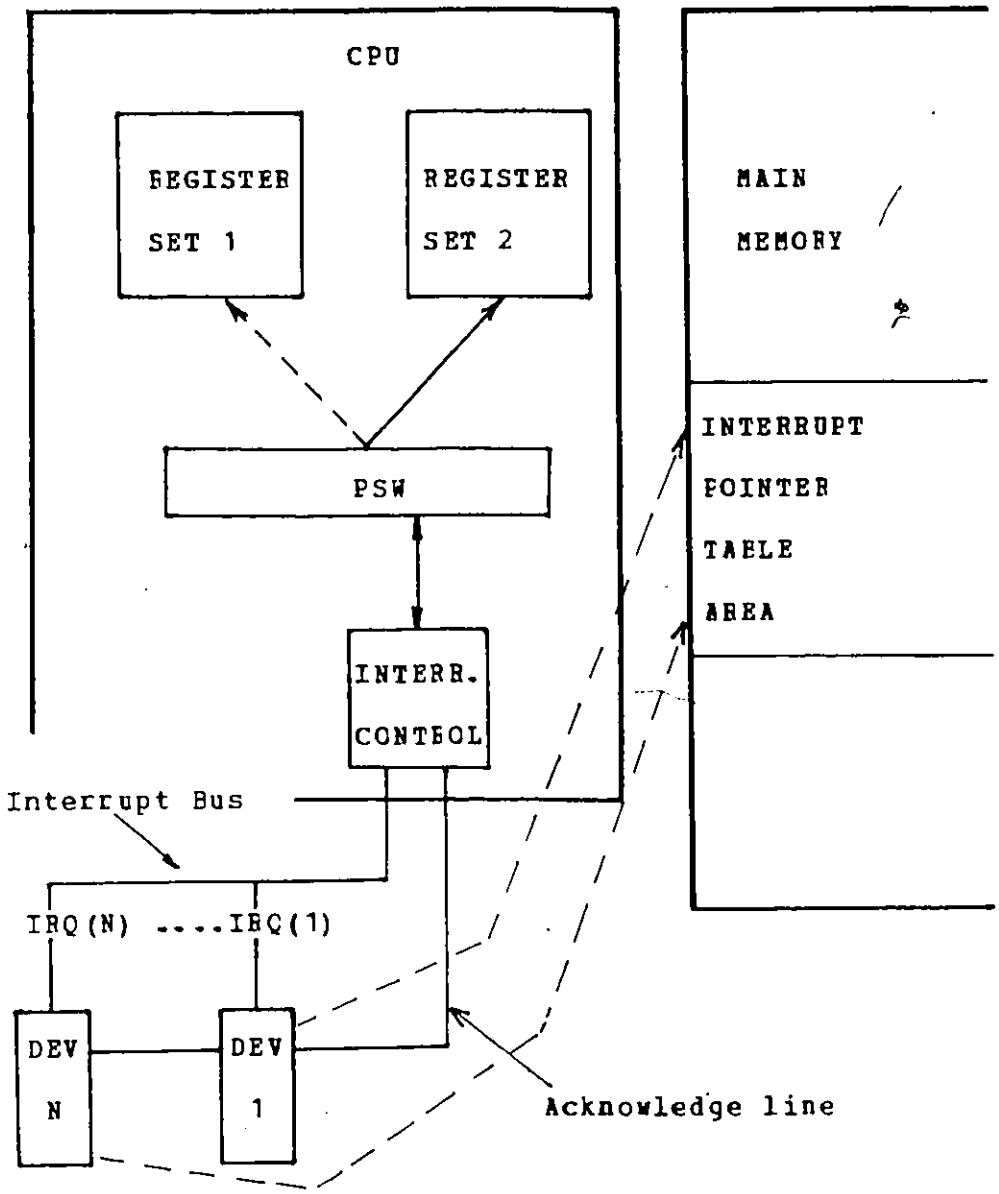


Figure 4-3. Organization of the interrupt mechanism elements as implemented in the Interdata 7/32 minicomputer. The dashed lines indicate transfer of control resulting from interrupt.

With one exception(the machine malfunction interrupt), on the occurence of the interrupt the current PSW becomes the

old PSW, and it is saved in a pair of registers belonging to the supervisor set. The machine malfunction PSW is stored in a reserved memory location. Again, with one exception, when a new PSW becomes the current PSW it is loaded from a reserved memory location. The exception is I/O interrupts. On an I/O interrupt, a CPU forces the PSW to the predetermined value which disables all interrupts except the machine malfunction interrupt, and selects the supervisor register set. The old PSW is stored in registers 0 and 1 of that register set. Hardware also automatically acknowledges the highest priority I/O interrupt request and obtains the device number and the status from the device. These are placed in registers 2,3 for the interrupt service task usage. The hardware control next automatically adds two times the device number to the starting location of the interrupt service table to obtain the address within the table that corresponds to the interrupting device. The value in the table becomes the current location counter.

4.4 Comparision of the interrupt structures of the computers PDP-8, NOVA-800 and INTERDATA 7/32.

The common feature of the interrupt structures of the above computers is a single line priority structure for detection and recognition of external interrupts. This means, that during the processing of one interrupt, the hardware circuitry does not allow the recognition of further

interrupts withcut destroying data important for current processing. If there is a desire to allow further interrupts to take place during the processing of an interrupt, some software mechanism must support this feature.

In the NOVA-800 and the INTERDATA 7/32 computers, the hardware circuitry recognizes priorities among interrupt requests on the interrupt lines(sometime called sub-level priorities). In the NOVA computer the number of sub-level priorities is limited to 16. The INTERDATA can have 1024 sub-level priorities. This feature is utilized by the single instruction 'Acknowledge Interrupt', which recognizes the highest priority active interrupt request and places its identity information into a CPU register for the interrupt service task usage. To achieve the same in the PDP-8, software is employed(skip chain), which tests in sequence interrupt flags to identify the active one.

Saving of the basic part of the status of the interrupted process is achieved by hardware control(PC into loc 0, or PSW into registers). The PDP-8 and NOVA have a small number of general registers, but they do not have fast facilities for storing and restoring them. The link register in PDP-8 must be first rotated into the accumulator and in NOVA computer only one register can be saved(restored) by an instruction at a time. In the INTERDATA 7/32 computer there is no need to save registers on the occurrence of an interrupt; only the switching of the CPU proccessing to a

different set of registers takes place. In the case of the need to save the register set, a single instruction for storing all of them is provided. Branching to the particular interrupt service routine is achieved by the software mechanism. The microprogramming control in INTERDATA 7/32 is autcmatically executing an indexing algorithm for branching to a particular service routine.

In summary one can say, that the single level structure is the simplest, and the least expensive. All of the basic tasks are executed on the CPU by software rcutines, except the Interrupt Recognition task. The response time and overhead for this structure are both very high due to the number of program steps that must be executed before the CPU begins to obey an interrupt request. While the system does test the highest priority request first, its high overhead may make the lower priority interrupts ineffective. This structure is called single-level since no interrupt can be recognized while one is being processed.

The multi level -single line interrupt structure has less overhead and better priority response processing than the structure described previously. This is due to the partial implementation of an identification of the interrupt source in hardware by a single instruction 'Acknowledge Interrupt'. This instruction identifies the Interrupt request that currently has the highest priority.

4.5 Multi level-multi line interrupt structure.

In this method, also called the matrix method, separate interrupt bus lines and separate go-ahead lines are provided for each device or group of devices. Simultaneous requests on lines are resolved by a hardware pricrity selection network. A block diagram of multi level-multi line interrupt structure is shown in Figure 4-4. Such an interrupt structure exists in almost all computers of medium and large size. This rather complex interrupt mechanism as used in the Sigma computer, is described below.

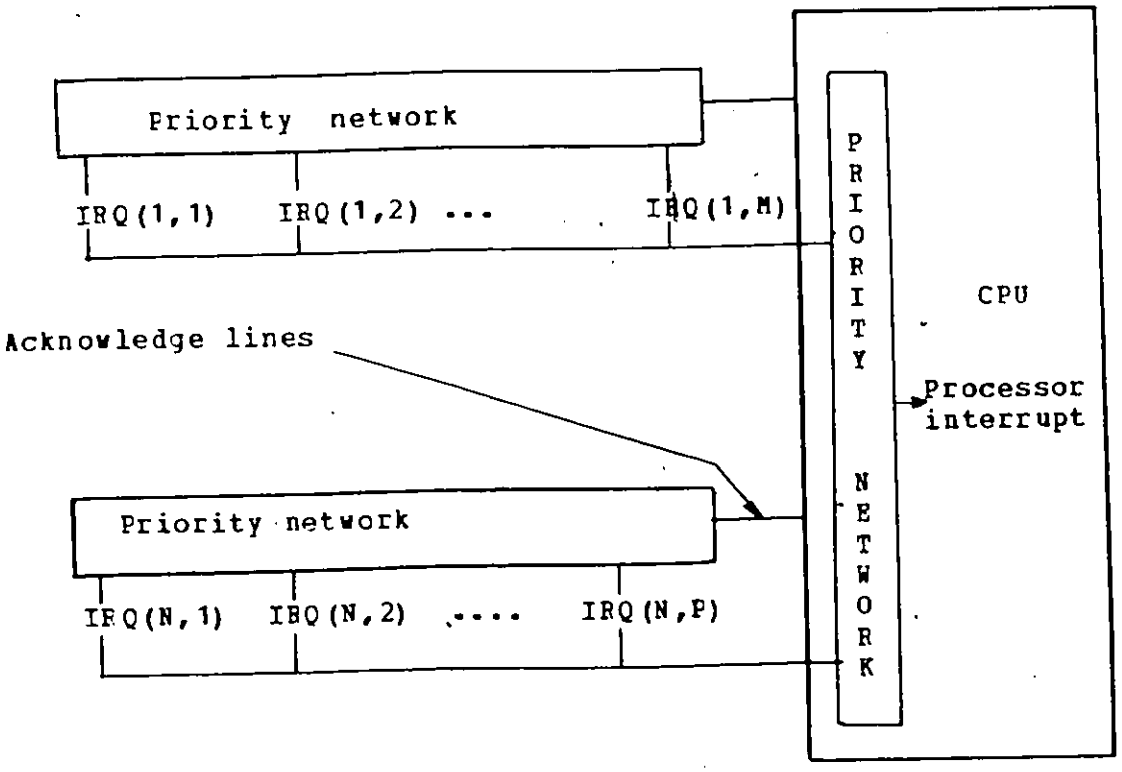


Figure 4-4. Multi level-multi line interrupt structure. Interrupt Requests are connected to interrupt

level indicators. The simultaneous requests on different levels are resolved by an automatic priority chain that causes the highest priority interrupt to be acted on first. If, during the execution of an interrupt, a higher-priority interrupt is received, then the lower priority interrupt will be deferred until it is again the highest. In this way the highest priority interrupt always receives immediate service.

