SYSTEM DESIGN FOR A MULTI-USER MICROPROCESSOR-
BASED APPLICATION

By

NEENENDRA R. PANDYA

Bachelor of Engineering

Bangalore University

Bangalore, India

1979

Submitted to the Faculty of the Graduate College
of the Oklahoma State University
in partial fulfillment of the requirements
for the Degree of
MASTER OF SCIENCE
December, 1981

SYSTEM DESIGN FOR A MULTI-USER MICROPROCESSOR-

BASED APPLICATION

Report Approved:

[illegible]

Report Adviser

[illegible]

[illegible]

Norman N Durham

Dean of the Graduate College

ACKNOWLEDGMENTS

I am grateful to my principal adviser, Dr. John R. Phillips, for his constant guidance and willing help throughout the course of my study.

I would also like to thank my committee members, Dr. G. E. Hedrick and Dr. M. Folk, for their encouragement and assistance. I would like to express my gratitude to Dr. D. D. Fisher, head of the department, for his advice and guidance.

I thank Mrs. Grayce Wynd for her excellent typing of the report.

I also wish to express my sincere appreciation to Dr. Meg Kletke for being very helpful and considerate during my position as a Graduate Research Assistant in the Department of Agricultural Economics.

I am deeply indebted to my parents, brothers and sister for their constant inspiration, encouragement and guidance; they have made all of my achievements possible.

TABLE OF CONTENTS

TABLE

LIST OF FIGURES

Figure Page

LIST OF ACRONYMS

CRT	- Cathode-ray tube
DMAC	- Direct memory access controller
I/O	- Input/output
iSBC	- Intel single board computer
MULTIBUS	- Intel single board computers' system interface bus
OS	- Operating system
PIC	- Programmable interrupt controller
PIT	- Programmable interval timer
PPI	- Programmable peripheral interface
RAM	- Random access memory
ROM	- Read-only memory
UART	- Universal asynchronous receiver-transmitter
USART	- Universal synchronous and asynchronous receiver-transmitter
μC	- Microcomputer
μP	- Microprocessor

LIST OF DEFINITIONS

Allocation technique - The method of providng a process with access to a shared resource.

Communcation expansion board - A board containing hardware to provide expansion of systems communications capability.

Direct memory access controller - A specialized processor designed to perform high speed data transfers between memory and the device.

DISKIO - The disk input/output module of the RMX/80 operating system.

Disk sector - Fixed length sections that a track is divided into on a disk.

Dynamic load - Refers to an electrical circuit where events occur that vary in amplitude and duration with time.

Expansion boards - Boards that provide system expansion and support through CPU, memory, and I/O boards.

Extent - A collection of several contiguous disk sectors.

IBM 5440-type disk drive - Top-load, cartridge-type disk drive with a controller drive interface, mechanically sectioned for 12 sectors.

Interrupt level - A unique level at which the occurrence of an interrupt can be identified.

Logic state analyzer - Used in debugging software and complex software-hardware faults. Presents the flow of the system's program by monitoring all important circuit points.

Master-slave programmable interrupt controller (PIC) configuration - A configuration used to handle more than eight external devices. One PIC is at a higher level than the others; it is called the master, and the others are called slaves.

Memory protection - A method of ensuring that the contents of main memory within certain variable limits are not altered or inadvertently destroyed.

Networking - Sharing and accessing of information between computer systems.

Non-contiguous allocation - An allocation method that assigns physically nonadjacent sectors to a file.

Object-oriented architecture - Makes an operating system easier to understand. It uses objects (data structures) as building blocks that the operators (system calls) manipulate.

Primitive - Operation provided by an operating system nucleus for use in synchronization.

Programmable interrupt controller (PIC) - A programmable device that handles interrupt requests from more than one external device.

Resource - Any device or item used by a computer, including special areas of memory such as buffers.

RMX/80 disk file system - A module of the RMX/80 operating system that provides for the filing and retrieving of data using disks.

RS232C - An Electronics Industry Association (EIA) standard that covers the electrical specifications for bit-serial transmission, as well as the physical specifications.

Serial input/output - Required for certain devices such as teletype, tape and disk communication. Each time a bit is passed on a single line according to a serial standard.

Time-slicing - A technique that shares processor time among several processes. The quantum of time allocated to a process is termed a time slice.

Universal asynchronous receiver-transmitter (UART) - A serial-to-parallel and parallel-to-serial converter.

Universal flexible diskette controller - A diskette controller capable of supporting virtually any software - sectored, single density diskette drive.

CHAPTER I

INTRODUCTION

Objective

This report develops a paper system design for a multi-user microprocessor based application. It gives the novice designer a feel for approaching and tackling the task of microcomputer system design. A few detailed applications illustrate the design process. None of the examples presented is intended to be a complete or specific design. A result of this report is guidelines for the systematic approach to the design process. Such an approach greatly simplifies the process of system design. Although terms are explained at a general level, this report deals with specifics quite often to elucidate certain design criteria.

Project Overview

When a designer embarks on a microcomputer system design project, he should take a systematic approach to the design process. In the section on "Approach to System Design," hints about hardware and software design are presented. Examples using the INTEL 8086 CPU with PL/M-86 as the application language are used throughout the design process. Use of diagnostic routines and discipline in software are also explained. Finally, the importance of economics, testing, and system integration in design are given.

The chapter on the hardware system first distinguishes essential hardware functions needed in the design. Specific boards to implement these functions are then selected along with the justification of their selection. The advantages of the processor and language selected are also given.

In the software systems chapter, first basic terminology used in operating systems is explained and the keypoints in selecting an operating system are given. The keypoints indicate the usefulness and power of an operating system. This section gives the designer a handle on selecting the operating system he needs. In the next section, a specific operating system is selected based on the points in the previous section. The selected operating system is shown to be very suitable to our design requirements. Considerations for reentrant programming are also presented. When and how procedures can be declared reentrant in a high-level language PL/M-86 for the INTEL 8086 chip are specified. These points would prove useful to a designer using PL/M-86 as the design language for his application.

After having defined the software and hardware systems, a few design applications are presented in sufficient detail to make the system design process clearer. The applications consist of a real-time alarm system and a text editor. Depending upon the application, the hardware components chosen earlier may not all be necessary.

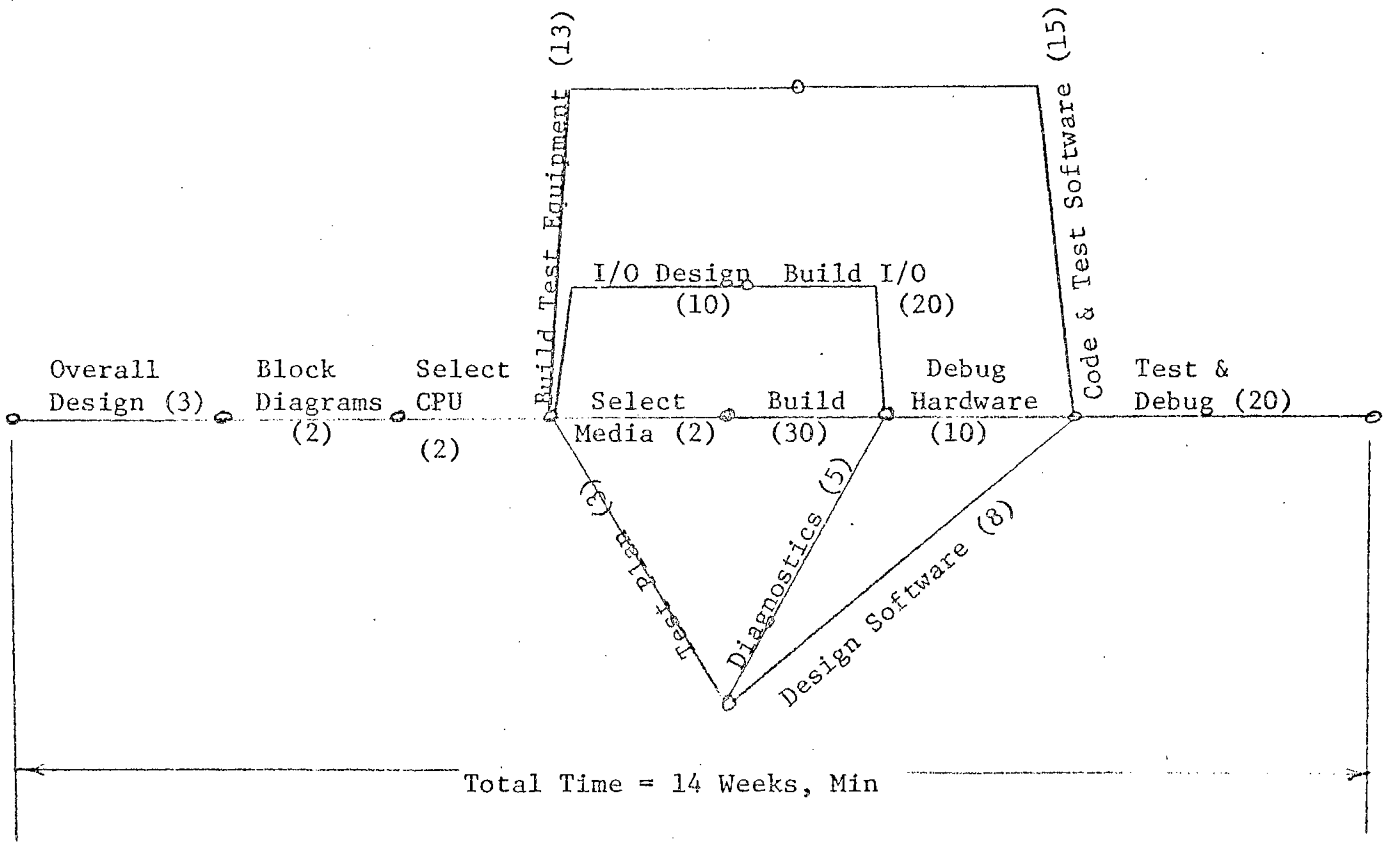
Approach to System Design

A systematic approach to microcomputer system design is necessary to achieve smooth implementation of the design process. The design process, especially the software part, takes some adjustment. For a first microcomputer design effort, it would be best to follow the key suggestions given below:

1. Pick a small job which resembles closely a design previously completed.
2. Study in detail and understand the manufacturer's literature on the selected microprocessor.
3. Select carefully microcomputer-development systems, logic state analyzers and other microprocessor test tools.
4. A key to successful microcomputer design project is to plan carefully the work to be done (22).

Figure 1 shows a critical path chart which is of great help in project planning. The chart shows what tasks must be completed and in the order they are done for a small design project. The first step, overall design, indicates that the designer should understand clearly what the system is supposed to do before selecting the microprocessor. The engineer can use pre-manufactured cards for ease in design, and hence can start designing necessary application dependent input-output hardware while waiting for the cards to arrive.

Most engineers start the hardware design by tackling the CPU first. CPU design usually starts with the clock. The clock should be designed according to the characteristics the specifications call for. The circuit is then checked under dynamic load. The designer may have to design and implement a manual operator's panel if the product requires a moderate sized program and more than a few integrated circuits. The next important hardware design aspect is the handling of memory. When designing a microcomputer's read-only-memory (ROM) and random-access-memory (RAM), one must always leave sufficient room for expansion. The program's basic definition may change or its length may have been under-estimated.



Note: (n) = Working Days to Complete Indicated Task

Figure 1. Critical Path Chart. The 14-week Minimum Total Elapsed Time is Typical for a Newcomer to μP's attempting to Perform a Relatively Simple Task with Which he is Reasonably Familiar

The microprocessor can be used to test the hardware with the help of diagnostic routines. A diagnostic program can diagnose some faults and report them. Typical diagnostic routines test memory and input/output equipment. When testing the hardware with these routines, they are placed in an accessible location for the debugging monitor, thus providing an on-line diagnostic tool. This helps to decide whether the hardware or software is at fault.

In software design, regardless of how the software and hardware design areas are assigned, the problem definition must be exact. It cannot be incomplete or imperfect, as that would cause grave problems later on in software development. Software development is at least as hard as hardware development. Paper designs are relatively cheap and easy to change and, in most cases, there are numerous alternative methods of design. Choosing the software design is normally based on economics, because the cost of software development in a microcomputer system project can be significant. In designing a microcomputer system from the chips up, software will cost about as much as hardware. But if the project develops around a microcomputer bought from a vendor, software costs exceed hardware costs.

Another important task to be performed in system design is testing and verifying the operation of the final microcomputer system. In many instances, the testing phase has not been planned well ahead, and results in a less reliable system. The test plan should be designed with the end-user's as well as the engineer's own viewpoint. It should reflect normal and abnormal situations expected in the field. A tester should bear in mind that a debugged program is one that has not yet received the inputs that will make it fail. Hence, the tester should try to design test cases to make the program fail. The crux of the testing issue can be stated as: the programmer and engineer debugging an imperfect system try to develop test cases that will make the system work. On the other hand, a test plan must aim at making the system fail (22).

System integration is the point at which the software and hardware designed are put together, and they work together. This is the point at which most failures appear. If the hardware and software were subjected to tests earlier, then only errors remaining at system integration time would be misinterpreted interfaces. On the other end of the line, integration can take months if the hardware designer completes his work assuming that the software to test his hardware will be correct, and the software designer expects debugged hardware to test his programs. Thus, a reasonable amount of time must be allowed in the design for this purpose.

CHAPTER II

THE HARDWARE SYSTEM

Selection and Classification of Essential Hardware Functions

In the system design for many applications, use of off-the-shelf components, such as single board computers, I/O expansion boards, can result in savings in time and increased efficiency on the system designer's part. INTEL offers a wide range of products, and in most cases all the designer must do is decide on the components he needs. At the same time, the design should be flexible so that future modifications and expansions would not pose a problem.

The task of designing the hardware system can be simplified by first classifying the necessary functional elements. These classes can be broadly categorized, as shown below:

1. A central processing unit powerful enough to handle many users.
2. An efficient input/output system.
3. Sufficient main memory and secondary storage.
4. Necessary expansion boards and peripheral controllers.

The INTEL 8086 CPU is chosen because it is a powerful processing unit with an architecture to handle high-level languages (HLLs) efficiently. Selection of a microprocessor language is influenced by cost-effectiveness and deciding which language is most suitable for a given application. Working with an HLL is much easier than working with assembly languages.

An HLL is easy to learn and can boost daily code production far beyond that of an assembly code. Programs written in HLL are generally portable with a few modifications. The disadvantage of HLLs are that they are less efficient than assembly languages, and require more memory and larger execution times (18). But PL/M-86, a systems implementation language, is designed specifically for the special requirements for microcomputers and also produces an efficient code. It is the language chosen in this multi-user application design because of its reentrant capabilities and several other features, including: high-level constructs for machine control, especially interrupt handling, direct port I/O and access to absolute memory locations; pointers and based variables; string manipulation; LOCKSET, a procedure for multiprocessing environments (20).

PL/M-86 makes it easy to divide a programming task into subtasks, thus encouraging top-down design. Modularity also simplifies program development and maintenance. PL/M-86 includes special features for writing systems software: I/O handlers, device drivers, system monitors--any executive program that directly controls hardware even if imbedded in application software. Unlike FORTRAN or PASCAL that require an operating system or run-time support to perform system-level functions, PL/M-86 does not need this support, resulting in savings in memory. Hence, PL/M-86 offers the memory efficiency of system-level code and the programming efficiency of an HLL.

PL/M-86 executes on the 8086 CPU which is the chosen processing unit for this design. Due to the HLL efficiency of the 8086, the PL/M-86 compiler increases memory requirements by no more than 1.45 times those of the ASM-86 assembler. The execution speed is only 1.19 times that of ASM-86 programs (depending on compiler options and the applications)(19).

The size of reentrant code can be reduced as much as 54% compared with the same code run through the PL/M-80 compiler. The 8086 CPU uses an optimum amount of registers to implement HLL efficiently. Based on a study by A. Lunde, it was shown that three index registers and six registers all together are used 90% of the time for forty-one scientific DEC system-10 programs. The 8086 provides eight general registers--four of which can be used as index registers. These are enough to support the required 90% usage. Too many registers increase the interrupt-service time. The 8086 handles the storage of constants to generate memory efficient code. It also provides all necessary addressing modes to support data manipulation in an HLL, from simple direct addressing to the complex base + index + displacement.

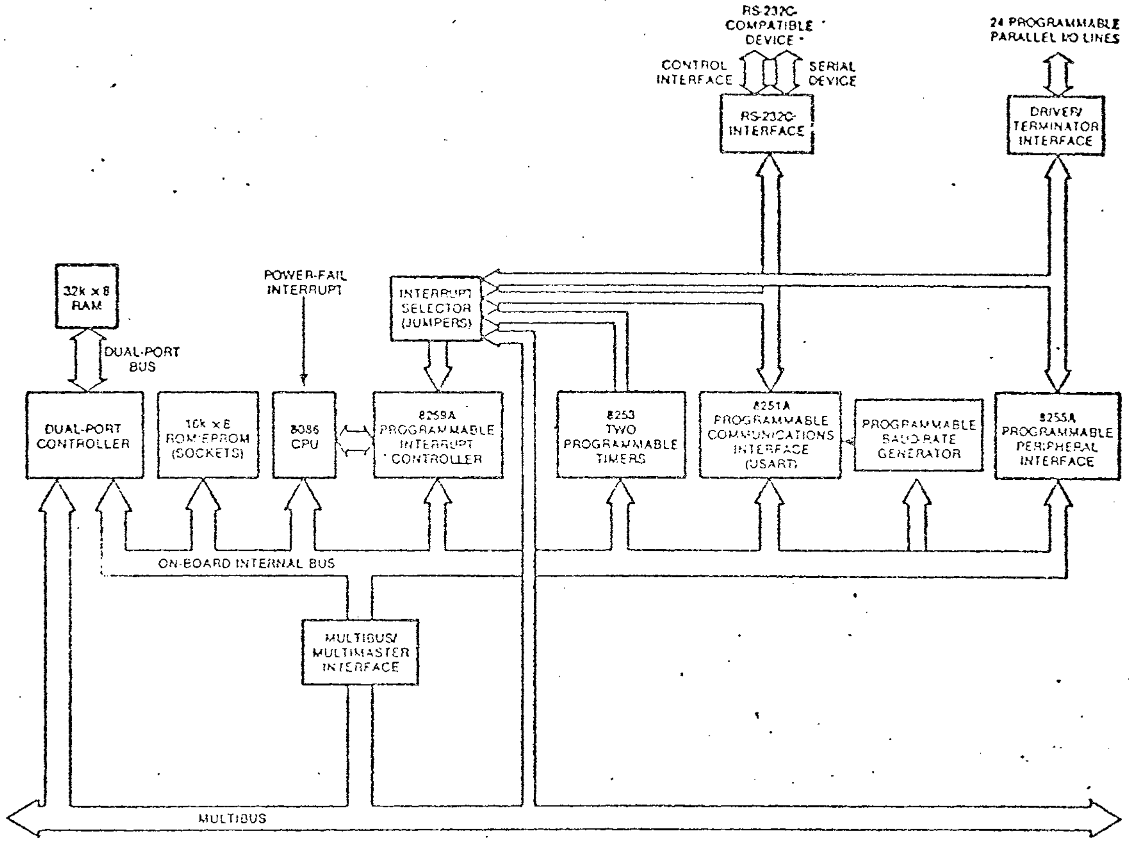
In addition to the above features of the INTEL 8086 CPU and PL/M-86 language, the following considerations led to selecting the hardware components of this design from the INTEL range of products. Before the advent of single-board computers and system expansion boards, system designers were hindered in two ways: hardware complexities and test requirements confined them to assembled computer subsystems or production volumes being low did not justify hardware and software development costs. But these hurdles can now be overcome by using families of single-board computers and expansion boards such as the INTEL SBC-80 series. These boards are ready-to-use, flexible, and inexpensive. These boards provide programmable parallel and serial I/O structures that provide design flexibility in I/O interfacing. The Multibus, the SBC-80 system bus allows modular performance expansion by providing a defined, standard interface between the SBC-80 single-board computers and expansion boards. Besides the wide variety of INTEL SBC boards, there are a number of non-INTEL SBC compatible boards that make system expansion and support possible. Expandable backplanes and card cages are also available to support multi-board systems (21).

Description of Specific Hardware Components

INTEL offers a line of single board computers that have limited RAM, ROM, and I/O capabilities (10). The iSBC 86/12A Single board computer has an 8086 16-bit HMOS microprocessor which, with the proper software and hardware, can handle many users efficiently (10, 11). The iSBC 86/12A also has nine levels of vectored interrupt control, expandable on board to 64K bytes. The system memory is expandable to one million bytes. Hence, the applications program and executive routines can both be resident in the system memory often. This computer also has the MULTIBUS interface on board, which allows for system expansion, thus not limiting the system's capabilities to those of the single board computer (12). A block diagram of the iSBC 86/12A is shown in Figure 2.

Having decided on the single board computer, the designer faces the problem of implementing an input/output system capable of handling many users efficiently. From Figure 2 it can be seen that the iSBC 86/12A has limited serial I/O capability. But by using appropriate serial I/O expansion boards, the designer can connect many terminals to the system.

Microcomputers are parallel systems, hence in serial communication, an eight-bit byte of data must be converted to serial form before output, and from serial to parallel form before input (8, 4). There are two ways to perform this conversion: by software, or with a UART (universal asynchronous receiver-transmitter). The advantage of a programmed implementation is simplicity and the elimination of external hardware.


RS-232C COMPATIBLE DEVICE
CONTROL INTERFACE
SERIAL DEVICE
RS-232C INTERFACE
24 PROGRAMMABLE PARALLEL I/O LINES
DRIVER/ TERMINATOR INTERFACE
32k x 8 RAM
DUAL-PORT BUS
POWER-FAIL INTERRUPT
INTERRUPT SELECTOR (JUMPERS)
DUAL-PORT CONTROLLER
16k x 8 ROM/EPROM (SOCKETS)
8086 CPU
8259A PROGRAMMABLE INTERRUPT CONTROLLER
8253 TWO PROGRAMMABLE TIMERS
8251A PROGRAMMABLE COMMUNICATIONS INTERFACE (USART)
PROGRAMMABLE BAUD RATE GENERATOR
8255A PROGRAMMABLE PERIPHERAL INTERFACE
ON-BOARD INTERNAL BUS
MULTIBUS/ MULTIMASTER INTERFACE
MULTIBUS

Source: Systems Data Catalog, Intel Corporation (1980).

Figure 2. System Architecture of the iSBC 86/12A Single-Board Computer

However, it is slow and might impair the microprocessor's performance. Also, no reliable delays can be implemented in a system using interrupts. A hardware implementation is required. Moreover, the INTEL 8251 USART (Universal synchronous and asynchronous receiver-transmitter) already exists on the iSBC 86/12A and serial I/O expansion boards.

Basically, there are three methods of input-output control (8). They are:

1. programmed I/O, or polling
2. interrupt-controlled I/O
3. direct memory access.

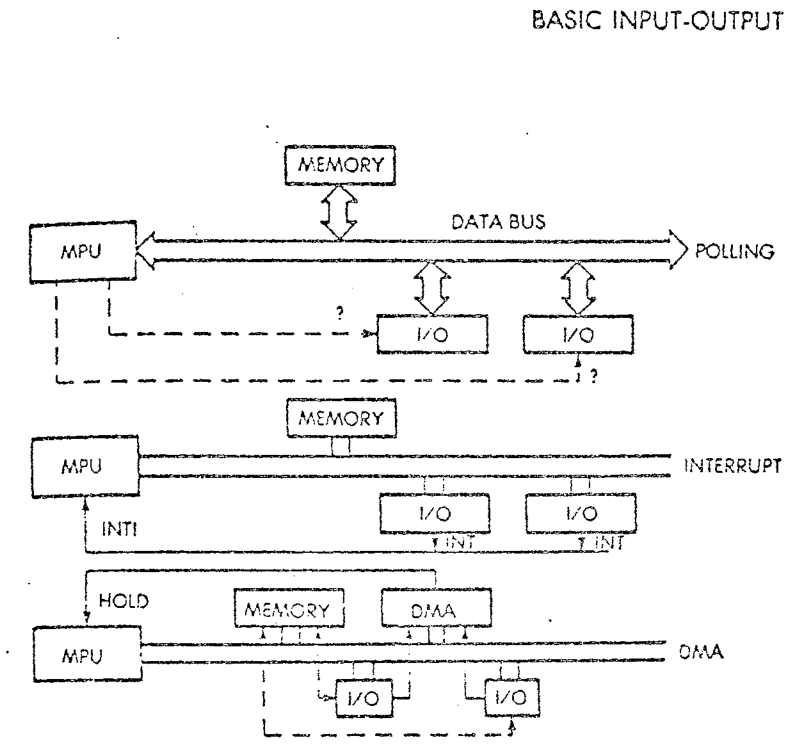
These three methods are illustrated in Figure 3.

Polling

In polling, all transfers to and from devices are performed by the program. The processor sends and requests data; all input and output operations are under control of the program being executed. The transfers must be coordinated by a "handshaking" process. This technique has two limitations:

1. It is wasteful of the processor's time, as it needlessly checks the status of all peripherals all the time.

2. It is intrinsically slow, since it checks the status of all I/O devices before coming back to any specific one. This cannot be tolerated in a real-time system, where a peripheral expects service within a specified time. Fast devices such as the floppy disk or a CRT require a near-instantaneous response time in order to transfer data without loss.



Source: Zaks, Rodnay and Lesea, Austin. Micro-processor Interfacing Techniques (1979).

Figure 3. Three Methods of I/O Control

Interrupt I/O

The interrupt-controlled I/O method guarantees the fastest possible response to an input-output device and is the method used in this system design (8). The iSBC 86/12A board provides nine vectored interrupt levels, the highest level being non-maskable is directly connected to the 8086 CPU. The INTEL 8259A programmable interrupt controller (PIC) provides vectoring for the next eight interrupt levels. A selection of four priority processing modes is available to the system designer as shown in Table I.

TABLE I

PROGRAMMABLE INTERRUPT MODES OPERATION

Mode	Operation
Fully nested	Interrupt request line priorities fixed at 0 as highest, 7 as lowest
Auto-rotating	Equal priority. Each level, after receiving service, becomes the lowest priority level until the next interrupt occurs.
Specific priority	System software assigns lowest priority level. Priority of all other levels based in sequence numerically on this assignment.
Polled	System software examines priority - encoded system interrupts status via interrupt status register.

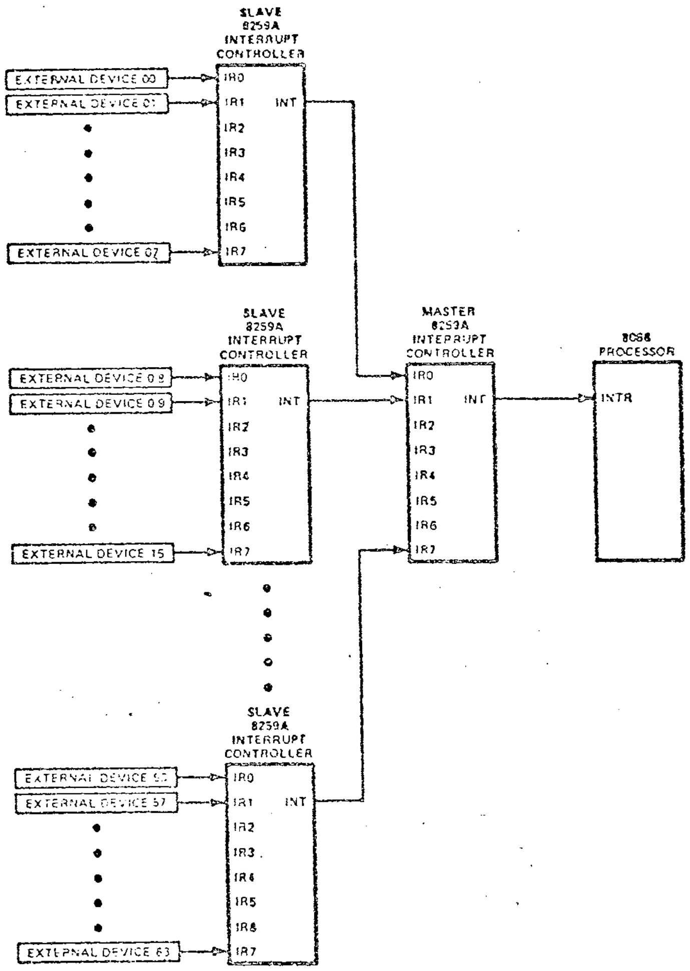
Source: _Systems Data Catalog_, Intel Corporation (1980).

Any combination of interrupt levels may be masked, via software.

The PIC generates a unique address in memory for each interrupt level. These addresses are equally spaced at four byte intervals. These locations contain unique instruction pointers and code segment offset values for each interrupt level. The PIC determines which of the incoming interrupt requests is of the highest priority, then determines if this request is of higher priority than the level being serviced and, if appropriate, issues an interrupt to the CPU. The CPU acknowledges the interrupt and obtains a device identifier byte from the 8259A PIC. It then stores its status flags on the stack and executes an indirect CALL instruction through the vector location (derived from the device identifier) to the interrupt service routine.

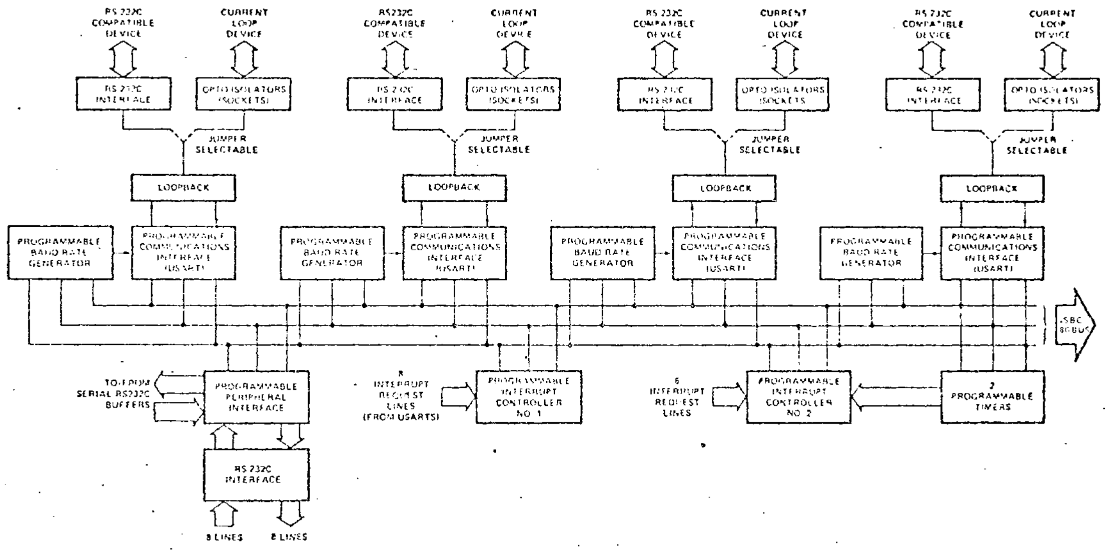
In the present system design, additional interrupt levels are required so that slave 8259A PICs can be interfaced via the system bus. A total of 65 unique interrupt levels may be generated by these additional vector addresses. Figure 4 shows the general concept of tying 8259A PICs in a master-slave configuration.