EXAMPLE: SIGMA 7.

The interrupt system of the Sigma computer consists of 237 levels. These levels are divided into the following groups: override group, external group and counter group. The override group, having the highest priority, cannot be disabled. This group of interrupt lines is assigned to emergency events (e.g., power off, etc.). Most of the remaining levels have very flexible control facilities for the storing of interrupt signals and the priority selection of interrupts between different levels.

Each level uses two flip-flops to remember interrupts and two flip-flops to control the current status of an interrupt level. Some of the possible state assignments are:

1. (00) - No interrupt will be allowed
2. (01) - An interrupt will be allowed, but no interrupt has been requested.
3. (10) - An interrupt has been requested , but

has not been granted by the computer.

4. (11) - The requesting interrupt has been granted , but processing has not been completed.

One flip-flop is a used as level enable-disable facility (ENABLE FF), acting between the waiting and activated state. The ARM flip-flop is a second software controlled switch for determination of the status of the level. In the disarmed state, no signal to that level is admitted. In the armed state , the level can accept and remember an interrupt signal. On receipt of an interrupt signal, the level advances to the waiting state and remains there until it is allowed to move to the active state. If the enable flip-flop is off, the level cannot move from the waiting to the active state and it is completely removed from the priority network that determines the priority of access to the CPU. When an interrupt moves from the waiting state to the active state, the CPU executes the contents of the pre-assigned interrupt location as the next instruction. The interrupt level remains in the active state until it is cleared by the execution of a special instruction.

The interrupt lines are positioned into the priority network based on the order of priority. This network causes the highest priority to be acted upon first. If, during the execution of an interrupt, a higher priority interrupt is received, then the lower level interrupt will be deferred until it is again the highest. In this way the highest

priority interrupt always receives immediate service.

EXAMPLE: PDP-11/45.

The main hardware components of the PDP-11/45 computer for the I/O interrupts processing are shown in Figure 4-5.

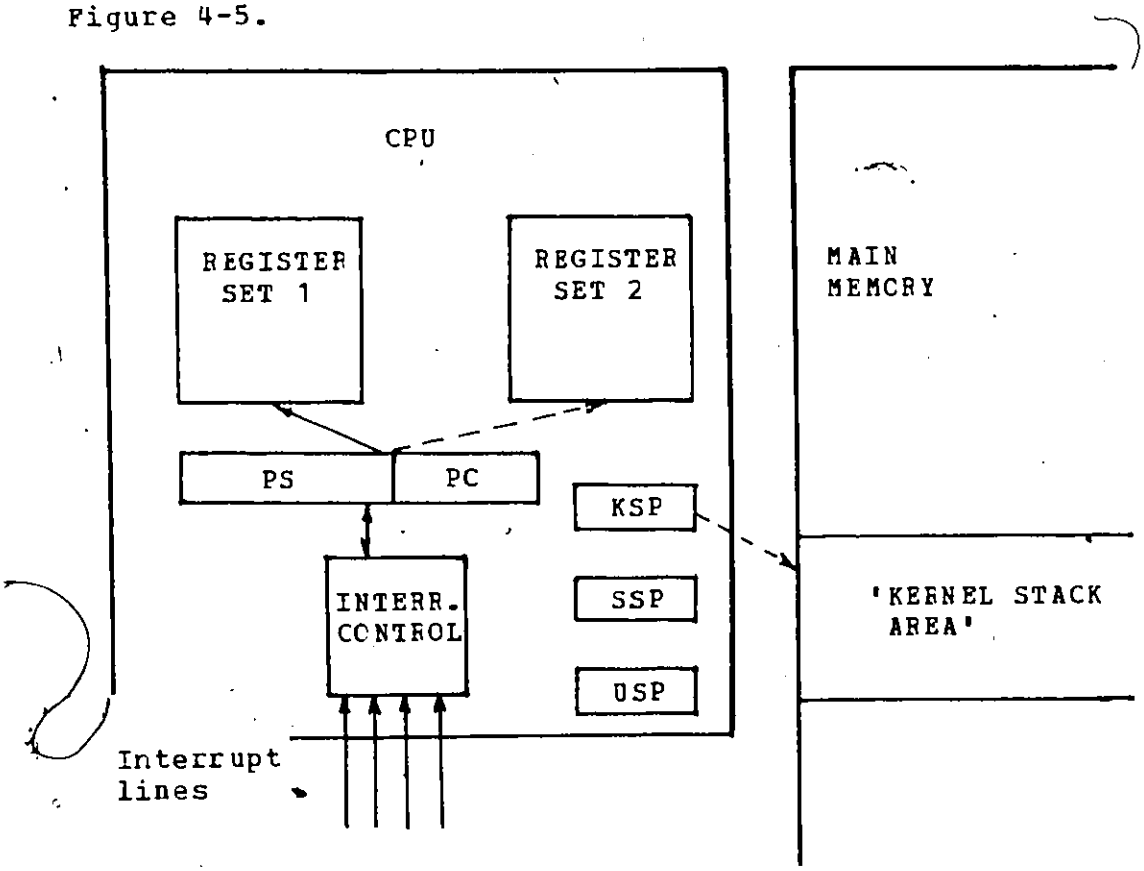


Figure 4-5. Organization of the interrupt mechanism elements as implemented in the PDP-11/45.

The basic information about the state of the running task is contained in the Processor Status Word register (PS) and the Progam Counter (PC) register. The PS contains the

information on the current priority of the running task(encoded in 3 bits). The priority level can be set under program control to one of the eight levels. This priority level inhibits granting of interrupt requests on lower levels. The CPU has two sets of registers, one Program Counter(PC) and three stack pointers registers. The stack pointers correspond to the operating mode - user stack pointer(USP), supervisor stack pointer (SSP) and kernel stack pointer (KSP). They indicate the last entry in the appropriate stack(a common temporary storage area with 'Last in First out' characteristic).

Interrupt requests from I/O devices are assigned four priority lines. When the interrupting device has a priority higher than the task running currently, CPU control automatically fetches a new PS and PC from the unique memory location (corresponding to the interrupting device). The old PS and PC are automatically pushed into a particular processor stack as indicated by the PS of the service task. The interrupt service task is then initiated.

4.6 Characteristics of the multiline priority interrupt structures.

The interrupt hardware detection-recognition structures of the computers as described before are categorized as multiline priority structures. The interrupt structure of the Sigma 7 computer has application in large real-time

environment. The interrupt structure of the PDP 11/45 is less extensive. It has 7 priority request lines. Both structures have flexible enabling/disabling(masking) facilities.

They have different implementation techniques used for saving and restoring operations of the basic status information for the interrupted program. The Sigma computer has reserved for each interrupt priority line fixed locations in memory for storing the old(interrupted task's) PSW and the new(interrupt service task's) PSW. When an interrupt occurs, the PSW of the currently running task is stored into location corresponding to that priority line and an interrupt service task. A similar mechanism is used in IBM 360 computer.

This type of mechanism does not allow another interrupt to take place on the same line, since the information of the interrupted task would be overwritten. This technique also seems to be 'logically unclean', because the status information of the interrupted task, does not have to be related logically to the task to which the interrupt request is pertinant.

The implementation of the save(restore) operations in the PDP 11/45 are based on hardware push down storage facilities. The basic information(PC,PS) of the running task on the occurrence of interrupt is automatically 'pushed down' into stack and 'pulled up' on the exit to the interrupted program. This feature enables the hardware to

handle multiple interrupts from the interrupt line. A similar interrupt structure is available on Burroughs computers(14).

An example of the interrupt hardware structure which has a separate register bank(set) for each interrupt line is the IBM System/7 minicomputer(21). Its primary application is in a process control enviroment. The status switching on occurence of interrupt is then accomplished by simply switching from one register 'bank' to another.

The multi level - multi line interrupt structure has most of the basic tasks implemented in hardware. Therefore this structure provides short response time and low overhead. Every allowed interrupt request is obeyed immediately, provided no higher priority request is presently being executed.

The greatest advantage of this structure is a near-optimum priority structure. The most important interrupt is either being serviced or is about to be serviced. The multi level-multi line structure ensures that the next process to be executed always has the highest existing priority, regardless of how many lower-priority requests remain only partially completed.

4.7 Summary of Interrupt mechanism features of some computer systems.

Interrupt Mechanism features of the computer systems which have been examined are summarized in Table 1. The entries in this table describe the basic characteristics of the systems.

The terminology in the table is based on the description in the preceding sections, to which one should refer for more detailed explanation. A basic explanation of the particular entries in the table is as follows:

Single level-single line. An interrupt detection-recognition structure in which interrupt requests do not have hardware enforced priorities. All interrupt requests are multiplexed into single line(physical wire) into the CPU.

Multi level-single line. In this structure all interrupts are again multiplexed into single line, but the interrupt requests on that line have enforced priorities by the hardware circuitry.

Multi level-multi line. The interrupt detection structure, in which the interrupt requests are multiplexed into two or more interrupt requests lines.

Multiple set of registers. An implementation of the interrupt switching operations based on the architecture of multiple set of general registers. When an interrupt occurs, only the switching of the CPU register set takes

place.

Stack feature. An automatic hardware storing of the basic state of the interrupted process into the LIFO temporary storage area on the occurence of interrupts.

COMPUTER	CHARACTERISTICS				
	SING LEV	MULT LEV SING LIN	MULT LEV MULT LIN	MULT GEN REG SET	STACK FEATURE
PDP-8	yes	no	no	no	no
INTERDATA 7/32	-	1024	no	yes	no
NOVA 800	-	16	no	no	no
PDP11/45	-	-	7	yes	yes
SYSTEM 7	-	-	4	yes	no
PDP-10	-	-	5	yes	no
IBM-360	-	-	5	nc	no
B-5000	-	-	48	no	yes
SIGMA-7	-	-	237	yes	no

TABLE 1.

V. INTERRUPT ACTIVITY MODEL.

5.1 An overview of the Interrupt Activity functions.

The investigation in the preceding chapters indicates, that the basic tasks of the Interrupt Activity are as follows:

1. To provide a mechanism for the synchronization of activities of some processes; i.e., the mechanism for passing signals when the progress of one depends on the other. On the hardware level, activities of the processes (computations) are represented by a sequence of changes in the state of the machine. Some changes(events) are in the interest of other processes. Interrupt systems continually monitor these events and notify an appropriate process about their occurrence.
2. To serve as a tool by which processor allocation decisions of the scheduler may be enforced. In this case it is the mechanism for fast switching of a processor among processes. The Interrupt mechanism is based on a priority driven scheduling method. Since the Interrupt requests are usually associated with high priority processes, the Interrupt mechanism should start a processor to execute instructions for a new process within a very small time delay after an interrupt event arises.

5.2 Basic tasks of the Interrupt Activity.

The preceding study also indicates that implementation schemes in different computers exhibit variations. However, there is an identifiable set of basic non-reentrant tasks common to all systems, which are the basis for our model of the structure of interrupt activity.