Only one serial device can be connected to the iSBC 86/12A board via the asynchronous RS232C compatible serial interface provided on board. To provide expanded serial I/O communication and implement the interrupt-controlled method of I/O, we can consider the iSBC 534, an INTEL product. The iSBC 534 is a four-channel communication expansion board that provides serial I/O expansion through four programmable synchronous and asynchronous communication channels (10). This board provides a flexible and easy means for interfacing iSBC 80 and system 80 based systems to RS232C compatible terminals. The block diagram of the iSBC 534 is shown in Figure 5. Two independent Intel 8259 PICs provide vectoring for 16 interrupt levels. Two jumper selectable interrupt requests (eight total) can



Source: Morse, Stephen P. The 8086 Primer - An Introduction to its Architecture, System Design, and Programming (1980).

Figure 4. Handling More than Eight External Devices


RS 232C COMPATIBLE DEVICE
CURRENT LOOP DEVICE
RS 232C INTERFACE
OPTO ISOLATORS (SOCKETS)
JUMPER SELECTABLE
LOOPBACK
PROGRAMMABLE BAUD RATE GENERATOR
PROGRAMMABLE COMMUNICATIONS INTERFACE (USART)
iSBC 80 BUS
TO/FROM SERIAL RS232C BUFFERS
PROGRAMMABLE PERIPHERAL INTERFACE
8 INTERRUPT REQUEST LINES (FROM USARTS)
PROGRAMMABLE INTERRUPT CONTROLLER NO. 1
6 INTERRUPT REQUEST LINES
PROGRAMMABLE INTERRUPT CONTROLLER NO. 2
2 PROGRAMMABLE TIMERS
RS 232C INTERFACE
8 LINES
8 LINES

Source: SystemsData Catalog, Intel Corporation (1980).

Figure 5. iSBC 534 Four Channel Communications Expansion Board Block Design be generated automatically by each USART when a character is ready to be transferred to the system bus, or a character has been received. Depending upon the number of users to be connected to the system, one or more iSBC 534 boards can be used.

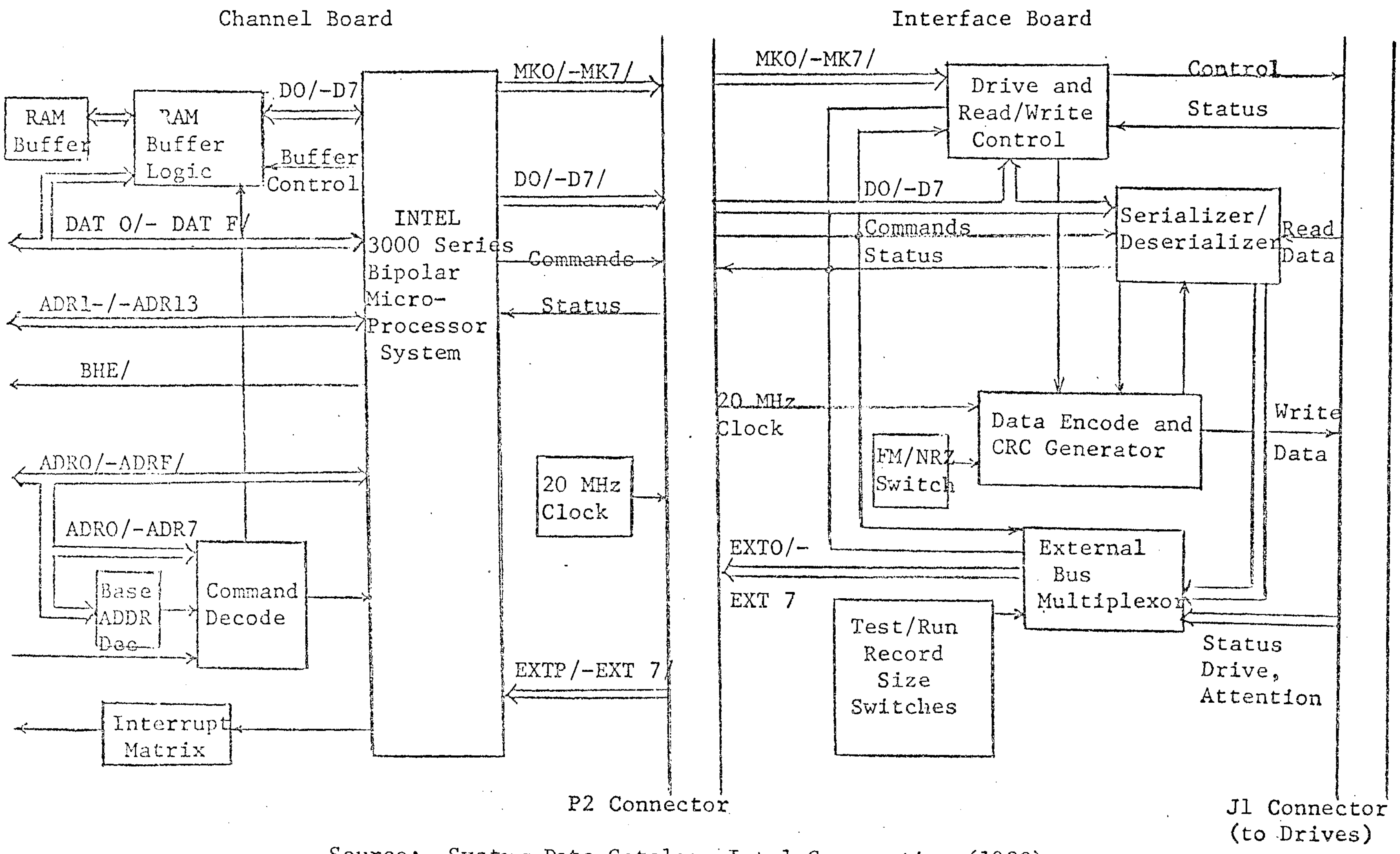
Direct Memory Access (DMA)

Although interrupts guarantee the fastest possible response to an input/output device, service to the device is still accomplished by software. This may still not be fast enough for processes involving fast memory transfers such as disk transfers. The solution is to replace software by hardware by using a direct memory access controller (DMAC), which is a specialized hardware processor.

INTEL offers two peripheral controllers, the iSBC 204 and the iSBC 206. These boards are fully compatible with the iSBC 80 and iSBC 86 single board computers (10).

The iSBC 204 is a universal flexible diskette controller capable of supporting four single density diskette drives. It has a wide range of compatibility without any sacrifice in performance. The iSBC 206 is a disk controller that can control up to four IBM 5440-type disk drives, providing up to 40 million bytes of storage. Though the iSBC 206 does not have as wide a range as the iSBC 204, it allows for a large secondary storage system which may well be necessary in a multi-user application. Figure 6 shows the block diagram of the iSBC 206 which actually is a two-board set, a channel board and an interface board.

Another attractive feature of the iSBC 206 is that it is supported by both the disk file and DISKIO levels of RMX/80s disk file system. The RMX/80 is INTEL's real-time multitasking operating system used



Source: Systems Data Catalog, Intel Corporation (1980).

Figure 6. iSBC 206 Disk Controller Block Diagram especially with INTEL's wide range of single-board computers. Files are named symbolically and may be created, deleted, or updated. All of the data transfers are carried out through DMA operations.

CHAPTER III

THE SOFTWARE SYSTEM

General Considerations in Operating System Selection

Currently available microcomputer operating systems range from simple noninterrupt-driven to full-blown multitasking/multiuser operating systems. To evaluate and to select the operating system that is best for an application, one requires knowledge of some basic microcomputer software terminology.

Microcomputer system designers now have very sophisticated hardware and are demanding equally capable software, both at the application and system level (14). The evolution of operating systems for microcomputers is much faster than that of minicomputers. There is a trend toward coding operating systems in high-level languages (HLLs) or a mixture of an assembly language and an HLL. Another advanced capability available in a number of operating systems is networking. Due to variability in vendors' descriptions of their operating systems, one must understand terms used to describe the software before he can compare vendors' offerings intelligently.

An operating system (OS) controls a microcomputer (μC) and acts as a mediator between the computer and its users; it relieves the users of the task of writing code that deals directly with system hardware resources.

A representative model of a total μC system is shown in Figure 7. At the innermost layer lies the hardware, surrounded by the operating system software. The next layer contains the application programs and system utilities - editors, compilers, assemblers, and debugging aids. The users interact with this layer and the OS fields all requests for services and handles all inputs and outputs, interrupt processing and task coordination. Hence, users need concern themselves with only their applications.

The user interfaces with the μC system by submitting a job - a collection of steps needed to accomplish a specific amount of work. These job steps are usually executed sequentially. Presented with a job, an OS might break the job up into tasks and create processes to service them. Each process is a computation that can occur simultaneously or concurrently with other processes. Multitasking denotes a system with several concurrently executing processes - either running, ready, or waiting (Figure 8).

Object-oriented Operating Systems

The complexity of many operating systems makes it difficult for users to grasp their organization. But OSs exhibiting object-oriented architectures have well defined mechanisms and are consistent, which makes them much easier to understand. For example, RMX/86 is a real-time modular object-oriented OS for the INTEL 8086 16-bit microprocessor. The RMX/86 architecture resembles that of 8086, as it has operators (system calls) that manipulate objects (OS data structures) much like the 8086's operators manipulate operands. Tasks, semaphores, mailboxes, connections, memory segments, and jobs are some of the OS's

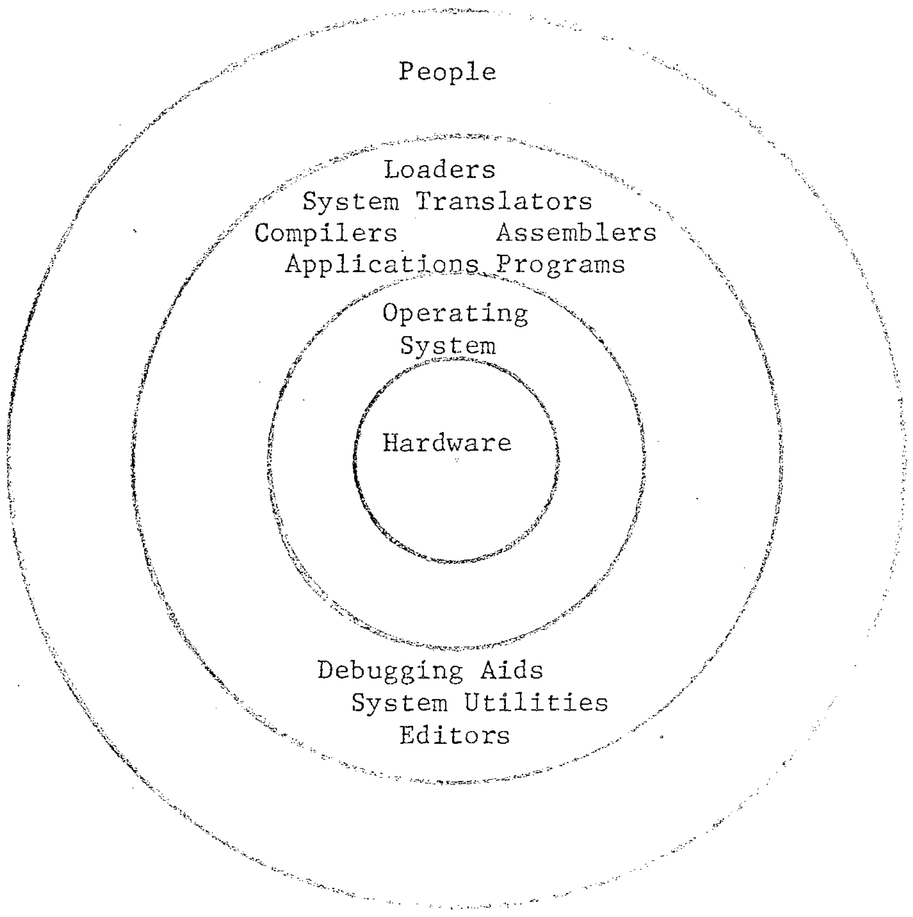
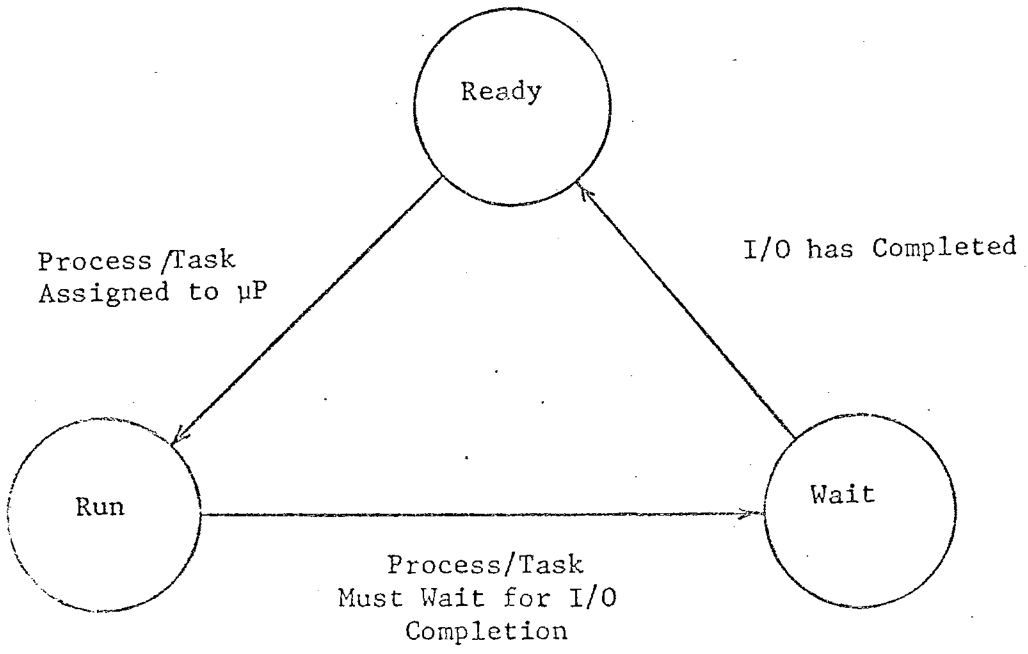


Figure 7. Hierarchy in a Computer System


Ready
Process /Task Assigned to μP
I/O has Completed
Run
Wait
Process/Task Must Wait for I/O Completion

Source: Hemenway, Jack, and Kotelly, George. "Microcomputer Operating Systems Directory." EDN, November 5, 1980.

Figure 8. A State Diagram of a Process

objects. RMX/86 treats data structures symmetrically, so users can create their own objects and write their own operators to configure custom OS functions. This type of architecture uses objects as building blocks that the operators manipulate. Each type of object has a specific set of attributes, and once one becomes acquainted with these attributes, he is familiar with all of the objects they describe.

The main advantage of object-oriented operating systems is that one can master the OS in a short time, and hence focus only on the objects he plans to use.

Types of Operating Systems

Operating systems can be classified in terms of user funtion as follows:

1. Development OS: This type of OS serves to produce software either to run on the host μC or another target μC. Intel's ISIS, Motorola's MDOS, Digital Research's CP/M are a few examples.

2. Real-time or Process Control OS: Generally serves to control industrial processes that require fast responses from the μC, because slow responses could impair or seriously degrade the processes. Computerized oil-refinery control is an example of a real-time application. Intel's RMX80 and RMX86, and Texas Instruments' RX are examples of such operating systems.

3. General-purpose OS: These are generally associated with business or scientific applications. For example, the CP/M OS is used in word processing, accounts-receivable generation, and mailing list maintenance.

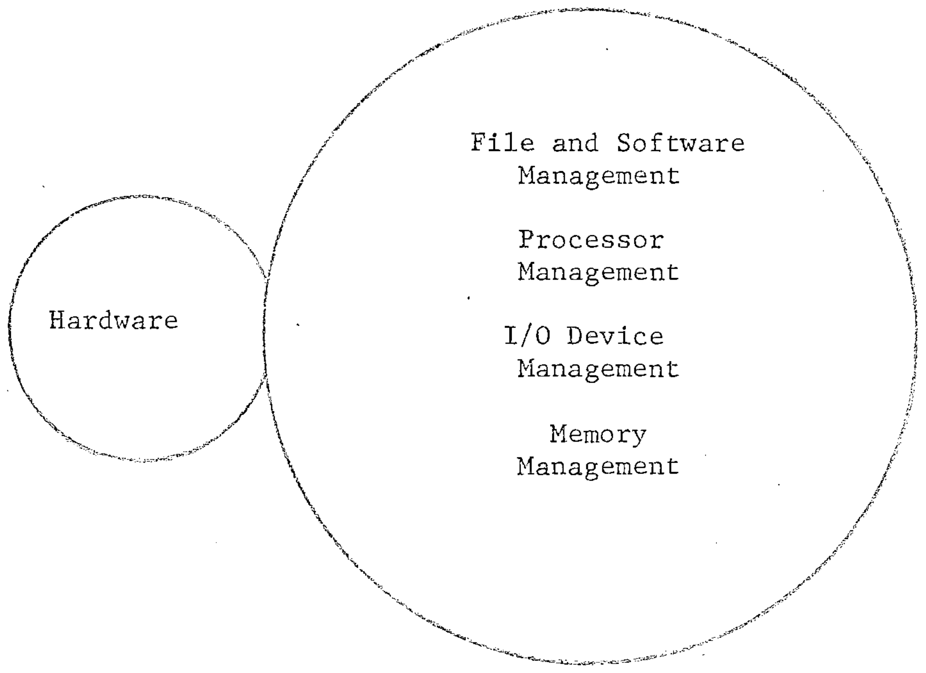
Many OSs provide capabilities in all three areas. A further classification is sometimes used to describe OSs. Any of the above three OSs can be either a multiuser or single-user system. A multiuser OS provides computational services to many on-line users. It achieves the effect of simultaneous service by sharing system resources in a round-robin fashion: each user in turn gets an allocation of system resources and no user gets more than any other - unless a priority scheme overrides someone's turn or allocates more resources to a higher priority user. A single-user OS, on the other hand, usually allows one user to submit jobs serially (14).

An Operating System's Resource Management

An OS manages resources by allocating them on the basis of user needs and system capabilities. It performs this task by maintaining lists of available resources and noting which user has control of a particular resource and who has the highest priority. As shown in Figure 9, an OS allocates four resources: process time, memory, peripherals, and files.

Processor-time Management

Efficient processor management is one hallmark of an effective μC operating system. Sequential execution of processes or jobs can be very inefficient because many of these jobs are independent. Hence, a better way to utilize processor time would be multiprogramming. Several jobs can be handled simultaneously by overlapping or interleaving their execution. Another technique known as multitasking is the same as multiprogramming, but applied at the task level (14). The number of tasks allowed in an OS gives an indication of its capabilities. An even



Source: Hemenway, Jack, and Kotelly, George. "Microcomputer Operating System Directory." EDN, November 5, 1980.

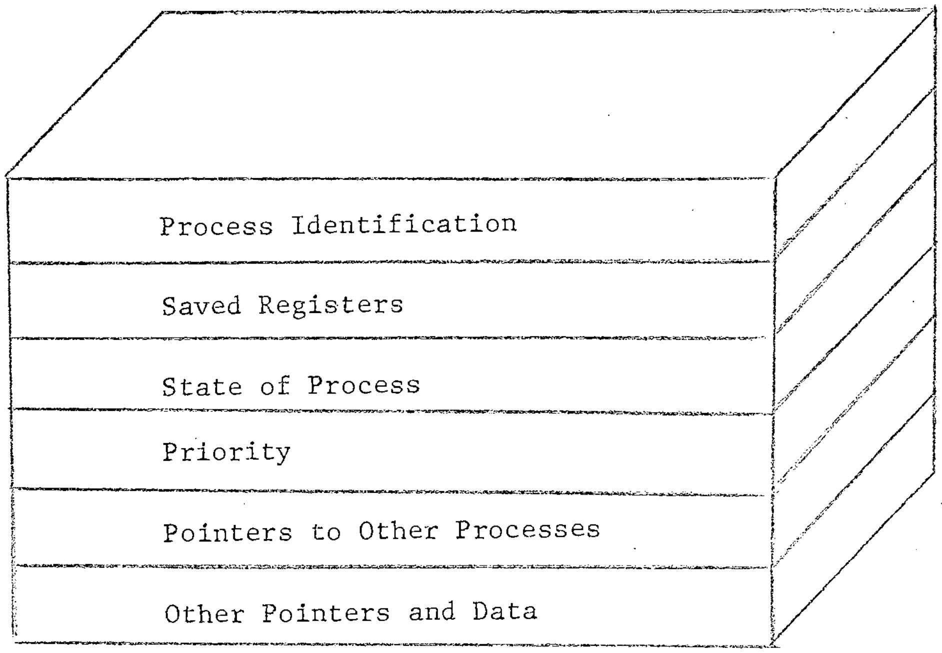
Figure 9. Resources to be Managed by an OS

more effective measure is whether the OS supports intertask communication. This communication is necessary to allow processes to synchronize their execution and pass data among themselves. A further indication of an OS's power is the ability of tasks to start, stop, or suspend other tasks.

Multi-user Capability. A multitasking OS can have many users. Such a time-sharing OS makes each system terminal a separate task, assigns exclusive memory to each user, builds in protection mechanisms to keep users from getting in each other's way, and makes most system utilities re-entrant. Interuser communication facilities, common data bases, and other features make an OS even more powerful. Thus, how many users an OS supports and how much memory is needed for each user provides a rough measure of the OS's power and expense. In addition, some OSs accommodate multiprocessing wherein a different microprocessor can serve each task. This structure permits true parallel processing, but it also complicates the operating system's processor-management routine.

A process is the basic computational unit within an OS. There are two types of processes - an abstract process and a software process. A software process is uniquely identified by a process control block (PCB), which contains a state variable, a save area, and information about resources required by the process and allocated to it (Figure 10). An abstract process is the computation that results from the execution of such a software process (14).

Synchronization Between Interacting Processes. Defining the rules by which processes interact is one of the OS's main taks. This interaction is controlled by synchronization and communication between the



Source: Hemenway, Jack, and Kotelly, George. Microcomputer Operating System Directory." EDN, November 5, 1980.

Figure 10. A Process Control Block

processes.

In a good operating system, concurrently executing processes are unaware of each other's presence. The only contact necessary between processes occurs when information must pass between them (14). To implement this contact, an OS can use different techniques. A typical scheme uses a physical entity called a lockbyte, or a semaphore, to represent each shared resource. By convention, a zero semaphore value indicates that resource is available and a one indicates that resource is in use. To access this semaphore, a microprocessor's resource request produces a communication signal, LOCK(X), release of the resource produces UNLOCK(X).

A general form of the above scheme uses semaphores that can assume a range of integer values. Two operations - P and V - change the value of a semaphore. This technique serves in systems that require more sophisticated traffic management, such as the producer/consumer model in Figure 11 (14).

Mailbox Communication. The mailbox is another form of synchronization. If a process needs to communicate with another process, it requests creation of a mailbox to connect the processes and pass messages (mail) between them. The message is then put in the mailbox and the second process can pick it up at any time. The mailbox ensures that the second process eventually receives the message. The mailbox is implemented by the OS as a data structure with associated rules to define its operation.

Another important factor in OS process or control is the monitor. It is a section of code enclosing critical code sections and allowing entry to the code only through defined entry points. It guarantees the exclusive use of a resource.

```
Producer Process ($P_p$)        Consumer Process ($P_c$)
Produce . . .                   Consume . . .
(Produce Item)                  $P(S_2)$
$P(S_1)$                        (Get Item From Buffer)
(Put Item in Buffer)            $V(S_1)$
$V(S_2)$                        (Consumer Item)
Jump to Produce                 Jump to Consumer
```

Figure 11. Interprocess Communication Using P and V Operators

Deadlocks and Their Resolution. With synchronization procedures, such as the use of semaphores, processes can block and unblock each other to synchronize their operation. But the OS must exercise careful resource management to avoid a condition wherein two processes hold each other's requirements and are unwilling to relinquish them. This condition is termed a deadlock and could halt a computer. Hence, every OS requires a policy to deal with deadlocks. Three approaches are prevention, automatic detection, and operator detection.

Prevention. The OS maps all processes, thus preventing deadlocks. A simple way to implement this would be to allow one process to execute at a time which is not a realistic technique. Another method is not to let the process execute until all of its required resources have been allocated to it - a technique termed preallocation.

Automatic Detection. This technique allows a system to reach the deadlock state, but provides a means of detecting such states once they occur.

Operator Detection. This approach assumes that deadlocks are infrequent and not worth worrying about. The operator generally must restart the system when a deadlock does occur.

Memory Management

Main memory is often the most critical and limiting resource in a μC system. Hence, an OS's method of allocating and administrating main memory is of importance to the system designer.

For a user wanting to run only one job at a time and not interested in maximizing the use of main memory, single contiguous allocation technique is appropriate (Figure 12). This method treats available memory as one unit and then divides it into several sections, two of which are the operating system and the user job areas. This method produces inefficiencies in memory-management under more demanding circumstances. Hence, most of OS's use other storage-allocation techniques to assign storage to a job. These techniques also apply to system software. A good OS would probably store some driver routines on disk until they are needed (14).

Other memory-management tasks making up a good OS are relocatable code generation and extension of available memory. Overlaying is a scheme of extending available memory. When a routine in memory is not needed, another routine residing on disk is called and replaces the

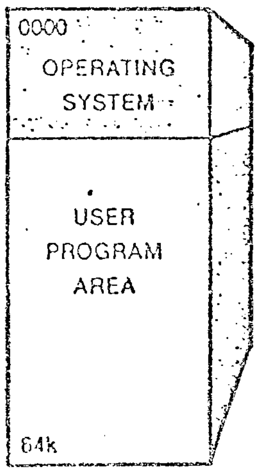


Figure 12. Single Contiguous Allocation

routine in the main memory. By this approach, the size of a μC memory can be less than the actual size of the OS. In overlaying schemes, a common area is established and data is transferred through this common area from one routine to the next. During evaluation of an OS, the designer should determine how an OS handles the overlay problem.

A second technique of extending available memory is called swapping. Unlike overlaying, swapping does not use any common area. Transfer of control is between programs rather than routines. This method is suitable in a time-sharing environment where control is transferred among user programs. When a swap takes place, the status of the interrupted program is saved and is restored when execution of the interrupted program resumes.

Another technique called segmentation provides automatic overlays. In this scheme, the logical parts of a program are placed in their own segment of memory which is a variable sized block of memory. Thus, the OS can store segments in non-contiguous memory locations. Though this is a flexible scheme, it usually needs special hardware to implement the addressing scheme efficiently.

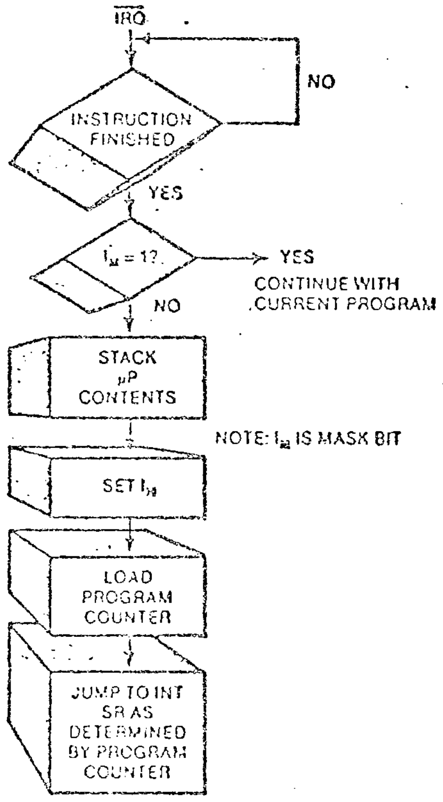
An essential factor one must consider before he selects an OS is its memory protection scheme. There are schemes ranging from a system / user mode classification to the IBM 360/370 scheme of associating protection keys with each 2K block of memory (14).

Peripheral Management

A good OS must handle peripherals efficiently and utilize them properly. Peripherals normally account for a good share of the cost of a μC system.

There are broadly two types of peripherals, those for input/output and those for storage. Line printers, card, paper-tape readers and CRTs fall under the first category. Peripherals such as disk and tape drives are storage peripherals. Under storage types there are different groups related to access times. A serial-access storage medium such as a cassette tape provides for serial access. Data can only be accessed sequentially and hence the time to access a particular data depends on its position relative to the start of the tape. Main memory, on the other hand, signifies a direct access unit. The time to access a particular datum in memory is independent of its position in memory. A compromise between the above two methods for accessing storage peripherals is denoted by disk peripherals. In this case, the μC system moves the disk drive's head to the appropriate track and waits until the desired sector passes under the head. It then reads the sector serially into a buffer block.

An OS often manages the peripherals by using an interrupt scheme. This makes the most efficient use of processor time (14). In this scheme, when a peripheral needs to transfer data, it sends an interrupt signal to the processor. The processor saves the status of all its registers and the program counter, and then transfers control to an interrupt service routine. After servicing the interrupt, control returns to the interrupted program by restoring the saved program counter and registers. Figure 13 shows the steps taken in interrupt handling. If the μC system consists of only one peripheral device, the OS's task of peripheral management is simplified. But when there are several peripheral devices,



Source: Hemenway, Jack, and Kotelly, George. "Microcomputer Operating System Directory." EDN, November 5, 1980.