Interrupt Activity covers the following basic tasks:

1. Interrupt Recognition (REC)
2. Interrupt Stacking (STC)
3. Interrupt Scheduling (SCH)
4. Control Switching (SWC)
5. Interrupt Servicing (SRV)
6. Post Processing (POST)

The following is a description of the basic operations of these tasks, with an overview of their implementation schemes as identified in the preceding chapter.

1. Interrupt Recognition Task (REC).

The purpose of this task is to examine if an interrupt request exists anywhere in the system and to activate the interrupt stacking task. The device or process seeking an interrupt sets the appropriate variables (flags) specific to it and shared with the REC task. The set of these variables is often called the Interrupt Request Register (IRR). The IRR

stores all the Interrupt Requests which are possible in the system until they can be processed. The requests may be examined(scanned) according to certain control parameters, for instance- priority numbers.

2. Interrupt Stacking Task(STC).

The function of this task is to obtain and record(implicitly or explicitly) the relevant interrupt information into some memory area (communication area). The organization of the information in this area can differ (e.g., the FIFO,LIFO,etc.), but most often it is in the form of a queue called Interrupt Request Queue. Interrupt Requests are accompanied by some control parameters. The block of control parameters could be extensive, as in the case of an I/O controller, or less extensive, for example, arithmetic operation overflow. Another element of the Interrupt Request Queue entry is the indicator (name) of the task, called the Interrupt Service Routine, which is responsible for servicing the selected Interrupt Request.

For the systems examined, the methods for obtaining information identity of the device requesting interrupt may be listed as follows:

a. Polling device interrupt flags by software instructions.

b. Device self identification on software command by putting its address on an I/O Bus.

c. Automatic hardwired identification of the interrupt and

branching into the location corresponding to the level of interrupt or specific interrupt request.

The methods for recording information relevant to the interrupt may be listed as follows:

a. software determination.

b. automatic storing into a dedicated location.

3. Interrupt Scheduling Task(SCH).

The function of the interrupt scheduling task is to select only one request in the case of multiple simultaneous requests and decide when an interrupt activity can proceed into the next state according to certain conditions. As was described in chapter 3 and 4, computer hardware and software is structured into different pricrity levels. The priority criterion is most often used for multiplexing(allocation) of the CPU among ready processes. Many systems have implemented priority selection criterion into fixed hardware circuitry. Since priorities of the processes could vary(chapter 3), most systems incorporate some arm-disarm or enable/disable commands to supply freedom or flexibility to re-allocate priority levels. With respect to the above statement ,we can speak of the level of control of the Interrupt Recognition structure. Enabling/disabling control is usually performed by associating flags (flip flops) with interrupt signals, which may be called control nodes. An example of a possible interrupt structure with an outline of the control (scheduling) nodes is shown in the Figure 5.1.

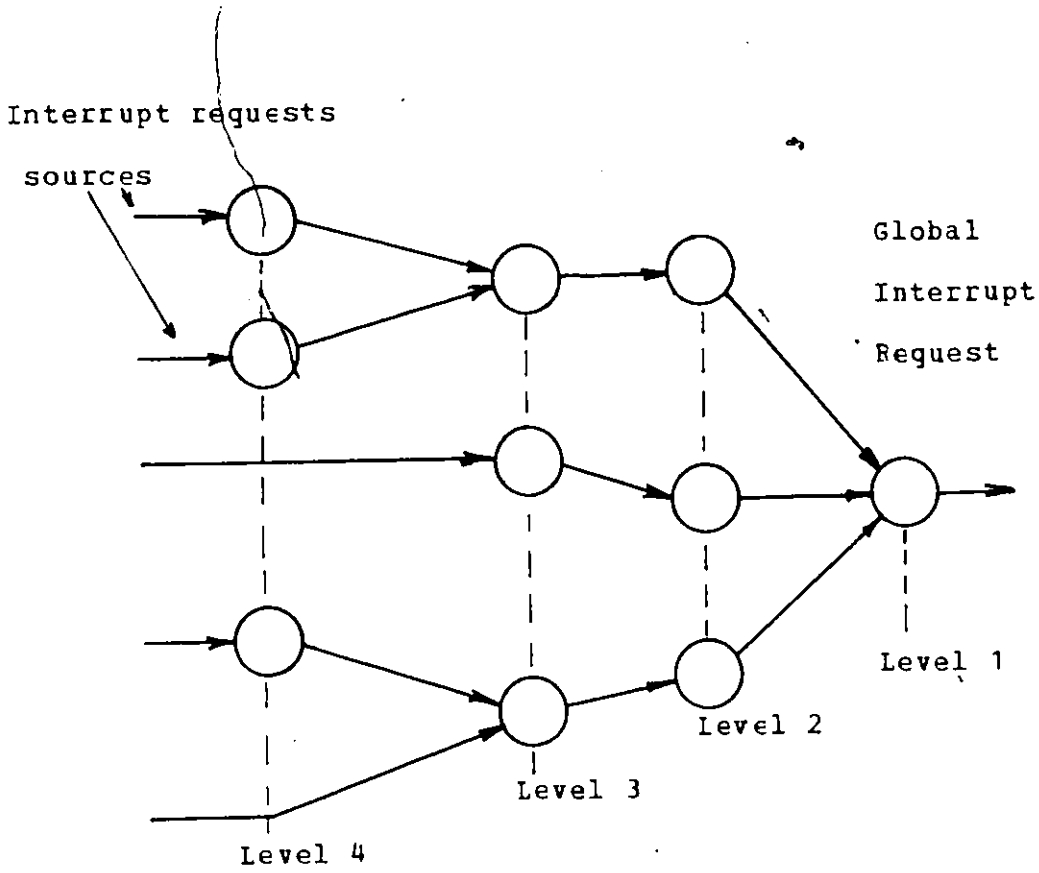


Figure 5.1 The diagram of control nodes in an interrupt mechanism organization.

In terms of the common usage(chapter 3) of interrupt enable/disable flags, the following interpretation applies.

Level	Function
1	E/D All interrupt system
2	E/D Group of interrupt signals-lines
3,4	E/D sub-group of interrupt signals,e.g., at a given level
3,4	E/D Individual interrupt sources

4. Control Switching Task(SWC).

When an interrupt request arrives, it is recognized and scheduled, control is transferred to the appropriate routine, which will perform the requested action. Conceptually what is involved in this is the act of saving the status information of the current process and loading the registers with corresponding status information of the other process. The status information, usually called stateword defines the process. The information it contains is necessary and sufficient to resume a process. Statewords may be different for different systems. The fundamental state word elements are implemented in hardware registers. For a central processor of conventional organization the basic stateword includes:

a. The contents of the instruction(program) counter.

b. The contents of the general registers of the processor.

5. Interrupt Service Task(SRV).

The function of this task is to allow the interrupt stack entry (if any) to be discarded. Interrupt service tasks are closely associated with the machine hardware or process requesting interrupt.

A typical example of the Interrupt service task is an I/O interrupt routine which may be described as follows(11):

The I/O interrupt routine upon entry checks the completion

of the I/O operation by the use of the Channel Status Word (CSW). If an error is detected, then a call would be made to an interrupt handling task for processing this error. Next, the channel scheduler queue is checked for any pending I/O requests. If none are pending an exit from the routine is made to the post-processing task. If I/O requests have been queued, then the next request for the free channel is fetched. The device routine is entered to determine the CCW (Channel Command Word) address. Following this the START I/O routine begins the operation if the device and channel are available. If the I/O request is not deferred, then this I/O request is removed from the queue. If the operation has been deferred because the channel control unit, or I/O device is busy, exit from the routine is made to the Interrupt post-processing(completion) task.

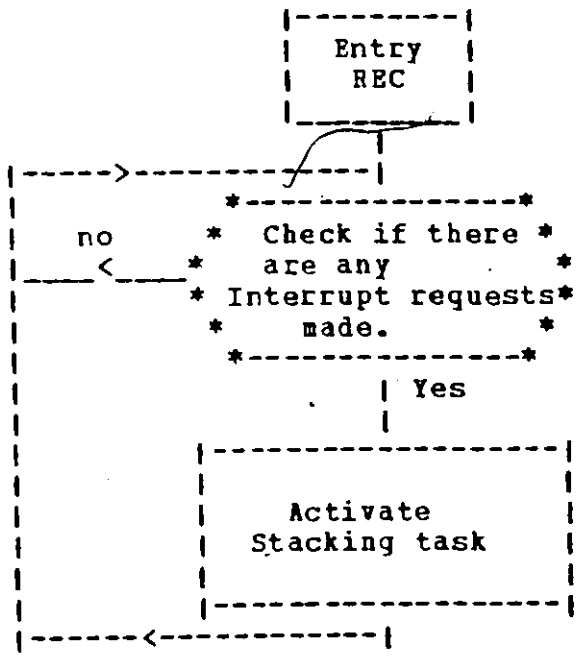
6. Post Processing Task(POST).

This task performs all operations associated with the completion of interrupt processing. Practical implementation of this task depends on the type of the multiprogramming environment(real time system,etc.). Upon completion of the I/O interrupt servicing, typical operation of this task would be to identify the process which requested I/O service, return the status of the completion of I/O(normal completion or error status), mark the completion of the I/O request in the Process Control Block and exit to the interrupted process or the

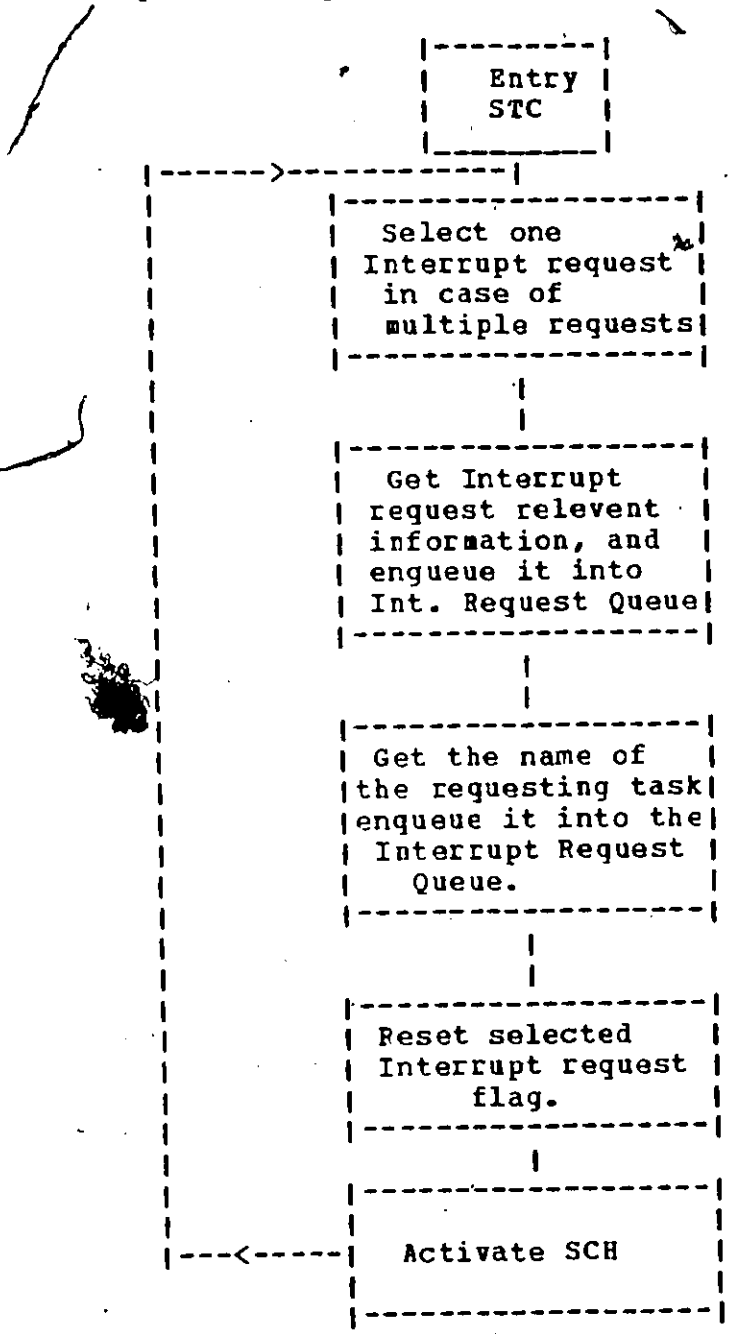
scheduler.

An algorithmic description of these tasks may be flowcharted, as shown below:

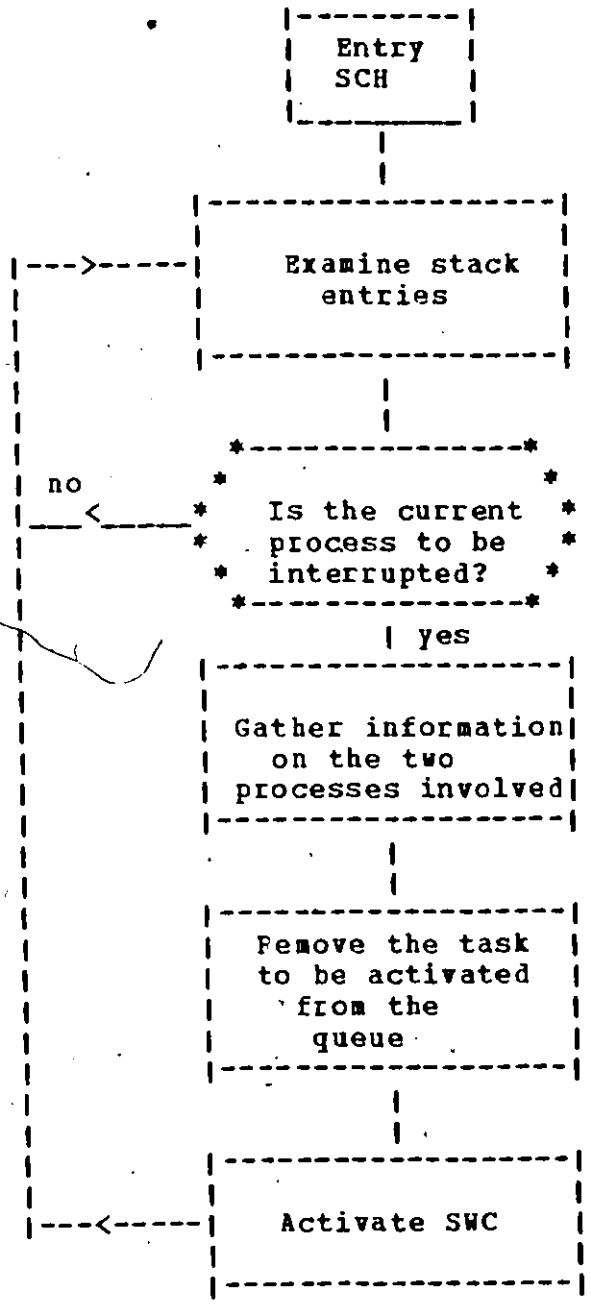
Interrupt Recognition Task:



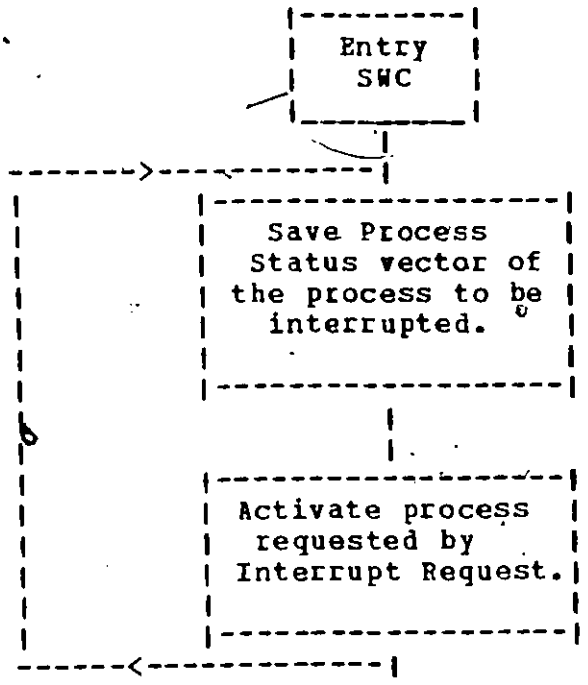
Interrupt Stacking Task:



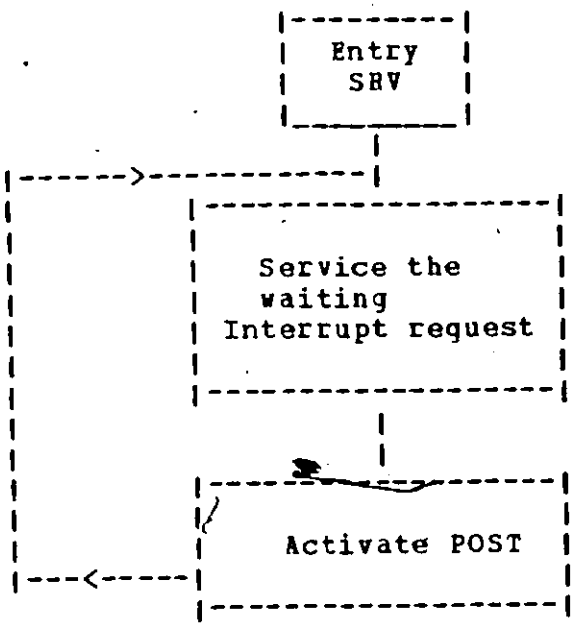
Interrupt Scheduling Task:



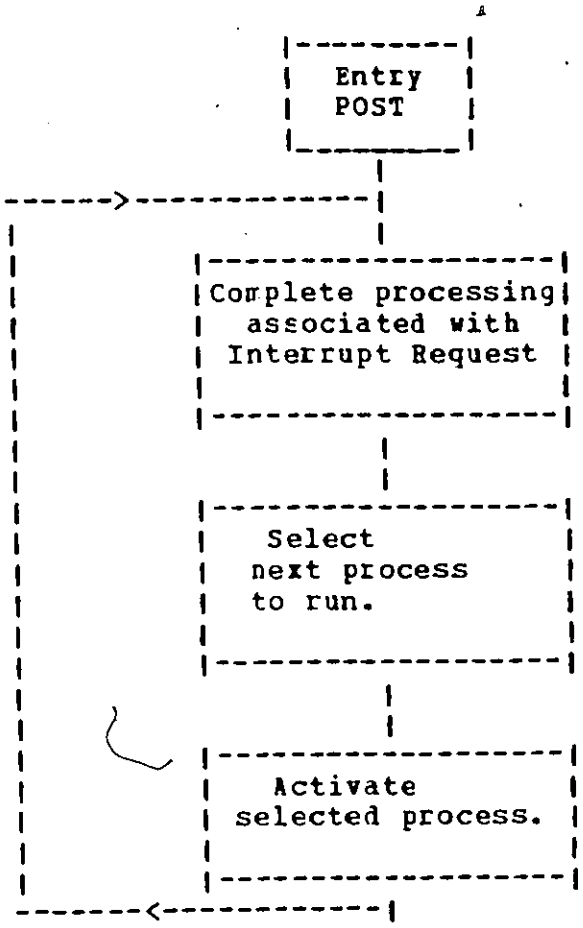
Interrupt Switching Task:



Interrupt Service Task:



Interrupt Post Processing Task:



For the simplified description of the Interrupt Activity tasks, a pseudo - Algol programming language is used (1). We define for this purpose the following symbols:

p(i)	process (i)
hreg	hardware registers image
irq	interrupt request
iarea	interrupt message area

n	number of interrupt requests
imes	message related to the interrupt
int	global interrupt request indicator
pr	priority
m	process enable mask
susp	suspended(interrupted) status
act	active status
cur	currently running process
prsa	process register save area

```
Interrupt_Recognizer:
    BEGIN DO FOR ever
        BEGIN
            /*scan all irq*/
            FOR i:=1 STEP 1 UNTIL n DO
            IF irq(i)=1 THEN
            int:=1;/*set global interr req indicator*/
        END
    END
```

```
Interrupt_Stacker:
    BEGIN WHILE int=1 DO
        BEGIN
        /*identify interrupt request*/
        FOR i:=1 STEP 1 UNTIL n DO
            BEGIN
            IF irq(i)=1 THEN DO
                BEGIN
                iarea:= imes(i)/*get interr related
                message and post it in interr area*/
                irq(i) := 0/*reset granted interr*/
                END
            END
        END
    END
```

```
Interrupt_Scheduler:
  BEGIN WHILE iarea_not_empty DO
       BEGIN
            select_one_of_the_messages;
            find_requested_process_p(req);
            IF (m(req)=1) AND (pr(cur)<pr(req))/*check
            mask and priority of requested process*/
            THEN DO
            BEGIN
                 send_the_message_tc_p(req)
                 p(cur) := p(susp)/*mark current process
                 for suspension*/
                 p(act) := p(req)/*mark requested process
                 for running*/
                 activate_switching_task
            END
       END
  END
```

```
Control_Switch:
   BEGIN IF process_marked_suspend DC
        BEGIN
        prsa(susp) := hreg /*save registers of the
             interrupted process status area*/
        hreg := prsa(act)/*reload registers with the
             new process register information*/
        END
   END
```

5.3 Processing states of the Interrupt Activity.

Each of the basic tasks of the Interrupt mechanism implemented in separate hardware circuitry can be one of two states - active or inactive. The basic tasks implemented in the software mechanism, since they must share CPU, can be in one additional state - waiting.