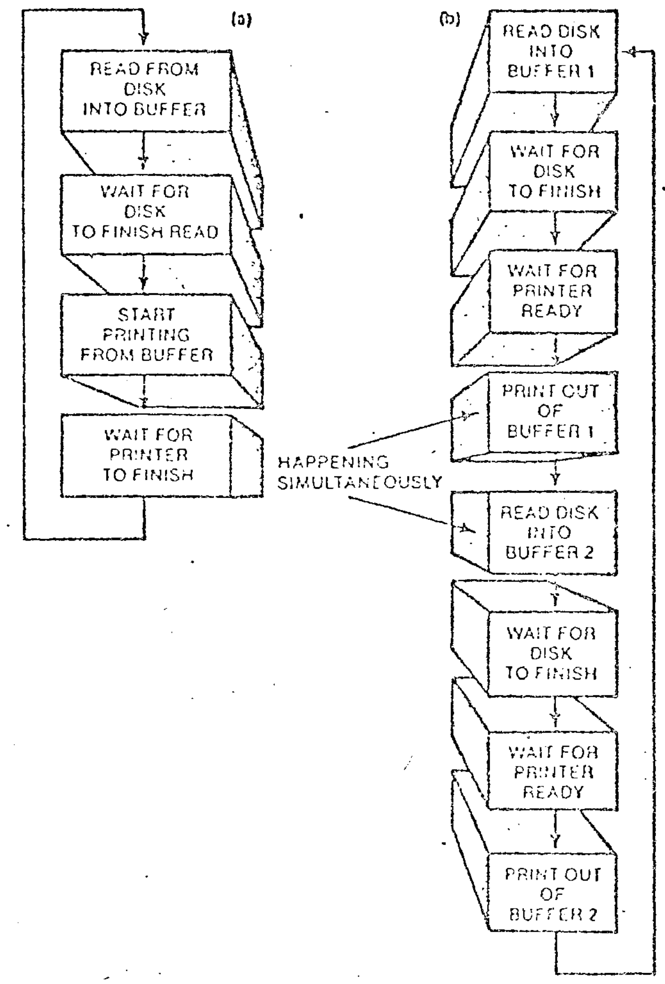
Figure 13. Interrupt Handling

there could be several interrupt request signals appearing at the same time. These interrupts can have scheduled priorities allowing the most important jobs to be performed first. Also, either the μC system hardware must provide vectored interrupts, or else a polling procedure is required.

Besides providing a good interrupt structure, an OS must provide synchronization between its data-transfer program and each device. The clocks used by peripherals are always asynchronous with the system clock. This synchronization is brought about in part by hardware such as controllers and by software. Additionally, properly formatted data and correct control signals must be supplied to each device by the μC system. A well designed OS minimizes the time consumed by this task.

In systems where high speed data transfer is to be achieved, direct memory access (DMA), a hardware function, can be employed. An OS can use a processor's time efficiently if it provides multibuffering. Multibuffering is the scheme wherein the peripheral devices are so scheduled that some of them can operate simultaneously. In this way, the processor and the devices are not kept idle. Figure 14 shows single- and double-buffering and the latter's advantage over the former scheme. The buffers could reside in the peripherals, in main memory or in both, depending on the buffer administration policies enforced.

Often a μC system could be slowed down due to slow peripherals. For example, low speed printers can bog down a μC system. A scheme known as SPOOLING makes a device that is dedicated to appear to be shareable. Files on a tape can be copied onto a disk and when a program tries reading from a tape, a SPOOLING routine converts it to a read from disk, which is a unit that can be shared. This method greatly improves the throughput of



Source: Hemenway, Jack, and Kotelly, George. "Microcomputer Operating System Directory." EDN, November 5, 1980.

Figure 14. Single Buffering (a), Double Buffering (b)

a system (14).

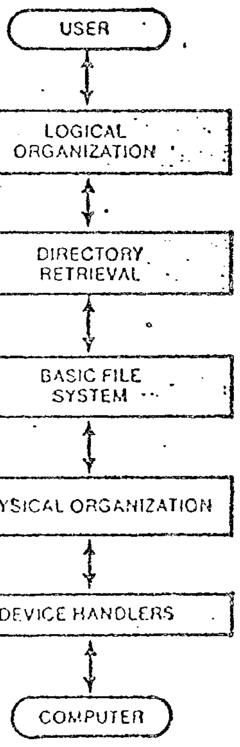
Another important feature of the peripheral management routines of an OS is the allocation strategies used. There are two types of allocation - static and dynamic. Static allocation is the allocation of devices before a program executes. The input/output scheduler handles this task. In dynamic allocation, devices could be reassigned within a program. Each device has a device handler to make these dynamic allocations. A shared device such as a disk appears as a group of dedicated devices to the user. It is the OS's responsibility to specify and allocate a definite portion of a device to the user program.

Device independence is another nice feature that some OSs provide. The user need not concern himself with a particular device's characteristics. The OS would even format the output to fit the console's requirements. This good feature of an OS improves programming productivity.

File Management

A file manager consists of routines which provide the interface between user programs and storage devices. The users do not have to be concerned with the details of data storage and retrieval. The programmer thus does not have to specify the actual mechanics of reading and writing a file, the location of files on storage devices, and the allocation and deallocation of storage space. Figure 15 shows a block diagram of a file-management system consisting of several levels of organization.

A file is basically a collection of records, each record containing some information. When referring to a file, all a user has to do is to give a symbolic name for the file. This name corresponds to an entry in a directory. The entry in turn translates the name to the file's



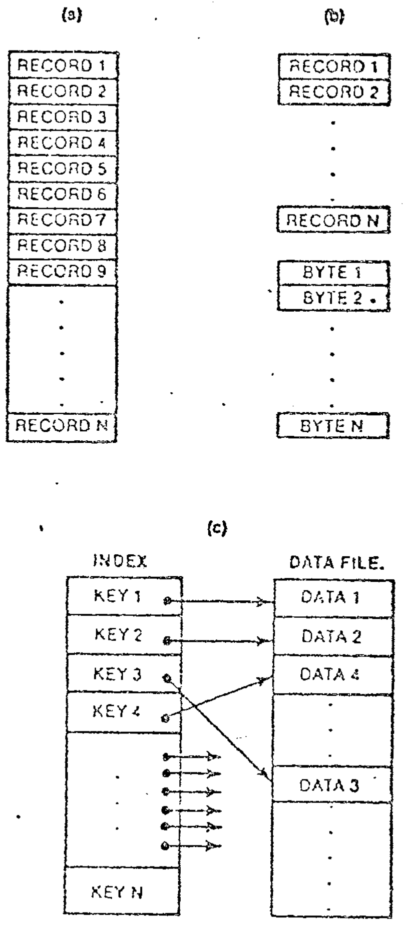
Source: Hemenway, Jack, and Kotelly, George. "Microcomputer Operating System Directory." EDN, November 5, 1980.

Figure 15. An Operating System's File Management System

attributes. In a single user system, one master directory is sufficient, but in a multiuser environment, an account identifies each user. The file's local name is looked up in the account directory. This in turn points to a unique entry in the master directory of all files. This allows for file sharing, which is an important feature that avoids unnecessary duplication of data. When selecting an OS, the designer should be aware of this fact (14).

Another important factor that affects a μC system's performance is how the OS allocates disk storage space. There are basically two methods - contiguous and noncontiguous allocation. The first method provides efficient file access due to minimal disk head movement. But its drawback lies in its deallocation. Holes are left in the disk, and hence the disk has to be repacked often; this consumes time. In noncontiguous allocation, although there is more disk head movement, the processing is more efficient. This scheme is implemented as either a linked list of disk sectors or by the use of extents. In a linked list technique, after the disk has been used for a while the linked list of sectors becomes fragmented, resulting in excessive head movement. In the extent method, the basic unit is an extent that consists of several contiguous sectors. A file consists of extents which are not themselves necessarily contiguous. Though this scheme also has the disadvantage of excessive head movement, it is to a lesser degree than in the linked list technique.

The logical organization of files must also be considered before selecting an OS. There are three main types: sequential, random, and indexed sequential, as shown in Figure 16. In a sequential organization, the records are read sequentially. There are two types of random files: byte and record addressable, depending on the type of application. The



Source: Hemenway, Jack, and Kotelly, George. "Microcomputer Operating System Directory." EDN, November 5, 1980.

Figure 16. Logical File Organization - Sequential (a), Random (b), and Indexed Sequential (c)

user can process subsets of a file without having to search sequentially. In an indexed sequential file organization, a file is organized into an index portion containing keys and pointers to sequential data in the other portion. This organization provides faster access than a sequential file organization.

Coding at the assembler level is often complex when trying to access a file-management system's facilities. A good high-level language (HLL) should provide constructs to access the file management facilities. Hence, all these facilities should be accessible from an HLL as is the case in most implementations of available HLLs.

The last function of the file-manager of an OS is its password and security-protection features. Files must be well protected from other users and the user himself. The protection could be delete protect, write protect, or both.

Selecting a Specific Operating System

The above methods to select and evaluate an OS can be used for the selection of a specific OS for this application design. The "Microcomputer Operating Systems Directory" article in the November 5, 1980, issue of the magazine EDN, contains a list of 69 operating systems available in the market, summarizing the essential features of each OS. This directory has been used in the selection of an appropriate OS.

The iSBC 86/12A single board INTEL computer will form the heart of the sample system. Other compatible components such as serial I/O expansion and interrupt handler boards must also be considered while choosing an OS. The OS should be able to support multiusers and PL/M86 and ASM86 languages. The application program may be coded in PL/M86 or ASM86 or a combination of both, depending on the type of application.

The foregoing techniques of OS evaluation have been summarized in an OS checklist (14). The same checklist has been used to select the OS needed for this application. The INTEL real-time operating system iRMX86 for the INTEL8086 will not be considered due to its inability to support multiusers. In this system design it would be necessary to have a real-time operating system.

The operating system checklist is reproduced below:

General

Name:
Date first released:
Primary application: Development ( ), process control ( ), general purpose ( )
Target processor:
Target model:
Sysgen program available:
Languages supported:
Language(s) system written in:
System residency: RAM ( ), disk ( ), RAM and disk ( )
System ROMable:
Minimum hardware required:
Networking supported ( ) How, and to what extent:
Sources available ( ) Price (single unit):

PROCESSOR-ALLOCATION MANAGEMENT

System vs User mode supported:
Multitasking ( ) If so, how many tasks allowed?
Intertask communication ( ) Tasks can start/stop/suspend other tasks ( )
Multiuser ( ) If so, how many users supported?
Minimum RAM needed per user:
Multiprocessing ( ) If so, how many μPs supported?
Synchronizing scheme used:
If multitasking or multiuser, type of synchronizing scheme used:
Semaphores ( ) monitors ( ) mailboxes ( ) other ( )

MEMORY MANAGEMENT

Single contiguous allocation ( )
Overlays supported ( )
Swapping supported ( )
Chaining supported ( )

Segmentation supported ( )
Static relocation supported ( )
Dynamic relocation supported ( )
When is binding done? Assembly/compilation time ( ) linking time ( ) loading time ( )
Memory protection available ( )

PERIPHERAL MANAGEMENT

Peripherals supported:
Interrupts used ( )
I/O multibuffering used ( )
SPOOLing supported ( )
DMA supported ( )
Device independence supported ( )
One configuration supports more than one type of mass-storage device (for example, a mix of hard disks and floppies) ( )

FILE MANAGEMENT

Named file system ( )
Sequential organization supported ( )
Contiguous organization supported ( )
Random organization supported ( )
Indexed sequential access method (ISAM) supported ( )
Multilevel directory supported ( )
Type of allocation used: Linked list of sectors ( ) extents ( ) single continuous ( )
File management systems accessible from an HLL ( )
Constructs in HLL to support file access ( ) or file access performed with assembler subroutine calls ( )
Password/security protection available.

Using the above checklist for a multiuser application, the main features an OS must have for this application are:

Development and real-time OS

INTEL 8086 CPU as the target processor

Support PL/M-86 and ASM-86 compilers

Allow multitasking

Support multiusers

Provide memory protection

Uses interrupts, supports CRTs and line printers

Applying the above features, and referring to the microcomputer operating systems directory, the following multiuser operating systems for the 8086 CPU are considered, DIOS, MICRO COBOL BOS, M^2SP/8086, POLYFORTH, XENIX and UMDS. MICRO COBOL BOS, M^2SP/8086, POLYFORTH, XENIX and UMDS operating systems do not support PL/M-86. XENIX and UMDS are not real-time operating systems. Hence the DIOS operating system by Systemathica Consulting Group, Ltd., was selected. It has all the desirable features for the system design. It supports the 8080 and 8086 CPUs. It is also a development, general-purpose and real-time operating system supporting the following languages: ASM 80/86 assemblers: BASIC and PASCAL (P-code) interpreters; FORTRAN, PLM, PL/I and BASIC compilers (14).

The essential features of the DIOS operating system are summarized below:

Name: DIOS

Software Manufacturer: Systemathica Consulting Group, Ltd.
4732 Wallingford St
Pittsburg, PA 15213
Phone (412) 621-8362

SYSTEM RESIDENCY: RAM and disk
ROMABLE SYSTEM: Yes
MINIMUM HARDWARD NEEDED: Depends on configuration-modular structured OS

RELEASE DATE: 1977 (8080 CPU version)
PRICE: Depends on configuration
TARGET MODEL: Disystem Series
SUPPORT LANGUAGES: ASM 80/86 assemblers; BASIC and PASCAL (P-code) interpreters; FORTRAN, PLM, PL/I and BASIC compilers
SYSGEN PROGRAM: Yes SOURCE CODE: Yes
LANGUAGE SYSTEM: PLM and assembler
NETWORK SUPPORT: Time sharing and multiprocessing

PROCESSOR ALLOCATION/MANAGEMENT

Automatic batch processing
Multitasking: number of tasks depends on system mapping using intertask communication
Multiusage: number of users depends on definable user/memory map
Synchronizing scheme: mailbox type

PERIPHERAL MANAGEMENT

Supports CRTs, character and line printers, floppy- and hard-disk drives, modems, magnetic- and paper-drives and plotters.
Provides DMA and spooling
Uses interrupts
Accommodates a mix of mass-storage devices

MEMORY MANAGEMENT

Overlays
Segmentation
Binding during linking and locate times

FILE MANAGEMENT

Named file system
Random organization
Allocation type: extents and linked list of sectors
Can access file system from HLL containing constructs
Password/security protection

COMMENTS: DIOS is RAM buffered (multitrack, multisector) and transparent to user. Most disk I/O executes at RAM-access speed.

Reentrant Program Considerations

PL/M-86, a high-level language for the Intel 8086 chip, has several nice features that make it applicable as the language in which the application code can be written. Since in this design multiple users are allowed and they have different priorities, a user with the higher priority can interrupt a lower priority user. Hence, it would be desirable to have some of the routines reentrant.