The Interrupt Activity mechanism as a unit in relation to other processing units in the system, can be one of the three states -

1 inactive

2 waiting

3 active

The inactive state represents the situation when no Interrupt request is received by the Interrupt processor. It means the CPU is idle or processing tasks which do not require services or co-operation of other tasks in the system.

The waiting state represents the situation when an Interrupt request is received by Interrupt processor, but did not fulfill the scheduling requirements for activation.

In the active state the Interrupt mechanism is processing(servicing) interrupt requests.

An example of the description of the interrupt system organization as shown in Figure 5-4 in the APL notation is as follows: the inactive state is defined by the expression

0 = V/IFQ (logical or) i.e. no bit set in the

Interrupt Register and INT=0. The waiting state is defined by

1 = $\bigvee$/IRQ and the INT = 0. The transition from the waiting state into the active state happens, when some of the waiting Interrupt requests are not masked.

i.e. 1 = $\bigvee$/(IRQ$\wedge$IMS), and CPU is enabled for interrupts (EN = 1). This state is represented by INT = 1, EN = 0.

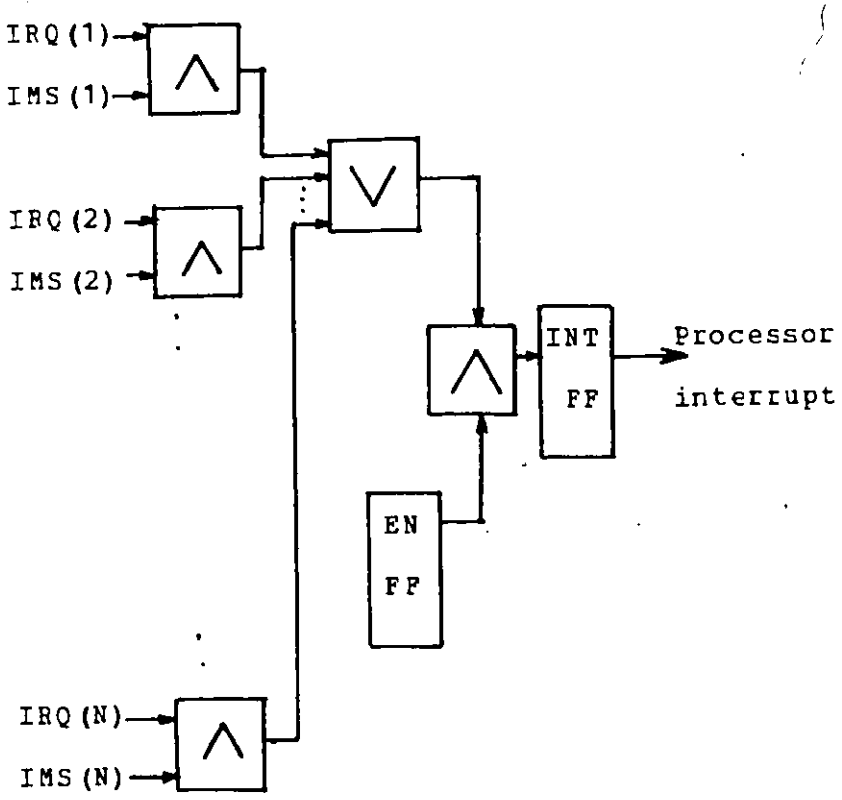


Figure 5-4. An example of an Interrupt mechanism organization.

VI. A STUDY OF SOME ASPECTS OF EFFICIENCY FOR INTERRUPT ACTIVITY IMPLEMENTATION TECHNIQUES.

6.1 Techniques for interrupt mechanism efficiency improvements.

The efficiency of the Interrupt activity implementation may be measured in terms of hardware cost, memory space required and real time overhead. It appears that with the present state of technology, the hardware cost and memory space are less important factors in the design of the interrupt mechanism than its timing characteristics. As was described in chapter 4, the parameters most often used in describing the interrupt mechanism characteristics are the response time, overhead processing time, priority and saturation criteria.

The importance of these parameters depends greatly on the application of the computer system. Present computer systems are used for a variety of types of problems. The optimization of these parameters in a system can lead to many design conflicts of the computer system. An example would be the relation between the response time of interrupts and the requirement on the number of general registers. As the number of registers goes up and monoprogram perfcrmance improves, the interrupt response time increases. Therefore, in most cases, some trade off in optimization of specific parameters is involved.

The study of interrupt mechanisms in the preceeding chapters indicates the basic techniques for optimization of these parameters. They are as follows:

1. An optimization of the operations of each of the tasks or subtasks.
2. An optimization in the scheduling of tasks or subtasks, and the enforcement of any possible parallel execution of them.
3. Enforcing possible parallel operations between the CPU and the interrupt mechanism.
4. Implementation of a larger part of the Interrupt activity into the hardware circuitry.

As described earlier, the interrupt Activity tasks are carried out partly on dedicated hardware circuitry and partly by the software routines on the CPU. Since the hardware part of the Interrupt Activity is fixed (hardwired algorithm), the scheduling of the software routines must be determined to take advantage of the existing hardware to minimize response and overhead times. Traditionally, basic interrupt tasks (Recognition task, part of the Stacking and Scheduling tasks) are implemented in dedicated hardware. The variations in interrupt service tasks prohibit an economical trade off in software for special hardware operations. The remaining tasks of the Interrupt Activity compete with other tasks of the computer system for CPU time. There is a desire to implement more of the interrupt tasks into dedicated hardware circuitry, which usually leads

to an increased cost of the system hardware. Also it is usually not obvious exactly what sort of hardware structures are needed for efficient implementation.

The basic aim of the design of the Interrupt Activity structure is to minimize response time. The interrupt response time - T(RES) in relation to basic tasks can be expressed as follows:

$$T(RES) = T(REC) + T(STC) + T(SCH) + T(SWC)$$

Each timing component of the Interrupt response time (time interval for execution of the recognition, stacking, scheduling and switching tasks) can be divided into additional timing sub-components. Examples of the important ones would be the saving of general registers operations, branching to the service routine,etc..

The purpose of this chapter is to examine which hardware/software features could improve the implementation of the Interrupt activity tasks.

6.2 Save-Restore Operations.

The most common source of overhead in interrupt processing is the saving and restoring of the machine status. Registers, the program counter and indicators must have their contents moved to a 'save' area in main storage. The new machine status must then be loaded for the interrupting (new) task before execution can commence. The procedure must be reversed when control returns to the

original task. Thus, this overhead is incurred twice per interruption. There is a need for rapid process changing facilities.

The size of this overhead is determined by the number of the registers to save and their size. Many computers have instructions for multiple storing and loading to reduce the time required to save and load the registers. Most of the present interrupt systems allow the hardware processor to be switched to the Interrupt Service routine with a minimum of state information being saved. The Interrupt Service routine can be given the responsibility of saving any other information with which it may interfere, such as various control or arithmetic registers.

The use of hardware mechanisms for automatic state switching introduces fixed overhead. Its implementation is desirable if this fixed overhead is less than the comparable variable cost in software.

To overcome the save-restore overhead, many computers provide a multiple set of registers, some for each priority level (e.g. IBM SYSTEM-7), some for each mode of operation of the computer system - for the user mode and the supervisor mode (e.g. PDP-11, Interdata 7-32). In the first case, when strict priority scheduling is used (interruption takes place only if an interrupt by higher priority task occurs), there is no need to save registers when the CPU switches to the register set corresponding to the priority level.

The same overhead saving effect is achieved in the second case, since the interrupt processor disables recognition of further interrupts during the servicing of a previous one.

In most computers the interrupt mechanism saves the status of the interrupted program in a memory area specifically reserved for the requested interrupt service task.

This technique is based on the assumption that the interrupted process will continue to run after the response to interrupt. But in general, the interrupt may cause the co-ordinator to preempt the interrupted process in favor of another process. It is then necessary to copy the execution state of the interrupted process from the locations associated with the interrupt to the process description block associated with the interrupted process. A more desirable solution is to store them in the block on the occurrence of the interrupt.

6.3 Source identification and the linkage to the service routine.

From the survey of the Interrupt structures (chapter 4), the most common procedure for the linkage of Interrupt Requests to the specific routine may be described as follows:

Each interrupt level is assigned a dedicated location in memory, to which control is automatically passed when the CPU accepts an interrupt. This location

belongs to the starting address (or has a branch instruction to it) for the centralized routine for that level, that processes and routes the Interrupt Request to the appropriate Service Routine.

The function of the central routine is to obtain the following information :

- determine which Interrupt Request on that level is seeking interrupt
- obtain the related status word

To avoid this time-consuming processing, some computers provide automatic sub-level branching (SYSTEM 7, INTERDATA 7-32). In the 7-32, special hardware circuitry is provided, which presents an encoded number for the particular request. When added to the base address of the level, this determines the address of the particular Service Routine.

Some computers provide a special status indicator ('summary' in the SYSTEM 7). If it is zero, the interruption is due to normal error and a no error exception condition is encountered.

6.4 Re-entrancy.

The concept of a priority interrupt structure in computer systems is very closely related to that of re-entrancy. In an environment where interrupts occur at random, it often happens that the routine which has been interrupted will be used by the interrupting program. In

such cases it is not sufficient that the interrupt structure (hardware and software combined) preserve only the registers and the central processor status of the interrupted program. It is also neccessary that the intermediate data created by the program is saved.

The entry into a typical subroutine requires that the parameters of the call and the return be supplied to the subroutine by modifying the addresses in the subroutine body itself. If a strictly ordered priority control scheme is used, the problem is resolved by having an explicit copy of the subroutine for every level that might require it. Such an arrangement could require a large amount of memory.

Another alternative is to burden the system with a complex executive that examines the status of every subroutine interrupted, and if necessary, stores and restores the calling sequence data. This however is not practical.

A further alternative, requiring slightly more time but less space than the first, is to store only the calling sequence and work area of the subroutine for each level that requires it. The subroutine body is never modified. It can be entered at any point, without fear of damage. The subroutine body is composed of 'pure code' (i.e., it contains only instructions and constants) and is said to be re-entrant. The calling sequence data are stored in a calling sequence area for that priority level.

There are two features which greatly facilitate the

implementing of re-entrancy. The first feature is the possibility of having multiple blocks of general purpose CPU registers. When an interrupt occurs, a change to a different block of registers takes place.

The second feature is the presence of the 'push down' facilities. These provide the capability, by the execution of one instruction, of saving all general registers in a reserved stack area that is maintained by the system. These facilities also allow the saving of the contents of the memory locations, in addition to the general registers, which have been set to some intermediate value by a routine.

Computers using stacking mechanisms seem to demonstrate a clear advantage in nesting. Interrupts can be nested in much the same manner that subroutines are nested. In fact, it is possible to nest any arbitrary mixture of subroutines and interrupts. This is desirable because of the random nature of interrupts. Use of the pushdown stack in the handling of interrupts and subroutine calls, results in the structure of the temporary memory area as shown in Figure 6-1.