A reentrant subroutine is a subroutine that may be invoked while it is already in execution from a previous invocation. A procedure calling itself is one way a procedure might be reentered. Another way is for a procedure to call a second procedure which, in turn, calls the original procedure. These two forms of reentrancy are called recursion (1). A procedure can also be reentered if an interrupt occurs while the procedure is being executed and during the processing of the interrupt service routine the procedure is called again. In PL/M-86, a procedure can be termed reentrant if it has the REENTRANT attribute in its declaration. Any procedures that might be entered more than once before returning must be designated as REENTRANT if they are to execute correctly. All the data such as local variables and parameters utilized by a reentrant procedure must have a unique memory location for each concurrent invocation of the subroutine; otherwise, data used by a previous invocation may be overwritten by data used in the current invocation.

The rules summarizing the use of the REENTRANT attribute in PL/M-86 are given below:

Any procedure that may be interrupted and is also activated from within an interrupt procedure should have the REENTRANT attribute. Any procedure that is directly recursive should have the REENTRANT attribute. Any procedure that is indirectly recursive should have the REENTRANT attribute. Any procedure activated by a reentrant procedure should also have the REENTRANT attribute (3).

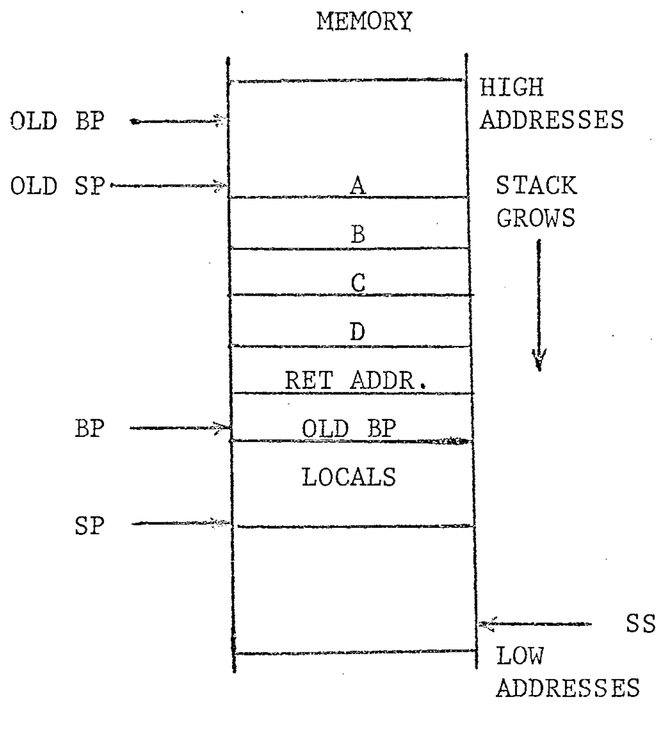
The REENTRANT attribute cannot be used in the same declaration as the EXTERNAL attribute. It may be used only in a PROCEDURE statement at the outer level of a module. A procedure declaration having the REENTRANT attribute may not have another procedure declaration nested inside it.

Most HLLs store local procedure variables on the stack which allows the procedure to be both reentrant and recursive (19). Storage is allocated dynamically during execution time rather than statically during compiling. As shown in Figure 17, all parameters are passed on the stack, and local variables are stored there.

```
CALL P (A,B,C)

     PUSH A
     PUSH B  PASS ARGUMENTS
     PUSH C
     CALL P

P    PUSH BP             SAVE OLD BP
     MOVE BP, SP         SET UP NEW BP
     SUB SP, LOCALS      ALLOCATE STACK
      "
      "
      "
     CODE FOR PROCEDURE P
      "
      "
      "
     MOVE SP, BP         REALLOCATE STACK
     POP BP              RESTORE OLD SP
     RET                 STRIP PARAMETERS
```



Source: Laphan, Stephen A. "The 8086 Microprocessor has the Architecture to Handle High-level Languages Efficiently." Electronic Design, March 1, 1980.

Figure 17. Stack Area Setup During a Procedure Call

CHAPTER IV

APPLICATIONS OF SYSTEM DESIGN

A Real-Time Alarm System

The hardware and software systems defined in the previous chapters could be applied to a real-time application to illustrate its use. The real-time alarm system described here is based on the real-time application of the October 20, 1979, issue of EDN magazine (15).

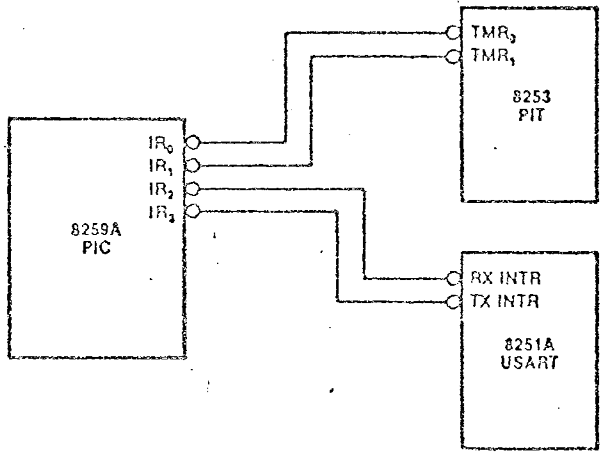
A typical example consists of events being monitored. The event being monitored could be the temperature exceeding a safe limit, or pressure exceeding a maximum limit in an industrial environment. There could be numerous other events being monitored. The events in turn can trigger an alarm scheme connected to the single board computer through its input ports. Important information regarding each event can be recorded by the computer.

The hardware system can consist of only the INTEL single board computer iSBC86/12A and a 7-segment LED display and a CRT. The additional hardware described in a previous chapter would not be needed in this application. The software system involves code pertaining to the particular application and selection of an operating system. The OS should be a development and real-time OS that supports the INTEL 8086 CPU. It must support PL/M-86, ASM-86, CRT and line printers. It must allow multitasking and provide primitives such as WAIT, SIGNAL, keyboard and CRT I/O. It should also monitor access to the CRT to resolve contentions that may arise. Applying the OS checklist to the operating systems directory, it appears that the DIOS, MTOS-86 and iRMX-86 are applicable in this case (14). On closer inspection, the DIOS operating system, although it has all of the desirable features for this application, has more capabilities than needed. It is sophisticated and not economical for this application. The MTOS-86 operating system is a multitasking, general purpose and real-time OS for the 8086. It supports PL/M-86 and FORTRAN-86 compilers. However, it is not a development OS and does not support ASM-86. The iRMX 86 operating system developed by Intel Corporation specifically for the 8086 CPU is a development, general-purpose and real-time OS. It supports multi-tasking, ASM-86, PL/M-86, and CRTs. It can readily be used in this application.

The system monitors the inputs from eight alarms, prints alarm messages on a CRT, and lights a 7-segment LED display with the number of the alarm currently being monitored. As shown in Figure 18, the 8259A programmable interrupt controller's interrupt request lines are connected to the 8253 programmable interrupt timer's outputs and to the 8251A USART's interrupt outputs. The ports of the 8255A programmable peripheral interface are used to interface with the alarm sensor's inputs and the 7-segment LED as shown in Figure 19. The alarm inputs are connected to port A and port B, the output port is connected to the 7-segment LED display. Port C is not used.

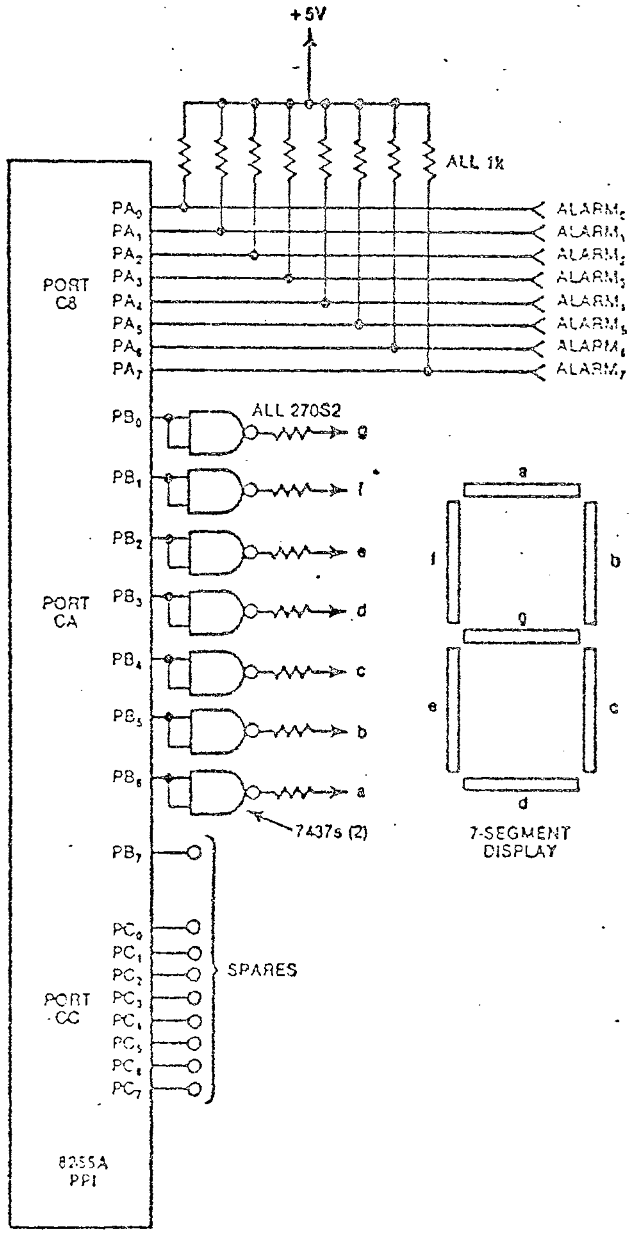
Before the system is used, it has to be initialized. The iSBC86/12A system debugger initializes the 8259A programmable interrupt controller and the 8251A USART. Hence, it is the designer's responsibility to write code that initializes the two timers and the 8255A PPI.

To ensure proper performance of the system, the different tasks to be



Source: Hemenway, Jack, and Teja, Edward. "Advanced Software Systems Design Course." EDN, October 20, 1979.

Figure 18. Connections to the System's Programmable Interrupt Controller



Source: Hemenway, Jack, and Teja, Edward. "Advanced Software Systems Design Course." <u>EDN</u>, October 20, 1979.

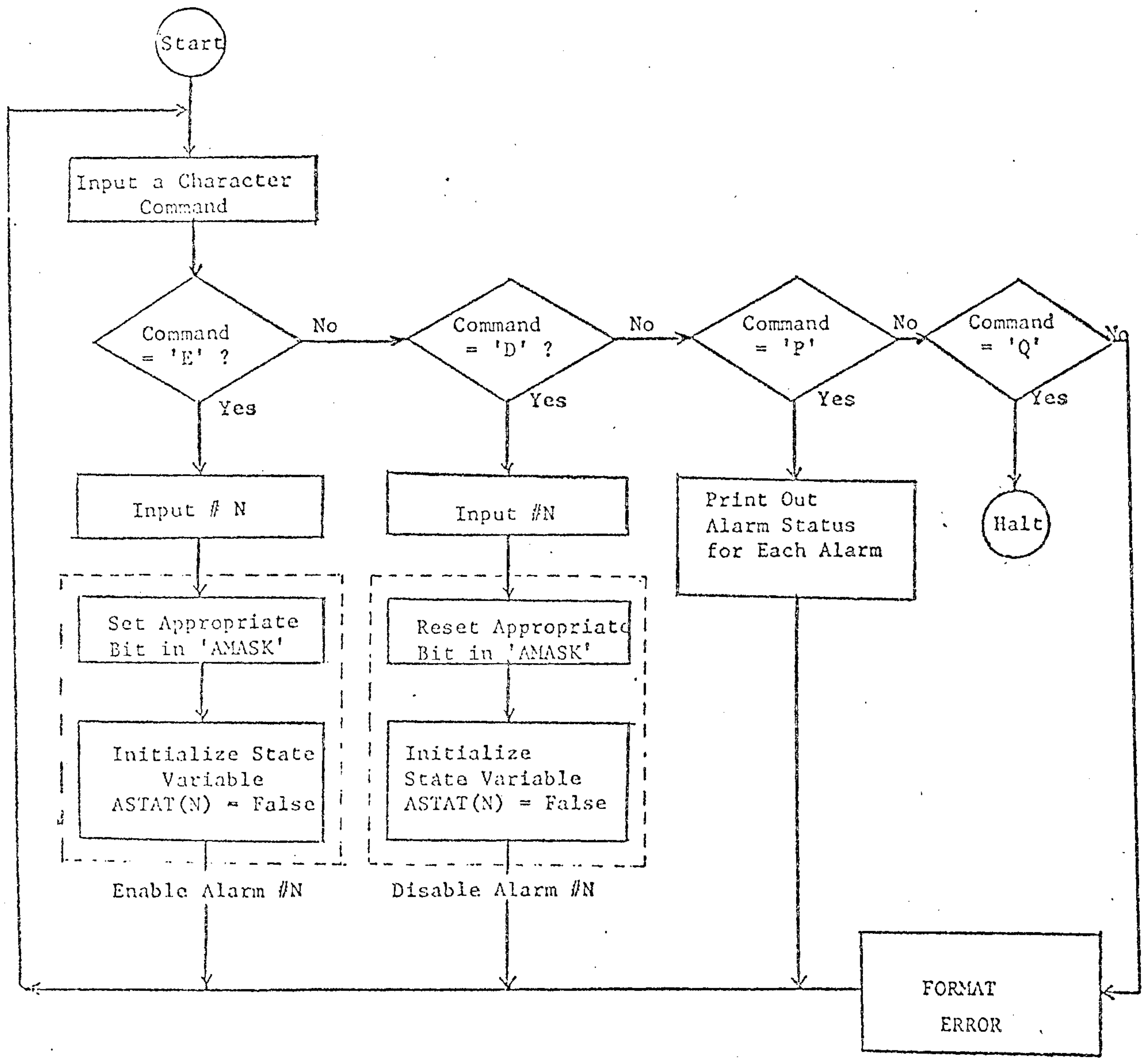
Figure 19. Connections to Ports A and B of the 8255A Programmable Peripheral Interface

accomplished by the system can be implemented as processes. Hence, the application code would be distinguished into code for each process. Figures 20 through 24 give the flowcharts of the different processes. Before explaining the action of each flowchart, it would be in order to list the symbolic names used along with their significance:

ALARM(N)	- represents the process executed to perform the function necessary for alarm N
AQUEUE(N)	- represents the condition queue for alarm N. It is a queue of blocked processes
AMSG	- the message displayed on the screen with the appropriate alarm number
AMASK	- the system alarm mask
ASTAT(N)	- the alarm number N state variables if 'TRUE' the ALARM(N) process is set; if 'FALSE' the ALARM(N) process is not set
AMSGINUSE	- state variable that controls access to the resource AMSG, resolving contention
AMSGAVAIL	- represents the condition queue of processes waiting to use the resource AMSG
ACQRT	- operating system function called to acquire the CRT
RELCRT	- operating system function called to release the CRT.

The interface between the user and the system is provided by the process 'EXEC' shown in Figure 20. 'EXEC' recognizes the following commands entered at a keyboard:

E, n	- Enable alarm #n
D, n	- Disable alarm #n
P	- Print alarm status



Start
Input a Character Command
Command = 'E' ?
No
Yes
Command = 'D' ?
No
Yes
Command = 'P'
No
Yes
Command = 'Q'
No
Yes
Input # N
Input #N
Print Out Alarm Status for Each Alarm
Halt
Set Appropriate Bit in 'AMASK'
Reset Appropriate Bit in 'AMASK'
Initialize State Variable ASTAT(N) = False
Initialize State Variable ASTAT(N) = False
Enable Alarm #N
Disable Alarm #N
FORMAT ERROR

Figure 20. User-System Interface Process 'EXEC'

Q - Quit, return to the monitor.

At startup time, all the alarms are disabled. The user can selectively enable or disable the alarms he wants. He can also print the status of each alarm at any time by typing 'P'. Finally, when the user wants to halt the system, he enters 'Q'.

When the command 'E' is input, the corresponding bit in the system alarm mask AMASK is set, and the alarm state variable ASTAT(N) is initialized to FALSE. These actions enable the alarm N. Similarly for command 'D', to disable alarm N instead of setting the bit in AMASK, it is reset. For command 'P', the status of each alarm (enabled or disabled) is printed on the CRT. The EXEC process will keep executing continuously until a 'Q' command is entered.

A read/set alarm process, 'RDALARM', shown in Figure 21 is always either running or on the ready queue. Its function is to read the 8255A PPI's alarm port (port A) and if there exists an alarm condition for any of the enabled alarms, the RDALARM process signals the appropriate alarm process. The RDALARM process continuously reads or senses the alarm inputs for an alarm condition, a necessary function.

When a process for alarm #N, ALARM(N) is signalled, the following are the actions performed by the process as shown in Figure 22. The status of the alarm ASTAT(N) is first checked to see if it is set. If it is not, then the process enters a wait state until the status is set. The process then tries to acquire the resource AMSG. After doing so, it sends the alarm number N to the 7-segment display and prints out the message with the alarm number. The resource AMSG is then released and the whole process repeats. The alarm processes are either waiting on the condition queue AQUEUE(N) for a signal from RDALARM or are active.

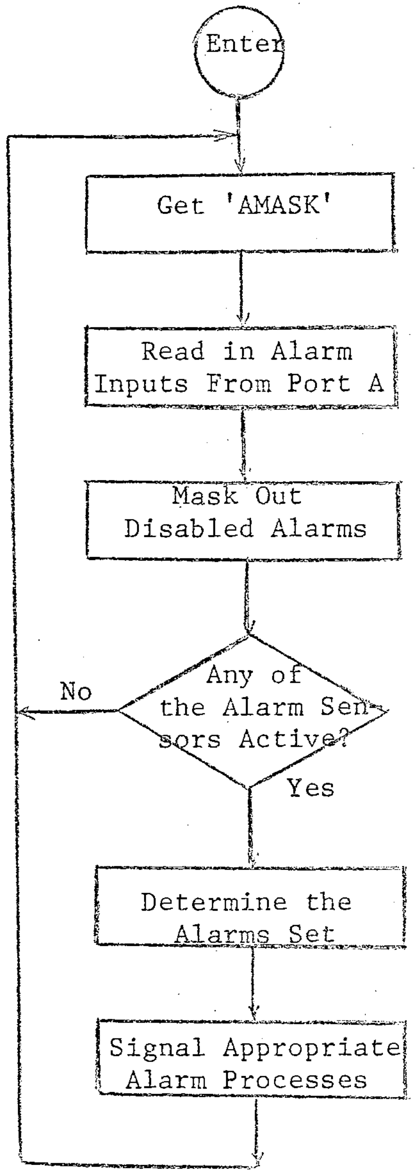

Enter
Get 'AMASK'
Read in Alarm Inputs From Port A
Mask Out Disabled Alarms
Any of the Alarm Sensors Active?
No
Yes
Determine the Alarms Set
Signal Appropriate Alarm Processes

Figure 21. Read/Set Alarm Process 'RDALARM'

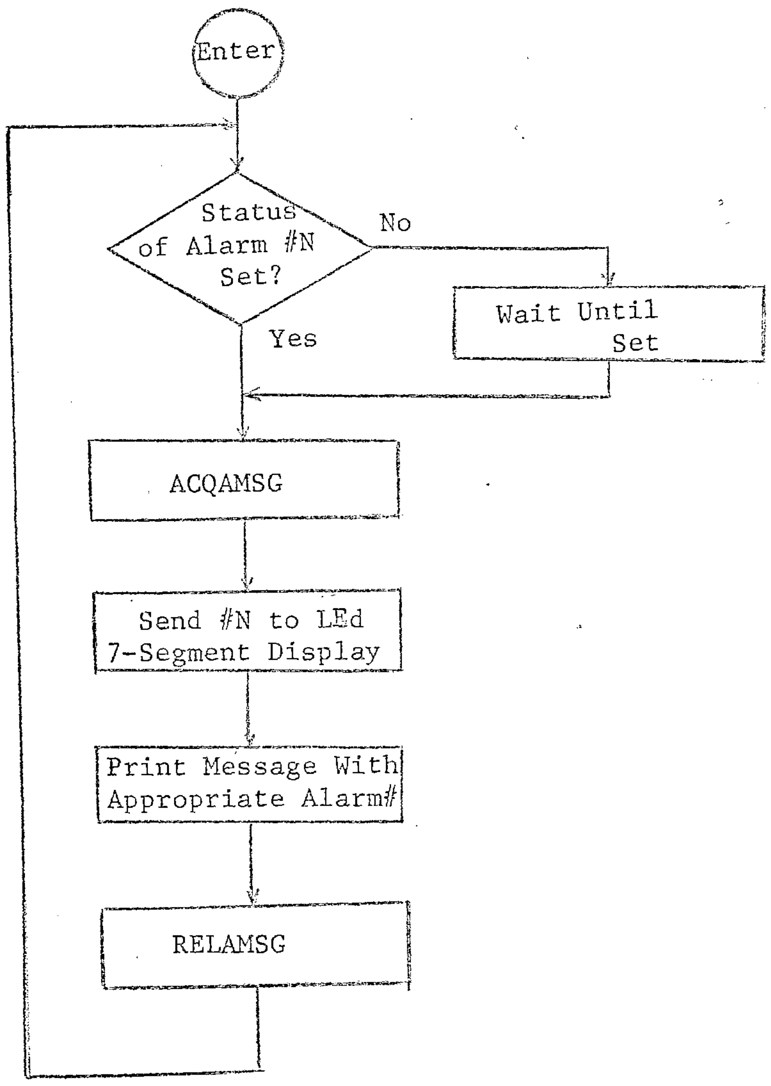

Enter
Status of Alarm #N Set?
No
Yes
Wait Until Set
ACQAMSG
Send #N to LEd 7-Segment Display
Print Message With Appropriate Alarm#
RELAMSG

Figure 22. Alarm Process 'ALARM(N)'

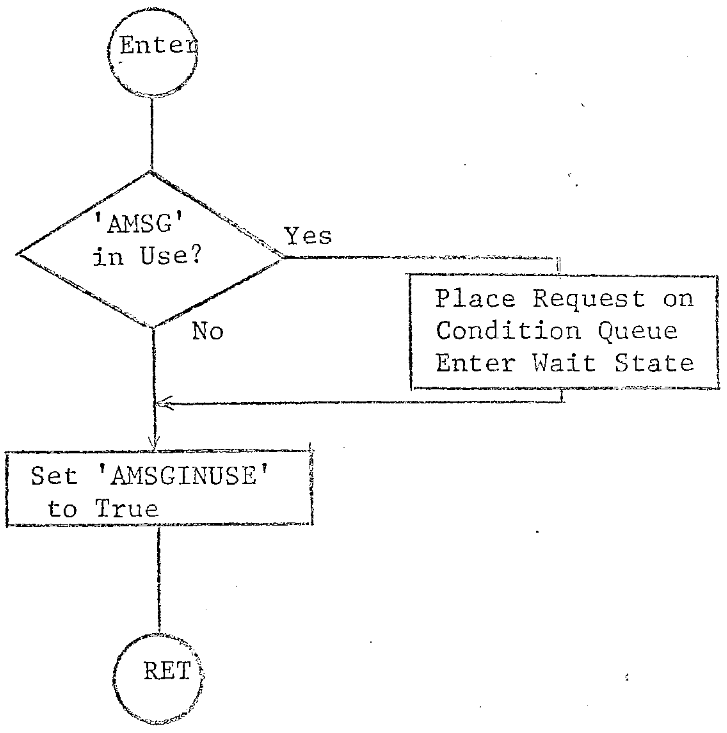
The alarm message to be printed, 'AMSG', is treated as a critical resource and hence the alarm number, which is critical data, is protected during message display. A monitor shown in Figure 23 is used to control access to the resource. The monitor has two entry points, 'ACQAMSG' and 'RELAMSG', that function as the names suggest. In 'ACQAMSG' if the status variable 'AMSGINUSE' is TRUE, then the resource is being used and hence the request is placed on a condition queue where it waits until a previous process relinquishes control of the resource AMSG. Then the state variable AMSGINUSE is set to TRUE, indicating that the resource has been acquired. 'RELAMSG', on the other hand, releases the resource by setting 'AMSGINUSE' to FALSE and signalling the next process waiting on the condition queue.

The routine PMSG shown in Figure 24 is used whenever a message is to be printed on the CRT. It first acquires the CRT by an operating system call ACQCRT. It then prints the message one character at a time on the CRT until the end of the message is reached. The CRT is then released by an operating system call, RELCRT.

It can be seen that the processes EXEC, RDALARM, ALARM(0).... ALARM(7) are continuously executing. The EXEC and ALARM processes could be waiting or active. These ten processes execute concurrently. This is achieved by time-sharing wherein each process is allocated a fixed amount of time, say, 25 milliseconds before control is transferred to another process. This is a very essential feature in real-time system.

The above system design illustrates a typical application of the design process presented previously.

'ACQAMSG' Entry



'RELAMSG' ENTRY

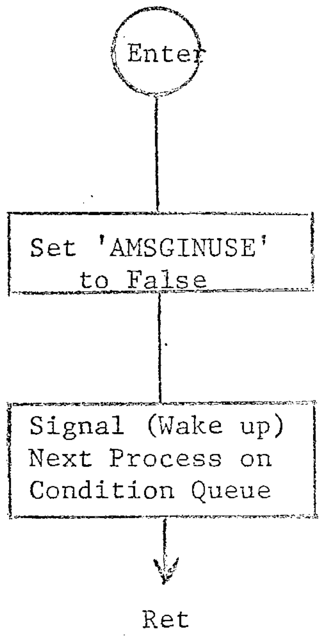


Figure 23. Monitor for Alarm Message 'AMSG'

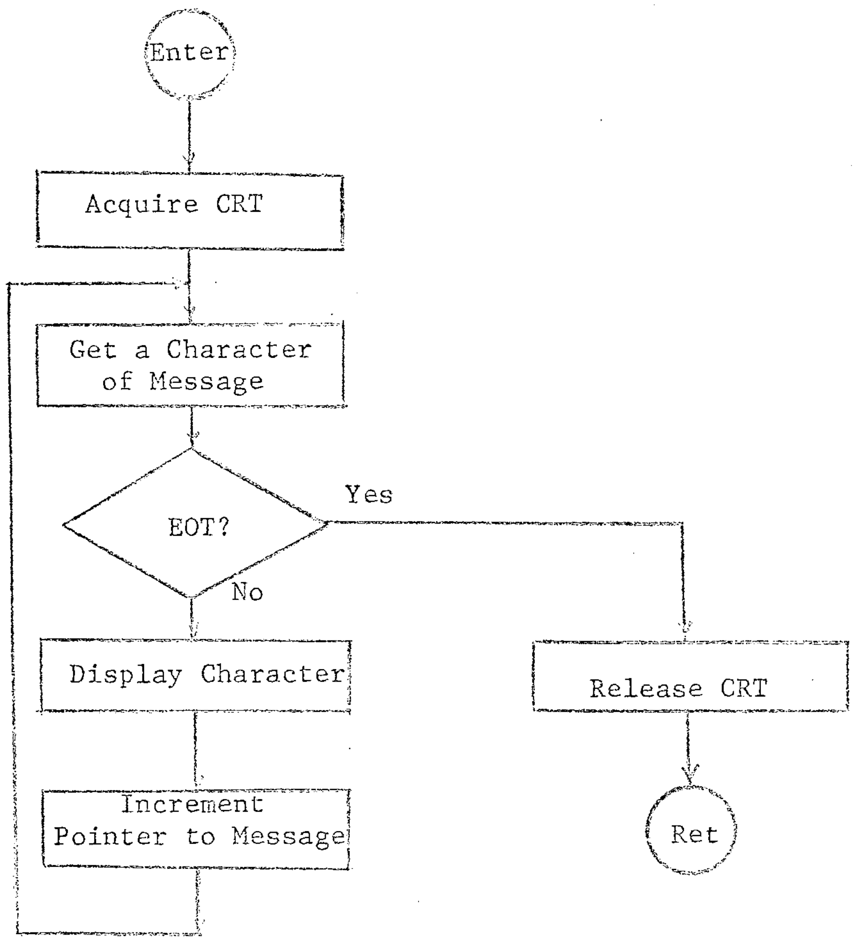


Figure 24. Routine 'PMSG' to Print Message on CRT

A Reentrant Text Editor

A text editor providing sufficient string-manipulation capability can be designed as an 8086-resident program making use of the INTEL 8086 CPU's powerful string manipulation instructions. By making the text editor program reentrant, multiple users can access the text editor.

The selection criteria in hardware and software systems design can also be applied here. In the hardware design, the 8086 CPU based single board computer iSBC86/12A is the main board. Due to its limited serial I/O capabilities, several serial I/O expansion boards (iSBC 534) can be used, enabling many users to be connected to the single board computer. Additional CRT terminals will be necessary, depending on the number of users. Secondary storage will be needed in a multi-user application. A diskette controller (iSBC 206) can be used, providing fast transfer of files from secondary storage to memory, and vice versa. A line printer will also be needed if the users wish to obtain a hard copy of the listings of the contents of their files and other operations.

In the software design, the first step involves selecting a suitable operating system. The OS checklist presented earlier is used. For a reentrant text editor, the operating system must have at least the following necessary features:

Primary application: Development or general purpose

Target processor: INTEL 8086 CPU

Languages supported: PL/M-86 and ASM-86

Processor management: Multiuser and multitasking

Peripheral management: Support CRT, keyboard, line printer. Use interrupts.

Using the above criteria and referring to the operating systems directory, the selection task can be narrowed down to four operating systems. They are the MTOS-86, iRMX-86, XENIX, and DIOS operating systems (14). MTOS-86 developed by Industrial Programming, Inc., is a general purpose and real-time operating system. It supports PL/M-86 and provides multitasking, but it does not support multiusers and ASM-86. XENIX, a level 7 UNIX operating sytem produced by Microsoft, is both a development and general-purpose OS. It also supports up to 25 users, CRTs, character and line printers, and uses interrupts. Though the XENIX operating system has several nice features, it does not support PL/M-86. The iRMX 86 operating system is a development, general purpose and real-time OS developed by Intel Corporation. Designed especially for the INTEL 8086 CPU, it provides multitasking, supports PL/M-86 and ASM-86. It does not support multiusers. The OS selected for this application is the DIOS operating system developed by the Systemathica Consulting Group, Ltd. The DIOS is a development, general purpose and real-time OS. It has all of the desirable features for this application. DIOS supports PL/M-86, ASM-86, provides multitasking and allows multiple users.