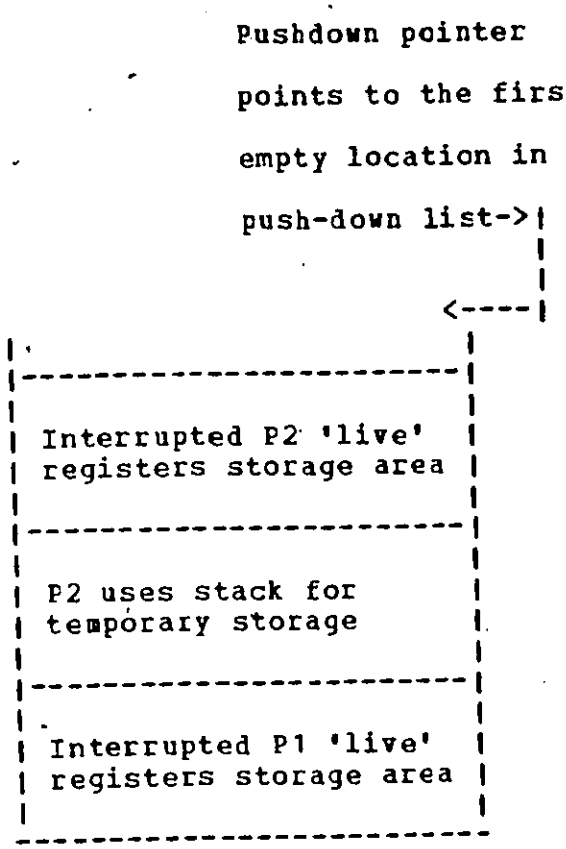


Figure 6-1. LIFO storage organization for the saving of the interrupted process's 'active' status information.

6.5 Number of Interrupt priority levels.

The number of hardware Interrupt priority levels implemented in computers differs. When more hardware levels are present, the price of the system is higher. With more levels implemented, more memory will be used for the dedicated areas. Common rules for Interrupt priority scheduling are:

1. An interrupt at a given level will nct interrupt the

processes underway at that or higher levels and

2. the Interrupt mechanism at each level will process all other interrupts at that level which were activated prior to the completion of the given interrupt before returning to the levels below.

When these rules are used, we can compare the effects of various numbers of interrupt levels, provided that we know the incident interrupt rates for each level, the processing time for each interrupt, and the non-interrupt processing that must go on at that level.

Each level can be considered as a state. By representing levels (states) as nodes, and the transitions caused by interrupts as links, we can represent a system in a state diagram, in the manner shown in Figure 6-2 . At any given instant of time, the system is in one state. It remains in that state (the self transition) until one of two things happen:

1. there is nothing more to do in that state, in which case it executes a transition to a lower state, or
2. an interrupt for a higher level occurs, in which case it executes a transition to a higher level state.

Associated with each upward transition is the transition cost equal to the cost of terminating the operation in the lower state and starting the operation in the higher state. Associated with each downward transition is the equivalent cost for the opposite direction.

It is evident that the effect of larger numbers of

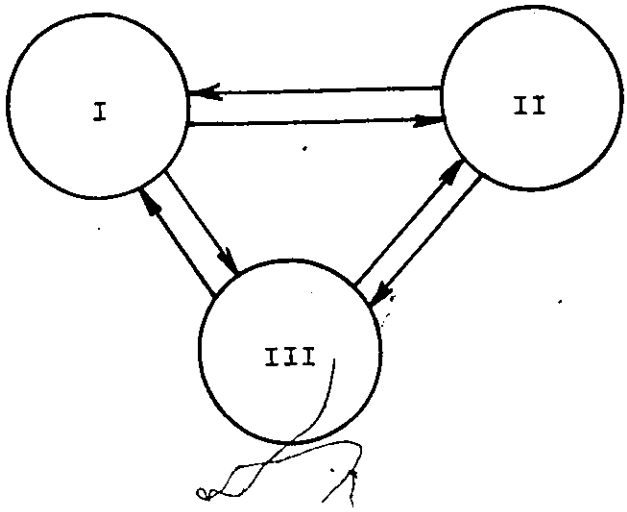


Figure 6-2. A state diagram of the interrupt priority levels processing.

Interrupt levels will be:

1. increased overhead time. Each level in the system will force futher increases in the number of transitions that will occur and reduce batch processing in each level
2. decreased Response Time for high priority processes at the expense of low-priority processes. High-priority processes will be resident for shorter periods, while low-priority functions will reside for longer periods.

If a level must be split into sub-levels, there still remains a choice as to whether hardware or software should be used for that purpose. If it is done by software, we must include the transitions, and compare these with the fraction

of the system facilities used for these additional transitions. If it is done by hardware, the cost is determined from the manufacture's catalogue.

6.6 System Saturation.

It is difficult to determine all the parameters for investigating the system saturation state. The detailed examination of this problem is investigated by means of queueing theory and requires a separate study. For the purpose of this study, we will introduce the following simplifying assumptions and parameters about an interrupt system, which has N Interrupt Request sources, numbered 1 to N in decreasing order of priority:

1. Interrupts occur relatively uniformly , with time interval TD(I) for the interrupt source I.
2. TC(I) is the crisis time for the source I
3. TS(I) is the time needed to execute the Interrupt Service routine for source I, plus involved overhead with it.
4. The interrupt mechanism is based on the single line priority structure with no interrupt recognition while one is being processed.

The Interrupt Activity mechanism will not reach saturation state, and will service properly each Interrupt Request, if two conditions are satisfied.

The first condition is :

The central processor must be fast enough to process all the necessary Interrupt Routines.

For each source I, the number of interrupts per second will be 1/TD(I). The time spent servicing this request will be TS(I)/TD(I). Since all the sources have to be serviced, the condition states:

$$\sum_{I=1}^{N} \frac{TS(I)}{TD(I)} < 1$$

The second condition states:

In no case should the waiting time for any source exceed its crisis time.

The worst case for any Interrupt Request IR(I) will occur if all the interrupts demand attention simultaneously, just after the control processor has started the longest interrupt routine, lasting TS(MAX). Interrupt Request I will have to wait for this routine to finish, and for all I-1 interrupts of higher priority to be dealt before it receives attention. The relation is:

$$TS(MAX) + \sum_{L=1}^{I} TS(L) < TC(I) \quad \text{for} \quad 1 \leq I \leq N$$

6.7 Scheduling of the Interrupt Activity tasks.

The simplest arrangement of the Interrupt Activity tasks processing is sequential (sequencing through each task servicing single interrupt).

Most of the systems have their interrupt recognition procedure implemeted by a hardware interrogating circuitry. Different computers exhibit variations in the hardware implementation of the additional tasks of the interrupt activities. The scheduling of the tasks implemented in software must be determined to take advantage of the existing hardware to decrease the overhead.

A typical Interrupt Activity program will stack one interrupt and then check to see if another interrupt has come in at that level. Also, most of the systems, after servicing one interrupt and before switching back to the interrupted program, check to see if another interrupt request occurred while servicing the previous one. Some computers perform the stacking task and the switching task simultaneously (example System 7). The diagrams in Figures 6-3 and 6-4 show in a simplified manner examples of scheduling of Interrupt activity tasks.

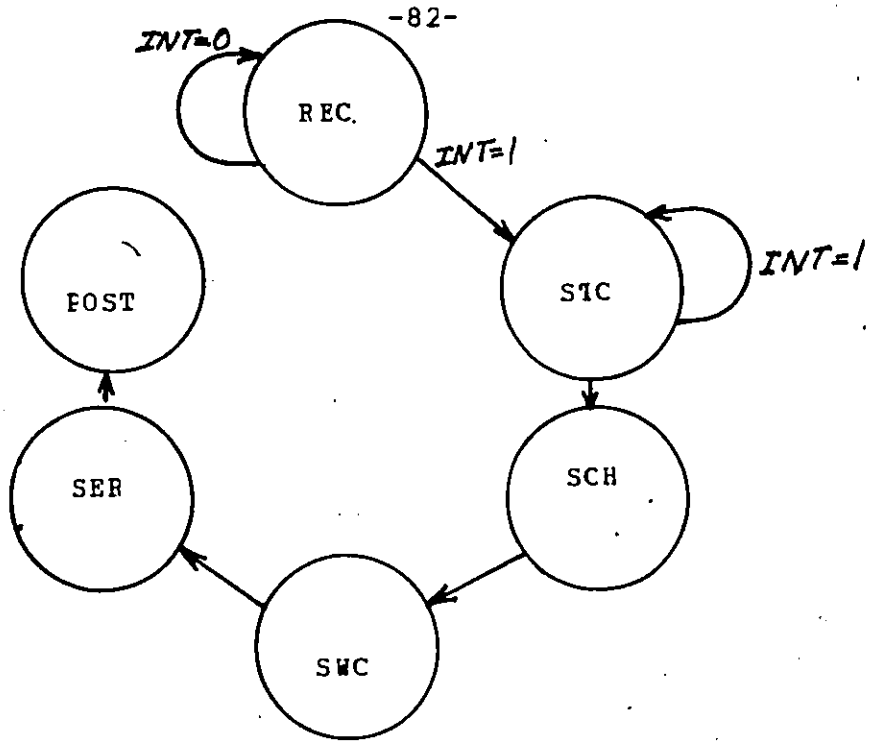


Figure 6-3. State diagram of the execution of the Interrupt tasks by simple sequencing.

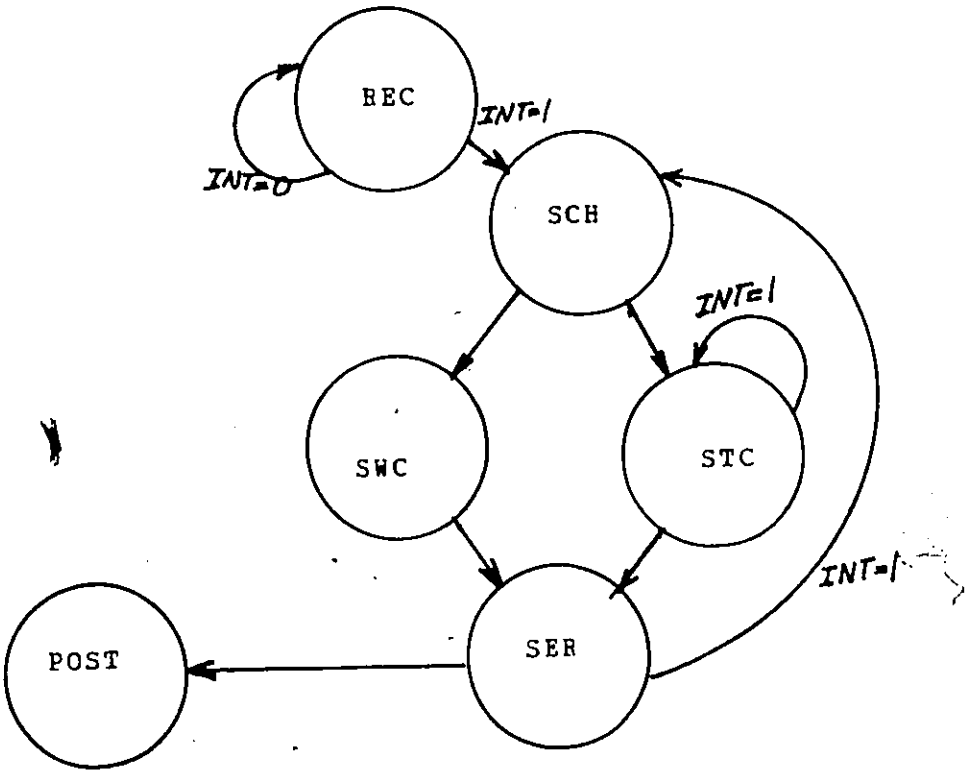


Figure 6-4. State diagram of the execution of the

Interrupt tasks by modified sequencing.