The second step in software design involves design of the application code. The operating system divides the memory, allocating storage to each user. Hence, each user has his own buffer area, protected from use by other users connected to the system. A user's file residing on secondary storage is read into the user's buffer area, where the text is edited. After the editing session, the text could be transmitted back to the secondary storage device.

In this application, each user is assigned equal priority and a simple time-slicing mechanism is employed. Each active user is treated as a process, and is allocated a fixed time-slice, say 25 milliseconds.

At the end of 25 milliseconds, an interval timer interrupts the CPU and control is transferred to an interrupt service routine. Depending on the number of users and the system response time, the time-slice is determined. For a reentrant program to work correctly, each time the program is reentered, the CPU registers and local variables in the program must be saved. Each user must have his own save area for this purpose. The process control block shown in Figure 10 has a register save area. A pointer to a save area for the reentrant program's local variables can also be placed in the process control block for each user. The save area needed for the variables has to be determined by the designer.

When a user logs on to the system, a user command process is created as shown in Figure 25. Part of the initialization procedure would consist of setting up a process control block. A prompt is then displayed, and the user enters a command. The user can enter any system command at this level. EDITOR can be one of the commands at this level. If the EDITOR command is entered, the local variables in the editor program are loaded with the values (appropriately initialized) pointed at by the process control block. Control then transfers to the editor program. For an unrecognized command, an appropriate error message is printed and the command prompt is redisplayed.

Only one process can be running at a time. The other waiting processes are placed on a ready queue. This is accomplished by a linked list of process control blocks. When a time-slice interrupt occurs, the interrupt service routine must perform the following functions in order:

Save registers in process control block of running process.

If the interrupted process was executing the editor, save editor's local variables in save area pointed at by process control block of the

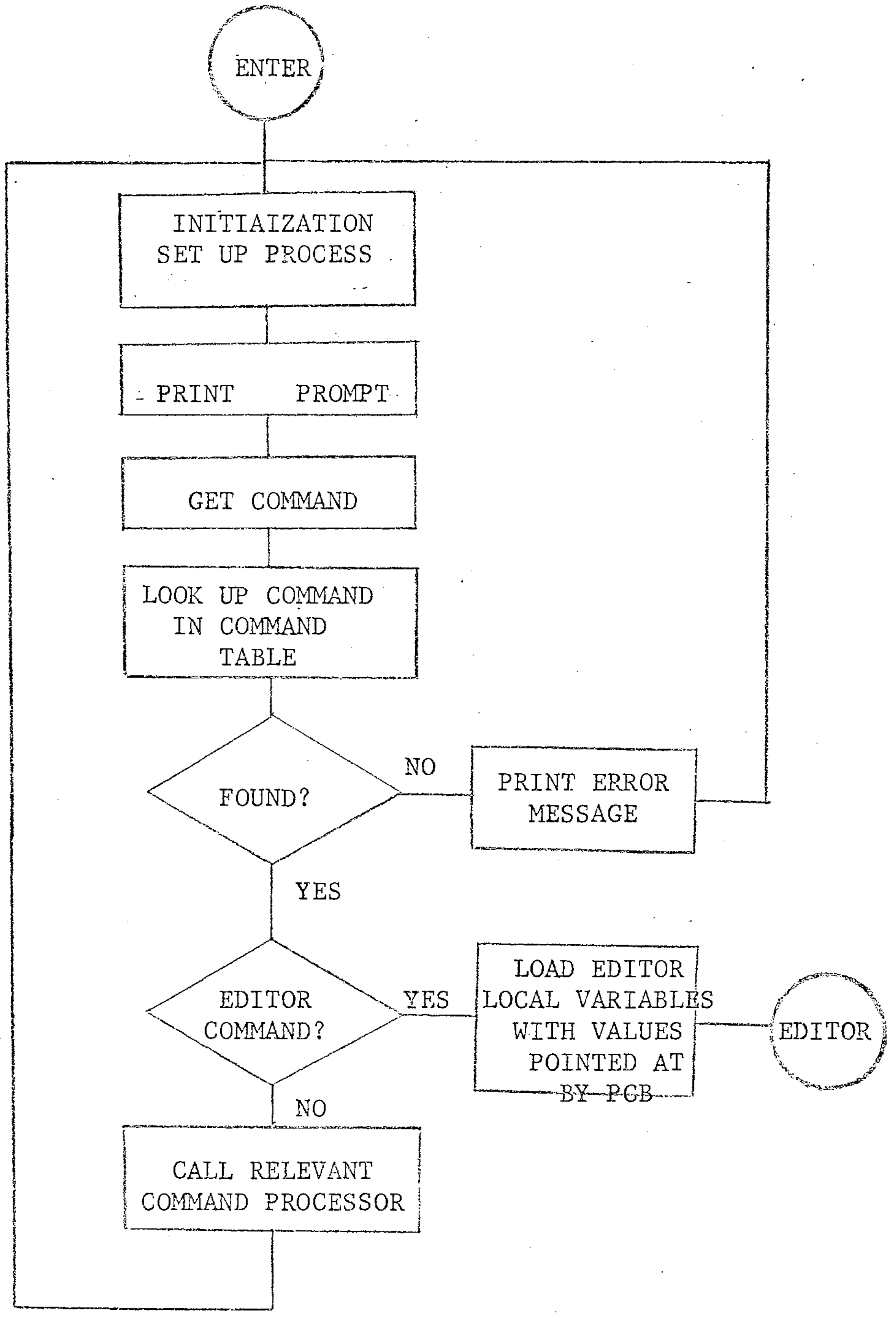

ENTER
INITIAIZATION SET UP PROCESS
PRINT PROMPT
GET COMMAND
LOOK UP COMMAND IN COMMAND TABLE
FOUND?
NO
PRINT ERROR MESSAGE
YES
EDITOR COMMAND?
YES
LOAD EDITOR LOCAL VARIABLES WITH VALUES POINTED AT ~~BY PCB~~
EDITOR
NO
CALL RELEVANT COMMAND PROCESSOR

Figure 25. User Command Process

interrupted process.

Place the interrupted process on the READY queue.

Issue non-specific end-of-interrupt to the programmable interrupt controller.

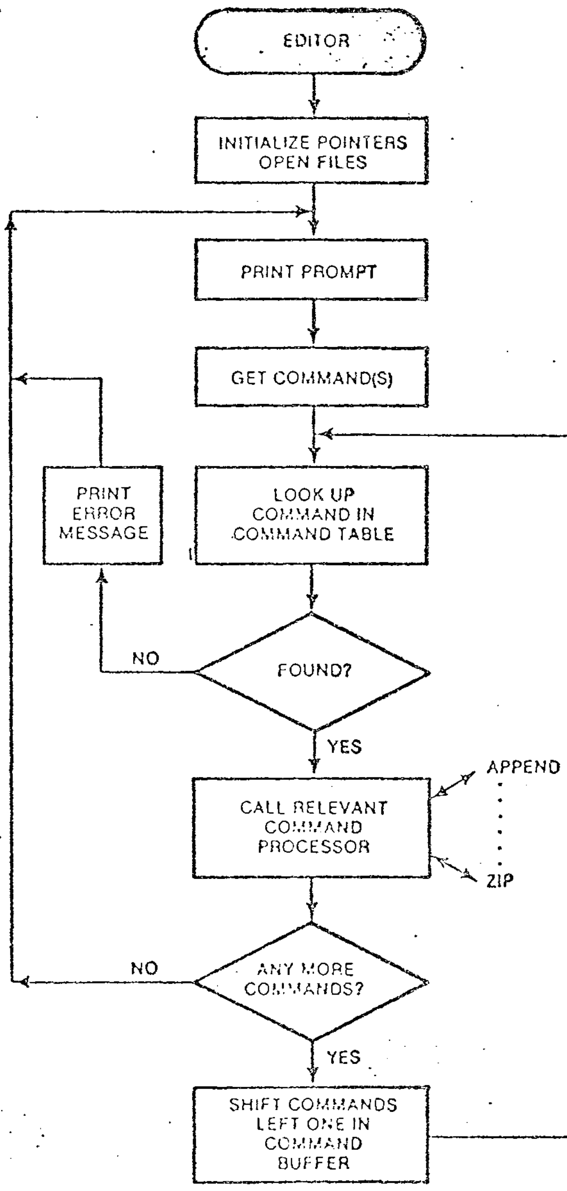
Remove new running process from READY queue.

Restore registers.

If the new running process was interrupted within the editor, restore editor's local variables from the save area pointed at by the process control block of the new running process.

Transfer control to new running process.

The text editor uses basic routines to perform the editing functions. Source files for programs or text can be created and altered by using the editor. By specifying commands (within the editor), routines are utilized to accomplish the necessary tasks. Logically, the commands fall under two classes: input/output and editing commands. The main controlling loop of the editor is shown in Figure 26 (13). Upon entry of the text editor, registers and pointers to the beginning of the buffer, end of buffer, current line, end of text in buffer are initialized. The editor prompt is printed and the user enters a command or commands that are read into a command buffer. After the command buffer is filled, a command table is searched for the requested command. If the command is not found, an error message is printed and the editor command prompt is redisplayed; otherwise, a call occurs to the relevant command processor. After executing the command recognized, the command buffer is checked for stacked commands. If there are any present, they are shifted left in the buffer, and the controlling loop executes as before, searching for and executing commands.


EDITOR
INITIALIZE POINTERS OPEN FILES
PRINT PROMPT
GET COMMAND(S)
LOOK UP COMMAND IN COMMAND TABLE
PRINT ERROR MESSAGE
NO
FOUND?
YES
CALL RELEVANT COMMAND PROCESSOR
APPEND
ZIP
NO
ANY MORE COMMANDS?
YES
SHIFT COMMANDS LEFT ONE IN COMMAND BUFFER

Source: Hemenway, Jack, and Teja, Edward. "Character String Manipulations Power a Text Editor." EDN, August 20, 1979.

Figure 26. Editor Flowchart

This design puts forth the concept of designing a reentrant text editor. The design of editor commands are not covered in this design and can be implemented at a more detailed design level.

CHAPTER V

SUMMARY, CONCLUSIONS, AND FUTURE WORK

System design is a carefully planned process. The designer must know precisely at the beginning what the system is expected to do. Hence, the overall design must be planned very carefully. The designer can use block diagrams to represent what the overall system is supposed to do.

Having put down his thoughts on paper, the designer's next task is to design the hardware and software system. Tradeoffs between hardware and software can be made at various places. Such tradeoffs are the system designer's decision. Some of the important factors affecting his decision could be speed of execution and cost of the overall system. Bearing this in mind, the software and hardware tasks are defined and designed. Throughout the design process, hardware and software tools such as logic analyzers and diagnostic routines are used to check out the system as it is being built. This makes the complete system consist of fewer errors during system integration. The designer should also bear in mind the wide range of off-the-shelf components available before he tries to design the hardware. Similarly in the software design process, software packages are available that free up the designer and allow him to concentrate only on the application code. The final product should be tested thoroughly from the end-user's and the designer's viewpoints. For a newcomer to microcomputer design, the total time spent in the design process normally exceeds the estimated time. To become a productive designer, one must put textbook techniques into action by writing code and constructing the hardware. Practice makes perfect.

Based on this report, some interesting design projects can be initiated. The real-time alarm system can easily be expanded. Port C of the 8255A programmable peripheral interface is presently not being used. It can be used to drive an audible alarm or it can be interfaced with a printer. The alarm process itself could perform a more complicated task. It could trigger a relay, for instance. Based on the text editor application given, one can actually go about the task of coding a reentrant text editor and also designing the editor commands. Hence, a microcomputer design project can be decided on, and the task of software and hardware systems design allocated.

LIST OF REFERENCES

(1) Morse, Stephen P. The 8086 Primer - An Introduction to its Architecture, System Design, and Programming. New Jersey: Hayden Book Company, Inc., 1980.

(2) MCS-86 Macro Assembly Language Reference Manual. California: Intel Corporation, 1979.

(3) PL/M-86 Programming Manual. California: Intel Corporation, 1980.

(4) Peatman, John B. Microcomputer-Based Design. New York: McGraw-Hill Book Company, 1977.

(5) The 8086 Family User's Manual. California: Intel Corporation, 1979.

(6) Madnick, Stuart E. and Donovan, John J. Operating Systems. New York: McGraw-Hill Book Company, 1974.

(7) Hayes, John P. Computer Architecture and Organization. New York: McGraw-Hill Book Company, 1978.

(8) Zaks, Rodnay and Lesea, Austin. Microprocessor Interfacing Techniques. California: Sybex, Inc., 1979.

(9) MCS-86 User's Manual. California: Intel Corporation, 1979.

(10) Systems Data Catalog 1980. California: Intel Corporation, 1979.

(11) iSBC 86/12A Single-Board Computer Hardware Reference Manual. California: Intel Corporation, 1979.

(12) Intel Multibus Specification. California: Intel Corporation, 1979.

(13) Hemenway, Jack and Teja, Edward. "Character String Manipulation Power a Text Editor." EDN (August 20, 1979), pp. 111-116.

(14) Hemenway, Jack and Kotelly, George. "Microcomputer Operating Systems Directory." EDN (November 5, 1980), pp. 276-338.

(15) Hemenway, Jack and Teja, Edward. "Advanced Software Systems Design Course." EDN (October 20, 1979), pp. 294-336.

(16) RMX/86 User's Guide. California: Intel Corporation, 1980.

(17) Ripps, David L. On Operating Systems. New York: Industrial Programming, Inc., 1980.

(18) Schnabel, Dennis. "MDL/μ - A New Language for Effective Microprocessor Software Development." Electronic Design (Sepbember 13, 1979), pp. 102-104.

(19) Laphan, Stephen A. "The 8086 Microprocessor has the Architecture to Handle High-Level Languages Efficiently." Electronic Design (March 1, 1980), pp. 97-99.

(20) Elmore, Mary J. "PL/M-86 Combines Hardware Access With High-Level Language Features." Electronic Design (April 20, 1980), pp. 181-186.

(21) Adams, George. "Reduce Your Microcomputer-Based System Design Time by Using Single Board Microcomputors." Electronic Design (February 1, 1978), pp. 56-66.

(22) Ogdin, Carol A. "Microcomputer Design Course." EDN (November 20, 1976).

(23) Ogdin, Carol A. "Software Design Course." EDN (June 5, 1977).

VITA

Neenendra R. Pandya

Candidate for the Degree of

Master of Science

Report: SYSTEM DESIGN FOR A MULTI-USER MICROPROCESSOR-BASED APPLICATION

Major Field: Computing and Information Science

Biographical:

Personal Data: Born in Madras, India, October 7, 1957, the son of Ramanlal and Bhadrabala Pandya.

Education: Graduated from St. Peter's High School, Maharashtra, India, in December, 1973, with an Indian School Certificate (Senior Cambridge); received Bachelor of Engineering degree in Electrical Engineering from Bangalore University, Bangalore, India, in July, 1979; completed requirements for the Master of Science degree at Oklahoma State University, Stillwater, Oklahoma, in December, 1981.

Professional Experience: Graduate Research Assistant, Department of Agricultural Economics, Oklahoma State University, Stillwater, Oklahoma, February 1980, to September, 1981; Software Engineer, Time Management Software, Inc., Cushing, Oklahoma, September, 1981, to present.