The desire to implement more of the Interrupt Activity mechanism into a hardware circuitry is easier in the computers with microprogramming technique control. The microprogramming technique offers more speed in comparison with software. It also offers more flexibility than conventional(hardwired) control because design changes are no longer costly to implement(22,23).

The above described techniques do not place enough importance on solving the problem of parallel operations of the CPU and the Interrupt Activity mechanism. The I/O instructions in the computer systems are usually the most time consuming instructions, and as a result, the CPU can spend a large amount of time getting Interrupt related information from the I/O interface.

The enforcing of parallel operations between the CPU and the Interrupt Activity mechanism is more or less a technological problem. It appears that associative memory techniques can be applied here for storing of interrupts and performing the basic processing on them. In a large multi-user computer system(24), a queued content addressable memory was used for the implementation of the interrupt mechanism. This unit forms a primary storage and processing facility for interrupt requests. Such an arrangement reduces greatly the time spent by the CPU for the processing of interrupt activities.

Another problem in Interrupt processing is the frequency of their occurrence. As was described in chapter 5 ,the scheduling sub-task constantly examines the scheduling parameters of interrupt requests and compares them with the scheduling parameters of the currently running process. Since most of the device interrupts are of high priority, there is large amount of switching overhead activity performed in the system. A possible limitation on the switching overhead activities can be made if the scheduling sub-task would be active only at certain intervals. This task can regularly examine the interrupt requests messsage queue at a frequency great enough to satisfy system response requirements.

To allow a processor to make reasonable progress on its scheduled workload, the external response requirements must also be re-adjusted in terms of time. Otherwise, a processor could be endlessly examining the request queue. By distributing more control power into I/O devices data loss can be avoided. With the present technology, it is possible to do this. Using the 'Large Scale Integrated Circuit' technology memories and microprocessors we should be able to limit the demand for processor service to some reasonable level that an operating system can handle. This and other similar systems could remove the need to interrupt the CPU processing except for critical errors.

VII. CRITICAL OVERVIEW OF THE EFFECT OF INTERRUPT FEATURES ON GENERAL INTERPROCESS COMMUNICATION FACILITIES.

7.1 Basic Process Intercommunication Functions.

As was described in the introduction, the basic aspects concerning Interprocess communication are mutual exclusion and synchronization problems. In the case of the mutual exclusion problem, there is a restriction on the interaction of processes, if a resource is used by one process, all other processes must be delayed (temporarily halted) from using the resource, until this resource becomes free.

In the case of the synchronization problem, one process must be delayed at a given point of execution until some event under control of another process has occurred.

The mechanism which provides these control functions in the computer system is often called the Interprocess Communication Facility (4,16,25,..).

The basic requirement for the Interprocess communication Facility is then to give the processes the following abilities:

1. A process should be able to become blocked until some special operation is (or has been), executed by another process.
2. A process should be able to execute a special operation which will cause a blocked process to be no

longer blocked (wakeup capability).

3. It must be possible for processes to gain exlusive control of a shared resource.

There are a number of mechanisms to provide the above capabilities. The Interrupt mechanism is one of them. A purpose of this chapter is to examine facilities, which in cooperation with the Interrupt mechanism provide interprocess communication functions in the multiprogramming systems.

7.2 Timing constraints of Interrupts on Interprocess Communication.

In current computer systems, there is another form of communication between tasks, in addition to the Interrupt mechanism. This method is based on reading and writing into common data areas (shared data). The main objective of these two methods is to co-operate in such a way that the activities in the computer system will be carried out in the proper way. It is difficult to analyze all the possible interactions of processes under these communication methods and prove their validity. There are some evident cases, where the processes will get into improper (interference) states, as is illustrated in these examples:

Consider a typical situation where two processes are sharing a common data base. When a process is going to access the common data base, it first checks the associated flag(lock). If the lock is not set the process sets the lock, then accesses and modifies the data base. After modification the lock is cleared.

The typical operation of testing a flag takes two instructions, one to load it in a register, and another to perform a test and a conditional branch on it. An example of the code involved on an Interdata M-4 computer is :

```
1) WAIT    LH R1,LOCK        GET THE LOCK VALUE
 2)        BNZ WAIT          WAIT ON ZERO LOCK VALUE
 3)        LHI R2,X'FFFF'    GET LOCK 'ON' VALUE
 4)        STH R2,LOCK       SET LOCK ON
           .....             MODIFY DATA BASE
```

By execution of this code the following sequence of events can occur:

1. Statement 1 is executed - the flag is read (which is unset), but before the test on the flag could be executed (i.e statement 2) the program is halted by the occurence of an interrupt.
2. The new program P2 now runs, finds the flag unset, then sets it and starts the I/O data base modification.
3. The old program P1 resumes operation, tests the flag that was read before the interrupt, finds it unset and proceeds to start the data base modification.

Similar 'interference problems' can occur by the following sequence of events between two processes:

1. Process P1, wishing to change the shared data, tests the associated flag and finds it unset, but is halted before it can set that flag to indicate busy status (statement 4).
2. Process P2 tests the flag, finds it unset, sets it and

starts altering the data area.

3. Before the changes are completed, P2 is halted (interrupted) and P1 is restarted. P1 executes the next piece of code; sets the flag and starts to change the data area.

The results in both cases are unpredictable, and probably unreproducible, since they depend on a critical time sequence, which is unlikely to be repeated.

These conflicts arise because processes P1 and P2 have no control over the Interrupt System. Interrupts are random, unpredictable in time, and hence the central processor can be interrupted at each instruction cycle.

These are some of the examples which show that one can not implement a multiprogramming system without ensuring a mutual exclusion of tasks during the inspection of common variables in order to achieve proper sharing of computing resources among processes.

The task of the designer of a multiprogramming system is to design mechanisms which will control communication among processes and prevent interference with each other.

The simplest method of a protecting critical sections of process from interference is to disable the occurrence of interrupts (process resets enable flag 'EN=0').

However, it is difficult to allow every process to have control over an interrupt mechanism for the following reasons:

1. A process can disable interrupts for such periods, that the systems response time requirements may lead to loss of data.
2. This technique is unnecessarily restrictive, since other unrelated processes are also prevented from running.

For these reasons, most of the systems have the so called supervisor mode of operation, in which the interaction code is executed as a high priority task. This solution is more satisfactory then the previous one, since high priority interrupts can be handled momentatily.

Some computers provide hardware instructions (primitives), for manipulating such global variables, as an indivisible operation. IBM 360 provides a 'TEST and SET' instruction. By execution of this instruction, the lock value from the memory is read and, on the same cycle that the most significant bit is set, it is written back. The condition code reflects the state of the lock. An example of its use:

```
WAIT    TS LOCK       TEST AND SET LOCK
        BC 1,WAIT     BRANCH IF MOST SIGNIF BIT IS SET
        ......
        ......        MODIFY DATA BASE
```

Sigma computers use an instruction 'XW' -exchange word

between memory and the CPU register, by which the proper and secure manipulating of global variables can be achieved. An example of such a sequence:

1. Preset register with the desired (lock) value.
2. Exchange this value with the value of the tested variable.
3. Perform the conditional branch.

In the case that this process is interrupted after statement 2, the new process always finds the global variable in a locked state.

The 'Test and Set' instruction or any similar type of instructions still do not produce a secure solution of mutual exclusion and synchronization problems.

One of the reasons is the danger that, when a process has already tested and passed on a global variable, it could be interrupted. In this case the 'lock' variable will stay 'locked', which will prevent other processes from the use of the associated facilities for an indefinite time.

Another reason for not using 'TS' instruction is that a process when testing and looping on the 'lock' could occupy the CPU for long periods of time. A more desirable solution would allow the processor to be pre-empted and reassigned. For these reasons the above types of instructions are not available to the users. They are instead used in system routines called by the interrupts which could be pre-empted by the processor if the variable is in a 'locked' state.

An example of such usage is in the Interdata OS/32 MT

operating system. The user task executes a supervisor call interrupt instruction supplying appropriate parameters to indicate the 'Test and set' function request.

The executive performs this function on behalf of the requesting task and decides on the allocation of the CPU according to the results of this test.

7.3 Design objectives for the Interprocess Communication Facility.

The Interprocess Communication mechanism should offer the following features(1,2,4,25,..):

1. It should provide a uniform method of communication between processes without restricting the number of communicating processes and without differentiating between types of events(e.g., hardware interrupts, software interrupts).
2. It should offer to a programmer the oportunity to design a code for a complex multi - process computation without a disproportionate attendant growth in complexity.
3. It should provide efficient hardware processor resource management.
4. It should have modular implementation and simplicity in its appplications.

From the discussion in the previous chapters of the typical multiprogamming systems, we can identify some of the basic features of the communication between elements and the structure of the multiprogramming system. These are as follows:

a. The various communication techniques used among the components of the system. The Interprocess Communication mechanism in typical present computer systems consists of these parts:

   1. Interrupt System.
   2. Lock/unlock mechanism
   3. Timer.

   The user processes communicate with the system processes via the Interrupt system, while the system and user processes communicate with each other via shared tables, queues and common variables.

b. Two modes of operations, variously named master/slave, supervisor /user mode, operating system /user mode... 'Enter' and 'Exit' to the operating system mode is associated with the new Process Status Vector, which either inhibit the interrupts entirely or enable interrupts of the highest priority to occur. Processes in user mode can not use 'privileged' instructions as for example:

   -instuctions to enable/disable interrupts

   -instructions for input/output devices

   -instructions to change the state of the processor

By building systems with all these varied communication techniques, we overcomplicate the operating systems and user programs design. The overall design of the systems tends to become obscure, and maintenance and system testing become exceedingly difficult. The communication facilities within the sytem, should be as uniform, simple, and as few in number as possible to create a useful and efficient computing tool. Considerable work has been done on alternate facilities for interprocess communication.

7.4 An Interprocess Communication Model.

A multiprogramming system chosen for the development of this interprocess communication model is composed of a set of processes P(1),P(2),.....P(N), which compute in parallel at arbitrary speeds.

The processes communicate among themselves through a special process, that we shall call the Interrupt Link - ILINK. The ILINK is based on the generalized Interrupt Activity mechanism developed in chapter 5.

A schematic organizational representation of such a system is shown in Figure 7.1. The Interrupt servicing task is not included in ILINK and is considered as one of P(I). This is different from the generalized Interrupt Activity Mechanism (developed in chapter 5), where the Interrupt service tasks were included in the Interrupt mechanism and treated

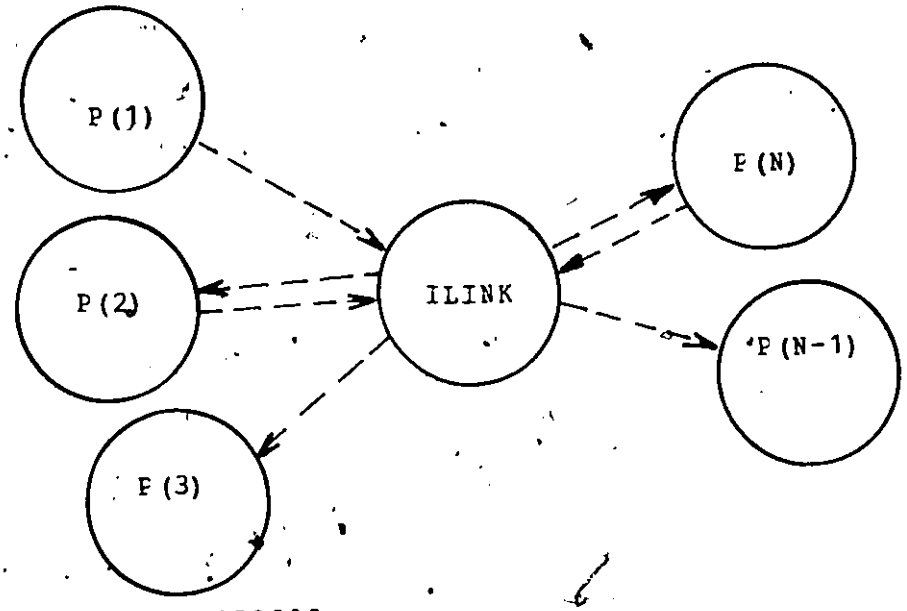


Figure 7.1. Organization of a system with ILINK as an interprocess communication mechanism.

seperately from the system's normal process-scheduling and priority assigning functions. From the previous analysis, of the way in which the interrupts signal the completion of activities, and the resulting action to be taken on account of such completion, it is considered desirable for the interrupt service task to be executed in co-operation with the other processes of the system.

The activities of the ILINK may be grouped into two major tasks as follows:

A. Interrupt Handler

B. Interrupt processor

The following algorithms describe functions of the Interrupt Handler and the Interrupt Processor:

```
Interrupt_Handler:
    BEGIN DO FOR ever
        IF any_interrupt_request THEN DO
        BEGIN
            select_and_identify_one;
            get_asscciated_message;
            record_it_into_communication_area;
            reset_granted_request;
            activate_interrupt_processor;
        END
    END

Interrupt_Processsor:
    BEGIN DO WHILE communication_area_contains_message;
        BEGIN
            apply_selection_algorithm_and_select_
            one_of_the_messages;
            do_appropriate_basic_processing_on_the_messag;
            send_the_message_to_the_waiting_process;
            delete_the_message_from_comm_area;
        END
    END
```

The realization of this model in present computer systems would require the modification described in the section 6.7. In present computer systems, scheduling of the interrupt service tasks is based on the crisis time. If the service task is not scheduled within the crisis time, the system can lose data from the I/O devices. The proposed modification (section 6.7) placing more control in I/O devices could prolong or disregard the crisis time as a limited factor for the organization of the software system.

From the analysis of the the proposed system some possible disadvantages may be identified in comparison with the system described earlier (chapter 5):

1. Possible decrease of the utilization of the hardware system resources. The immediate initiation of peripheral operations after receiving Interrupt request is prolonged, since the requested process must compete with other processes of the system for re-scheduling.
2. A possible small increase in overhead of the processor switching activity, since Interrupt service processes have to be associated with the Process Descriptor Block, that can prolong the time to switch the hardware processor to the interrupt service process.

The advantage of such a system is in the simplicity of the communication procedure between processes. This results in ease of the design of an operating system (as was described in preceeding chapter). Also, if the interrupt

response requirements could be re-adjusted to a longer time periods, the global processor overhead will be deacreased.

Many types of communication primitives were developed. Dijkstra has developed a generalized, unified approach that treats both the synchronization of parallel activities and the interlocking of mutually exclusive code sequences. He used this in the design of 'THE Multiprogramming system'(8). Dijkstra's method invclves the use of special variables called semaphores that are common to a number of processes(activities). There are only two operations that can be carried out on a semaphores- namely P- and V- operations. P operation represents potential delay; if an attempt is made to perform a P operation a on semaphore that has negative value, the process making the attempt will be held up until some other process performs a V- operation on that semaphore. The process executing V operation on that semaphore, finding negative value, will activate a waiting process. Since interrupts are basically process activation signals, involving possible reallocation of the CPU to the activated process, we can consider interrupts as V operations.

Communication among processes in the Multics is achieved by an exchange of messages via a commonly accessible mailbox (shared data base) whose identity is known to each process by common convention(25). Processes communicate among themselves by notify and wait event oriented primitives

which operate on mailboxes.

The multiprogramming system, which has been written on tthe computer RC 400 by Hansen(5) is based on system nucleus. The nucleus incorporates a mechanism for interprocess communication, which is similar to the ILINK mechanism proposed in this section.

VIII. RESULTS AND CONCLUSION.

The major objective of this study was to examine the problem of interprocess communication and processor sharing among processes by means of an Interrupt Activity mechanism in the multiprogramming environment. The examination was focussed on the efficiency aspect of such a mechanism and its influence on the organization of the software facilities in the computer system. For this purpose, a detailed study of some implementation schemes was undertaken. On the basis of this study, basic components of the interrupt mechanism were identified, a general model of the Interrupt Activity mechanism was developed, and the efficiency and organizational aspects investigated.

The results indicate that while logical functions, such as, interrupt recognition, stacking, scheduling, control switching and post processing are easily identifiable, their hardware implementation exhibit variations which have significant influence on process switching flexibility.

The design of the hardware interrupt mechanism in computer systems is also facing many design conflicts, for example, the relation between the response time of interrupts and the requirement on a number of general registers. As the number of registers goes up and monoprogram performance improves, the interrupt response time deteriorates because of the time required to save state information for the running process. The provision of multiple sets of registers that switch

automatically as each interrupt occurs requires more control software. Thus the optimum interrupt structure should be evaluated based upon the particular application intended for the computer system.

The following are some of the identified built-in hardware features, which are important in determining the overhead time spent in the multiplexing functions of the operating system:

- push-up,push-down operations with automatic addressing
- multiple register set
- hardware priority network
- the ability to individually inhibit or 'mask' any interrupt request
- size of the processor status vector
- use of microprogramming techniques
- separation of the I/O control and operation from the operation of the CPU, therefore removing the need to interrupt CPU processing

The Interrupt mechanism features complicate the interprocess communication aspects of the multiprogramming system since interrupts are random (unpredictable in time). The active process can be interrupted at each instruction cycle, which can often lead to deadlock or other irreproducible error situations. Most of the present systems therefore treat interrupts by a special mode of the

operation (supervisor mode) and do not allow user programs to manipulate the interrupt system.

An Interprocess Communication mechanism has been proposed. This mechanism will provide uniform communication among processes and conceptually simplify the structure of the software system. The realization of this mechanism would require some modifications in the computer hardware architecture, as for example distribution of more control power into I/O devices. This could be a subject for further study.

IX. REFERENCES.

1 Dijkstra, E., Hierarchical ordering of sequential processes, Acta Informica, 1, 115-138,1971.

2 Goldberg, R.P., Survey of Virtual Machine Research, Computer, 7, 6, 34-45, 1974.

3 Wirth, N., On Multiprogramming, machine coding and computer organization, Comm. ACM 12,9, 489-498, 1969.

4 Saltzer, J.H., Traffic control in multiplexed computer system., PhD Dissertation, Massachusetts Institute of Technology, 1969.

5 Brinch Hansen, P., The Nucleus of a Multiprogramming system, Ccmm. ACM 12,4, 293-242,1970.

6 Lampson, B.W., A scheduling philosophy for multiprocessing systems, Comm. ACM 11,5,347-360, 1968.

7 Liskow, B.H., The design of the Venus operating system, Ccmm.ACM 15,3,144-149, 1972.

8 Dijkstra, E.W., The structure of THE multiprogramming system, Comm.ACM 11,5,341-346,1968.

9 Tsichlitzic D.C. and Bernstein P.A., Operating Systems, Academic Press 1974.

10 System 360 operating system, supervisor and data management services, IBM, White Plains, N.Y., 1968.

11 OS-32 MT System Description, Interdata Inc., 1974.

12 Yordon E., Design of on-line computer systems, Prentice-Hall,Inc.,1972

13 Borgers E.P., Characteristics of priority interrupts, Datamation, 6, 31-34, 1965.

14 Bock B.V., An interrupt control for the B 5000 data processor system, Proc. FJCC ,229 - 241,1963.

15 Introduction to programming PDP-8 family computers, Digital Equipment Corporation, 1970.

16 Watson R.W. Timesharing system design concept, McGraw - Hill Bock Company.

17 How to use the NOVA computers, Data General Corporation, 1970.

18 32 bit series reference manual, Interdata Inc.,1974.

19 PDP 11/45 Processor handbook, Digital Equipment Corporation,1971.

20 Mendelson M.J., and England A.W., The SDS Sigma 7: A real-time time-sharing computer, Proc.FJCC,51-63,1966.

21 Davis M.I., Processor interrupt scheme speeds response and minimizes overhead, EDN, 9,49-53,1972.

22 Jones L.H and Mervin R.E., Trends in Microprogramming: A second reading, IEEE Transaction on Computers, 8, 754-759, 1974.

23 Aqrawala A.K. and Rauschen T.G., Microprogramming: perspective and status, IEEE Transaction on Computers, 8, 817-835, 1974.

24 Erwin J.D. and Jensen E.D., Interrupt processing with queued content adressable memories, Proc. FJCC.,621-627, 1970.

25 Spier M.J. and Organick E.I., The Multics Interprocess Communication Facility, Second Symposium on Operating Systems principles, 83-91, 1969.

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA