The Minicomputer in On-Line Systems

WINTHROP COMPUTER SYSTEMS SERIES

Gerald M. Weinberg, *editor*

MOSTELLER
Systems Programmer's Problem Solver

HEALEY AND HEBDITCH
The Minicomputer in On-Line Systems

CONWAY, GRIES, AND ZIMMERMAN
A Primer on PASCAL, 2nd ed.

BASSO AND SCHWARTZ
Programming with FORTRAN/WATFOR/WATFIV

SHNEIDERMAN
Software Psychology: Human Factors in Computer and Information Systems

GRAYBEAL AND POOCH
Simulation: Principles and Methods

WALKER
Problems for Computer Solutions Using FORTRAN

WALKER
Problems for Computer Solutions Using BASIC

CHATTERGY AND POOCH
Top-down, Modular Programming in FORTRAN with WATFIV

LINES AND BOEING
Minicomputer Systems

GREENFIELD
Architecture of Microcomputers

NAHIGIAN AND HODGES
Computer Games for Businesses, Schools, and Homes

MONRO
Basic BASIC

CONWAY AND GRIES
An Introduction to Programming:
A Structured Approach Using PL/I and PL/C, 3rd ed.

CONSTABLE AND O'DONNELL
A Programming Logic

CRIPPS
An Introduction to Computer Hardware

COATS AND PARKIN
Computer Models in the Social Sciences

EASLEY
Primer for Small Systems Management

CONWAY
A Primer on Disciplined Programming Using PL/I, PL/CS and PL/CT

FINKENAUR
COBOL for Students: A Programming Primer

WEINBERG, WRIGHT, KAUFFMAN, AND GOETZ
High Level Cobol Programming

CONWAY AND GRIES
Primer on Structured Programming Using PL/I, PL/C, and PL/CT

GILB AND WEINBERG
Humanized Input: Techniques for Reliable Keyed Input

The Minicomputer in On-Line Systems

Small Computers in Terminal-Based Systems and Distributed Processing Networks

Martin Healey
David Hebditch

Winthrop Publishers, Inc., Cambridge, Massachusetts

Library of Congress Cataloging in Publication Data

Healey, Martin.
The minicomputer in on-line systems.

Includes index.
1. Minicomputers. 2. On-line data processing.
3. Electronic data processing—Distributed processing.
I. Hebditch, David Leroy. II. Title.
QA76.5.H444 001.64 81-589
IBSN 0-87626-579-4 AACR2

17 Dunster Street
Cambridge, Massachusetts 02138

Printed in the United States of America.

10 9 8 7 6 5 4 3 2 1

To Jill

Contents

Foreword

Computing, like any high technology business, is burdened with unanswered questions, some serious, others comparatively trivial. A splendid example of the latter category has been the endless debate over who invented the first stored program computer. A more critical inquiry is posed in discussions about just what a minicomputer is, or for that matter what a terminal, a small business system, or a terminal computer is.

Debate questions are always fun. Engaging in deep dialog is a lovely way of spending time. It is at least as good as attending conferences, writing papers, and otherwise avoiding the need for work. Discussing matters whose resolution is largely a semantic issue clearly permits the practitioner the luxury of avoiding technical decisions due last week or last month. Since it is not considered cricket to terminate the conversation of people who debate these lofty and near-impenetrable matters, the clever gamesman can survive by avoiding decisions, with their consequent risk of error.

It has long been known that certain MIS professionals have very bad attitudes on these matters. They attempt to solve problems by creating workable solutions. This evil practice should be stopped, because it might force the debating societies to stop debating and go to work—the last thing the talkers desire. If one is forced to bet one's reputation on a path or a fork in the road, the chances for error are high. Better to debate than to do!

Books about minicomputers invariably seem to fall into the trap of spending several hundred pages attempting to define minicomputers. The matter used to be very simple for pragmatic types who didn't worry about elegance. The parameters were fairly simple: (1) physical size, and (2) manufacturer's name. Consequently, a computer that could be picked up by an average-size individual, one manufactured by anybody other than a traditional mainframer, was by definition a minicomputer. Unfortunately, some recent developments have destroyed this easy definition. There aren't many people who can get their arms safely wrapped around an H-P 3000 or a DEC PDP-11/44. Anyone who *can* is a candidate for several Olympic awards. The other side of the equation has also come apart: IBM entered the minicomputer business directly, Univac joined by acquisition, and Honeywell reenergized its long dormant minicomputer operation. End of one pass at decision-making criteria!

So what is a minicomputer? Can a machine with a 32-bit-wide data bus, a 300 ns cache memory, and a million bytes of main store be called a mini-anything? Of course it can't. The truth is that yesterday's definitions are now nearly meaningless. There is a tidy, smooth curve of machine power beginning with microprocessors and running exponentially up to a Cray-1. Somewhere along this curve the traditional independent observer can decide that these are micros, these are minis, these are maxis, and everything else is a mainframe. You pays your money and takes your choice; who cares what it is called?

Having thus neatly solved the question of defining the minicomputer, it is necessary to find the separation between one of them, whatever they are, and a terminal. This should be a good deal easier because the parameters are better known.

A terminal is a box. That box is remote from a computer. So far, so good. Does a terminal have: (1) intelligence, to process information, (2) private store, (3) mass storage even in limited quantities, (4) human intervention devices such as keyboards, and/or (5) connections to the computer? These questions define the various classes of terminals. If there are enough yes answers, perhaps the box in question is not a terminal at all but a computer. Can the answer be determined numerically? Does a single yes mean terminal and three or more mean computer?

As with the case of the minicomputer definition, I am now involved in a semantic snarl-up at least partially of my own making. Indeed, does it really matter? Isn't it more than enough to say that a terminal is a remote electronic device that has the power to somehow communicate something to somewhere in one of a variety of forms? Enough!

None of this semantic footwork really matters very much to the dirty-handed workers in the trade. People take a machine, grab some devices that work remotely and go to work building systems that provide intelligence to the end user community residing at the periphery. That is what it is all about. How the goal is reached and what it is called should be relegated to the Oxford Union.

Having placed definitions in the category of a problem that has been solved (a longtime ploy of politicians who believe that if the problem is said to be solved it is), we must turn to the real snags in computing during the next few years.

The data communications explosion begun so promisingly a few years ago continues to run into roadblocks. Most of the obstacles are erected by longtime communications carriers whose general approach is to try to bar all the new freedoms the FCC carefully nursed into life. The common carriers are not stupid. They see major revenue sources flying away as their aging, analog-based plant rapidly falls into disuse before its natural accounting life is ended. There is little quarrel with the notion that a forty year write-off on technological equipment is an invitation to disaster. However, the fiscal policies and practices of the carriers were established and firmly set long before computers were invented. The repeated, public predictions by Howard Anderson of the Yankee Group that the common carriers will be an easy target when the computer people really start after them seem to be coming true. But this is another of those debating-society issues.

The carriers are digging in for a long fight. It isn't merely a matter of matching their services to those the independents are making available. To quote myself from a near-forgotten *Datamation* article, "If MCI and Datran didn't exist, we would have had to invent them to keep AT&T honest." What has happened is that the carriers have reacted savagely enough to make the future bleak for all but the best financed independents. Datran is gone, even though it cost AT&T $50 million in settlements. By its own admission MCI is only marginally profitable. The regional carriers are gone. Telenet was acquired by GT&E.

The traditional carriers are defending on many levels. First, they take every legal action possible to delay the implementation of new services by independents. Second, they are providing their own competitive services. Third, they are attempting to remove from service functional capabilities already in general use—for example, distance pricing and private line networks. Finally, they are always just a little slow in connecting local loops and repairing outages.

All of this means that the natural, normal, and highly proper convergence of data processing and data communications is a good deal more difficult than it should be. The problems are political and economic, not technical. This book describes systems in being and how they are built. Neither it, nor any other book on the market, can forecast the political climate that may well undercut a sound technical performance.

The clash between a slow-moving, regulated industry and a very rapidly changing, thoroughly unregulated business is of increasing concern to the user community. Knowing how to build a system is of little value if a very large traffic cop keeps saying, "You can't do that" at regular intervals. The cop wears many disguises. He may appear as a British labor leader who has instructed his men not to install non-GPO modems, or as a Nigerian minister who refuses to allow any encoded data on public transmission facilities, or as a Canadian telecommunications minister who believes that all necessary transmission can be accomplished on public networks.

Who suffers from these actions? Not the mythical general public whose interests are allegedly being protected. The loser is the end user at whose behest remote, terminal-based systems are being constructed. The solution is political. Corporations don't have the influence when it comes to a clash between the need for more effective business communications and an impact on the plain old telephone service.

A sense of outrage continues to exist in the computer community over the restrictions, real and apparent, arbitrary and legislated, that grow ever more widespread. The game is at a high level because the revenues involved are very large, billions of dollars. Even the most astute technical people are likely to be blind to these affairs; their interest is in getting the job done.

Before 1976, the direction and pace of the data processing and data communications convergence were fairly clear. The technical job to be accomplished was, and still is, becoming ever clearer. Books like this describe what has to be done. But the whole effort will fall apart if the industry's technical experts do not take the time and energy to come to grips with political and legislative forces. Things are happening that may undermine even the best planned efforts. It was previously considered reasonable to assume that telecommunications regulations would be eased. In actuality, matters have gone quite the other way. The French PTT still wants to pull cable in an era when satellite transmission is completely routine. Fiber-optical transmission is regarded with suspicion by those who grew up with twisted pairs of copper wires. Digitally based communications equipment cannot approach peak effectiveness when inhibited by operating parameters in a thirty-year-old analog carrier. Voice, data, and image transmissions are still seen as separate entities when all three have become little more than the movement of digital bit streams.

The technology of communications-based systems is expanding at a rapid pace. Each new development, however, is delayed by the defenders of the past. The excuses vary but the most commonly heard are: "unproven technology," "too great an impact on existing services," and "no tariff has yet been filed."

The common carrier attack on remote systems has been sustained, but at worst their efforts can only slow down implementations. But there is another emerging threat that has the possibility of totally destroying effective remote computing—the concept that data should be held within national boundaries as a national resource.

Remote computing systems by their very nature weaken the idea that all data should be maintained locally. In fact, while much of day-to-day operational data should properly and for good technical reasons remain local, there is a valid set of aggregate data that make little sense locally. How is a modern corporation to optimize its use of scarce resources if arbitrary, often capricious, regulations are placed on its ability to move the data to the location where it is needed?

The issue of transborder data flow is rapidly growing in its impact upon remote computing systems. It is beyond the scope of this book to discuss whether it is a real issue or one dreamed up by a tiny minority of social activists as another means for controlling large corporations. However, it is an issue with a major potential for mischief.

Of still more recent generation is the issue of national vulnerability to computer/communication outages. The thesis as evolved by a Swedish governmental committee revolves around the notion that each national society must have all of its own facilities for keeping the computer/communications complex going. Why? Because the big, bad wolf, presumably an American-made wolf, will come along and reduce a nation to third-rate status by failing to provide spare parts or the technical know-how to keep the complex functioning. It is hard to take such scare tactics seriously. Yet it would be foolish to assume that the threat will simply disappear.

The point in both of these cases is the same. It is no longer sufficient for a technician to stay purely technical. The construction of computer-based systems is full of social, cultural, economic, and political questions that can no longer be cheerfully disregarded. Enough good systems have been built for the alarm-raisers to have had their interest awakened. A cynic might suggest that we ought to have failed more often because if that were the case, the politicians would leave us alone. So perhaps some of the energy and effort devoted to trying to decide what a minicomputer is, or what a terminal is, should be applied to the process of maintaining freedom to implement. The definitions are only of importance to lexicographers, but the politics are critical. The deeper people delve into remote computing systems, the more the impact will be if the system proves "culturally unacceptable."

We used to think that technology was hard. It isn't. It's still not easy or off-the-shelf. The multiple forces that pull and tug within a system are not that easy to see, and it is still hard to find all the trade-offs. But Healey and Hebditch have pulled out most of the issues into plain sight and tried to dispel the mystery. It's not a black art, nor does it require implementers capable of leaping tall buildings.

Minicomputer systems can be thoroughly constructive for the corporation in which they are based. They can also be major traps for the unwary. There are rules of the game that must be obeyed even as with large-scale mainframes. Healey and Hebditch detail the structure for minicomputers in the distributed environment. Their game plan is sound. Now it is up to all you implementers to follow their precepts.

PHILIP H. DORN

Preface

This book is the culmination of a discussion that has been going on for some four years between the authors. During the mid-1970s we have seen the steady convergence of computer and telecommunications technologies. In particular, the data processing industry has continued to move away from batch processing to transaction processing, and networks of terminals have improved user access to computer systems.

Conspicuous in these trends has been the emergence of the minicomputer. These compact and powerful processors, which originated in control systems, have found a new role in the commercial field as data concentrators, message switchers, front-end processors, terminal controllers, and special-purpose transaction processors. A common characteristic in these systems is the use of interactive and telecommunications facilities for data transport.

Although the cost/performance ratio and general modularity of minis make them very difficult to ignore for many applications, the problems of system design, development, and implementation can be numerous.

The authors are consultants, one majoring in minicomputers, the other in communications systems, both having a reasonable knowledge of the other's specialization. During many discussions (in just as many watering holes), we managed to convince ourselves that if we had an *average* understanding of these problems, then at least half the data processing profession was really struggling!

Minis are ideally structured for handling data communications and terminals, but that does not mean that they always do it well. This book is intended as a review of minis, communications, and how the two work together. Many products are described by the way of illustration; where a specific device or software package is mentioned, we do not intend to imply that it is any better or worse than comparable components. merely that it is representative of a certain approach.

If this book is for anyone in particular it is for the COBOL programmer whose boss is planning to buy a six-pack of minicomputers on which to base a network of interactive terminals. But we hope that technical managers, designers, systems programmers, and application programmers in user companies, manufacturers, and software houses will find it helpful in the development of effective systems as well as in the development of their personal expertise.

MARTIN HEALEY
DAVID HEBDITCH

1

The Role of the Minicomputer in Data Communications and Distributed Processing Systems

This chapter reviews the various roles minicomputers play in terminal-based data processing and communications systems to provide a background against which we can consider the technical problems involved in implementing minis in such roles. For the time being we shall define the mini as a small-scale, general-purpose computer such as the Digital Equipment Corporation's PDP-11 or Data General's NOVA series. Small business computers (such as the IBM System/3 and ICL 2903) and special-purpose distributed processing products (like the Burroughs B776 and IBM 8100) are excluded. A more complete definition of the minicomputer will be discussed in chapter 2.

Some Typical Systems

When minicomputers were first developed, they tended to be used in industrial process-control and data-acquisition systems. In the former case, the nature of the application demanded a processor capable of acting in "real time," in the original sense of the expression, a processor that could economically and efficiently interface to a number of local and/or remote telemetry and control devices. Most minicomputers were designed, therefore, to operate in a highly responsive on-line environment. Their development engineers had probably never heard of batch processing (or data processing, for that matter). Universities and research bodies also quickly saw the potential of the mini, which typically is used for one-off jobs generally developed in batch or (more often) interactively, using high-level languages such as FORTRAN or BASIC. One of the earliest uses of minis outside the industrial and research fields was in message switching. Clear-text messages received over telegraph circuits are transmitted to destination teletypewriters/teleprinters according to routing information held in standard or nonstandard headers. Disc or drum storage may be used to implement a delayed-delivery facility.

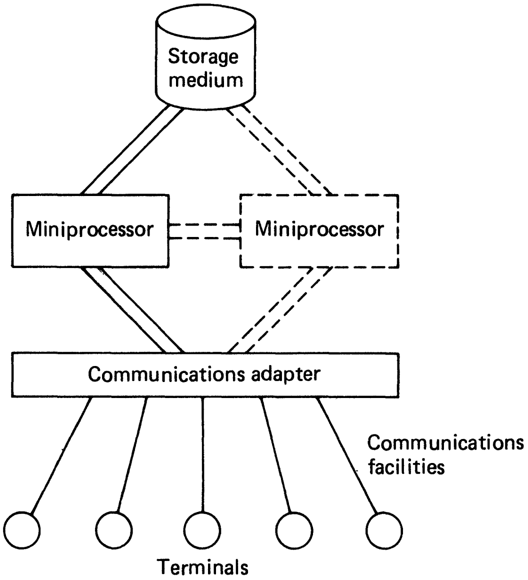


Figure 1.1
The minicomputer as the central site processor (in single or multiple configurations)

All the above systems are typical of the mini used in a stand-alone situation (as shown in figure 1.1). Until recently, the application of the mini as the main computer in commercial data-processing systems was somewhat limited mainly as a result of the lack of suitable software. Today the problem is no longer a lack of software but the choice of a package/language from the plethora of programs now available for business applications. Most such packages assume application as terminal-based transaction processing systems (the structure and relevance of this approach will be considered in chapters 7 and 8). The two earliest and most widespread uses of minis in commercial teleprocessing were as front-end processors (figure 1.2) and as remote batch terminals (figure 1.3).

The first generation of communications control units (CCUs) typified by the IBM 2701, 2702, and 2703 were hardwired devices, inflexible and expensive. In order to achieve a more adaptable, cost-effective solution that would relieve the mainframe of some of its steadily increasing control-program load, many software houses (and some users) developed their own software to work in one of three ways: to emulate the CCU (for example, the IBM 2703); to emulate another peripheral controller (for example, the IBM 3803 magnetic tape controller); or simply to support the mainframe input/output (I/O) channel.

Because of its market size IBM was inevitably subject to attack by this approach: its response was that because central processors and storage were becoming so cheap, there was no point in shifting control program functions into

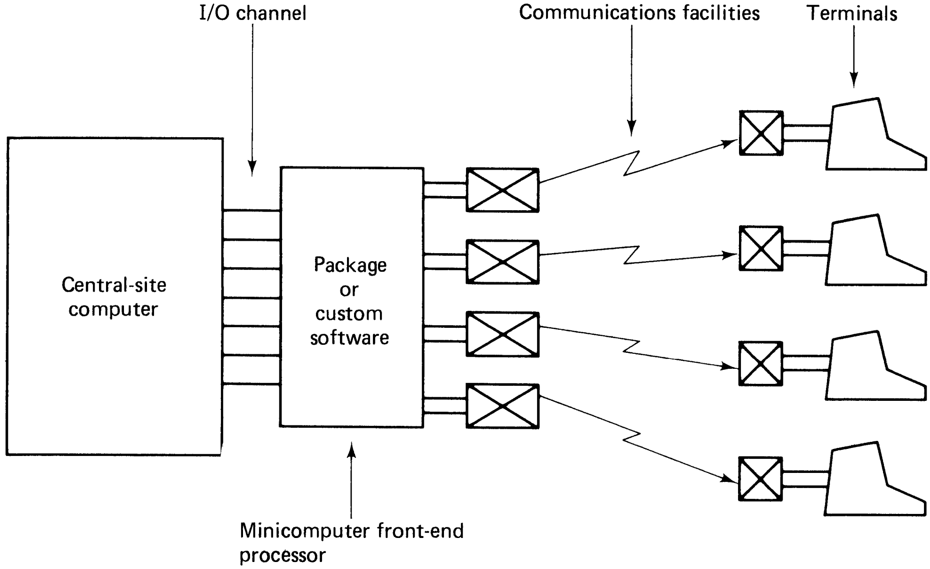

I/O channel
Communications facilities
Terminals
Central-site computer
Package or custom software
Minicomputer front-end processor

Figure 1.2
The minicomputer used in a communications control unit (front-end processor)

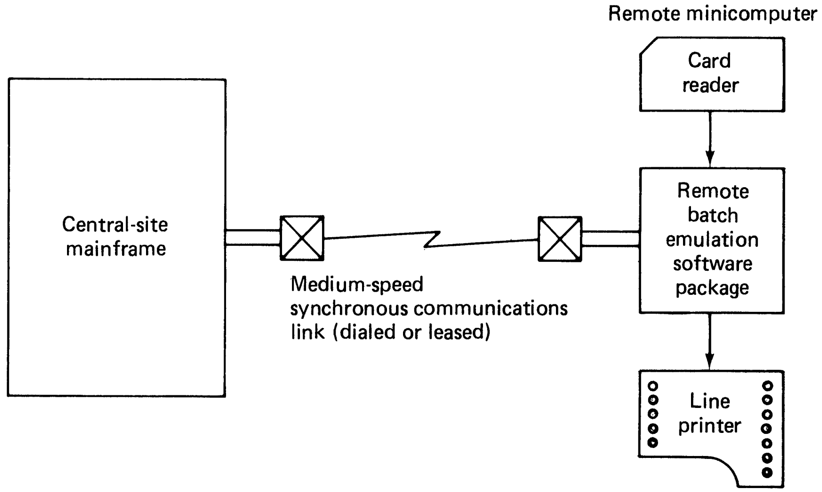

Remote minicomputer
Card reader
Central-site mainframe
Remote batch emulation software package
Medium-speed synchronous communications link (dialed or leased)
Line printer

Figure 1.3
The minicomputer used to emulate a special-purpose remote batch terminal (potential distributed processing system)

the communications control unit. Soon after that, IBM announced the 3704 and 3705 programmable control units—not necessarily because the company had changed its mind about front-ending, but because it was becoming impossible to fulfill users' increasingly heterogeneous teleprocessing needs in a hardwired controller. In fact, since the 370X was introduced, the control software in the central processing unit (CPU) has become larger, not smaller. In spite of the 370X, companies still find it cost-beneficial to front-end their mainframes with a "foreign" mini. Other suppliers who have been front-ended include Control Data Corporation, Univac, and Burroughs. (Honeywell is front-ending with its own minis.)

The minisuppliers and systems houses have also focused their attention on another communications product: the remote batch (or remote job entry) terminal. Originally the RBT was also a hardwired device, typically providing communication with remote input/output peripherals such as card readers and line printers. Independent suppliers soon found that they could put up an alternative package that included similar peripherals but used a mini-based controller and a program that emulated operations used by the mainframe manufacturers' products. Such emulators have been produced for the IBM 2780/3780, ICL 7020, CDC UT-200, Univac DCT-2000, and others.

One advantage of the mini solution is that merely by changing the emulator program one can work to a Univac data center for part of a day and to an IBM center for the rest. Moreover, once the mini is installed other peripherals (including interactive terminals) can be connected for applications. Even in its remote batch configuration, the system can be used for local vetting of the input. This is the beginning of distributed processing, which will be discussed later.

Figure 1.4 illustrates the use of minis in another type of terminal; the interactive visual display unit (VDU) or keyboard printer. In such devices the miniprocessor was closely integrated into the product. In practice many commercially available units (Datapoint, for instance) employ specially made processors, often micro-based. The processor and storage are used to refresh the cathode ray tube (CRT) and to implement various facilities (variable and protected fields, selective transmission) as well as to control the communications function to the CPU. The minis still are relatively low in power because of the limited demands made upon them. As with RBTs, such programmable terminals (especially the VDUs) were provided with emulators so they could be sold in competition with the equivalent units from the mainframe company. Such units included the IBM 2265, IBM 3275, and ICL 7181 stand-alone displays. Peripherals were generally limited to compact media such as cassette tapes and discettes (floppy discs).

Many remote terminal products use the cluster concept in which a number of terminals located in one office block or plant site can share the logic of a single local controller (see figure 1.5). Rather than using expensive modems, the terminals are connected locally via coaxial or multi-core cable (V24 or current loop). Such cheaper circuits generally give a maximum terminal-to-controller distance of less than 1000 meters without amplification or regeneration. The

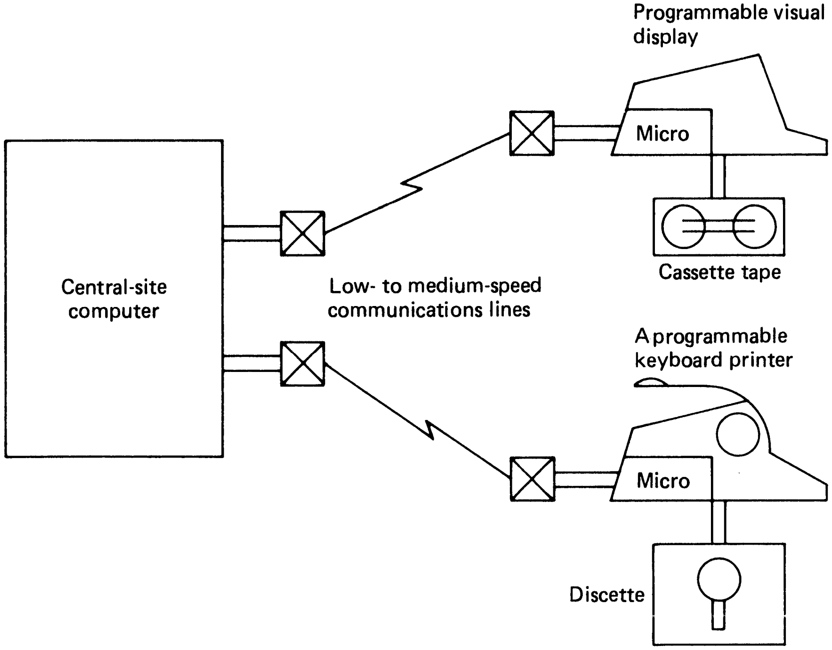

Programmable visual display
Micro
Central-site computer
Cassette tape
Low- to medium-speed communications lines
A programmable keyboard printer
Micro
Discette

Figure 1.4
The minicomputer and small special-purpose computer in programmable (intelligent) terminals (distributed processing system)

cluster approach is often used for general-purpose VDUs but can also be encountered in more special-purpose configurations such as controlling point-of-sale terminals in a department store or shop-floor terminals in a production plant. If the VDUs are attached through video cables, the screen may be refreshed from a buffer that is an area of mini main storage rather than a component of the display itself. This reduces the cost of the terminals but increases the overhead on the controller. If, for example, each CRT needs refreshing 50 (or 60) times per second, then an interrupt needs to be raised once every 20 milliseconds for each terminal so that the data can be "rewritten" to the screen. This usually imposes such an overhead that many such cluster controllers rely on a hardware component for refreshing the screen, a solution that steals memory cycles rather than processor cycles. These problems do not, of course, exist with the special-purpose terminals (unless they have unbuffered CRTs incorporated in them). Hardware implementation of clustered VDUs are typified by the IBM 3270 range. With some terminal systems, the companies developing the mini-based cluster controllers often provide software that enables them to emulate the 3270, thereby making it possible to compete with IBM. Such suppliers include Harris and Raytheon. The use of an emulation package sometimes precludes the development of

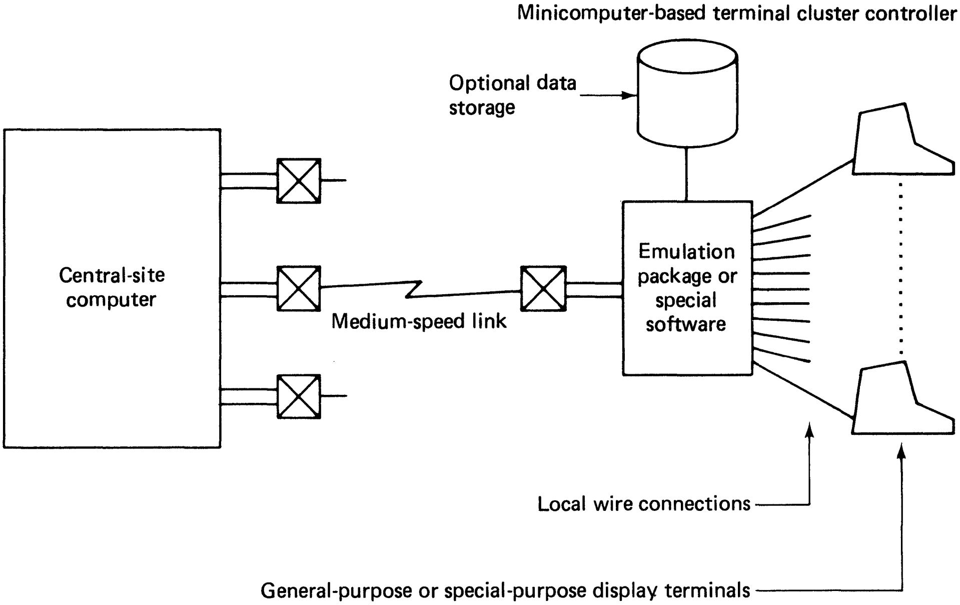

Minicomputer-based terminal cluster controller
Optional data storage
Central-site computer
Medium-speed link
Emulation package or special software
Local wire connections
General-purpose or special-purpose display terminals

Figure 1.5
The minicomputer as a remote terminal cluster controller (potential distributed processing system)

applications routines for the cluster controller. On the other hand, many suppliers now positively encourage the insertion at least of data vetting routines by providing a high-level programming language (Mohawk Data Science's MPL, for example).

Also falling loosely into the category of cluster controller are the key-to-tape (key-to-disc) systems featuring a medium- to high-speed communications adapter which can be used for the onward transmission of data to a central site. Most such key-to-disc systems are based on minicomputers packaged in by the supplier. These systems are marketed by companies like MDS, CMC, and Entrex (Redifon/Nixdorf). In addition to providing local power to terminals, programmable cluster controllers also reduce the cost of communications facilities to the central site. Minis are sometimes used to perform this function exclusively, as shown in figure 1.6. In this case two minis are being used as transparent time-division multiplexors (TDMs). This technique avoids the need for separate dialed or leased lines (telephone or telegraph) from each terminal to the CPU. Instead, a terminal (usually a low-speed interactive device such as a Teletype or IBM 2741) is connected to the nearest available remote multiplexor. Each port at the multiplexor is allocated one or more character positions in blocks of data which are exchanged between the remote multiplexor (MUX) and one of almost identical configuration at the CPU. The in-house MUX receives the transmission blocks and distributes

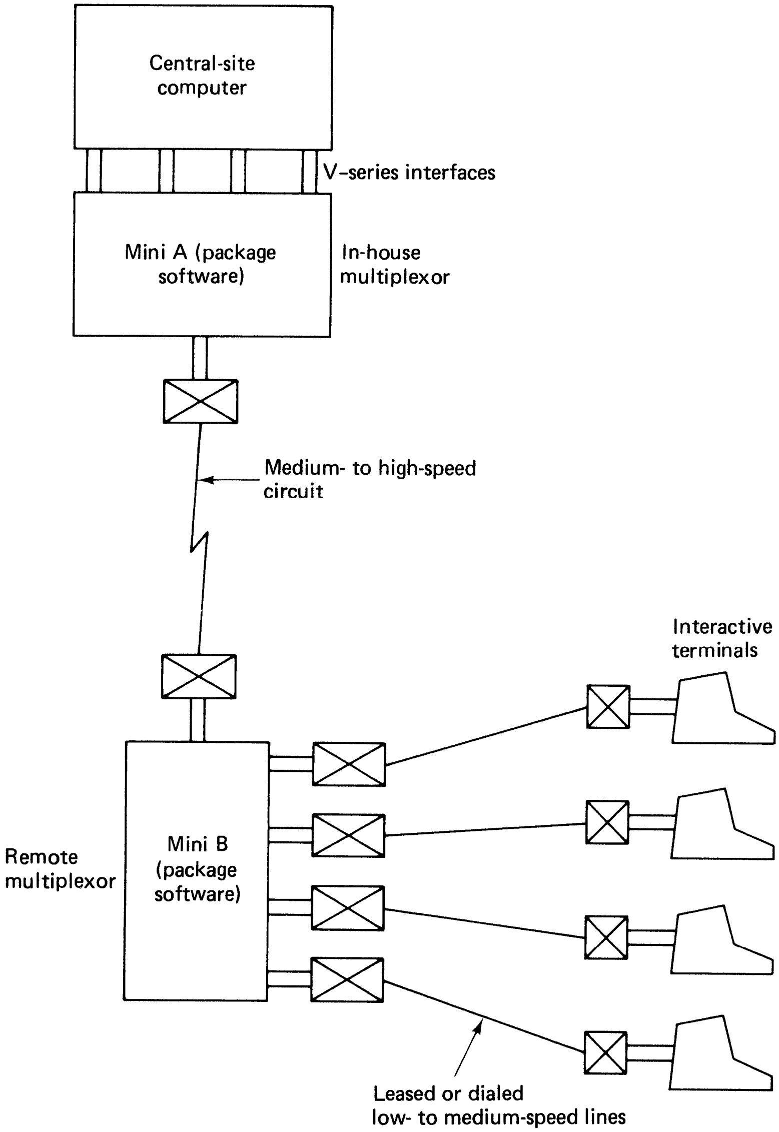

Central-site computer
V-series interfaces
Mini A (package software)
In-house multiplexor
Medium- to high-speed circuit
Interactive terminals
Remote multiplexor
Mini B (package software)
Leased or dialed low- to medium-speed lines

Figure 1.6
The minicomputer as a transparent time-division multiplexor (TDM)

each character to its relevant port (the one through which the CPU expects to communicate with that terminal). Similarly, the ports between the MUX and the CPU are scanned for outgoing characters and these are blocked together for transmission to the remote unit which will similarly distribute them onward to the terminals. Normally, each terminal has positions in a block permanently allocated to it. The faster the speed of the terminal (or line, if it is buffered) the more positions will be allocated. This approach can be wasteful, because even if a terminal is inactive (albeit momentarily) a null character still needs to be sent in order to maintain the positional identification of the subsequent characters. To overcome this, the software packages written for some mini-based multiplexors avoid positional significance by sending two characters, the line number and the data character itself. Although this would appear to double the occupancy of the medium- to high-speed circuit linking the multiplexors, the net effect is often beneficial, since interactive terminals usually transmit and receive very few characters while they are switched on. This is generally known as "statistical multiplexing." At the central-site computer, the multiplexor may be a separate device interfacing to a communications control unit or it may be a front-end processor in its own right. In the latter case, the connection to the CPU will be via the I/O channel rather than CCITT V-series (EIA RS-232) interfaces.

It is most unusual for application functions, even very low-level ones, to be programmed into a TDM. These are essentially transparent devices (neither the terminal nor the CPU needs be aware of their existence in the network) and the timing constraints tend to prohibit the meaningful intervention of processing activity. *Concentrators* are nontransparent devices that also serve to reduce line costs but at the same time enable application processing and control functions (such as those possible in mini-based cluster controllers) to be implemented (see figure 1.7). Unlike terminal controllers, concentrators are used when the terminals are remote from the concentrator and connected over telegraph or telephone lines (which may be dialed, leased point-to-point, or even leased multipoint circuits).

Whereas time-division multiplexors are *character-interleaved* systems, concentrators work on a *message-interleaved* basis. To each incoming message, the concentrator software adds a prefix containing the source-line address and/or terminal address. Messages can then be formed into larger blocks for onward transmission to the central site. The same arrangement in the opposite sense is used for data going from the computer to the terminals. Note in figure 1.7 that the concentrator does not require an equivalent device at the center and is treated by the CPU as though it were a terminal itself. Reducing the number of physical terminals (as against logical terminals) directly connected to the CCU has the effect of reducing processor overheads.

Emulation techniques are sometimes used with concentrators but not nearly as often as with cluster controllers. Whether emulation is a factor or not, the use of minicomputers as concentrators provides substantial opportunities for applications programming. The peripherals available—including disc (especially low-cost cartridge), tape, and line printers—serve even further to facilitate the development of relatively sophisticated application functions. Minis also provide greater flexibility in the type and variety of terminals that can be attached and the method of communicating to the central site. For example, the main link may be duplicated or the concentrators themselves may be interlinked to provide an alternative path to the CPU in case of line failure.

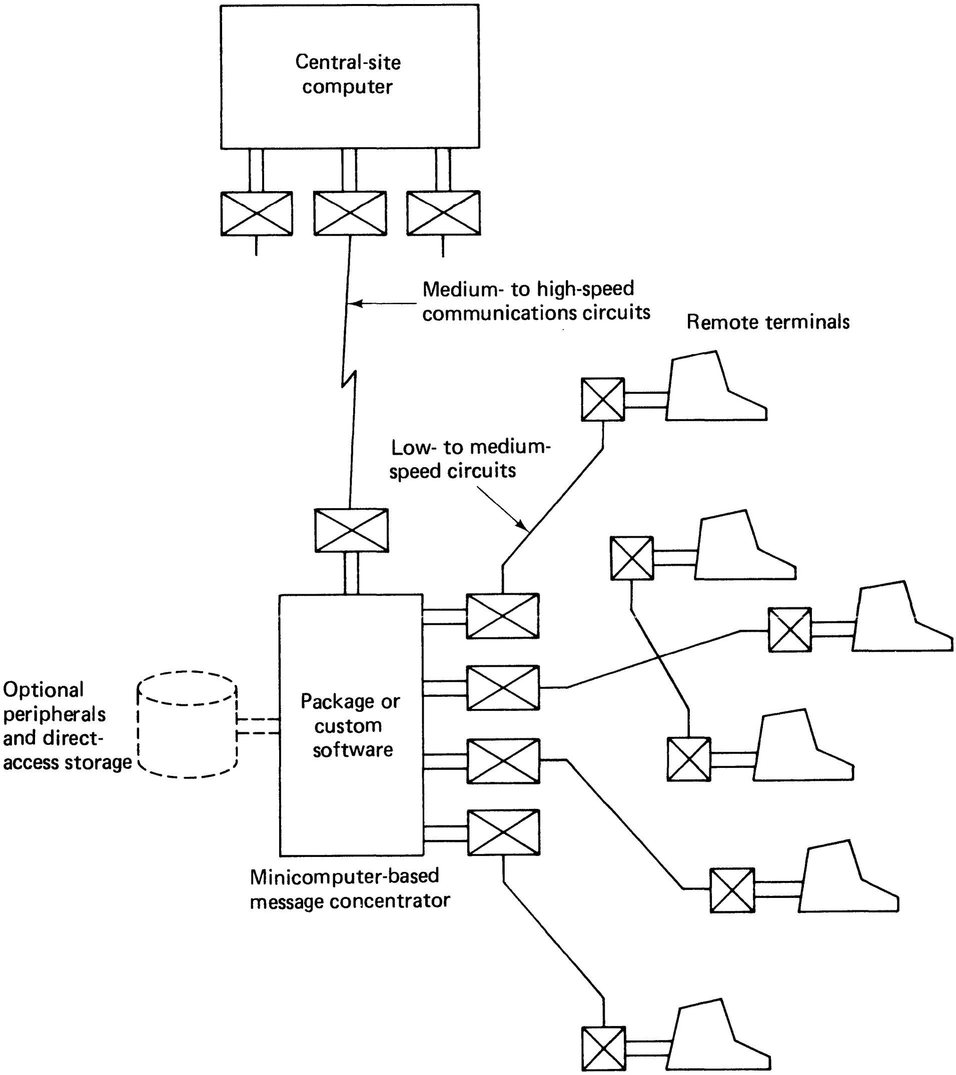

Central-site computer
Medium- to high-speed communications circuits
Remote terminals
Low- to medium-speed circuits
Optional peripherals and direct-access storage
Package or custom software
Minicomputer-based message concentrator

Figure 1.7
The minicomputer as a communications concentrator (potential distributed processing system)

The use of minis in some of the communications networks described (in terminals, cluster controllers, and concentrators) clearly opens up possibilities of developing *distributed processing* systems. Such systems are characterized by the implementation of *application* functions (in addition to *control* functions) in two or more system components that are linked together in some way. These systems are often confused with *distributed switching* networks such as the packet-

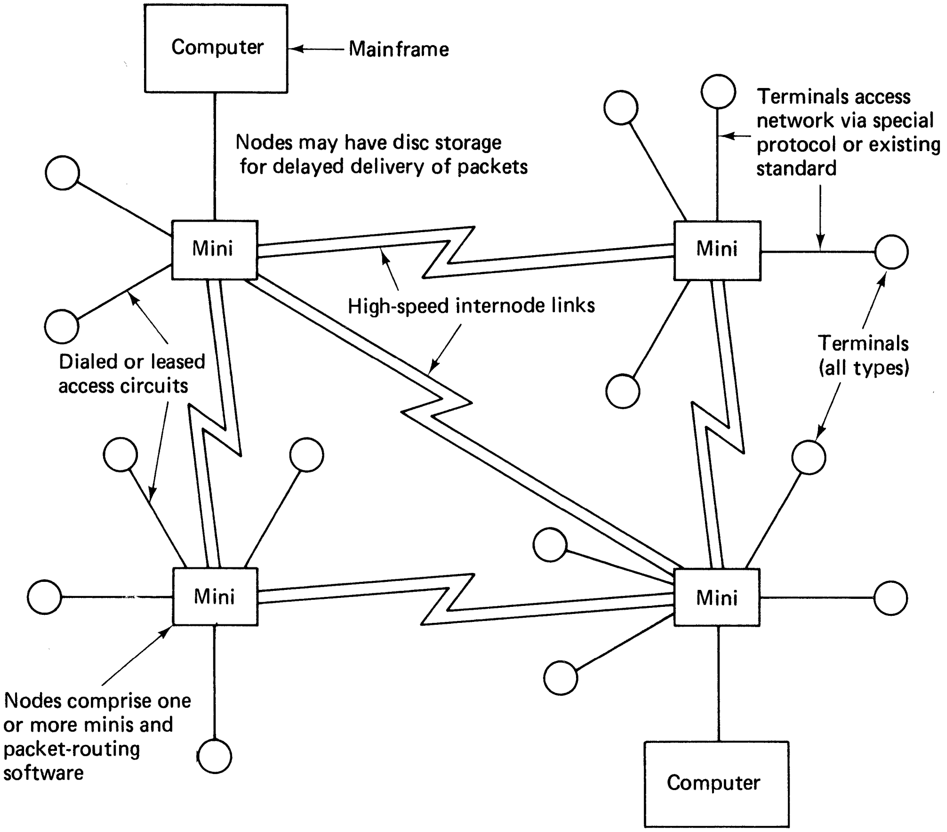

Computer
Mainframe
Nodes may have disc storage for delayed delivery of packets
Terminals access network via special protocol or existing standard
Mini
Mini
High-speed internode links
Terminals (all types)
Dialed or leased access circuits
Mini
Mini
Nodes comprise one or more minis and packet-routing software
Computer

Figure 1.8
The minicomputer as a packet-switching node (distributed switching system)

switching network shown in figure 1.8. All packet-switching networds do (or should do) is improve the performance and reduce the cost of data transmission. Of course, a distributed processing system may use a packet-switching network for communications between terminals and CPU, but no applications-related code can exist in the switching nodes. In almost all packet networks installed to date, minicomputers have been used for the switching nodes. Packet-switching will be discussed further in chapter 3.

Why Use a Mini?

When confronted with the problem of selecting the most appropriate hardware for a particular communications function the system designer can choose from four main classes of equipment:

	STRONG FEATURES	WEAK FEATURES
General-purpose mainframes	Power* Expandability* Supplier stability Resilience	Price Flexibility* Adaptability* Architecture* Modularity*
Small business computers	Supplier stability* Resilience*	Price Power* Flexibility Expandability* Adaptability Architecture* Modularity
Special-purpose communications products	Adaptability* Supplier stability* Architecture Resilience	Price* Power* Flexibility* Expandability* Modularity*
Minicomputers	Price* Power* Flexibility Expandability Adaptability Architecture Modularity	Supplier stability* Resilience*

*Debatable point

Figure 1.9
Advantages and disadvantages of four main classes of equipment used in communications systems

—general-purpose mainframes such as the IBM 4341 or ICL 2960;

—small business computers such as the IBM System/34 or ICL 2903;

—special-purpose communications products like the IBM 8100, Univac UTS-700, or Burroughs B776; and

—minicomputers such as the DEC PDP-11/34, Prime 100, or DG NOVA.

Figure 1.9 summarizes the major benefits and disadvantages of each approach in general terms. Although this represents the authors' subjective assessment, it is based upon practical experience with all four categories. For specific systems it may be necessary to give certain criteria much heavier weighting (for instance, need for a single supplier), which may in turn downgrade the importance of some deficiencies.

The general criteria used should be interpreted as follows:

—*Price:* The price the customer has to pay for a typical configuration of the right scale for the job.

—*Power:* The processor power (not just speed), storage capacity, and throughput capability (not the same as processor power) provided for the price.

—*Flexibility:* How rigidly structured the system is in both hardware and software terms. Can alternative supplier terminals be attached easily?

—*Expandability:* How easily the capacity of the system can be increased and how small the increments are.

—*Adaptability:* How easily the system can be adapted to meet precise application needs. Conversely, does the application need to be changed to fit the system component?

—*Supplier-stability:* Whether the supplier (manufacturer or agent) is going to be around to support the system through the whole of the project life.

—*Architecture:* Whether the architecture (particularly the I/O and interrupt structure) is relevant to a data communications environment. How about the software?

—*Resilience:* How proven and stable are the hardware and software being used?

—*Modularity:* How modular is the system? (The modularity of a system contributes to its failsafety as well as its flexibility, expandability, and adaptability.)

General-purpose mainframes are included in figure 1.9 as a reference point (usually only in the largest networks are they considered possible concentrators and so on). Most small business computers (SBCs) are made by the same companies that make the medium to large mainframes. Perhaps for this reason SBCs tend to be structured like scaled-down mainframes. This is not true of minis, which although ostensibly smaller than mainframes often have greater throughput, particularly in communications-related jobs. In other words, the architectural differences can be much more important than those of processor power. Today, "mini" is used to describe the type of computer, not its power or size. Most special-purpose communications products were developed by the mainframe companies in response to the threat of minis in the teleprocessing field. Although these products boast architectures more relevant to communications tasks, they tend to lack both flexibility and adaptability.

Problem Areas

Our aim is not necessarily to promulgate the virtues of minicomputers in data communications. On the contrary, there seems to be something of a bandwagon effect stimulated by the all-too-clear advantages that minis sometimes have over alternative solutions. This enthusiasm ignores many of the pitfalls that can threaten the success of a project, and it is on these problem areas that much of our text will concentrate. This will be done in a positive way; not only will the danger areas be identified, but possible alternative solutions (where they exist) will be expounded.

Problem areas may be summarized as follows:

—processor overheads associated with communications I/O
—peripheral switching and sharing
—support for nonstandard terminal types (e.g., polled terminals)
—suitability of instruction sets for commercial applications
—disc storage access times
—file access methods and shared file support
—hardware and software failsafety features
—operating system structures
—interrupt structures
—multitasking
—storage management
—programming languages
—macrocode or microcode?
—addressing problems and the
—use of specialized packages

Although our review of the way minicomputers are being used in data communication systems has been comprehensive, it is clear that many new roles will be found for minis as the capabilities of these compact but flexible processors increasingly become understood by systems designers.

References

Davis, D. W. and Barber, D. L. A. *Communications Networks for Computers*. London: John Wiley, 1973.

Healey, M. *Minicomputers & Microprocessors*. London: Hodder & Stoughton, 1976.

Hebditch., D. L. *Data Communications: An Introductory Guide*. London: Paul Elek, 1975.

Martin, J. T. *Telecommunications and the Computer,* 2nd ed. Englewood Cliffs, N.J.: Prentice-Hall, 1976.

United Kingdom Post Office. *Handbook of Data Communications*. Manchester: NCC Publications, 1975.

2

The Minicomputer: Technology and Architecture

What Is a Minicomputer?

Before 1976 a minicomputer was simply a small computer made by a specialized company (see figure 2.1). These machines were marketed to "other equipment manufacturers" (OEMs) for inclusion in larger products, such as the controllers in petrochemical plants. Essentially they were merely sophisticated components. Technological advances were largely limited to hardware, with software lagging far behind. The cost-effectiveness of the mini also appealed to universities and other research institutes, which helped to widen the range of available software and—more importantly—served to spread the word. Commercial data processing users were still relatively unimpressed: the software was barely adequate for specialist researchers and engineers, let alone the "end-user" oriented DP men who required an easy to use, easy to understand system that would not distract them from the real problems of processing business data. The mini was, however, a most useful component in a data communications system and as such was first brought to the attention of the DP fraternity.

By 1976 the scene had changed. Healthy sales in the OEM area had sponsored continuing development of software systems and rapidly advancing hardware technology was being incorporated into the product. The OEM discount system developed for engineering and instrumentation users was available to companies building small business computer systems and to software houses. With a mainframe computer the end user is serviced by the manufacturer, with the software house doing add-on work. With a mini, the software house can expand its image into a turnkey systems house, dealing directly with the end user, selecting a specific mini on behalf of its client, and getting a discount on the hardware. Suddenly minis were a success in commercial data processing.

While the marketing aspects show a clear differentiation between minis and mainframes, there are also technological differences. An overriding feature is

1957	Digital Equipment Corporation (DEC) manufactures circuit boards/ modules for testing and laboratory automation.
1959	DEC PDP-1: first programmed processor-$120,000. For graphics displays, message switching, instrument control, process control.
1960-1962	Special 'aerospace' small computers (Burroughs D210, Hughes HCH 20L, Univac Add-1000)
1965	DEC PDP-8, first true mini-$20,000.
1966	PDP-8S; first 4K word computer under $10,000. Interdata and Varian enter market.
1968-1969	LSI becomes available. 16-bit minis introduced (PDP-11). Hewlett Packard, General Automation, Computer Automation, Data General enter market.
1970	40-50 vendors active.
1971	Memory management extends physical address space.
1972	Serious use of minis in data processing; full range of peripherals supported.
1973	Intel 8080 microprocessor introduced. 32-bit Interdata and SEL machines.
1975	Altair introduces S100 bus microcomputers.
1976	Prime virtual operating system introduced. Honeywell back with Level 6.
1977	IBM Series 1. Tandem multi-processor system.
1978	DEC VAX 11/780
1979	16-bit microprocessors available (8086)

Figure 2.1
Evolution of minicomputers

versatility. The mini is designed to be used in a wide variety of applications—not only commercial data processing—with OEM customers adding their own external equipment. Further, bearing in mind its historical development as a process control computer, the mini is designed for "real-time" applications. In process control external events must be serviced on a priority basis, as they occur, not when the computer can schedule service. Other events must be time initiated, for example, read an analog-to-digital converter every 100 milliseconds. Thus the concept of an interrupt, used in mainframe computers only for major events such as errors or completion of block data transfers, has been refined to accept interrupts from such trivial events as single keystrokes on a terminal keyboard. Interrupt servicing overheads have been brought down to as low as 10 to 30 microseconds. This feature adds a whole new dimension to the techniques of terminal-based data processing systems which we will examine in detail.

Five years ago a mainframe computer produced a relatively high level of performance by using sophisticated architectural techniques. Nowadays an equivalent performance can be achieved using a comparatively simple (and cheap) minicomputer architecture, merely by employing the faster electronic components readily available, aided by microprocessor controlled input/output channels. Thus the "small" constraint of the mini has largely disappeared with machines like the PDP-11/70 and MODCOMP IV, matching small mainframes like the 370/125 and yet retaining their superior terminal handling characteristics and cost effectiveness.

The leading mini maker is Digital Equipment Corporation (DEC), which in 1977 became the second largest computer manufacturer (after IBM) in terms of annual revenue, topping Honeywell and the rest. The other major mini specialists are Data General and Hewlett-Packard, with Texas Instruments in a very powerful position as the leading component manufacturer. Figure 2.2(a) is a list of the leading companies in the United States and Europe. Note that with its Series 1 IBM joined the list in 1977 and that Honeywell, long established in the mini field, has renewed flagging interest with its Level 6. Univac has bought in via Varian Data Machines. The other traditional DP computer suppliers, Burroughs and NCR, have simply called their latest small business computers (the B80/90 and 8200) minis, but in no way do these machines possess the general-purpose, versatile I/O and software necessary for the range of interactive applications considered here. They are nevertheless good terminal-oriented small business computers in competition with many other suppliers, some of which are listed in figure 2.2(b). Note that some dedicated SBCs incorporate OEM minis (the older Nixdorf 8870 with a Data General NOVA). The mini manufacturers also package their own machines into small business systems such as Computer Automation Inc.'s SyFA and DEC's COS 300 and 500 range. Some new mini products are aimed directly at the growing commercial systems market. These include the Data General ECLIPSE and Hewlett-Packard 3000.

The bottom end of the business computer systems market has been serviced by glorified accounting machines with data stored on magnetic stripes on a ledger card. These visible record computers (VRCs) are now being superseded by terminal-based systems using floppy discs as random-access bulk data stores. Minis can be used in such systems, but here the microprocessor is the dominant device. The microprocessor is simply a small CPU fabricated on a single silicon chip, which is used to build a dedicated processor. Micros are already being used in intelligent terminals and as I/O channel controllers on bigger computer systems. As micros become more powerful, they will form the heart of the next generation of "minis"; our comments are thus equally relevant to today's minis and tomorrow's micros. Possibly new names will be added to the lists in figures 2.1 and 2.2—such as Zilog and Intel—and many of those listed will be absorbed or amalgamated as the vibrations caused by the drastic cost reductions in computer hardware impinge on the fortunes of computer systems suppliers. Large-scale integration and microprocessors are discussed later in this chapter.

(a)

American

Computer Automation (CAI)
Data General (and DCC)
Digital Equipment (DEC)
General Automation
Hewlett-Packard
Honeywell
IBM
Microdata
Modular Computers (MODCOMP)
Perkin-Elmer Data Systems (Interdata)
Prime
Systems Engineering
Texas Instruments
Univac (Varian)

European

Digico
Ferranti
GEC (UK)
Nord
Philips
SEMS (Telemechanique/CII)
Siemens

(b)

Alpha Micro
Burroughs
Computer Technology (CTL)
Datapoint (Ventek)
Diablo
Harris (Sanders)
Hewlett-Packard
Honeywell
IBM
ICL (Singer)
Inforex
Jacquard
Kienzle
Lockheed
Mohawak Data Services (MDS)
NCR
Nixdorf
Olivetti
Ontel
Philips
Raytheon
Reality (CMC)
Triumph-Adler
Univac
Wang

Figure 2.2
Leading manufacturers. *(a)* Minicomputer manufacturers *(b)* Small commercial data processing systems

To illustrate the size of the minicomputer market it should be noted that by 1978 Digital Equipment had supplied over 40,000 each of its PDP-8 and PDP-11 models. These machines are sold rather than rented, which means that with strong competition, each supplier must continually strive to improve his product, absorbing any new technology as soon as possible. Contrast this with the constraints imposed on the traditional computer suppliers by the existence of rented systems. If IBM makes significant improvements to the System/34, it will obsolete the System/3 too soon; rented System/3s would then be returned, since the new IBM product is competing with the old IBM product as well as those of other manufacturers. Inside IBM this situation is complicated by the Series 1 minicomputer. It is now possible to buy an IBM 3780 from IBM's Data Processing Division, or a Series 1 with 3780 emulator from its General Products Division!

This chapter is a review of typical minicomputer hardware.

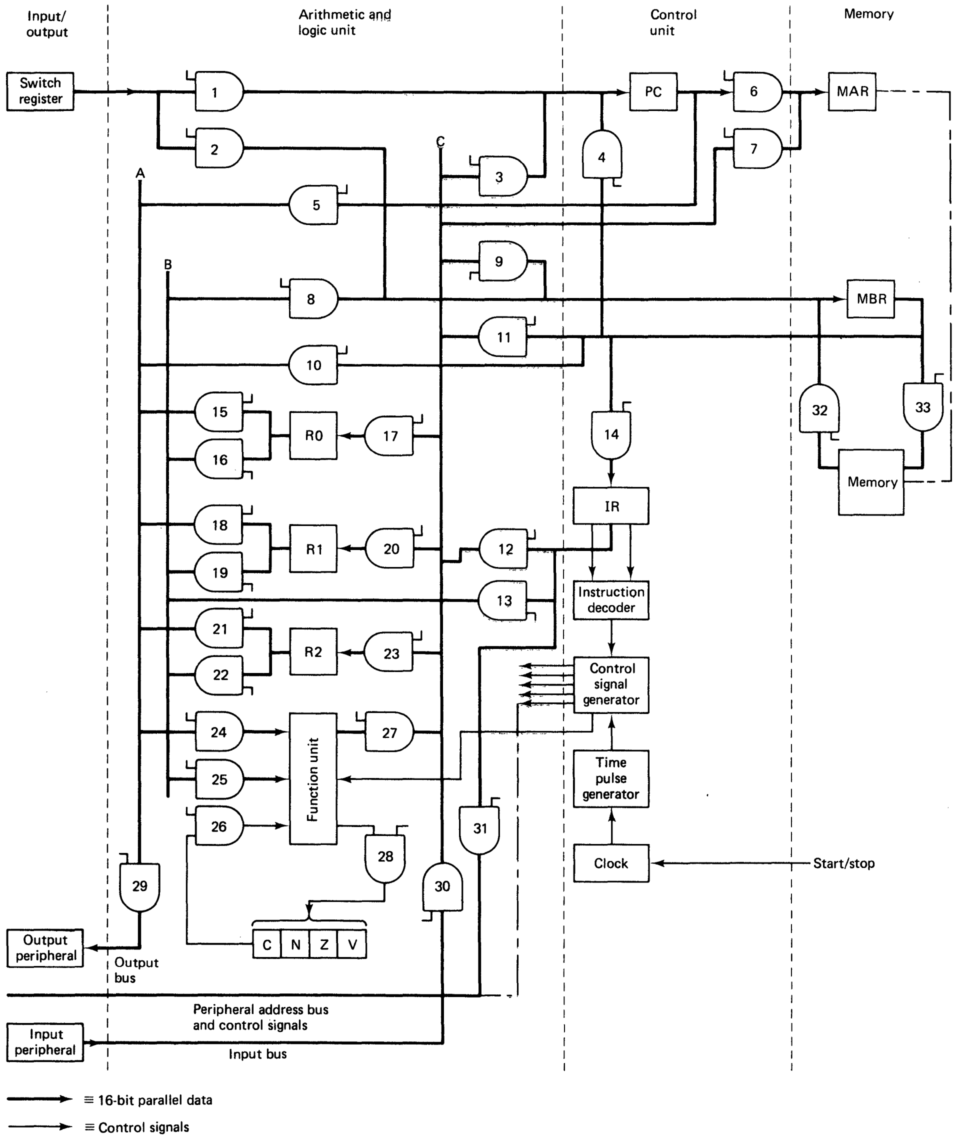
Architecture

The Processor

The processor of a "typical" minicomputer is shown in schematic form in figure 2.3. All operations and data transfers occur in bit-parallel sequence. Figure 2.4 is the same machine rearranged to stress the internal bus structure. The standard word length is 16 bits, although 8, 12, 18, 24, or 32 may be encountered. Typical memory cycle times lie between 500 nanoseconds and 2 microseconds, although it must be stressed that this is only one factor affecting the overall speed of a machine. The number of program-usable registers varies from 1 to 16, excluding the essential program counter, instruction, memory address, and memory buffer registers. Seldom are all registers generally available to the programmer because of constraints placed on the instruction set by the limited word length.

In the simplest machine the single register functions as an accumulator, all other data being retained in memory. Most machines have at least a second register which is used as an index or base in memory addressing. Further registers are used as multiple indexes and extensions of active word length, for instance, two registers to store the 32-bit product of two 16-bit numbers. If registers primarily used as index registers are generally accessible they may be used as scratch-pad registers. The control and status flags (carry, overflow, zero, negative, and so on) are commonly grouped into one effective register which may be stored and restored in multitasking applications. The instruction length is often effectively increased by special flags such as a single- or double-length data indicator.

Byte operations use the same registers, with specific treatment of the other half word dependent upon the operation (sign extended, zeroed, and so forth). All arithmetic and logic operations are performed by the arithmetic and logic unit (ALU), which incorporates shifting operations. The ALU may also be used during an instruction execution to calculate effective memory addresses. The condition flags are set as a result of the last operation performed by the ALU (except address calculations) so that conditional branching is executed by testing the appropriate combination of flags. Thus a compare instruction is a subtract in which the result is ignored, but the condition flags are set.



PC:	Program counter register
IR:	Instruction register
MAR:	Memory address register
MBR:	Memory buffer register
R0, R1, R2:	General register (typically 4 to 16 on current minis)

Figure 2.3
A typical minicomputer central processor

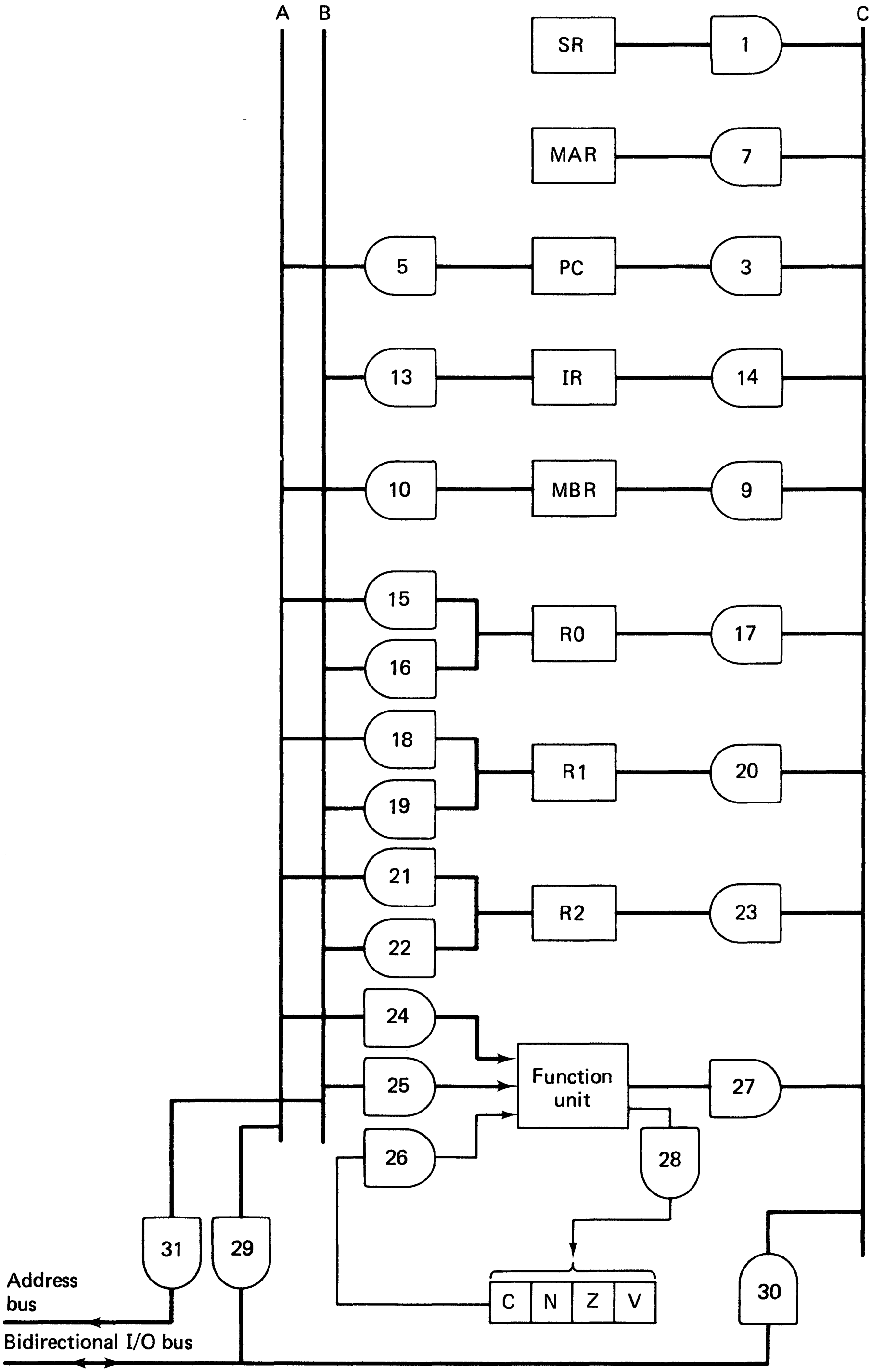


Figure 2.4
Minicomputer CPU rearranged to stress internal bus structure

Bus Structures

Data is communicated between units of the computer along a bus or data highway. The usual arrangement, shown in figure 2.5, employs a number of buses for specific interconnections. Two buses are normally required to connect CPU with memory and CPU with peripherals. Each bus must comprise sufficient wires to transmit data—addresses, control, and test information—in both directions. Often the logic is organized to use the same wires for sending and receiving data (bidirectional), taking advantage of the fact that both will never be required simultaneously. A typical I/O bus will use up to 64 wires.

A technique pioneered in the PDP-11 is the single bus shown in figure 2.6. In this case the peripherals are treated as specific "memory" locations (virtual I/O), avoiding specialized I/O operations. Data can be passed direct from one peripheral to another as simply as from one memory location to another. Direct memory access is similarly performed. The apparent simplicity of this technique is offset by the extra logic and time required to give appropriate allocation of the bus. The single data highway is also an obvious bottleneck, setting a top limit on

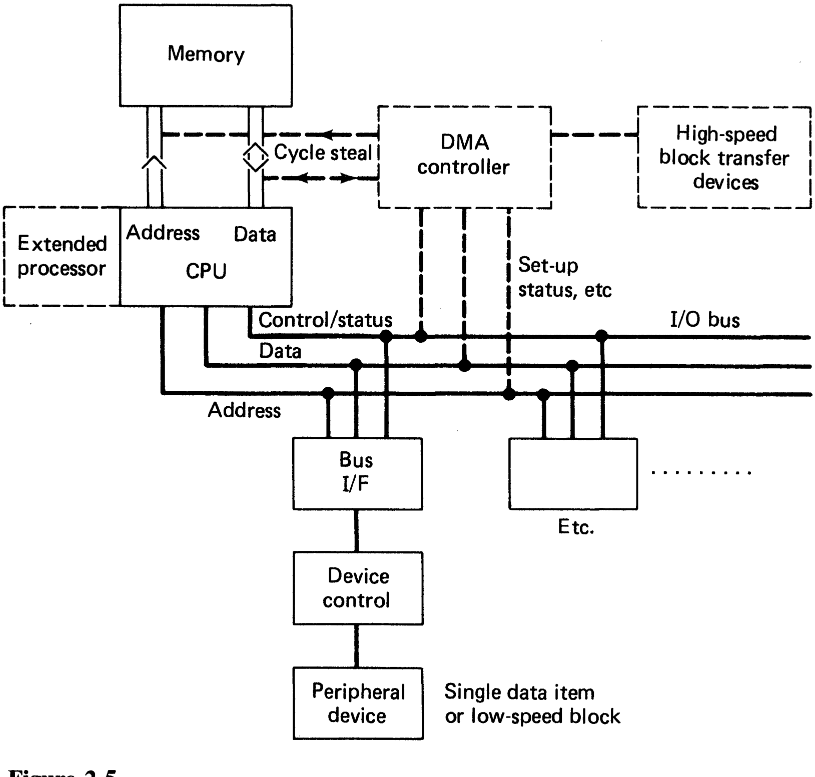


Figure 2.5
A typical minicomputer with separate memory and I/O buses

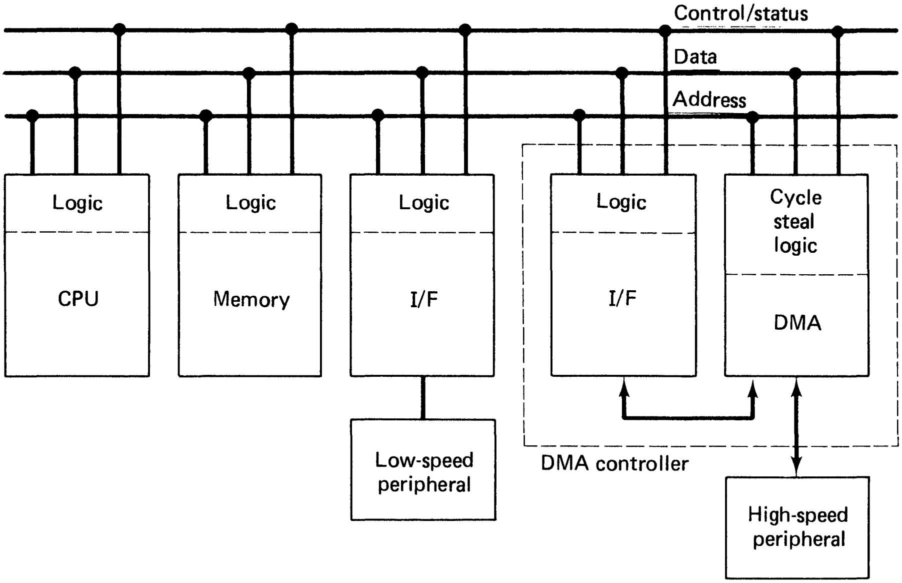


Figure 2.6
Single bus system

performance. Many microcomputer systems use a simple version of a common memory and I/O bus, called the S100 (Altair, North Star Horizon, Cromemco).

Some machines, such as the GEC 4000, feature as options a more complex architecture involving a number of multiplexors to route data between CPU, memory, and peripherals. The bottleneck in the single bus system is reduced by allowing more than one multiplexor and employing multiport memory (figure 2.7). This it should be noted is a multibus mini when only the basic multiplexor is used, but resembles a mainframe when I/O processors (channels) are configured.

Inside the CPU, instructions may be of differing lengths, for example, for the PDP-11, move register to register is 16 bits, move memory contents to register 32 bits, and move memory contents to memory 48 bits; the latter two instructions incorporate 16-bit addresses. The memory however is addressed by the 16-bit word, with a 16-bit data highway, so that the effective 48-bit instruction requires three memory accesses (to consecutive locations). Some machines feature 8-bit highways, with most instructions requiring multiple memory accesses (8080/Z80/6800/6502 microprocessor based machines), while some of the more recent machines feature 32-bit highways (Interdata 8/32). Thus a machine may feature a double-precision floating point (64-bit) multiply instruction, which will multiply the number stored in four consecutive memory locations by a number stored in a 64-bit internal register (usually four standard internal registers treated as one for this instruction). The 16-bit machine requires four-memory accesses for the one data word.

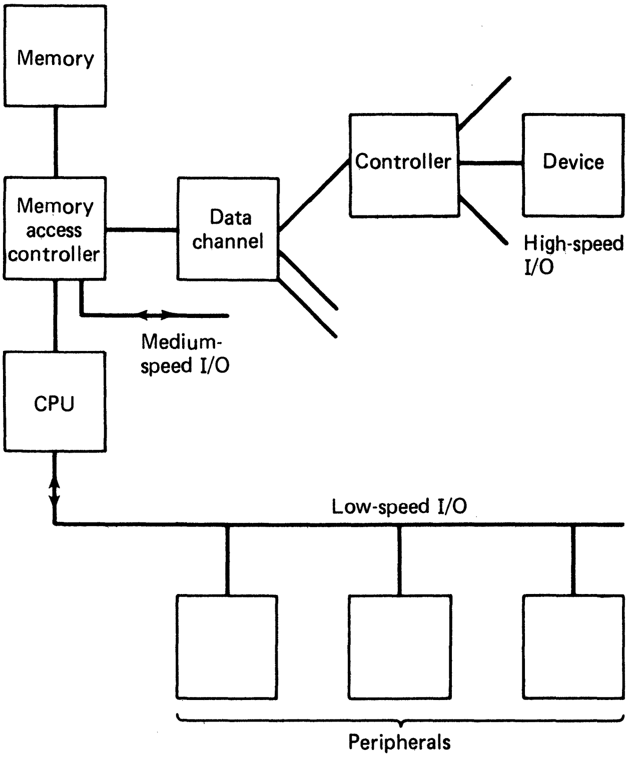


Figure 2.7
Advanced system with active I/O

In most minicomputers the memory is organized in 16-bit words numbered evenly. Odd-numbered addresses then refer to the 8-bit byte, as shown in figure 2.8. Without this feature byte manipulation can only be performed by fetching the full word and masking and/or swapping half words inside the processor. Thus most minis have a virtual address space of 64 KB, but a few with word addressing have a 128 KB (64 KW) space.

	15 8	7 0
Word 0	Byte 1	Byte 0
Word 1	Byte 3	Byte 2
Word 2	Byte 5	Byte 4

Figure 2.8
Word/byte addressing

Memory

Magnetic core and semiconductor (MOS) memory were both common in minicomputers, although MOS now dominates. Most processors take advantage of the read/write nature of core to execute some instructions, as when incrementing is performed in a read/modify/write mode. Thus mixing core and semiconductor memories raises problems. The solution is to use a different bus interface for the semiconductor memory with additional logic to emulate the read/modify/write mode for the appropriate instructions. Most machines use asynchronous transfers between CPU and memory so that alternative speed memories are offered for the same basic processor.

Memory was traditionally supplied in units of 4K 16-bit words, but 32 and 64KB blocks are now common. Read-only memory (ROM) is available on some machines but is used only in rare cases. The common exceptions are bootstrap loaders and microprograms discussed later. Multiport memory (figure 2.7), sometimes available as an option, allows simultaneous block transfer through one port and normal program access through another. Otherwise cycle stealing is employed, as is discussed below.

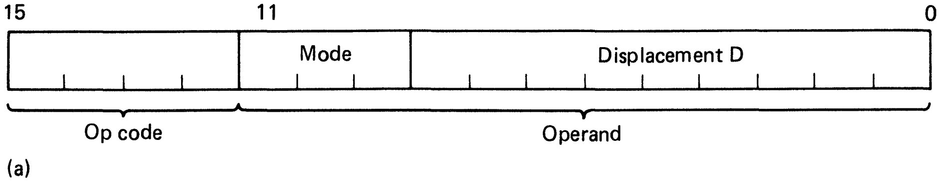
Instruction Sets

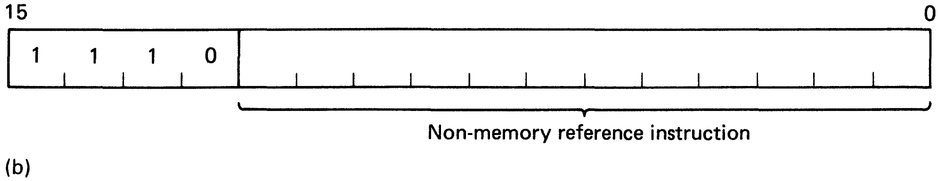
In most machines the instruction set can be broken into three categories: memory reference, nonmemory reference, and input/output. A simple instruction arrangement, using a 4-bit operation code (or order), is shown in figure 2.9. This setup allows 14 memory reference instructions. The logical allocation of the bit significance is seldom so straightforward in practice.

Memory Reference Instructions

Most machines use single-operand instructions, as in figure 2.9 (a), the second operand (when needed) being a CPU register. Often only the accumulator may be used, but if a choice of register is allowed the operation code must specify which. The combination of the mode bits and the "displacement" D are used to locate the specified memory cell, as discussed below. The location following the instruction is occasionally used to store a 16-bit literal or an indirect address, effectively extending D.

The PDP-11 is a notable exception to the above rule, since it allows some double-operand instructions such as move contents of one memory location to another, as shown in figure 2.10. Three bits are used to specify one of 8 modes, and another 3 bits one of 8 registers, one of which is the program counter (PC). For each operand the selected register is either the actual operand location or a pointer to a memory location, depending on the mode. Offsets and literals are stored in locations following the instruction. Thus MOV R1, R2 occupies one word, MOV R1, A two, and MOV A, B three words. This multiword technique





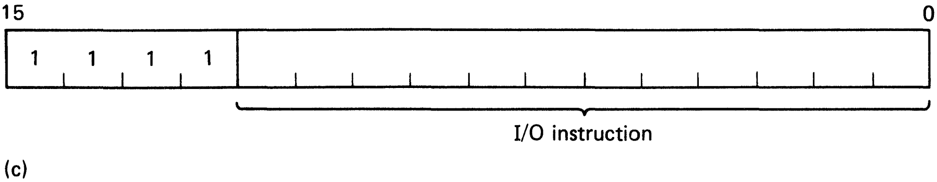
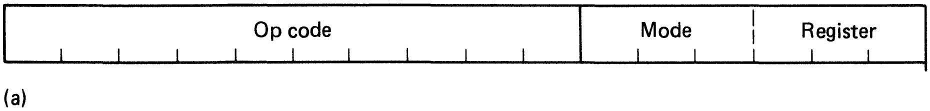


Figure 2.9
Simplified single-word instruction set organization

has now been adopted by Texas Instruments and IBM. Interdata and others use multiwords to achieve an effective 32-bit machine. Single-word-instruction machines are simple and ideally suited to earlier minis, but are limiting on newer machines. Thus, for instance, Data General have employed the NOVA instruction



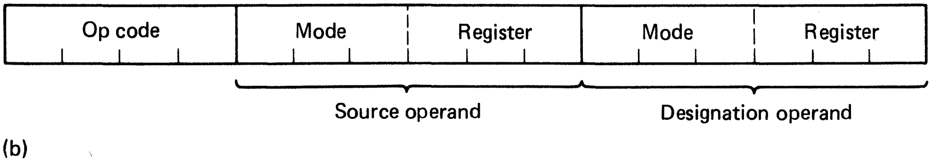


Figure 2.10
Single- and double-operand instructions in a multiword instruction set

set on the ECLIPSE but have used one specific bit pattern to cause the processor to read the next word and treat it as an extended instruction set.

Byte or word data is handled directly. Some machines feature instructions that test or modify specific bits of a word, although this is usually handled with a program employing masking operations. Character-string manipulation instructions are also encountered, for instance, scan a string starting at a pointer set in one register, through a number of characters set in another register, but stop when the data matches the content of a third register. Such instructions have obvious advantages over a programmed loop since only one instruction fetch cycle is required; there are some problems with interrupts, however, since such an instruction may take numerous memory cycles to complete. Other instructions such as byte-string-move and translate do occur but all too infrequently. The IBM Series 1 and Data General ECLIPSE are new machines that support character-string instructions and this will hasten market-leading Digital Equipment Corporation into producing new versions of the PDP-11 to compete, VAX-11/780 being the first of these machines. In general mini instruction sets are good at binary arithmetic and logic but weak on character handling. Program control operations such as jump and jump-to-subroutine also come under the memory reference category. With a JSR, current PC contents are automatically saved for return; dedicated register or memory locations may be used, or the first word of the subroutine, but stacking offers advantages discussed below.

Nonmemory Reference Instructions

This group of instructions contains operations on data stored in the internal registers. Arithmetical and logical operations are grouped with branch and shift instructions. A wide variety of conditional branch instructions are encountered; the branch point may be indicated by a PC-relative displacement in the instruction, but many machines use the skip-type operation. With a skip instruction the PC is incremented by one (two with byte-addressed memory) if the logical condition is met. Otherwise the next instruction in sequence is executed, normally an unconditional Jump. Increment (or decrement)-and-skip-if-zero instructions are used for loop counting. This may also be a memory reference instruction, with the counter stored in memory rather than in a register. Signed integer multiply and divide are always available but often as part of an optional extended instruction set.

Both arithmetic and rotational shift operations are employed, with and without the carry bit. In many machines only a one-place shift can be executed by one instruction; other machines allow shifting by a specified number of places.

With single-word instruction machines it is usually possible to create some combinations of shift, skip, set, clear, increment, and other operations into one instruction. This assumes that no conflict can arise and the order of execution must be clearly determined to ensure the correct operation of the combined instruction. This technique was referred to as microprogramming in an earlier machine, a word that now has broader connotations, as we will discuss.

Input/Output Instructions

All input/output instructions must identify the peripheral device by number. Four types of instructions are needed:

1. Read data (peripheral-to-CPU)
2. Write data (CPU-to-peripheral)
3. Test
4. Control

The test and control instructions are required to activate and deactivate devices and to check the current status of a peripheral. The data transfer instruction must also reference a data location in the processor. In the simpler machines a specific register is used, a further register-memory transfer being required to save the data. Alternatively a register can be used as an index to an array in memory.

Single-bus systems do not use special instructions, since peripheral buffers and status/control registers are treated as memory locations and conventional memory reference instructions are used for input/output. This is called virtual I/O.

Numeric Data

Numeric data can be used in the computer in a number of forms. The basic instructions assume 16-bit integer numbers in 2's complement form. Software routines for double-length (32-bit) integer arithmetic are common, one word forming the most significant half and a second the least significant.

Floating-point representation uses 32 bits, typically a sign, 24-bit unsigned, normalized mantissa (approximately 7 decimal places), and an 8-bit excess 200 exponent. A binary exponent is common ($x = \pm m.2^e$), so that the normalized m always has a 1 in the left-hand bit; hexadecimal ($x = \pm m.\ 16^e$) is also used when up to three leading zeros are allowed in m. Double-precision floating point (64-bit) increases the mantissa to 56 bits, equivalent to about 15 decimal places. Software routines are normally used, but optional floating-point processors are available.

Decimal (BCD) arithmetic is rare in minicomputers although software packages are available for machines used in commercial applications. It is quite common for high-level languages, COBOL for example, to employ ASCII character strings. Most direct arithmetic operations can be used with byte or word data, a flag being set to indicate the specific type. On some machines, bits of the code are used for this purpose while other machines use a separate flag. The latter effectively extends the instruction length and allows more scope, although the flag must be set by a previous instruction. On such machines the same instruction performs differently, depending on the data word-length flag setting. Software routines are always provided to convert ASCII character strings to internal binary representation and vice versa.

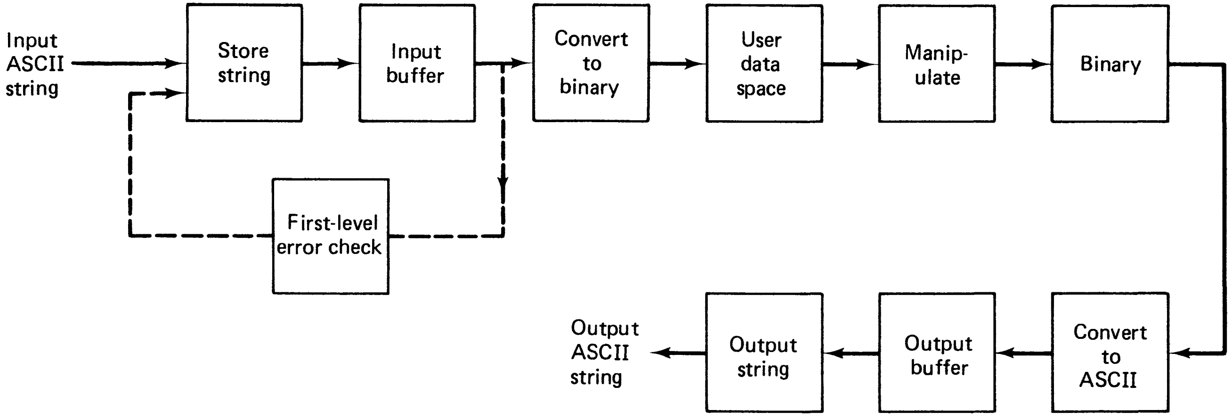


Figure 2.11
Human/machine interface to numeric data

Figure 2.11 indicates that all data processing requires human input, for example character strings, conversion to binary, manipulation, and conversion back to character strings. The simpler the internal representation, the simpler the conversion routine. ASCII strings, for instance, only pack the ± sign into the least significant character. The simple internal representation is, however, more wasteful of space and takes longer to manipulate. Commercial DP is mostly input and output, while scientific work is nearly all processing. Hence the preference for ASCII (or better, BCD) for the former and floating point for the latter. Double-precision integer with implied decimal point is also suited to commercial work. Simply because of availability, double-precision floating point (8-byte) is commonly used in commercial work on minis; single precision does not have enough significance.

Memory Addressing

As has been explained, most machines, even though they employ 16-bit data highways, are capable of addressing individual bytes in memory. With a 16-bit word this sets a limit of 64K bytes. Any bit operations must be executed by fetching the appropriate byte and and decoding this under program control in the CPU. The addressing modes encountered are here discussed with reference to figure 2.9(a). One bit of the mode is used as a direct/indirect flag, others for various modes. Not all the following modes will be found in one machine, but they cover the range. They are summarized in figure 2.12.

Direct/Indirect

For direct addressing the information is stored at the indicated location; for indirect addressing the indicated location contains the address of the location of the required information. An extra memory access is involved for indirect addressing, which is typically used for array processing, the content of the addressed location being incremented each time.

Absolute or Page Zero

The displacement, D, is treated as the actual address. Since D is typically about 8 bits long, only 256 locations can be addressed. On most machines this is the first 256 bytes, but on others it is the first 256 data items, particularly in machines with a separate data type flag; thus for byte data D refers to the first 256 bytes, while for double-precision floating point data the first 256 × 8 bytes are referenced.

Immediate

The actual data is in the location following the instruction.

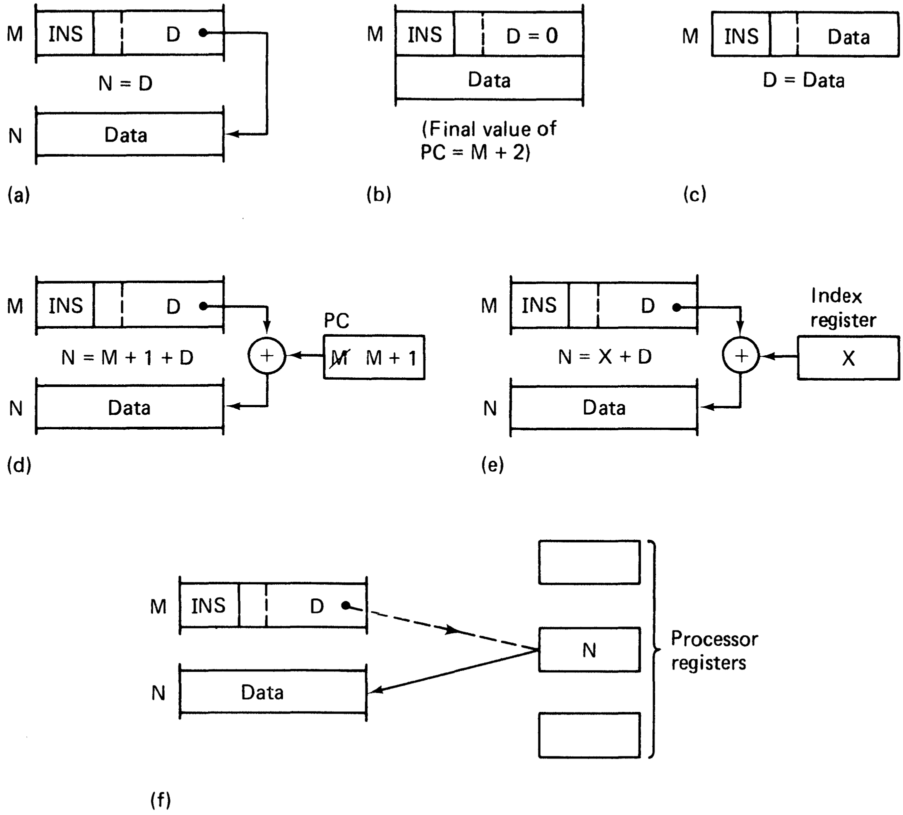


Figure 2.12
Some common addressing modes

Literal

D is the actual data. Note that the use of "immediate" and "literal" terminology on minis conflicts with such usage on some mainframe computers.

Program Counter Relative

D is treated as an offset, and is added to the PC contents to give the memory address. Sometimes D is treated as a positive integer and at other times as a signed integer, allowing back-referencing. This mode is the most common since it creates relocatable code.

Indexed or Base Relative

D is added to the contents of a specified register to give the memory address. This is used for working with arrays. If D is kept constant and the register contents incremented, it is termed indexed mode, while if D is varied and the register contents kept constant it is termed base relative mode. There is no difference in general except where indirect access is used, in which case the register points to a memory location, the contents of which are added to D to give the required address, and the register is clearly being used as a base pointer and D as an index. One General Automation machine allows both a base and an index register, so that D is added to the base register contents to find the base pointer to which the contents of the index register are added to give the desired address, a two-dimensional array technique.

Auto-Increment, Auto-Decrement

With these modes the index register contents are incremented or decremented after each operation. They must be a true pair in the sense that they must increment *after* the address has been used (postincrement) and decrement *before* the address is used (predecrement), or vice versa. These modes are used in creating stacks and for working through lists. All registers in the PDP-11 can be used for auto-increment/auto-decrement but most machines only use a specific register. In all machines certain operations (JSR) use one register by default, often called the *stack pointer.*

Stacking is probably the best way of entering a subroutine. On excuting a JSR instruction, the PC contents and any other critical register contents can be placed on the stack (pushed) by successive auto-increment mode move instructions; on return, the auto-decrement mode move instructions in reverse order reinstate (Pop) the CPU conditions. Subroutine calls can now be nested, since this creates a "last-in, first-out" stack. Stacking can also be used to create reentrant code by ensuring that all data is separated from the code and that pointers to the data are stacked with the JSR routine.

Pointer or Register Indirect

The contents of a selected register form the address required. This technique is used extensively with machines like the PDP-11 and TI 990 when D becomes the register pointer. This is combined with such techniques as indirect addressing and indexing to produce a very versatile addressing system. It is very common in microprocessors.

Input/Output

Programmed I/O

The normal I/O instructions have been mentioned. Each peripheral device requires an interface to communicate between the I/O bus and the device itself. A simple system is shown in figure 2.13. Since a peripheral is essentially slower

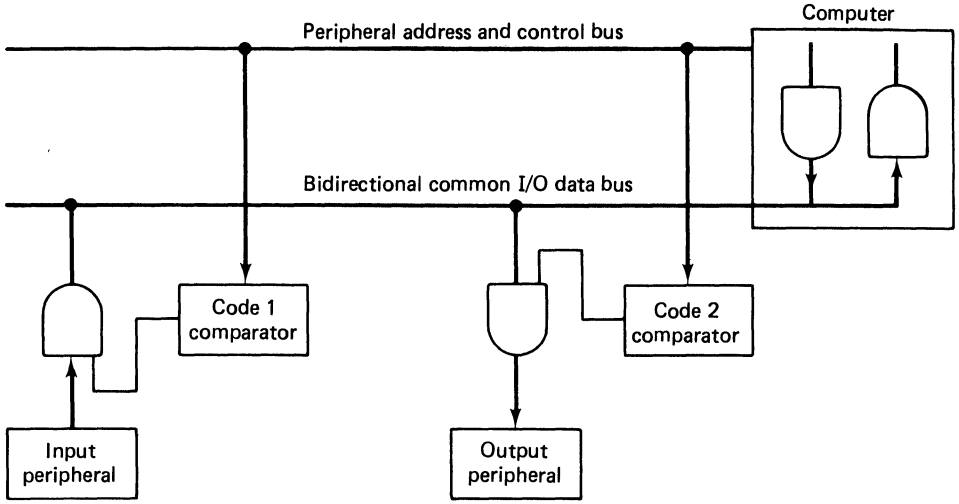


Figure 2.13
A simple party-line I/O bus system

than a processor, some means of checking the status of the device must be provided. At the simplest level a busy flag is required, set to 1 while the peripheral is active. The processor repeatedly tests this flag until it clears, when a new data transfer can be initiated.

Interrupts

The approach described above means that the processor is continually waiting for the peripheral to complete. To overcome this problem it is common to use an interrupt system. The peripheral action is initiated by the processor, which then proceeds with another task. When the peripheral has completed, its busy flag is cleared and a done flag is set; no further action can be initiated until the done flag is cleared, usually automatically by the next data transfer. The setting of the done flag causes a request for an interrupt to the processor. The processor checks the priority of the interrupt and if it is higher than the current task (and any other peripherals which have requested interrupts), the current task is shelved and the device serviced. The task of shelving a program is similar to a JSR, except that it may occur anywhere in a program. Stacking is thus an advantageous technique in handling interrupts.

The interrupt system must handle two problems. It must determine which peripheral requested the interrupt, and then establish a priority for the interrupts (for example, a Teletype must not be allowed to interrupt a high-speed analog-to-digital converter).

Two techniques dominate the servicing of interrupts.

Software Polling. When an interrupt request is accepted, a routine is entered causing each peripheral in turn to be tested until the one awaiting service is detected. The order in which the devices are polled forms a priority system. When the interrupting device has been identified, the interrupt service routine references a table which it uses to pass control to the appropriate device service routine.

Vectored interrupts. A specific memory location is physically associated with each possible interrupt requesting device as shown in figure 2.14. When an interrupt is requested, if it is of high enough priority (it may wait in a queue), the program currently executing will be interrupted. As with a JSR, the PC and active register contents must be saved for return from the interrupt service, and the service routine entered. If the start address of the routine is previously stored in the memory location dedicated to the peripheral, then this can be copied automatically into the PC. As with subroutine calls, stacking allows simple nesting of interrupts. This technique is much faster than polling. The stack is simply popped back into the PC and status register to completė the return from the service routine. Alternative vectoring techniques relate a vector location to each priority level rather than to each device. Another technique forces the vector location contents into the instruction register, allowing a single instruction execution with no overheads (for example, increment a counter; a JSR instruction effectively vectors to a normal service subroutine).

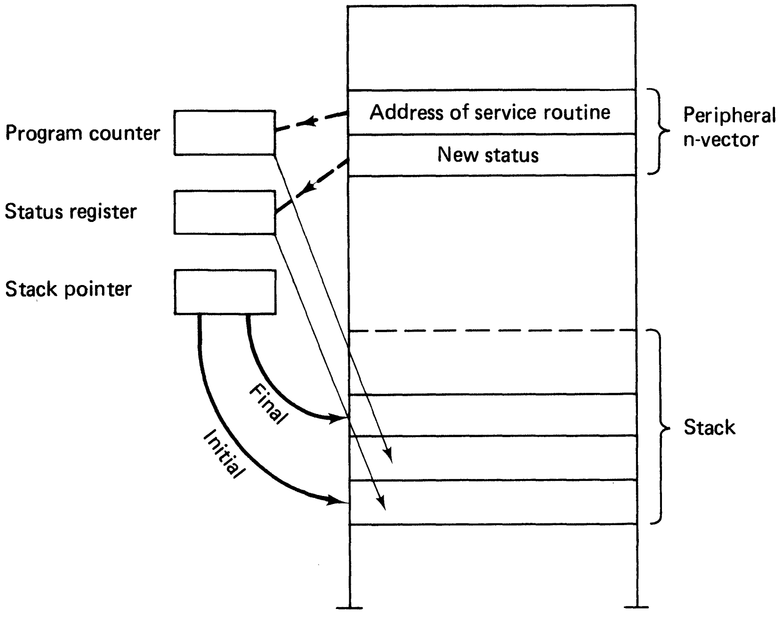


Figure 2.14
Vectored interrupt structure

Priority

Figure 2.15 shows a generalized priority system. Each peripheral has its own interrupt request mechanism, which can be blocked by an interrupt enable (one per device), which can be independently software controlled. There are a number of hardware priority levels, usually from 4 to 16; an interrupt on a higher level will preempt a lower-level service routine. The interrupt enable flags are provided to dynamically stop a normally high-level interrupt from suspending a lower-level routine. The mask serves a purpose similar to the individual enable flags, except that it controls interrupt levels rather than devices and can be set in one instruction. Multiple devices can be "daisy chained" to one level, in which case the device physically nearest the processor has the higher priority. The interrupt-on flag on the CPU turns the whole interrupt system on or off dynamically. On the more sophisticated systems the CPU has a set of priority levels, matching the hardware levels, which are software controllable. In this case the priority of an incoming request is compared with the CPU priority, rather than the level at which the interrupt being serviced was attached. Thus if a service routine has a high-priority initial part, but once having collected volatile information can proceed at a lower priority, this can be achieved by initially setting the CPU priority high and then lowering it in the service routine itself.

Internal interrupts such as power fail detection have the highest priority. Many computers support executable instructions that behave as interrupts (software interrupts). These are used for fast context switching, as when a user program executes a supervisor call for executive services, as detailed in chapter 7.

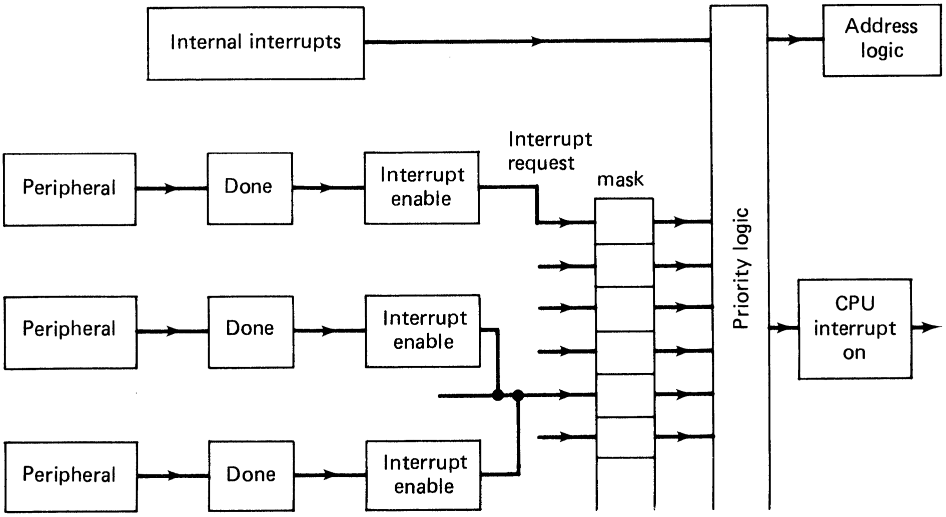


Figure 2.15
Priority interrupt structure

Autonomous data transfer

With program-controlled I/O, an instruction is required for every data transfer. Physical devices, however, regularly need to transfer a block of data, as a line of characters on a line printer. Thus many machines allow block data transfer to or from contiguous locations in memory. The start address and word count are set in registers and the transfer initiated. The address is loaded into the memory address register (MAR) and the data is transferred. The address pointer is then incremented and the word count decremented and tested for zero. The process continues until the word count reaches zero, when an interrupt is requested to signal completion of the block transfer. If the pointer and counter are stored independently from the normal registers, the block data transfer can continue while another program executes, simply pausing that program at the end of an instruction to transfer the data, the program continuing until the next piece of data is ready. Thus apart from the repetitive pauses in the program, it and the data transfer are independent of each other, hence the term autonomous data transfer. The pointer and counter may be stored in memory, in which case they must be accessed for each data transfer. Better, they may be located in special registers in the peripheral interface, in which case the address is entered with each data transfer.

Direct memory access

With high-speed data transfer it will be impractical to request an interrupt and to use programmed I/O operations. Even the necessity to await the completion of an instruction, as in block data I/O, will be intolerable. Special channels are therefore provided to permit direct access to the memory, as shown in figure 2.16. Direct memory access (DMA) can now use the memory whenever it is not required by the processor, which may be partway through the execution of an instruction.

The DMA is said to "cycle steal". If the processor requires memory access while the DMA is active it must hesitate until that transfer is complete. Note, however, that if the DMA transfers at full memory speed it will steal all the cycles, holding the processor up. Since each peripheral connected to the DMA will have its own address and counter registers, multiple devices can be connected to the same channel. Thus a priority structure may be needed and DMA controllers can become quite complex. All autonomous data transfers must be set up by normal programmed I/O instructions such as initialize address-pointer and word-counter registers, and an interrupt raised on completion. The IBM Series 1 has a sophisticated set of instructions that reference device control blocks in memory to set up and control DMA transfers; multiple transfers can be chained. With multiport memory, if the software can organize the DMA to use one port while the processor uses another, then cycle stealing is avoided. While DMA was essentially used for disc and tape block transfers we find increasing use of intelligent controllers employing DMA for supporting multiple terminals. With the 8089, Intel have introduced a single chip LSI channel controller.

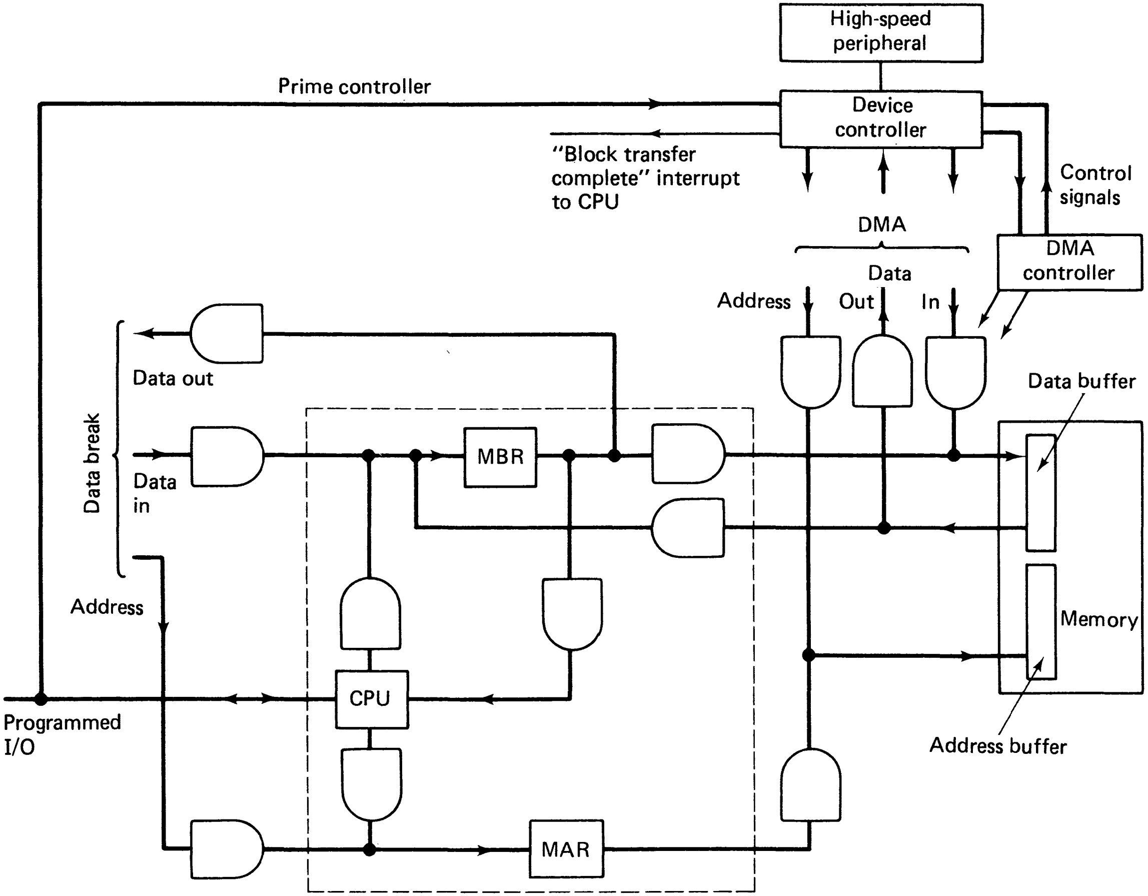


Figure 2.16
Autonomous and DMA access to memory

Memory Management and Protection

A memory management unit is a hardware device containing a set of address relocation registers. A memory address issued by an executing program is modified by the contents of one of the registers to create a new address. The first address, issued by the active program, is called a *virtual address,* the latter a *physical address.* It is the latter which is actually used. Thus the memory management unit "maps" virtual addresses to physical addresses. Note that this is not dissimilar to base-relative addressing, but it is external and transparent to the user program. The mapping registers and any associated arithmetic unit must be very fast; typically the mapping adds an overhead of 50 to 250 nanoseconds to each memory access. On most minis the virtual address space is confined to 64 KB (16 bits), but the physical address space has no such bounds. Between 18 bits (256 KB) and 24 bits (16 MB) are encountered. A typical system is shown in figure 2.17.

The immediate impact of memory mapping (figure 2.18) is that while any one program still has a 64 KB address space, multiple programs of any size up to 64 KB can coexist in memory, up to the limit of physically available memory. Each program internally has a virtual address starting at zero. The switch from execution of one program to another is achieved simply by changing the mapping

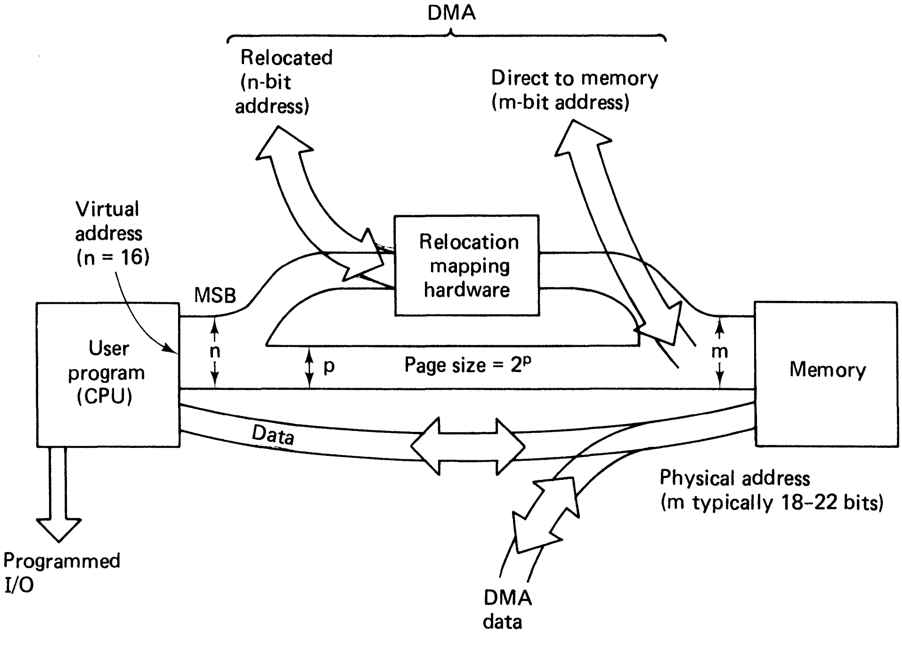


Figure 2.17
A typical minicomputer memory management system

registers. Most machines support multiple sets of mapping registers to increase the switching efficiency, but some special instructions must be available to the executive (not the user) to set up and modify these registers at run time. A technique commonly employed is to use one set of mapping registers exclusively for the executive and other sets for user programs. Certain instructions (typically software interrupts), when executed, cause an automatic context switch from the user map to the executive map and as such are used to make supervisor service calls.

It is a relatively obvious expansion of memory mapping to incorporate a check that physical addresses generated by an executing program are within the allocated space for that program. With an unmapped system memory protection traps can be instituted when a program is loaded, but there can be no protection from run-time addressing errors. With mapping hardware every memory access is checked for validity at run time, with no software overheads other than error servicing routines. A simple protection scheme is also shown in figure 2.18.

By using multiple mapping registers rather than a single unit as in figure 2.18, the one contiguous virtual address space can relocate into a number of noncontiguous physical blocks. Further different programs can now map part of their virtual address spaces to the same physical space, thus sharing code or data. Physical memory utilization is also better with the system shown in figure 2.19, since smaller contiguous blocks are required. As programs are swapped in and out, loaded programs may have to be shuffled in physical memory to regroup fragmented free memory into a contiguous block; if the basic segment is small, this will be unnecessary. On the other hand the smaller the segment the greater the

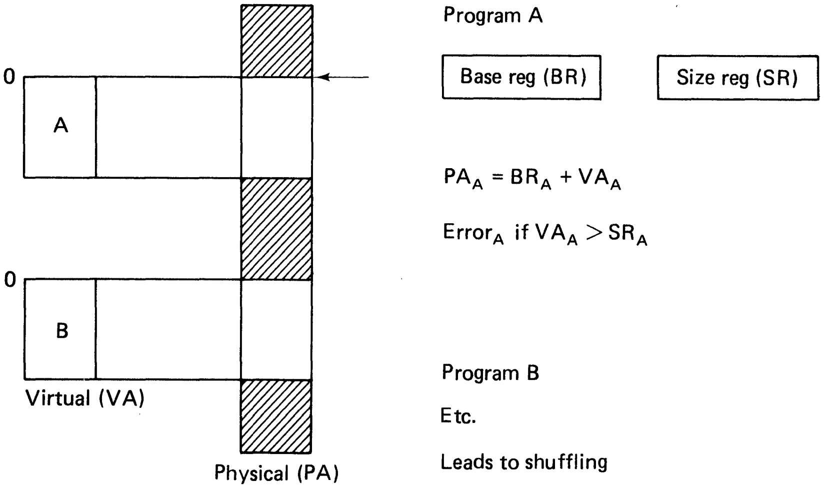


Figure 2.18
A simple mapping and protection scheme

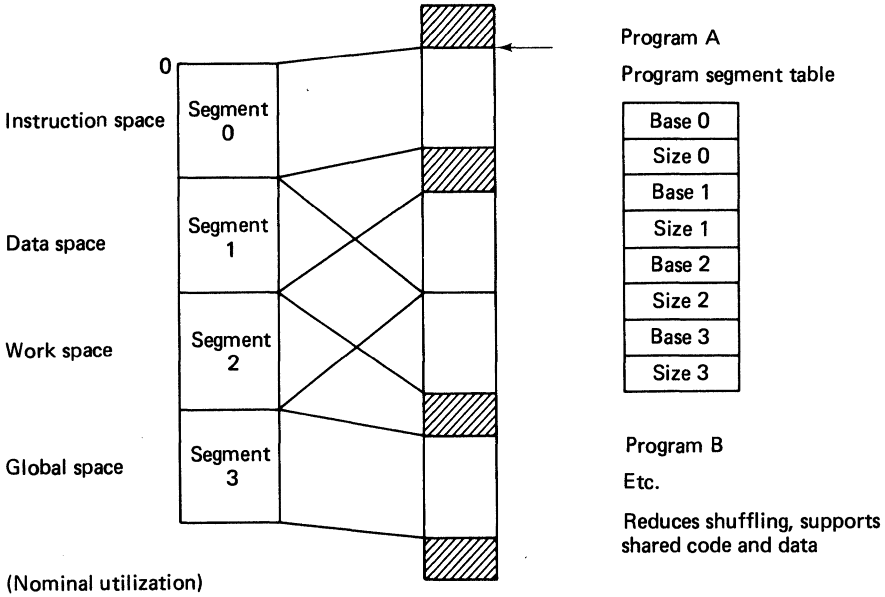


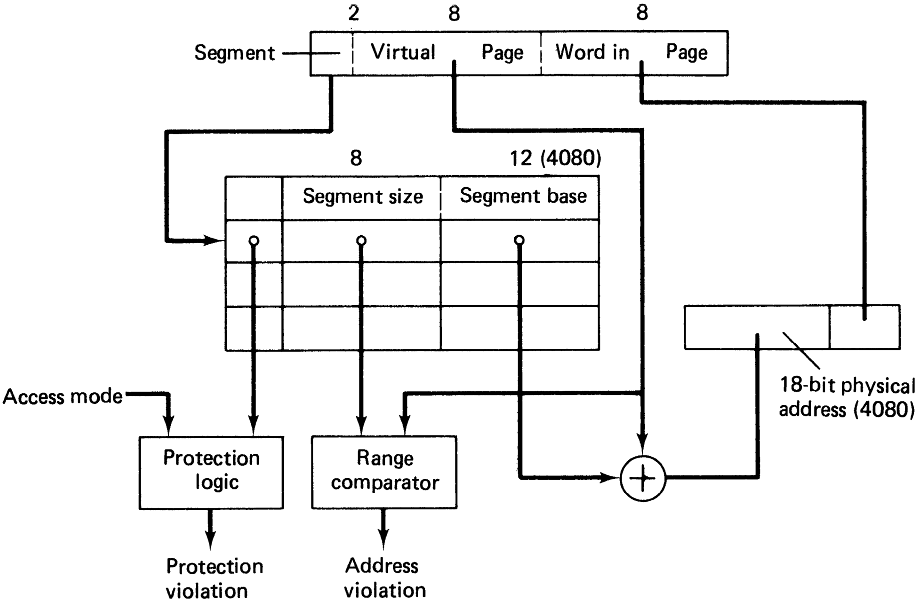
Figure 2.19
Mapping by segmentation

number of mapping registers required, and therefore the executive mapping register maintenance overhead is higher.

With few execptions (Prime 400/500, VAX-11/780) minicomputers adopt the *segmentation* approach, rather than *paging* in the classical mainframe sense. With segmentation the *virtual* address is broken into logical blocks, each of which is mapped to a specific physical block of memory. While a segment has a maximum length, the actual amount of physical memory used is variable, depending on the running program. To ease physical memory allocation it is handled in units called pages, as shown in figure 2.17, rather than individual words. Thus for instance if physical memory is treated as 256-byte pages then the 8 least significant bits of the virtual address are not translated, since they simply define the byte within the page. Thus to accommodate a segment of 8 KB with a 256-byte page, 32 contiguous physical pages are required; to accomodate 8.1 KB segment, 33 pages are needed. The smaller the page the less physical memory is wasted, but more bits are then needed in the address translation hardware.

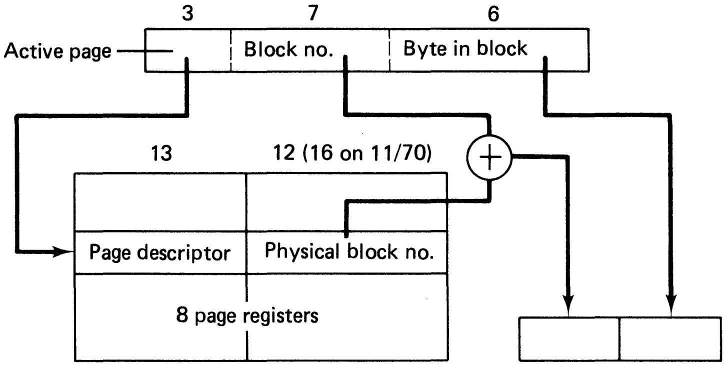
For reference, a paging system (as opposed to segmentation) retains mapping hardware pointers to each *physical* page of memory, storing the identification and virtual address of the user allocated to that page. A virtual page address is then compared to the contents of the page address registers (a content addressable memory), a match-in which identifies the required physical page. A large physical memory would need too many page address registers, so real systems maintain only a most-recently-used subset of pointers (a look-aside buffer). On modern machines one set of hardware supports segmentation of the virtual address space and another set controls the physical paging of the memory. Note however that while the 16-bit virtual address of the mini constrains the maximum individual program to 64K, this is a blessing in disguise, since the number of possible pages involved is low enough to permit a physical relocation register per page if needed.

The concept of memory mapping is critical to the efficiency and security of a multitasking system of the type we are considering. The diagrams described here indicate the techniques commonly used by minis. Figure 2.20 shows the segmented approach used by the GEC 4000 range. The two most significant bits identify the segment so that the virtual space is broken into four equal 16 KB segments. The relocation hardware contains a segment size entry to indicate how much of the available 16 KB has actually been allocated. Any upper part of the segment not used need not be allocated physical memory. Each segment can be located independently in physical memory starting on 64-byte page boundaries (only the twelve most significant bits are stored by the relocation or segment base register). The PDP-11 uses a somewhat similar technique (figure 2.21), breaking



– Segments start on 16 K virtual boundaries
– Segment size from 64–16 K
– Physical segments located on 64-byte boundaries

Figure 2.20
The GEC 4000 memory management system



Physical address
– 18 bits (22 on 11/70 model)
– Variable page size from 1 to 128
64-byte blocks on 64-byte boundaries

– 8 page registers
– 2 sets 11/32, 11/40
– 6 sets 11/45, 11/70
– (Kernel, user, supervisor) X (Data and instruction)

PAGE DESCRIPTOR

Abort
– 3 bits access control - Read only } With or without trap
Read/write
– 7 bits page field length (number of 64 bytes). Error if BN > PLF
– 1 bit "written into"
– 1 bit trap indicator
– 1 bit expansion up (program) or down (stacks)

Figure 2.21
The DEC PDP-11 memory management system

the virtual space down into eight equal-sized segments of 8 KB. Digital Equipment Corporation defines what others call a segment as an active page, and what others call a page as a block.

Note in figures 2.20 and 2.21 that the mapping hardware contains further data per segment to control access rights and so on. The "written-into" bit is set whenever a segment is modified; if not set this segment need not be rolled out to disc when swapping.

The Interdata 8/32 has a 20-bit implemented virtual address broken in sixteen equal-sized segments, each 64 KB. An intriguing variant is the Intel 8086. This single-chip microprocessor uses a full 16-bit segment size, with four possible segment spaces, one each for code, data, stack, and common. Each segment has an associated base relocation register which form the 16 most significant bits of a 20-bit address, to which the 16-bit virtual address is added. Thus any program can comprise 4 × 64 KB segments, relocated independently anywhere in a 1 MByte address space on 16-byte page boundaries.

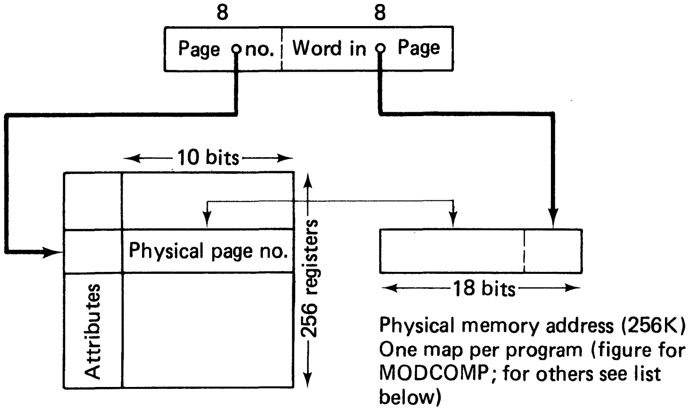
The Texas Instruments 990/10 breaks the 64 KB virtual space into three segments called zones. Unlike most systems the segments are not fixed length and comprise any number of 32 byte pages which, however, must be contiguous—there can be no "holes" not allocated physical memory at the top of each segment. It has three sets of mapping and limit registers, one reserved for executive use and one for moving data between virtual spaces. It relocates to a 21-bit (2 MB) physical space in 32 byte pages.

Both the TI 990/10 and the DEC PDP-11 use virtual I/O, which is allocated to the top of the physical address space. By mapping only the executive virtual address to this physical space, user programs can be forced to execute I/O via supervisor calls (see chapter 4).

Most of the other minis use a slightly different technique commonly called "paging," but not to be confused with virtual memory paging. One way of looking at the technique is to consider something similar to the system shown in figure 2.20 with many more segments, each one page long. To constrain the number of segments and therefore relocation registers to manageable proportions, the page is bigger than the 16 to 64 bytes previously discussed. Some typical values are shown in figure 2.22. With the page technique the larger page size means that the relocation boundaries are coarser but contiguous physical pages are never necessary and shuffling is completely eliminated.

The IBM Series 1 has one further feature, in that with double-operand instructions, each operand can use a different set of mapping registers, a great aid to context switching and parameter passing in supervisor calls. The Intel 8086 also uses separate relocation registers for code and data.

Of the systems discussed, only the 32-bit Interdata 8/32 with its 20-bit virtual address and the Intel 8086 with 4 × 64 KB segments can handle programs over 64K (64 KByte on, say, a TI 990/10, but 64 KWord or 128 KByte on a word-oriented machine like the MODCOMP IV). This concept has been developed further on the DEC VAX 11/780 and Prime 400/500 to full virtual memory concepts on a par with mainframe computers (see figure 2.23). The scheme is too complex to discuss here in detail, since it requires memory and disc resident look-up tables for mapping data. Briefly, however, the Prime uses 16-bit segment addresses (128 KB, since it is word oriented) and a 12-bit "segment-number" extension to the address. Further, it generates another 12-bit "process number" for multiprogramming user identification. With $2^{12} = 4K$ possible segments and process numbers and each segment mapped into 64 pages each of 2 KB, it is impossible to maintain one relocation register per segment and page. The machine instead keeps a dynamic reference to 64 pages in a "look-aside buffer". The virtual page address is hashed and compared with the entry in the buffer: if it matches, the relocation address is immediately available; if not, a fault is recorded and the hardware uses a double memory access (a segment table that points to a page table) to update the buffer from memory resident tables. A 97 percent hit rate is claimed in practice. The hashing algorithm is an interesting variant on the content-addressable technique often used on mainframes.



Physical memory address (256K)
One map per program (figure for MODCOMP; for others see list below)

MODCOMP IV:	4 sets of 256 pages, each 256 words (64 KW) 10-bit page register ≡ 256 KW physical address
Varian V77:	16 sets of 64 pages, each 512 words (32 KW) 11-bit page register ≡ 1 MW physical address
Data General:	4 sets of 32 pages, each 1 KW (32 KW) 7-bit page register ≡ 128 KW physical address
Hewlett-Packard:	4 sets of 32 pages, each 1 KW (32 KW) 10-bit page register ≡ 1 MW physical address
IBM SERIES 1:	8 sets of 32 pages, each 2 KB (64 KB) 13-bit page register ≡ 8 MW physical address

Figure 2.22
Mapping by pages

One final point concerning memory mapping relates to DMA transfers, which are, of course, to physical addresses. Two alternatives are shown in figure 2.17. The first is to use the mapping unit to set up physical addresses in the disc controller; the second is to allocate a map set to the DMA transfer. The latter ties up a set of mapping registers, but the former can only transfer to a contiguous block of memory. Note in either case that roll-out cannot be allowed if the program concerned has initiated a DMA transfer.

Hardware Enhancements

More and more of the features of systems previously implemented in software are, for efficiency reasons, being introduced into hardware. Some of these are outlined below.

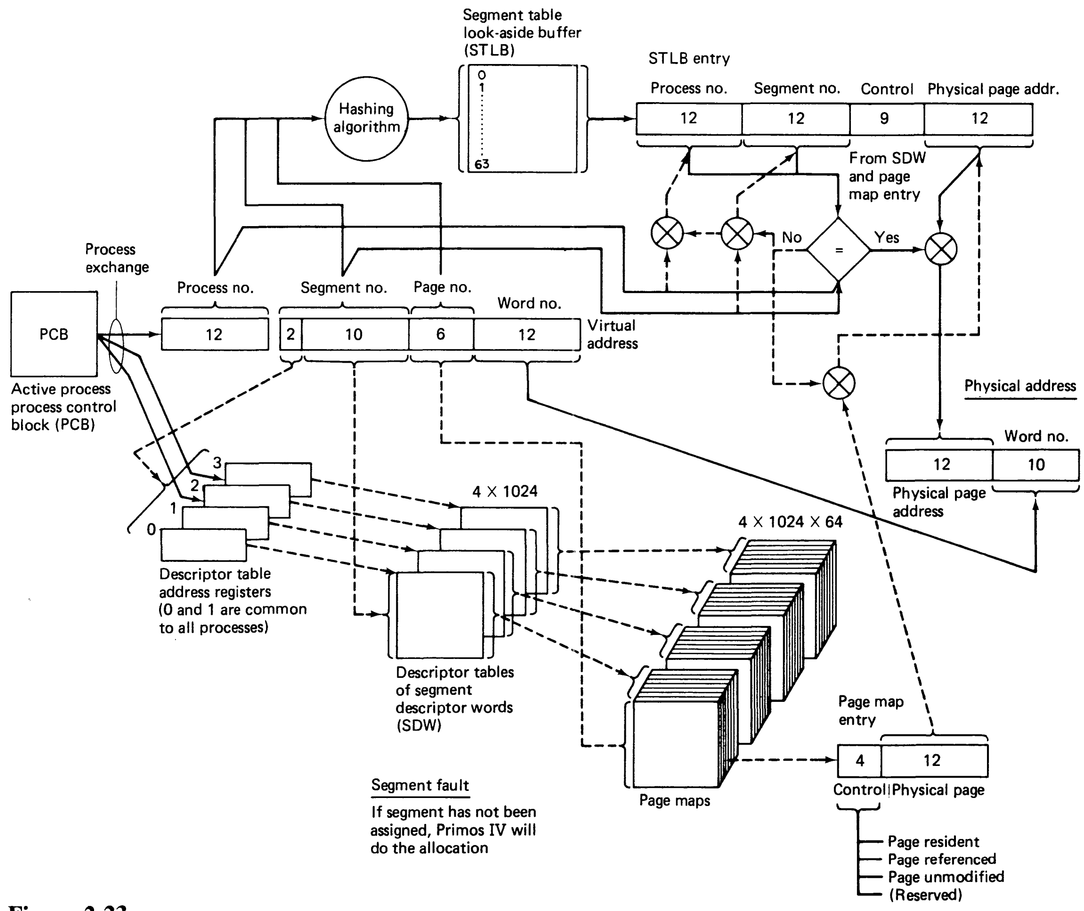


Figure 2.23
The Prime 400/500 paged and segmented memory management system

Executive Functions

The heart of any operating system is the executive, a set of routines that effectively control the processor functions and resource allocation. It is normal to write the executive as a software program to be loaded as a memory resident unit. This program could be written into ROM rather like a glorified bootstrap, still taking up some of the normal memory address allocation. An alternative is to implement the nucleus of the executive as an extension of the normal instruction set. Once this concept is accepted, as in the GEC 4000, it affects the more normal considerations. The more conventional machine has standard I/O features that are utilized by the executive. With an integral executive, I/O is one of its functions, negating the need for conventional instructions. This is meaningless for a single program operation but is extremely efficient in a multiprogramming environment. The executive can also handle mapping and protection, particularly interprogram corruption. One problem that arises is that executive calls are made from the program as though they were normal instructions. Thus specific bit orders are required, reducing the number available for other instructions. This is a real problem in 16-bit instruction sets, since there are never enough combinations.

Arithmetic units

The simpler machines do not support multiply or divide instructions. However since most machines are micro programmed (see later) these operations are offered as part of an optional extended instruction set, implemented by a new microcode ROM. 16-bit integer operations are common but the range can extend in some machines to full floating-point; BCD arithmetic is now a common extension. Microcoding an extensive operation such as floating-point multiply results in a relatively slow execution time. Thus some machines offer an expanded high speed hardware arithmetic unit which is nevertheless controlled by microcode and thus acts as a part of the CPU. New LSI parts such as the Intel 8087 are designed to be paralleled up with the 8086 CPU. An alternative scheme is to provide the arithmetic unit as a high speed peripheral device; input data is loaded and results extracted by I/O instructions. In some cases it is possible to overlap the execution of the longer arithmetic operations with other instructions.

Multiple Register Sets

The content of the program accessible processor registers must be stored whenever the program flow is altered. This can be avoided by using multiple register sets, switching a pointer to the active set when necessary. This could be used at JSR level, but would then put a constraint on the number of nested calls. The technique becomes much more powerful in a multiprogramming system, when one mode is used by the executive and another by user programs.

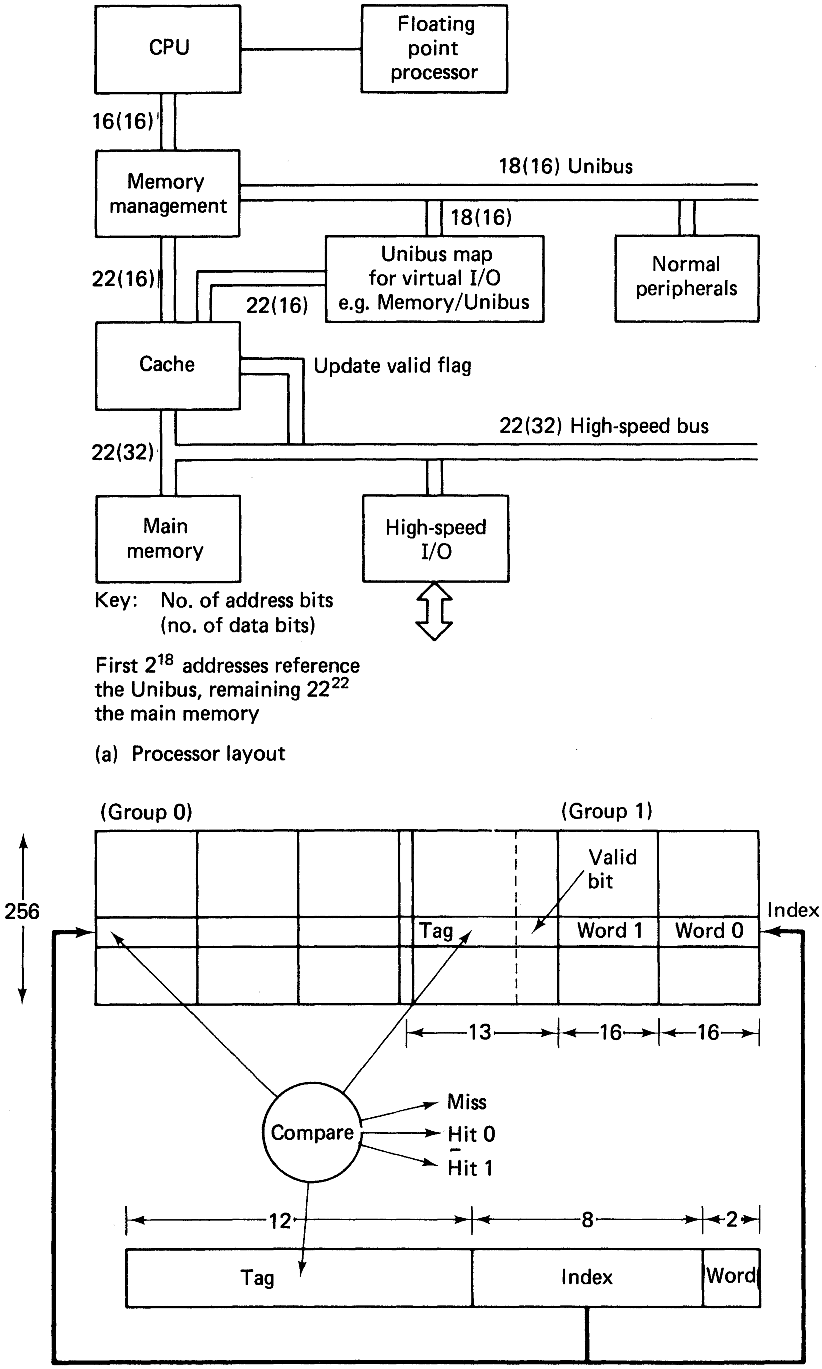
Cache Memory

A cache memory is a block of high-speed bipolar memory located between the processor and main memory. When memory is read initially, data is transferred as usual, but it is also written into the cache together with a marker that indicates which memory location it is copying. The next time memory is read the cache is first checked and main memory is accessed only if the required data cannot be found. If a "hit" occurs data is available at, say, 300 nanoseconds; for a "miss," normal memory speed of, say, 1 microsecond is required. All writes must update main memory, usually checking the cache at the same time and marking a hit as invalid cache data. Thus, if—and the figures quoted are reasonable—80 percent of memory accesses are reads and 80 percent of cache hits are achieved, the average memory speed for the above system is 550 nanoseconds. All DMA writes must also check the cache and mark any hits as invalid.

Different organizations and techniques for checking cache are employed. If multiple cache blocks are employed then the hit rate is improved, since instructions and data tend to be in contiguous but separate blocks. A further improvement results with incorporation of "instruction look-ahead" when two or more words are copied into cache locations for each actual read from main memory.

Figure 2.24 shows the cache employed by the PDP-11/70, which uses a 22-bit physical address. Four bytes are read at a time from memory to cache and there are two independent groups of cache registers, each of 256 locations. Each location stores the two words plus a "valid" bit and 12-bit "tag" field and 6 parity bits. Eight bits of a memory address (the index) are used to point to one of the 256 cache locations. The most significant 12 bits (the tag) of the address are compared with the two stored tags, provided the valid bit is clear, to determine whether to read group O, group 1, or memory. If a hit is achieved the least two significant bits define which word to transfer. For a miss, memory is accessed for the required word, but at the same time the 2 bytes transferred plus the next 2 are written into cache together with a copy of the tag. When writing the same check is made, the valid bit being set if a hit occurs. The PDP-11/60 uses a similar cache but with only one group of 1024 locations. Two bytes only are transferred and stored (no instruction look-ahead); the machine uses an 18-bit physical address with a byte bit, a 10-bit index, and a 7-bit tag.

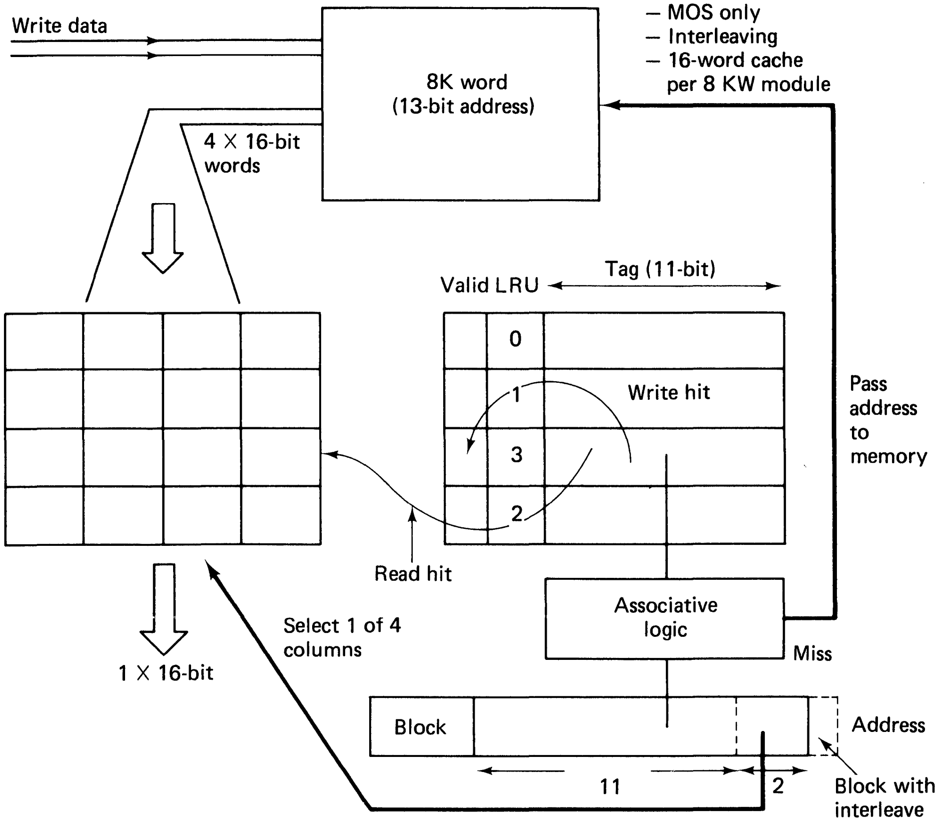
The Data General ECLIPSE cache, figure 2.25, uses a more conventional content-addressable technique. There is one 16-word cache per 8K word block that is split into an 11-bit address and a 2-bit one-of-four word indicator. For each of the four cache locations there is an associated 11-bit address register, a valid bit, and a 2-bit least-recently-used (LRU) marker. The content-addressable (associative) logic determines which of the four locations, if any, has the desired data. If a hit is achieved the appropriate word is transferred on read or the valid bit is set on write. If on read a miss occurs, four words are transferred to the cache location which has the "oldest" LRU rating. For all accesses the LRU bits are updated.



(a) Processor layout

(b) Cache

Figure 2.24
The DEC PDP-11/70



All reads via cache: Hit direct
Miss from memory } Update LRU during next cycle

All writes to memory: Hit update LRU and set invalid

Figure 2.25
The Data General ECLIPSE cache

Memory Interleaving

Core memory requires a read and write operation for every access. If even addresses are physically arranged to lie in one block and odd in another, then the read cycle of the next access can commence while the write cycle of the current access is completing. Since many accesses are sequential, the effective memory transfer speed can be increased. Interleaving to more than two levels is available, increasing the probability of nonconsecutive access to the same physical block.

Interleaving of semiconductor memory is still useful but not as effective as with core.

Multiport Memory

As already discussed, multiport memory can minimize cycle stealing in DMA transfers.

Parity and Error-Correcting Memory

While most memory uses either 8 or 16 bits per unit of data, memory is readily available with extra parity bits, typically 1 per byte. The 9-bit word is checked for parity whenever it is accessed, a detected error causing a flag to be set and raising an interrupt. For more specialized applications memory units are available with 21-bit words, 16 data and 5 for an error correcting code (ECC). With 5 redundant bits in 21 an error in any one bit can be detected *and* corrected. Since 4 extra bits are required to error-correct an 8-bit data unit, ECC is normally applied to the 16-bit word.

Tightly Interconnected Machines and Shared Peripherals

Much of this text is devoted to intermachine communication as a means for moving applications-oriented information from one point to another. For purposes of system reliability and "up-time," however, it is occasionally necessary to employ redundant processors and peripherals. In such cases standby components must be switched into the system either automatically or manually. Such machines require tight interconnection, being capable of sharing resources at high speed bus data rates without any appreciable time overheads. The most common solution is to use multiport memory systems with two processors sharing the same memory, the porting hardware on the memory system acting as a multiplexor. Machines like the PDP-11 which does not support multiport memory must use special bus interconnection hardware systems, in DEC's case by linking two Unibuses. A special hardware box is also provided, effectively a Unibus extension, into which peripherals are connected. This box can be switched into one or other of two Unibuses, and hence peripherals can be shared between two processors. A much more sophisticated MODCOMP system is described in chapter 5.

Diagnostic and Monitor ROM

Most minis use a small bootstrap or initial program load (IPL) program stored permanently in a read-only memory. Initialization of the bootstrap causes the operating system to be copied down from the disc into memory and started. Most often this ROM uses a fixed part of the user's address space but is also quite often built into the microprogram ROM. The facilities of a program provided by such a ROM can be quite advanced. Two such applications are diagnostics and monitors. Typically on start-up the processor executes a diagnostic routine that checks all registers and memory ports, and flags any errors it finds. A monitor routine can allow the operator to use the console terminal to communicate direct machine control functions such as changing and displaying the contents of specific memory locations in near English text rather than using switches and lights on a front panel. Numbers can be printed in hexadecimal rather than binary. Because even the simple microcomputers use good ROM-based monitors, IBM's complicated panel console on the Series 1 comes as a distinct surprise on a modern machine.

Error Logging Systems

All processors report detected errors by setting bits in status registers. At the simplest level only limited errors such as arithmetic overflow are detected, and it is the sole responsibility of the applications program to check these flags. As machine sophistication increases more and more system functions are checked by hardware. Detected errors can be coupled to the interrupt system to activate an operating system support program to check whether a program must be retried, aborted, or allowed to continue. Error and status reporting from intelligent I/O controllers can also be incorporated in the system. Logging of errors can be an operating system software function, but the bigger minis are using microprocessors (the Amdahl 470 uses a Data General NOVA minicomputer for the same purpose) to log errors to a floppy disc, which can be used by a centralized maintenance department. Remote access to the microprocessor system via a communication line is also possible.

Power fail

A device is commonly used to monitor the power supply. When the supply is broken the power-pack capacitors allow the machine to function for a further fraction of a second; the power fail protection unit then generates an interrupt which overrides all others. This initiates a program to save the contents of all CPU registers in nonvolatile memory locations. When power is resumed another interrupt is generated to activate a program restoring the processor to its original state. To be effective with MOS memory, backup battery power supplies must be provided to overcome the volatility.

Clocks

Many routines require timed event indicators varying from time-of-day printout, through operating system time sharing of resources, to program control of a sophisticated data logging subsystem. These markers are generated by a binary counter driven by a fixed-frequency oscillator that can generate an interrupt after a specified number of ticks. The clock is commonly locked to power supply frequency but faster and more accurate units use crystal-controlled oscillators. The more sophisticated clocks can be program controlled to set rate and mode of operation. Multiple clocks may be used by a bigger system.

Microprogramming

Microprogramming is a technique for implementing the instruction set. The processor so far described is itself treated as a programmed computer, a processor within a processor as shown in figure 2.26. At the inner level the instruction set is of low sophistication with, for instance, direct control of gates. Each machine-level instruction calls an inner machine program, a microprogram, to execute that instruction. To be effective this micromachine must be very fast. Thus long word lengths (they need have no relation to the full machine word length) of 30 to 100 bits are used so that the maximum number of concurrent, rather than sequential, operations can be executed. The microprograms are written into high-speed ROM with cycle times of a few nanoseconds. Executing a machine instruction by running a microprogram is essentially slower than with a dedicated, hardwired logic system. The advantage of microprogramming is simply versatility. By changing the microprogram the main machine instruction set can be changed, and thus the machine can be tailored to suit the application. Actual tailoring is seldom used, since appropriate operating systems will have to be generated. However, machines can now be marketed with a standard instruction set that can be

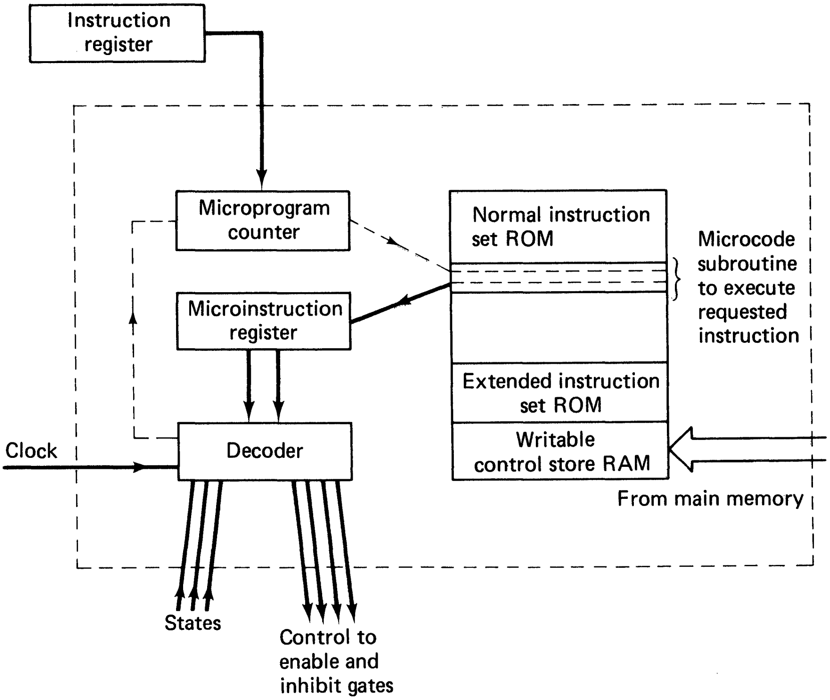

Instruction register
Microprogram counter
Microinstruction register
Decoder
Clock
States
Control to enable and inhibit gates
Normal instruction set ROM
Microcode subroutine to execute requested instruction
Extended instruction set ROM
Writable control store RAM
From main memory

Figure 2.26
Microprogrammed machine instruction decoder

extended (within the constraint of available bit patterns) by extending the microprogram. Indeed, if a section of high-speed RAM is included in the micromachine memory, this could be loaded by the operating system to change the extended part of the instruction set at run time.

Microprogramming also helps the machine manufacturer to develop new products, since a unit can initially be microprogrammed to emulate the model being replaced, using the software already available.

32-bit Machines

Most minicomputers are internally 16-bit orientated. This is somewhat confused by features such as multiword instructions (MOV A,B on a PDP-11 is a 48-bit instruction) and memory management systems generating 18 to 24-bit physical addresses with 4-byte wide cache. Nevertheless all registers internal to the PDP-11 are 16 bits long and the sub-division of the 48-bit move instruction is on 16-bit boundaries; a virtual address still cannot exceed 16-bits. Thus there are a number of new machines appearing with longer internal words. These are often called 32-bit minis but they are direct competitors to mainframes, retaining nevertheless much of the mini's I/O flexibility.

The leaders with this trend were System Engineering Laboratories with the SEL 32 and Interdata with the 7/32 and 8/32 machines. Interdata still used a 16-bit memory system but defined an architecture and instruction set similar to an IBM S/360. In particular although instructions are fetched from memory in 16-bit words, limiting offsets and displacements to 16-bits, internal program counter and index registers are 24-bits long giving a virtual address space of 24-bits (16 MB). The memory management was very simple, merely providing sets of 24-bit base relocation registers. Since the instruction set is defined as multiples of 16-bits, downwards compatibility with the earlier 16-bit machines was retained. Operations on 32-bit long data items are included in the instruction set, but this is a feature of 16-bit machines such as the GEC 4000 and machines with added floating-point processors.

With the introduction of VAX 11/780 DEC moved firmly into 32-bit machines. VAX 11/780 has two complete instruction sets, the 16-bit unprivileged PDP-11 set and its own 32-bit native mode set. The machine uses a common internal bus as did the PDP-11, but with a maximum transfer rate of 13.3 MB/second (synchronous) with support for up to 4 Unibus and 4 high-speed (2 MB/second) I/O subsystems, The machine has a 32-bit virtual and a 30-bit physical address space. Memory management hardware is organized to support the extensive virtual operating system, a feature more typical of a mainframe than a minicomputer. The VAX 11/780 instruction set is one of the richest on any computer, making a number of features available to programmers which are part of executive functions on mainframes (Que and deque, procedure call, polynomial evaluation). This is one of the few machines to support triple operand addressing ($A = B + C$ for example) and features extensive string handling instructions (move a byte string from one location in memory to another, translating in the process via a memory resident table as one instruction).

Microelectronics in the Computer Industry

Integrated Circuit Technology

The concept of producing large numbers of transistors and resistors on a single chip of silicon to form a functional unit is now well established. Starting with small-scale integration (SSI), as in a quad two-input AND gate, the process moved through medium-scale integration (MSI), as in a shift register and adder, to large-scale integration (LSI) such as 4K-bit memory chips, simple CPUs, and so on. The near future offers us very large scale integration (VLSI) with some remarkable products that promise to have a spectacular impact on the structure of the computer industry and, one hopes, a marked improvement in cost-effectiveness and access for the end user.

Figure 2.27 illustrates the trend in packaging components into single chips. One can readily see by comparing the requirements of, say, a PDP-11/34 CPU that a device of such complexity was available by 1979 as a single VLSI chip (8086, 68000, or Z8000). Equally important, the 64 K bit memory chip was introduced by 1980 and predictions are being made of a megabyte of read/write memory on a single board for only a few thousand dollars by the early 1980s. Figure 2.27 does not tell the whole story; other dimensions such as speed, power consumption, and cost must also be considered. In general the faster devices consume more power and are available at lower packaging densities. In practice a number of manufacturing techniques are used. Sometimes one technique improves so as to obsolete another; more often, as this year's slow technique becomes as fast as last year's fast technique, then this year's fast technique is an order faster still. Figure 2.28 shows a rough assessment of the different technologies. Metal-oxide semiconductor (MOS) devices effectively use field-effect transistors (FETs), the others Bipolar transistors.

Transistor-transistor logic (TTL) is still the common technique for most computers, emitter-coupled logic (ECL) for bigger, faster processors, and MOS for LSI components such as memory chips and microprocessors. Some TTL LSI components are available now (the Ferranti F100 microprocessor, for one). Silicon-on-sapphire (SOS) MOS technology offers potential for VLSI and speed in the near future, but has suffered badly in attempts to put it into production (the General Automation processor manufactured by Rockwell and since withdrawn). Hewlett-Packard claims to have achieved a breakthrough for immediate use in its desk-top computers. Other MOS technologies (VMOS, HMOS) are giving dramatic speed increases.

Some wonderful technology has gone into reducing the size of individual components in LSI circuitry. VLSI makes even greater demands, down to electron-beam techniques for the finest control of element size and position.

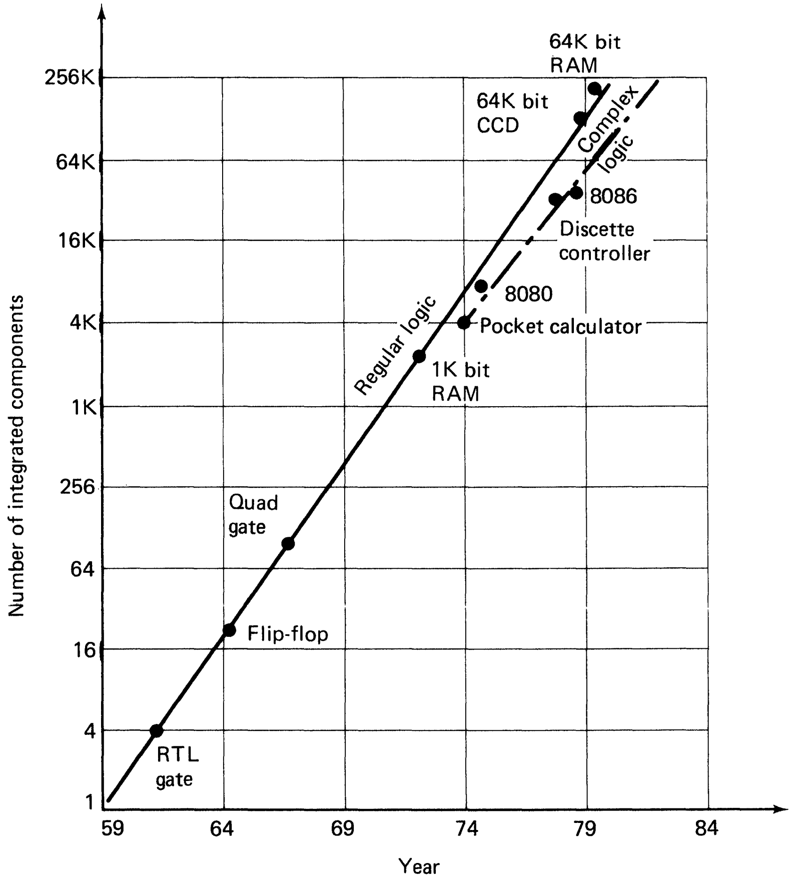


Figure 2.27
The progress of integrated circuit technology

Memories and Bulk Data Storage

Random access memory chips are the largest-volume requirement for LSI components. While there is still some life in the magnetic core market, semiconductor memory systems made from MOS LSI chips are now the standard. Typical speeds range from 50- to 500-nanosecond cycle time with 50- to 400-nanosecond access time. MOS memory is volatile and must have battery backup if the system cannot tolerate reload from a disc. There are new developments in nonvolatile MOS memory, but these are more expensive than the bulk-produced n-channel (NMOS) chips. A now more common technique is the use of a small high speed TTL memory as a cache to a main MOS memory.

MOS memory chips are available as static or dynamic parts. The dynamic chips offer higher densities (in 1980 4K-bit static versus 16K-bit dynamic RAMs

Technique	*Speed*	*Comment*
	(nanoseconds)	
MOS		Most LSI and VLSI components
PMOS	200-1000	Oldest and slowest; obsolete
NMOS	50–300	Most common variant
CMOS	50–500	Very low power version
SOS, VMOS, HMOS	1–100	Latest and fastest
TTL		Conventional computer systems
Low power	25	
Medium speed	8	Normal MSI, some LSI
High-speed Schottky	2	
I^2L	1	
ECL	0.2	Very fast MSI processors

Figure 2.28
Types of LSI technology

were standard) but require external refresh logic circuitry to rewrite the stored data every two milliseconds. Most chips are bit oriented so that eight 64K RAM chips are required, plus refresh logic, for a 64KB memory system. Note however that address decode logic is internal to the chip.

Read-only memory (ROM) chips are readily available, from production masked versions through fusible-link and electrically programmable (PROM) to ultraviolet erasable EPROM (this last reuseable) types. The latest EPROMs run at about 250 to 500 nanoseconds; microprogram ROMs must use much faster devices. Note also that fast Schottky TTL RAM (random-access memory) is needed here for writable control store, not MOS.

Two other integrated circuit devices, the charge-coupled capacitor (CCD) and the magnetic bubble memory, are currently creating a stir. These are effectively shift registers, not RAM, and are intended as bulk stores. In effect they can be considered as a sector on a disc with near-zero access latency. They are now slow and expensive, but their potential is high. Probably the same arguments of bubble vs. CCD will rage as did MOS vs. Core: as users we do not care who wins as long as the technology offers advantages. The most obvious use for CCD or bubble stores is as multiple 512-byte shift registers to replace fixed-head discs or drums; ironically there is a ready availability of commercial solid state systems using slightly modified MOS or core RAM! Much more work is needed in system and architectural design to see how best to take advantage of these new devices.

While the fall in the cost of memory has an obvious impact on system cost, the overall effect must be much more dramatic. With the new generation of microprocessors (let alone conventional machines) being able to address megabytes of memory, then the concept of sophisticated memory/disc management schemes such as paging will simply be unrealistic. Thus the low cost of memory will impact on software development in the future.

Microprocessors

While memory chips were the obvious target, and still are, for development of commercial LSI, the intriguing follow-up was the pocket calculator. For once the demands of millions of schoolboys (as opposed to spacecraft and missiles) determined a push forward in technology. The calculator chip, while no higher in gate requirements than a memory chip, is a complex random logic device compared to the repetitive pattern of a RAM. The techniques acquired in supplying the vast market for calculator chips provided the technological springboard for the initially more circumspect market for general purpose CPU chips, which are now as commonplace as calculators. These CPUs on a chip are known as *microprocessors*.

Pins on the chip are obviously required for power supplies, system clock, data, and addresses. Since the internal components of the CPU are not externally accessible, some pins are required to indicate the current internal status so that external components can be synchronized. Typical signals indicate memory or I/O operation, Read or Write, and timing information (which sub cycle). This becomes more important since due to the limited number of pins, their function is time multiplexed (a common example is eight pins carry the least significant eight bits of the address for one sub cycle, followed by eight bits of data in the next sub cycle); demultiplexing is provided externally, controlled by decoding the status data. Finally, some pins are required to feed external control signals into the CPU, the common ones being reset, hold (for DMA), wait (slow down the processor to slower memory speeds), and interrupt. Hold and interrupt acknowledge status signals are also required. From this it should be realized that pin-out is a major limitation on microprocessor design and that multiplexing of pins is always used.

Figure 2.29 shows the internal architecture of a typical microprocessor, the Intel 8080. This shows that the processor uses an 8-bit data word but by use of double internal registers can generate 16-bit addresses. This (and the other 8-bit microprocessors) uses multi-byte instructions. The first byte is fetched and decoded, reading subsequent bytes if needed to complete the instruction fetch. Thus increment a register is a one-byte instruction while jump requires three bytes, one for the op code and two for the address. The instruction set as a result is surprisingly rich, far more powerful for instance than the old PDP-8, which although a 12-bit machine, codes every instruction into one word. The range of addressing modes is limited, extensive use being made of pointer-mode; thus on the 8080, to add the contents of a memory byte to the accumulator requires the address to be set into register pair H and L followed by a single byte add-to-accumulator instruction. Auto-increment/decrement modes allow common use of

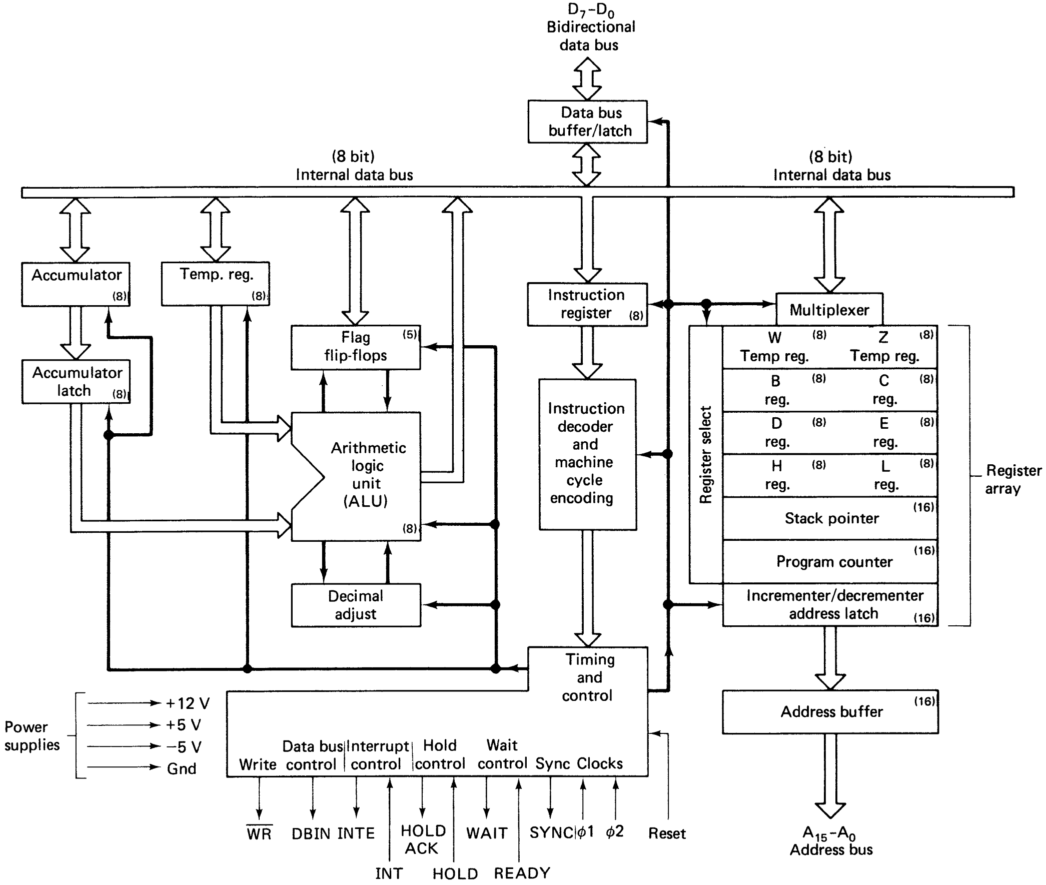


Figure 2.29
8080A CPU functional block diagram

stacks. The Zilog Z80 is a superset of the 8080 featuring an additional relative addressing mode and byte string operations.

Since MOS is the only technology available for mass-produced LSI parts (originally PMOS, commonly NMOS, and lately variants such as SOS, VMOS, and HMOS) computers built around microprocessors are relatively slow compared to 16-bit minis. Clock rates of two to four MHz result in typical instruction execution times of one to ten microseconds (remember that the 3-byte jump instruction requires three sequential memory access just to fetch the full instruction). Further, the data unit manipulated by the instruction is only 8 bits; thus software routines for a 16-bit integer multiply can take 150 microseconds and a 32-bit floating-point multiply 10 milliseconds.

While the low cost and ease of interfacing of the 8-bit micros has made them ideal devices for terminals and simple computers, there are requirements for more processing power. Technological advances saw the introduction in 1979 of the new generation of 16-bit microprocessors, featuring as many as 30,000 elements on the single chip. The earlier 16-bit micros like the National Semiconductor IMP 16 and DEC LSI-11 are in fact built on chip sets, typically a register set, an arithmetic and logic unit, a microcode decoder/controller, and a microcode ROM. The LSI-11 uses the Western Digital MCP-1600, as does the Alpha Micro and the Pascal Micro Engine.

Data General and Texas Instruments both produced single-chip micros with instruction sets compatible with their minis. The Data General chip is used in the Micro Nova, while the Texas Instruments 9900, although used in the 990/4 and 990/5 computers is a general purpose CPU. In fact the more powerful 16-bit 9900 was available about the same time as the 8080 and 6800 but failed to appeal to system builders because the other supporting LSI components, discussed below, were not forthcoming. A somewhat more intriguing variant is Fairchild's 16-bit micro, which emulates the Nova instruction set. This was clearly aimed at helping system builders to use existing Data General software without buying processors from Data General and as a result led to a lawsuit. In any case this concept has been negated by the new 16-bit general purpose micros since they all have far more powerful features than the old and simple Nova. Three 16-bit processors are competing to dominate the single-chip general purpose market, the INTEL 8086, the Motorola 68000, and the Zilog Z8000. As with the 8-bit processors, Intel are well ahead with the overall system concept and more advanced development system support. Zilog with the Z8000 are basically producing a single-chip implementation of a typical minicomputer CPU with privileged instructions and vectored interrupts. The 8086 is much more a component in a new systems concept along with the 8087 arithmetic processor and the 8089 I/O processor. The multibus bus system specified by Intel for their development systems and single board computers features multi processor support; there is in fact a single-chip bus arbitration logic component.

The 8086 architecture is shown in Figure 2.30. In fact it is not just a CPU but a CPU and a simple memory access controller combined in a single chip.

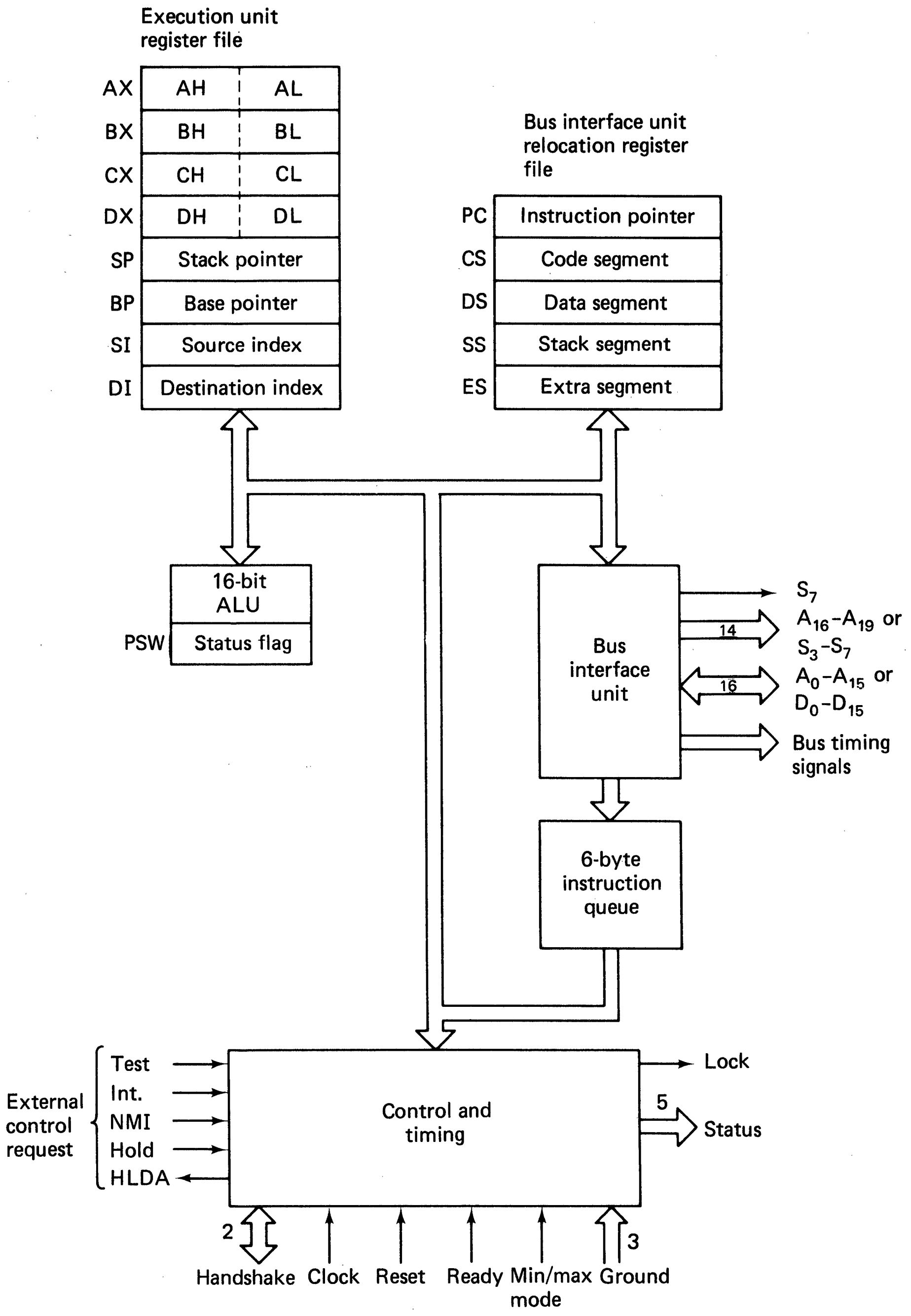


Figure 2.30
INTEL 8086 architecture

Instructions are loaded into the buffer direct from memory while the CPU fetches instructions a byte at a time from the buffer to decode and execute them. Thus apart from initial instruction fetches and breaks in program sequence (jumps), instruction fetch and decode/execute overlap. Memory is byte addressable accessing 8 or 16 bits as required. The internal registers are 16 bits long; however, addresses are generated 20 bits long by adding normal 16-bit addresses to a 20-bit base relocation register. In fact there are four base registers, only the most significant 16 bits of which can be set, the lower four being zero. Thus a 64 KB segment (16-bit) can be created anywhere in a 1 MB physical memory space (20-bit) on a 16-byte boundary. One base register is used for the instruction fetch (code), one for data items, one for stack, and one for a further data area (common). This extremely flexible architecture does not however provide memory protection features or paging. The Zilog Z8000 supports a separate memory management chip, providing multiple register sets, paging and protection. Both the 8086 and Z8000 have maximum segment sizes of 64 KB while the M68000 uses direct 24-bit addressing. These 16-bit processors feature multi-byte instructions, multiply/divide, byte string move, and translate operations. As a result, they are more powerful than many minicomputer CPUs.

Microcomputers

To build a computer àround a microprocessor a bus is normally used to synchronize the external chips to the CPU. All the timing and internal status data from the CPU must be latched and decoded to enable the other components at the correct instants in time. A simplified bus structure is shown in Figure 2.31. In practice, in order to provide drivers and decoders for a full system, as many as forty SSI/MSI chips are required. Thus the manufacturers developed special support components to replace commonly used combinations; the 8224 and 8228 shown in Figure 2.31 are typical. It is this concept of a family of components that is all-important to a system builder, lack of which hampered the acceptance of the TI 9900. Intel now offer the 8085 microprocessor which is effectively the 8080, 8224, and 8228 combined, an example of technological advances simplifying system design rather than the more obvious performance enhancements.

The semiconductor manufacturers have developed a most impressive range of special-function LSI chips, usually part of a family for ease of interconnection, all of which make for easier design of computer systems. Some typical parts are described briefly later.

There are a number of levels on which microprocessors can be packaged with other chips into microcomputers. Obviously in building a small business machine, a VDU, or a printer controller, a special printed circuit board is required. However, by constructing a physical bus on a motherboard, distinct functional units can be constructed on single plug-in cards. Typical cards would be a CPU, RAM (8K, 16K, 32K, or 64K), EPROM, serial I/O, printer controller, disc controller and so on. Each card has all the necessary logic to decode and drive the

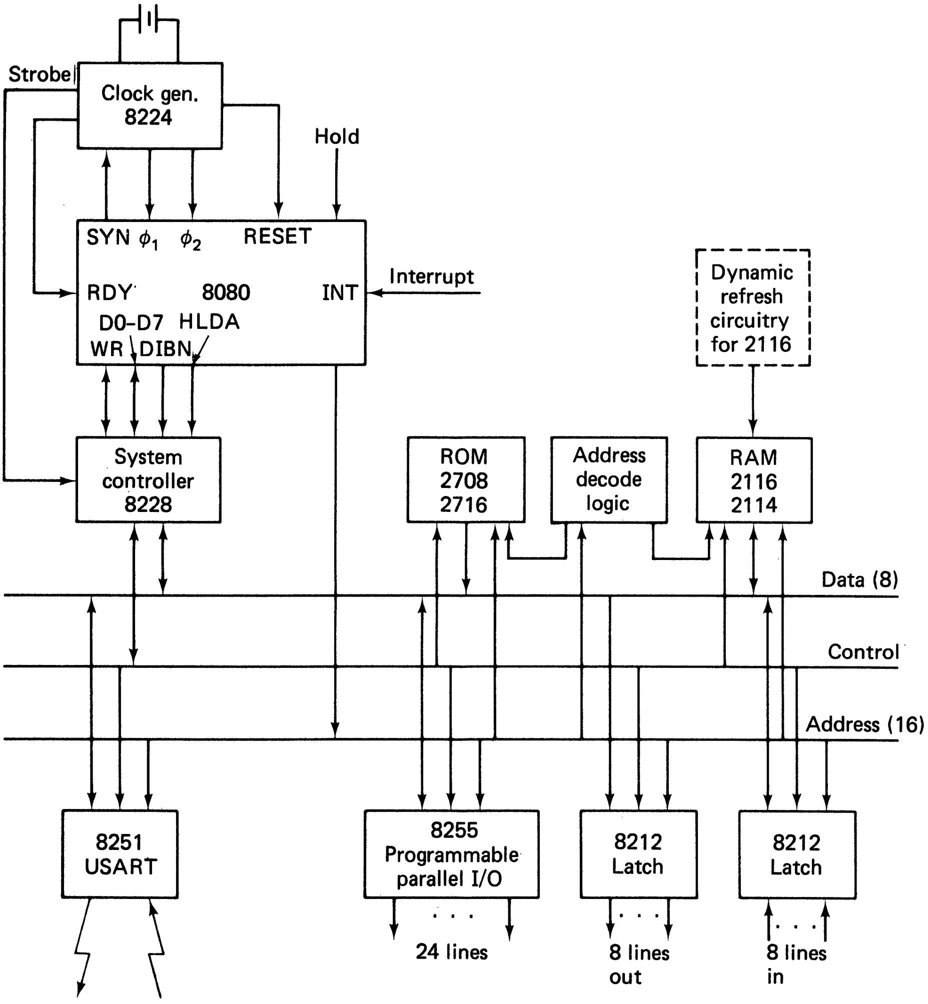


Figure 2.31
An Intel 8080 System

bus, interconnecting with other cards. Of course, this construction technique is used in all computers, but the general purpose availability of microprocessors to new computer manufacturers has led to the adoption of standard buses, in sharp contrast with the rest of the computer industry. Multibus (defined by Intel) is the more sophisticated example, but the most common is the S100 bus defined by MITS for the Altair, but now with an official IEEE specification. Intel first introduced the multibus for a range of off-the-shelf prototyping cards. With recent advances in LSI technology far more components can be packed onto the one board; in particular some RAM, ROM, and simple I/O can be built onto the CPU card leading to the concept of a single-board computer (SBC), albeit still retaining the bus interface logic for expansion. Personal computers such as the Commodore PET, or Intertec Superbrain achieve their low cost by eliminating the overhead of a general purpose bus with a loss of flexibility.

While in general microcomputers are not made by the component manufacturers, they do in fact make single-terminal discette-based systems intended as engineers' design tools. These are called Microprocessor Development Systems (MDS). An MDS supports program development software and special debugging aids. They could be used as general purpose microcomputers but are far too expensive. The specific features of an MDS are support for real-time programming (system) languages such as PL/M, PL/Z and MPL (all manufacturer-specific), and modular run-time executives (Intel's RMX80). More important, however, is an in-circuit emulator (ICE) which allows a target hardware system to be temporarily linked into the MDS. (The target machine microprocessor is removed and an umbilical cord plugged into the socket to link to the MDS.) The ICE allows all the facilities of the MDS to be used on the target machine. Thus the MDS RAM can substitute for the target machine ROM allowing breakpoint debugging via the MDS VDU. The ICE also maintains a running trace of the last hundred or so bus operations which can be examined after a trap to detect program logic errors. Finally, the MDS allows debugging by symbolic reference to the labels used in the original source program by referencing the symbol table on discette.

With the exception of an MDS, a computer built around a microprocessor must be evaluated as a small business computer, low-end mini, or whatever, based on performance, software, facilities, peripherals, etc. Single-chip CPU versus MSI multi-chip CPU is of secondary importance. Note, however, that with microprocessors numerous computers are being built which will support basically the same software. (Microsoft BASIC in one form or another is used on Tandy, Altair, National Panasonic, Horizon, Cromemco, PET, and many others.) This is an important trend in reducing software development costs by sharing across multiple computer manufacturers.

Other special purpose devices are more commonly using microprocessors. Some use multiple microprocessors to achieve high performance with modularity. An early example of this was the Codex 6000, an intelligent network controller, which allows a network of terminals to be connected to various lines with various protocols. By using multiple processors sharing a common high-speed bus, one group can be reconfigured while the others continue to function normally. A schematic diagram is shown in figure 2.32.

LSI Components for Serial I/O

The basic operations of converting a parallel byte of data into a bit serial stream for output (transmission) and collating an incoming (received) bit serial stream

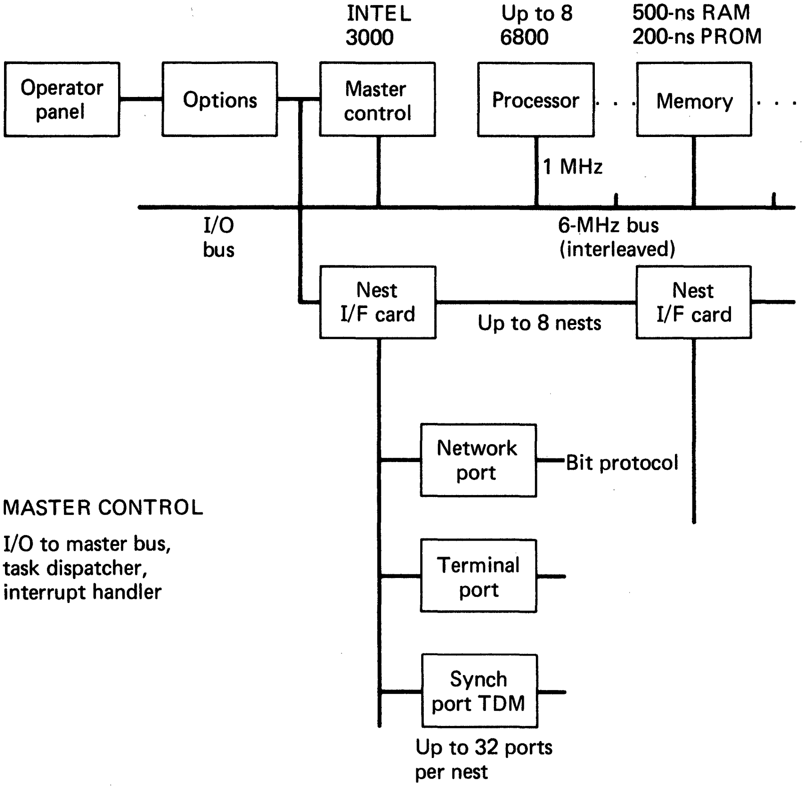


Figure 2.32
The Codex 6000 intelligent network processor

into a parallel byte are now implemented in LSI parts. Earlier computers actually involved the computer in bit at a time I/O, so-called *bit-banging*. The LSI I/O chip provides serial I/O to the line drivers and a parallel port to the computer bus. On the computer side these chips are provided with chip select and other timing and status pins to ease the interface design; often they are specifically designed to be easy to interface to a given micro, the family concept mentioned previously (8251 USART with 8080 or Zilog SI/O with Z80). The chip will contain a register to accept an output byte and another to store the received byte; most often double buffering is supported so that the computer can handle one byte while another is being serialized/deserialized. The buffering means that the computer is only involved for the specific I/O instruction and can easily be interrupt-driven, since data received or transmitted status bits may be provided. These devices are usually MOS parts but with the usual TTL pin compatibility at low power levels. They must thus be clocked to synchronize the bit timing. For asynchronous transmission (see discussion in next chapter) the clock is provided independently at each end of the line. Each character has a unique start bit which restarts the timing, and therefore the two ends need to stay in step only over a period of, say, ten bits. A local crystal oscillator is more than adequate except that one end could coincidentally clock on the bit transition boundary. To avoid this all asynchronous parts are clocked at 16 (or 32 or 64) times baud rate; inside the chip this is divided and the first bit synchronized to the middle of the bit interval. With synchronous transmission the modem uses a self-clocking technique, thus providing synchronized timing signals to each end of the line, which are used directly by the I/O chip. Some of the available chips are now discussed.

UART (Universal asynchronous receiver transmitter). A UART provides double buffering of both transmit and receive characters, both channels being independent for full.duplex operation. They handle independent receive and transmit clocks at 16 times baud rate. The actual logic inside the chip provides for a choice of 1, 1½, or 2 stop bits, 5, 6, 7, or 8 data bits and odd, even, or no parity. The particular requirements of a given application are selected by wiring specific pins either to ground or +5 volts. Pins are also provided to report status information for data received and transmitted and also to report receive errors such as overrun, parity, or framing. A schematic diagram of an interface using a UART is shown later in figure 5.4.

USRT (Universal synchronous receiver transmitter). A device similar to a UART except that no start or stop bits are involved and the clocks run at baud rate.

USART (Universal synchronous/asynchronous receiver transmitter). A USART combines the features of a UART and a USRT. In general they also provide further logic for direct support of modem control functions (request to send) and status functions (data set ready); with a UART these features are provided by additional TTL logic. However, with an increase in the number of logical lines to be supported and in the range of options provided, there would be too many pins required. Thus a more powerful technique of a programmable chip is adopted on USARTs. They are not programmable in the sense that a sequence of instructions can be executed by a processor, but that the current configuration from within the range of options is defined by internal control register settings. Further, as well as the data buffers, the status information is returned in an internal register. The control and status registers can be accessed through the data port as can the data buffers, a single pin on the chip set to 0 or 5 volt by the processor differentiating between data or control/status. A schematic diagram of a USART is shown in figure 2.33.

Line protocol chips. A number of programmable USRTs have also been developed which handle extra functions of synchronous line protocols such as SYN character detection for Bisynch and bit stuffing and stripping for HDLC/SDLC. Some of these parts will work at bit rates over a MHz. Other chips have been developed to support the USRT for such operations as calculating cyclic redundancy check (CRC) characters. The most sophisticated parts incorporate all these features in the one chip. The most complex interface component in the 1970s was the Zilog SI/O chip which could be programmed for asynch, bisynch, HDLC,

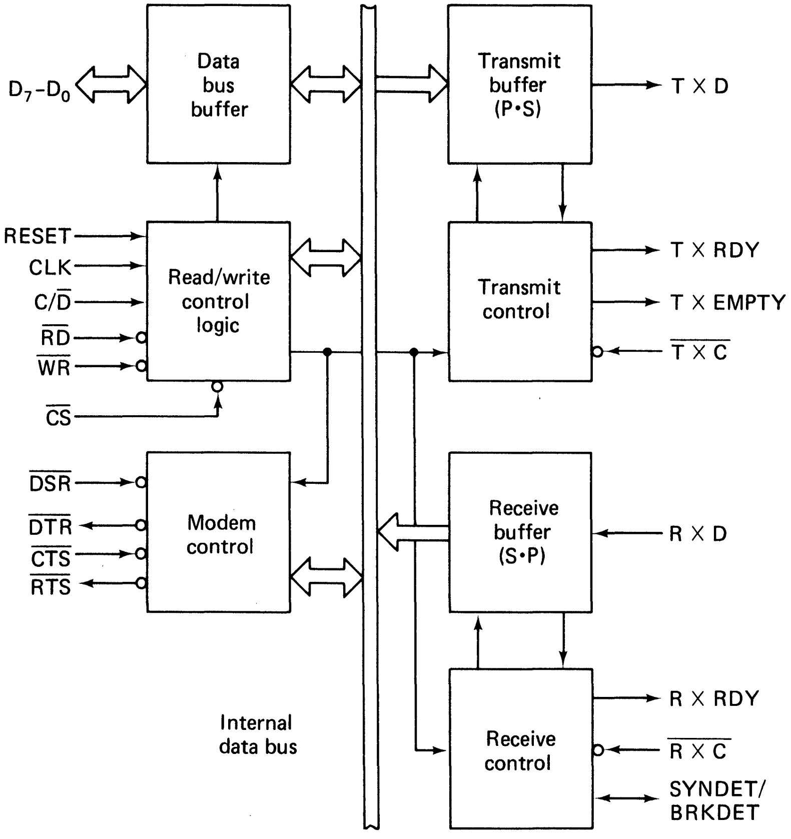


Figure 2.33
8251 programmable USART

or SDLC (including CRC calculations) and supported two independent channels plus modem control.

Baud rate generators. Various baud rates are used in communications, with multiples of the baud rates used on asynchronous UARTs. These are generated by dividing down a high-frequency crystal-controlled clock. Chips were produced whereby the division ratio could be selected by strapping pins to 0 or 5 volt. Nowadays programmable clock chips are available, the division ratio of which can be set by loading a control register in the chip. These chips usually support multiple dividers so that other timing markers can be independently generated from one crystal.

Drivers. The LSI parts work at TTL low power levels. To drive lines over any distance amplifiers are required. Often the voltage levels must also be changed, requiring additional power supply rails. Driver chips have now been developed which both amplify and convert signals from 0 to 5 volt to ±9 volt for V24 interfacing. Four lines per chip are common.

Other LSI Components

Other functional, regularly used circuits have been built onto LSI chips, some MOS but many in bipolar form. Typical parts are:

Parallel I/O. These range from simple 8-bit latches to programmable devices. The 8255, for example, has three sets of 8-bits which can be set by control registers to act as three ports. Two of the ports can be either input or output, while the third can be set to two 4-bit groups, either in or out.

Direct memory access. DMA requires independent maintenance of an autoincremented address register and an autodecremented word count register. The 8257 is an example of a support chip which provides four pairs of registers, set by I/O instructions via a data port, with four sets of DMA request and accept signals for peripheral control, plus logic for generating the CPU hold and hold-acknowledge handshake. Other DMA chips allow two channels to be set up for memory-to-memory string moves (most microprocessors feature only single-memory-resident operand addressing). Far more powerful is the Intel 8089, designed for use with the 8086. This chip is an I/O channel in the IBM S/370 vein. It is in fact an I/O processor, with dual channel DMA, with its own dedicated instruction set. It can be configured such that its program shares 8086 memory or executes from its own local memory. Unlike a simple DMA controller which allows the peripheral to place data onto the memory bus, the 8089 actually handles the data, hence providing the ability to process as well as route data streams.

Interrupt Controller. Microprocessor interrupt mechanisms are quite simple. These can be enhanced by external logic to provide priority and vector address generation, now available as a single chip.

Memory refresh. Dynamic RAM is cheaper than static, but must be periodically refreshed. To avoid putting the CPU into a hold state for the refresh period, logic is provided to refresh the memory during sub-cycles of the basic CPU cycle which are not used for normal memory access (the memory refresh is then transparent, even to DMA). The refresh control logic is available as a single chip. It is expected that self-refresh logic will be built into future generations of dynamic RAM (psuedo-static RAM).

Disc Controller. Most of the digital logic for a discette controller can be obtained on single chips. These devices are dedicated processors. They feature a number of internal registers for control and status, as usual accessible by the CPU via normal I/O operations. Serial/parallel data handling, header and tailer generation, CRC calculations, and other features are program-selectable to allow any format to be used. Timers and counters are maintained for mechanical control features such as stepper motor drive for head positioning and loading and sector identification. Control of multiple drives, some with overlapped seek on one disc with read/write on another, are available. Some parts provide some of the analog signal handling too, in particular data-separator logic for reliable separation of the data stream into binary form for various recording formats. Similar chips are appearing for control of bigger, faster hard discs.

CRT Controller. Much of the logic for repeatedly converting an array of ASCII characters into a corresponding set of dots, with all timing logic, has been built into an LSI part. The translation from a row of eighty ASCII characters into a set of, say, ten rows of dots for a raster scan, is done by a ROM-based translation table, which defines the character set displayed. The more complex chips also provide extra features for generating attributes such as cursor, underline, blink, various intensities, etc. The new parts are programmable so that the number of characters per line and other features can be varied.

Bit-slice microprocessors. Sets of high-speed bipolar LSI components are available to construct full-scale computer CPUs. Typical components are register sets and arithmetic units. These are typically 4-bit units; four are connected in parallel to construct a 16-bit mini. The control unit of these systems is microprogrammable so that the instruction set can be tailored and implemented in a specific ROM. The Intel 3000 used in the Codex 6000 is one of the originals, but the AMD 2900 series is the leader and is used in many modern minis (Honeywell Level 6). It will be interesting to see whether the cost-effectiveness of generations of single-chip micros with 16- and even 32-bit word lengths will make these devices obsolete. Some of the simpler MOS micros are implemented on three or four chips, one of which is the control ROM, which can be microprogrammed to suit. National Semiconductor's IMP 1600 is an early example but the 4-chip set Western Digital MCP-1600 used by DEC in the LSI-11 must be the best-known current implementation.

The fast LSI components can also be used in applications other than processors, modems being a good example, disc controllers another.

Special-purpose LSI chips. The cost of the micro is kept down by manufacturing volume. If a production run is sufficient a special-purpose chip may be possible, whereas three or four chips may have been needed using a micro. A typical example is the special chips used in modern multiplexing modems. As new techniques like uncommitted-logic-arrays progress, this approach will become more popular. Electrical instruments such as digital voltmeters are another area where special-purpose LSI dominates, rather than microprocessors. One-bit word length Boolean processors and field-programmable logic arrays (PLA) can also be used for applications that do not use arithmetic or character data such as process control with relay closure input data and solenoid activation output. PLAs are also used to replace groups of TTL logic in many computers.

Microcomputers. Devices incorporating the processor, a small RAM and ROM memory and a simple I/O port on one chip are already in use. These are intended for simple, bulk, dedicated applications such as washing machine controllers, but with a little advance will incorporate enough memory to be useful in terminal controllers and the like.

Peripherals and Interfaces

The very essence of a minicomputer is flexibility. There is a wide range of system software and hardware to choose from. The S100 bus microcomputers are even more flexible than minis in terms of hardware flexibility but are far more constrained in software features beyond single-user systems. Virtually any peripheral device can be interfaced to any mini or micro. There is no difference between the CDC disc drive used by, say, ICL or by Texas Instruments. However, the low cost and high volume of the mini, but more significantly of the micro, has led to the development of cheaper, simpler products such as floppy discettes, matrix printers, and eight-inch Winchester discs. There was a time when minicomputers were synonymous with small discs (the DEC RK05 at 2½ MB); now 10 MB discs are common on simple microcomputers.

To attach a peripheral device to a computer an interface is required. This will plug into the computer bus, communicating with the CPU and memory either by programmed I/O or by DMA as required. The interface must then pass data, control, and status information to the device controller, which in turn couples to the physical peripheral device. The interface and controller may be simple enough to be constructed on the one card; others require multiple cards. Often the non-computer-specific part of the controller will be housed in a separate case with a relatively simple interface card for a specific computer. In the simple category are serial communication interfaces, the UARTs or USARTs being mounted directly on the interface card with edge connected cables to sockets mounted on the rear of the computer case. Multiple V24 lines on single cards are common. A disc controller, on the other hand, is a common example of a multicard unit; many disc suppliers make universal controllers which link directly to a relatively simple interface card designed for a given range of computers.

The interface itself actually passes parallel words to and from the computer, translated to actions by the logic of the controller. Thus most mini and microcomputer manufacturers supply general-purpose parallel I/O cards. Each card supports multiple I/O ports which allow a general-purpose controller to be attached. Note, however, that software drivers in a specific operating system are designed to relate to detailed bit patterns in dedicated registers (two or three in a simple serial I/O card, but ten to twenty in a complex disc controller). Thus it is common practice for plug-compatible peripherals to be sold with controllers which are compatible with manufacturers' controllers for software commonality. A nonstandard controller and disc for a given mini may be cheap hardware but could require expensive software modification.

The trend for plug-in interfaces and interface/controllers has been taken up with microcomputers with particular emphasis on the S100 bus. The leading operating system (CP/M, described in chapter 7) is designed to be user tailored to any controller.

The range of peripheral devices available is all the standard industry peripherals. Terminals are of great importance, particularly VDUs, which are usually the asynchronous non-buffered type, relying on the mini's exceptional interrupt handling capability for direct support. It is the ability of the mini to provide cheap attachment of cheap terminals that is a major attraction. Terminals are so important to this text that they are covered in a broader context in chapter 3.

Matrix printers and line printers up to about 400 lpm are common on minis. Printers from any manufacturer are easy to support, the simpler ones using V24 serial interfaces, the faster ones a dedicated parallel interface/controller.

Floppy discs are more common on micros than minis, ranging from the cheap five and a quarter-inch 90KB drives to high density eight-inch double-sided drives of over a Meggabyte. IBM compatibility is most useful. Cartridge discs, often the one fixed, one exchangeable (FED) type of around 10 MB capacity were the norm in the 1970s, but eight-inch Winchester disc technology and storage module drives (SMD) provide discs from 10 to 300 MB. The bigger drives use multi-surface disc packs. Fixed-head discs are still used in some cases for fast swapping (no head movement latency) but these are being replaced by bigger RAM memories, either in the main address space or by fixed-head disc emulators. CCD and magnetic bubble stores will affect this area.

Industry standard (half-inch) magnetic tape is common for back-up, but not for processing. A large capacity disc and a tape is a common mini-based business computer configuration. The fixed nature of many of the eight-inch Winchester drives has led to the development of high capacity cartridge tape (quarter-inch) drives for back-up systems.

Any card or paper tape equipment or specials like facsimile (FAX) scanner/printers, or computer-output-microfilm (COM) can be easily attached, possibly using a general-purpose parallel interface card.

Devices required for connecting the computer to the physical world fall into two categories, digital and analog. Both input and output devices are needed. Digital I/O subsystems are either single-digit (lighting lamps, closing switches) or multibit (coded data such as direct input from a keyboard). Analog signals are converted to an equivalent binary representation by sampling and using an analog-to-digital converter (ADC). A digital-to-analog converter (DAC) is used to set a hold circuit to produce a proportional analog voltage output. Multichannel signals are multiplexed, with the more sophisticated devices under program control from the computer (channel select, repetition rate, and amplifier gain). Signal processing is a highly specialized topic beyond the scope of this book.

3

Data Transmission and Terminals

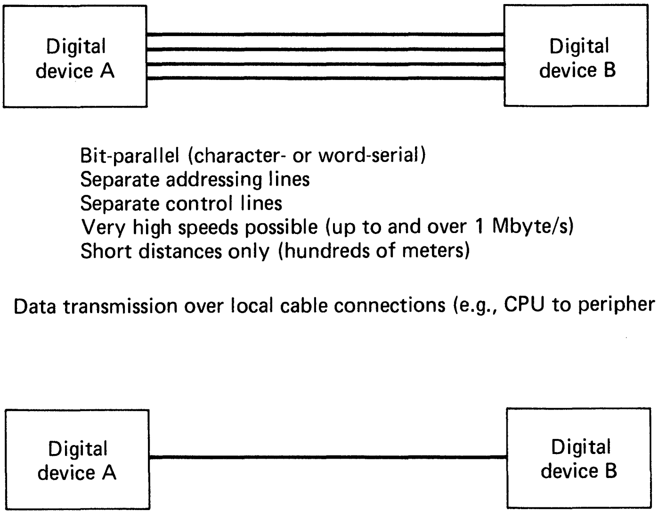
The design of teleprocessing systems is primarily concerned with providing a more effective interface between computers and their end users. The main vehicle for achieving this is the interactive terminal, which may be installed at the user's place of work and connected on a permanent or occasional basis to the computer. The computer itself may be in the same building as the user or on another continent. Clearly, some facility for enabling the two devices to transmit data from one to the other is essential, but the data communications aspect of teleprocessing systems needs to be kept in context. For many organizations the ability to move data rapidly and conveniently over long distances is beneficial in its own right but it is in the ability of the user to interact directly with the computer that so much potential lies. This chapter, then, is concerned with reviewing the basic concepts and techniques of data transmission. It also describes the main types of terminal available. For a more detailed treatment of this subject, the reader should turn to the books listed as references. If you are already familiar with the subject it still may advisable to skim the chapter to be sure that our terminology agrees with yours.

Data Transmission and Distance

All computer systems involve some element of data transmission. Traditional batch systems involve the movement of data between various system components over multiwire cables. Such techniques are highly efficient. However, the cost of such cables usually bodes against their use over distances of more than a few hundred meters at best. A further complication arises when it is necessary to move data from one building to another. If this requires transmission over land not owned by the company concerned, then in most countries, there may be a legal requirement to make use of transmission facilities provided by a government-licensed supplier of telecommunications services.

Once distance is a factor in data transmission, the characteristics of the facilities available become markedly different. Figure 3.1 compares the main features of transmission over short-distance cable connections and over telecommunications links. Transmission over local cables usually involves sending data character by character (or word by word) in a bit-parallel fashion with separate leads for addressing and control functions. The speed of transmission on such cables can be very high and with extremely low error rates (usually caused by hardware malfunction in the channel-driver logic).

In contrast, data on telecommunications circuits is transmitted in bit-serial form and addresses and control characters need to be embedded in the data stream. Transmission speeds are low in computer terms, with data rates in excess of 9600 bits per second being relatively expensive, and sometimes difficult to obtain. Furthermore, telecommunications circuits are engineered to lower stan-



Bit-parallel (character- or word-serial)
Separate addressing lines
Separate control lines
Very high speeds possible (up to and over 1 Mbyte/s)
Short distances only (hundreds of meters)

(a) Data transmission over local cable connections (e.g., CPU to peripheral)

Bit-serial
Addressing characters embedded in data stream
Control characters embedded in data stream
Only low speeds readily available (up to 9600 bits/s)
Relatively high error rates
Transmission over long distances possible
(thousands of kilometers)

(b) Data transmission over telecommunications links

Figure 3.1
Data transmission characteristics

dards than most computer equipment and error rates are high enough to require the use of special techniques to detect and correct errors through intermittent data loss. This problem of error rates and error handling is discussed later in the chapter.

Telecommunications Services

With a few exceptions, companies wishing to employ data transmission techniques have to obtain the circuits required from the government-approved supplier of telecommunications services to the community at large. In most countries the only services available for data use are the telephone and telegraph networks. As these systems were not designed for the job of data transmission, their use by computers tends to pose many problems to both the user and the supplier. Figure 3.2 shows the main classes of service available. Each of these—telegraph, telephone and, where available, data—can be subdivided into *dedicated* and *switched* facilities. There are few people left in the technologically advanced parts of the world who are not familiar with the dial-up telephone system. Similarly, in the business community the dial-up (or switched) telegraph system, usually called Telex, has become a feature of everyday life, especially in the face of a general deterioration in postal services.

Both telephone and telegraph circuits can be obtained in the form of permanent connections between two locations. Such links use the same circuits as dial-up links, but all the switching exchanges are bypassed. Such dedicated lines

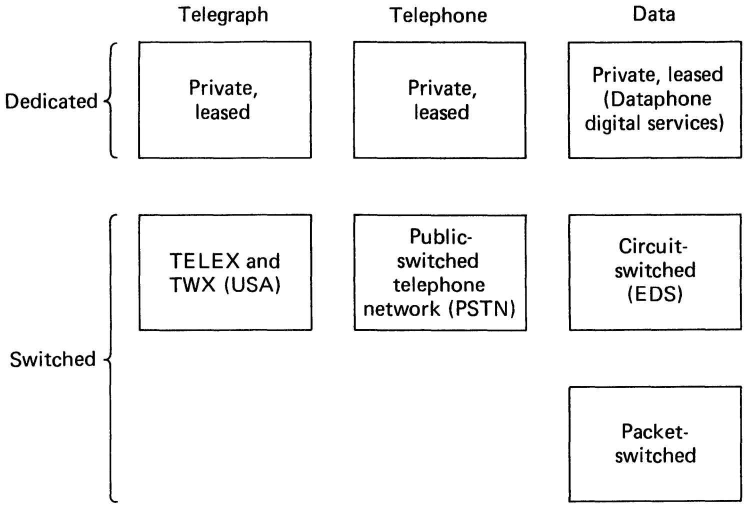


Figure 3.2
Data transmission services

are leased from the supplier in return for an annual charge based on circuit length and quality rather than usage.

In the 1980s it is expected that there will be extensive developments in the area of public data networks. It may seem fatuous to observe that the telephone and telegraph networks were not designed for use by computers, but it is important for the system designer to keep this in mind—it is the major constraint on the performance and availability of such circuits. In order to overcome such constraints (including the time to establish calls on switched circuits) authorities are actively pursuing the installation of services offering the wider range of transmission rates, faster connect times, and higher reliability demanded by computer-based systems. New data networks sometimes involve the exploitation of new technology (for example, digital transmission) but more often, as a contingency, use new techniques (notably packet switching) based on existing analog communications facilities. The characteristics of these three types of telecommunications services (telegraph, telephone, and data) will be described later in this chapter.

Suppliers of Telecommunications Services

Any discussion of the organizations involved in the supply of telecommunications services tends to require separate consideration of the United States and the rest of the world. In the United States, suppliers of telecommunications facilities are generally known as common carriers. These are privately owned and operated companies licensed by the government through the agency of the Federal Communications Commission. Although there are nearly 2000 such common carriers the market is dominated by the massive American Telephone and Telegraph Company (AT&T), which operates over 80 percent of all installed telephones. The next largest telephone company, General Telephone and Electric, holds a mere 7.8 percent of the marketplace.

The telephone network controlled by AT&T is known as the Bell System and is operated by twenty-three wholly or partially-owned regional subsidiaries. In general, the services supplied by non-AT&T companies (known generically as the Independents) are fully compatible with the Bell System, thereby making it practical to make intercarrier calls. Switched telegraph services comprise the internationally-standard TELEX service and the Teletypewriter Exchange (TWX) network. Both of these are operated by the Western Union Telegraph Company.

In the 1970s the United States has experienced the growth of specialized common carriers of which there are two types. The earliest, typified by Microwave Communications Inc., developed and installed their own networks. These were designed primarily to provide wide bandwidth channels in direct competition with existing Bell intercity trunk services. The other type of specialized common carrier made use of existing telecommunications circuits (usually leased from Bell) and configured them into networks suitable for data transmission by adding computer-controlled switches. These are known as value added carriers, and the most well established in the late 1970s were TELENET and TYMNET.

In response to this competition for such a potentially lucrative sector of the marketplace, AT&T developed the Dataphone Digital Service (DDS). Dataphone is the brand name under which Bell provides data communications services based on the existing telephone system. DDS, however, employs digital transmission technology to provide leased circuits between most major American cities.

It now seems that the monoply position of AT&T in the United States will be somewhat weakened (at least for data transmission users) over the next decade. With the exception of Canada, which has a telecommunications supply industry similar in structure to that of the United States, the rest of the world is likely to have to continue to live under a monopoly system, because in most countries such monopolies are both exclusive and government controlled. Such telecommunications authorities usually embrace postal services as well as telephone and telegraph systems. Post, telegraph, and telephone administrations (PTTs) are either government departments (as in France) or public corporations (as in the United Kingdom). Strictly speaking if the administration is *not* a government department it should be referred to as a Recognized Private Operating Agency (RPOA), but as this would embrace both AT&T and the British Post Office most telecommunications specialists tend to use PTT to refer to *all* government-owned administrations.

Before embarking on the development of a communications-based system it is important to contact the relevant administrations of the countries in which the system will be based and obtain details of the services offered, the prevailing tariff, and the regulations controlling the use of the facilities provided. The regulations in particular vary from country to country, especially with respect to the devices that can be attached to circuits. Some PTTs insist that certain attachments be supplied only by themselves, others provide nothing beyond the circuit terminations, and still others labor under a confusion of policies between these extremes.

In the case of international connections, it is necessary to deal with the PTT that will be responsible for one end of each circuit. For example, either the Deutsche Bundespost or the Italian PTT will undertake to provide a circuit between Frankfurt and Rome. Life gets complicated, however, when the user company is headquartered in Frankfurt but also wishes to install a circuit between London and Paris. In this case the Bundespost will not be able to help and it will be necessary to contact the French PTT or the United Kingdom Post Office. As such multinational networks expand in scope, the problems of development and management become more complex, often requiring the appointment of a full-time communications manager. International communications is, if anything, even more fraught with regulatory and political issues than domestic systems. For example, it is not currently permitted to transmit data on dialed calls between Europe and the United States. All data traffic into and out of the United States has to be handled through one of three approved record carriers: ITT, RCA Global, or Western Union International. In some parts of the world where there is political friction between neighboring countries it may be necessary to route lines through a neutral country. Considering the scope and complexity of telecommunications systems worldwide there is a surprising level of standardization, a situation mainly attributable to the work of the International Telecommunications Union (ITU), an agency of the United Nations.

The ITU was established as long ago as 1863 and currently has 124 nation-members and a secretariat and conference center based in Geneva. The agency comprises the International Frequency Registration Board (IFRB), which allocates and records radio frequencies; the International Consultative Committee on Radio (CCIR), which approves standards relating to the use of radio communications; and the International Consultative Committee on Telegraphy and Telephony (CCITT), which develops standards for use in telecommunications systems. CCITT has no power over members, and standards are therefore formulated as recommendations, some of which will be referred to later in this chapter. The recommendations are developed on an ongoing basis by various study groups, working parties, and *rapporteurs* and submitted for approval to plenary sessions held every four years.

Until the 1960s international telecommunications was carried out via cables (some of them submarine) and limited-capacity radio links. This changed with the launching of Early Bird and the advent of satellite communications. The International Telecommunications Satellite Organization (INTELSAT) was established in 1964 with a membership of eleven countries, which by 1974 had grown to eighty-five. Although new countries can join the organization, INTELSAT tends to be dominated by an American company, Communications Satellite Corporation (COMSAT), which must own not less than 50.6 percent of the international body. INTELSAT's main objective is to develop satellite communications, and COMSAT's main role in this is the development, launching, and operation of new satellites. Currently in use are INTELSAT I (Early Bird), II, III, and IV. Another recently established (1959) body is the European Conference of Postal and Telecommunications Administrations (CEPT). CEPT is a club of some twenty-six European PTTs and, although much of its work in the area of policy and tariff structures affect the data communications user, it is not an open body like the CCITT. CEPT does not in any way displace CCITT, but contributes to it and works in parallel with it to implement recommendations in Europe.

A further group that develops standards relating to data communications is the International Standards Organization (ISO). The ISO (and its various national members such as the American National Standards Institute and the British Standards Institute) has been particularly concerned with the development of link protocols (procedures for using telecommunications facilities) and codes. Another international body with a significant impact on the way in which we transmit data is IBM. The size of IBM's share of the world computer market is so large that this corporation tends to impose de facto standards in this field.

Circuit Arrangements

The circuits provided for the transmission of data may be characterized in a number of ways:

—in-house/in-company, or
—common-carrier/PTT supplied

—switched/dialed or
—leased/dedicated/private

—simplex
—half duplex or
—full duplex

—two-wire circuits or
—four-wire circuits

—point-to-point/two-point or
—multipoint/multidrop

It is necessary to obtain circuits from a common carrier or a PTT only when data is going to be transported over land not owned by the user. (The restriction may also apply to the transmission of data between two companies, even if their land is adjacent.) Within the confines of a single building or site an organization may install its own telecommunications lines. (In some cases the common carrier might be prepared to undertake the design, commissioning, and servicing of such in-house facilities.) Traditionally the characteristics of in-company circuits are essentially the same as those provided by PTTs, but new techniques based on loops, rings, or busses are being introduced, as explained later.

Each of the three main classes of service—telegraph, telephone, and data—are obtainable on a switched or leased basis. Leased lines are sometimes called dedicated or private circuits and are nonswitchable connections between fixed customer locations. The user pays an annual rental charge (usually related to the length and quality of the line) regardless of how much or how little the connection is used. Dedicated lines bypass switching centers (but use the same local and trunk circuits), thereby providing a higher level of quality. A further advantage of leased lines is that the circuit is instantly available for use. In the case of switched lines, it may take up to 60 seconds for the connection to be established (either by dialing or automatically). Telegraph and telephone systems are switched through a series of local and regional exchanges. Figure 3.3 illustrates the general structure of a switched telephone network. British and (in parentheses) American terminology is employed. Equipment installed on customer premises (which may vary from a single telephone instrument up to a private branch exchange) is connected via one or more two-wire local ends (local loops) to a local exchange (central office). Calls between nearby subscribers are switched in the local exchange. If the number of calls justifies it, neighboring local exchanges may be intercon-

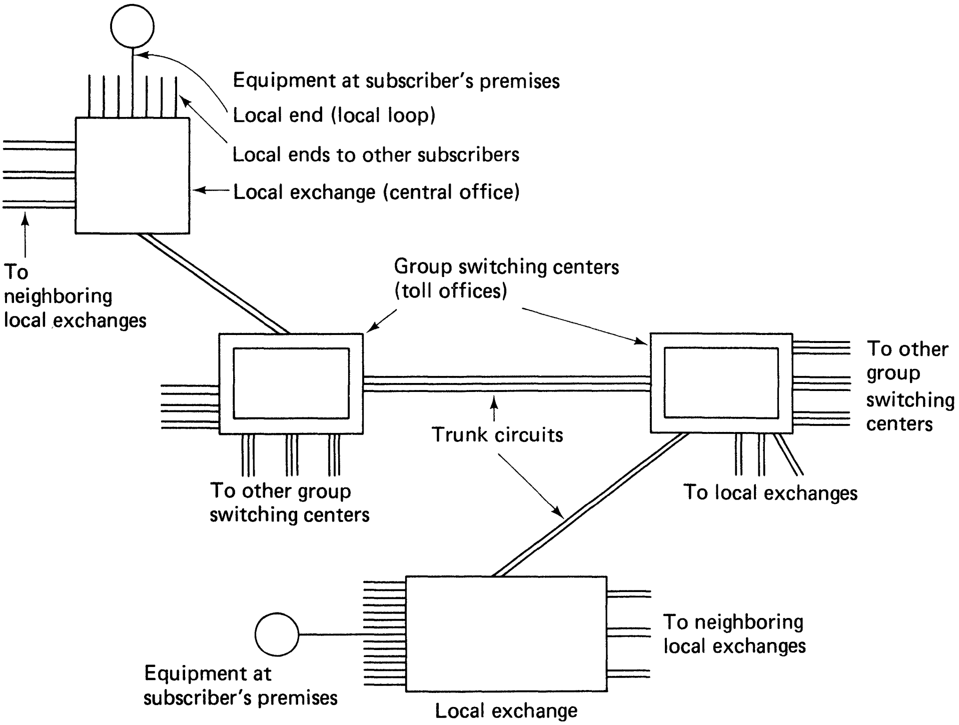

Equipment at subscriber's premises
Local end (local loop)
Local ends to other subscribers
Local exchange (central office)
To neighboring local exchanges
Group switching centers (toll offices)
To other group switching centers
Trunk circuits
To other group switching centers
To local exchanges
To neighboring local exchanges
Equipment at subscriber's premises
Local exchange

Figure 3.3
General schematic of a public switched telephone network

nected so that calls between customers attached to each exchange can be established without going up to the next level of the network.

If a long distance call needs to be made, the subscriber precedes the number with the appropriate area code (STD code in the United Kingdom). This causes the local exchange to route the call on to the next level of the system, a group switching center (toll office). The call set-up may then route down to the relevant local exchange or onward to other exchanges until it reaches the called subscriber. When the caller replaces his handset, the call will clear down through each exchange, releasing the intermediate circuits for other calls. There is almost always a degree of alternate routing between exchanges so that secondary paths are available in case of circuit failure or congestion. Switched telegraph networks operate in a similar fashion, sometimes using special plant, in other cases sharing plant with the telephone system. TELEX, for example, tends to be a physically separate network, but TWX in the United States relies on telephone switches and circuits. Although some data networks have been conceived to operate separately from the telephone system, it is more likely in the long term that circuit-switched data services will be integrated with new digital speech networks employing computer-controlled exchanges.

Another technique for providing switched data services is *packet switching*. This is so radically different from traditional circuit switching that it caused considerable consternation among the PTTs when it was first proposed. However, the private networks that pioneered the technique have been so successful that many administrations in Europe have announced plans for packet-switched public data networks, and the United States already has the first such service from a common carrier in TELENET. The CCITT has produced a recommendation, X25, which sets out standards for user access to packet-switched networks. Circuits are provided with differing arrangements for the direction of transmission. These arrangements can have a significant influence on the performance of the circuit. Three schemes are possible, as illustrated in figure 3.4. *Simplex* transmission (one direction only) is rarely used in teleprocessing systems. This is because of the need for the receiving device to be able to acknowledge the arrival of a message to the transmitting device. *Half duplex* working is a very common arrangement that enables data to be transmitted in either direction but not simultaneously. On the other hand, *full duplex* circuits enable data to be transmitted simultaneously in both directions. (It should be noted that CCITT definitions of these terms are different, but because the definitions used here are the ones most universally accepted they will be used throughout the book.)

The local ends in telephone systems consist of two-wire loops. Speech circuits at the next level of the network (interexchange) tend to be four-wire links, one pair for each direction of transmission. On modern systems employing high-capacity coaxial cable and microwave links this four-wire arrangement tends to be functional only. The PTT or common carrier will, however, provide a four-wire arrangement end-to-end, thereby giving a full duplex capability. Full duplex working can also be achieved on two-wire links by multiplexing, a technique described later in this chapter. Switched circuits are always two-wire, but leased lines can be two or four, at the customer's option.

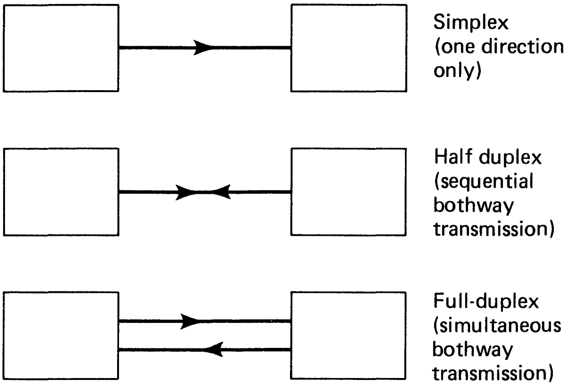
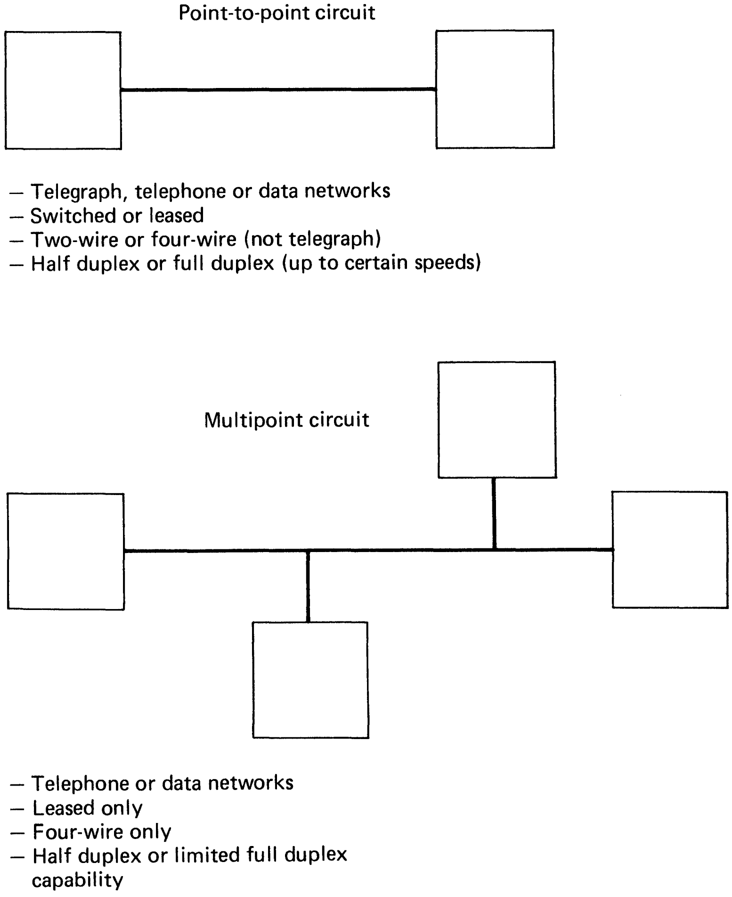


Figure 3.4
Directions of transmission

A dialed call establishes a simple point-to-point connection (a two-party circuit). Although computer-based (or stored program controlled) exchanges make it possible to set up a conference call with three or more users sharing a line, it is not clear whether data transmission systems are likely to use such techniques. The simplest form of leased line is point to point (or two point), but most PTTs and common carriers are able to provice multipoint (or multidrop) private circuits whereby a number of subscriber locations (usually up to twelve) may share a single facility in order to reduce costs. This is illustrated in figure 3.5.



- Telegraph, telephone or data networks
- Switched or leased
- Two-wire or four-wire (not telegraph)
- Half duplex or full duplex (up to certain speeds)

- Telephone or data networks
- Leased only
- Four-wire only
- Half duplex or limited full duplex capability

Figure 3.5
Two-point and multipoint circuits

Circuit Characteristics

This section deals with the characteristics of public telegraph, telephone, and data networks and discusses some of the problems associated with attaching digital devices for data transmission.

Electrical Interfaces

Electronic circuits used in computers work on TTL voltage levels; nominally logical 0 is represented by 0 volts and logical 1 by 5 volts. TTL circuits do not provide sufficient power for other than short distance transmission of signals, and in any case 0 volts could be a broken wire rather than logic 0. Power amplifiers are used to drive TTL-compatible signals over lines of up to ten meters or so, typically for, say, a parallel printer interface. Thus the TTL voltage levels are nearly always converted to a more suitable signal just before the plug on the equipment. The most common signals used are those specified for RS232/V24 interfaces, namely between +6 and +15 volts for logic 1 and between −6 and −15 volts for logic 0; 0 volts is now an idle period, not logic 0. An earlier standard used to connect Teletypes, which used mechanical relay contacts, not electronics, was the current loop principle; logic 1 forced a current of 20 (or 60) milliamp down the wire loop and logic 0 a null current. The voltage across the line for, say, 20 mA was small so that lines longer than those possible with V24 could be used, and so they are still occasionally in use for electronic asynchronous terminals. Telegraph signals for transmission over long-distance public wires use ±80 volts at slow speeds. For longer distances than a computer to a local terminal, some form of modulation must be used to retain undistorted data at the receiving end. For public transmission (adopted for many long in-house runs), a modem is used as described next. Less common methods adopted in-house use coaxial (or video) cables with special modulation techniques such as the pulse-width modulation at 1 M bps used by IBM to interconnect 3277 VDUs and the 3271 controller of the 3270 cluster system. Manchester encoding is another technique occasionally encountered and others will be introduced with the new in-house networks such as Ethernet. It must be stressed that pre-defined standard modulation techniques must be used on public networks for commonality, although unfortunately American and European modulation techniques are sometimes incompatible.

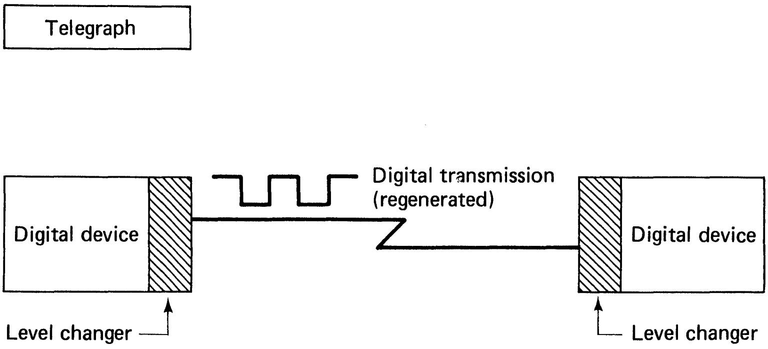
Telegraph Circuits

At first sight it would seem that telegraph networks are eminently suited to the transmission of data. Within computer components data is both recorded and manipulated in digital form and bits are transmitted in the same fashion on telegraph circuits. The only significant difference is in the signaling levels employed. Interfacing to a simple leased circuit can, therefore, be achieved with the aid of a simple level changer that converts 0/5 volt TTL internal signals to ± 80-volt signals for the line and vice versa (see figure 3.6). Current loop or RS-232 are often used in-house.

The disadvantage of using DC signaling on the communications link is that distortion can impose limitations on transmission rates. The nice, square shape of the pulse that leaves the adapter is subject to distortion caused by the capacitance of the line. The greater the distance transmitted, the more severe is the distortion until finally the signal cannot be reliably interpreted as '0' or '1'. This problem is dealt with in telegraph systems through the use of regenerative repeaters. These repeaters interpret a signal on the line and create a fresh, clean version of it for onward transmission.

In essence the problem is this: the higher the transmission rate required, the more repeaters are needed and the higher the cost of the network. As teleprinters (teletypewriters) need operate only at very low speeds, most telegraph networks operate in the range of 50 to 200 bits per second. TELEX, for example, is a 50-bps service and TWX 150 bps. Now for telegraph systems to become a practical medium for general-purpose data communications, a much higher range of speeds will be required. But for common carriers to achieve this would require heavy additional costs (for the extra repeaters) which would have to be passed on, in part, to the majority of users, who are not going to benefit from the enhancement.

In spite of the speed restriction, switched and leased telegraph services are occasionally used in computer systems as a low-cost method of data transmission. DC signaling techniques are also used fairly extensively on an in-house basis. In



- Point-to-point
- Half duplex
- Asynchronous timing
- Public network speeds to 110/200 bits/s
- In-house limited distance speeds to 9600 bits/s (20 or 60 mA current loop)

Figure 3.6
Data transmission on telegraph circuits

this case, using RS232 ±12 volt signals, transmission rates of up to 9600 bps are possible so long as the cable length is kept below about 200 meters. For longer distances, repeaters may have to be used or, alternatively, slower speeds accepted.

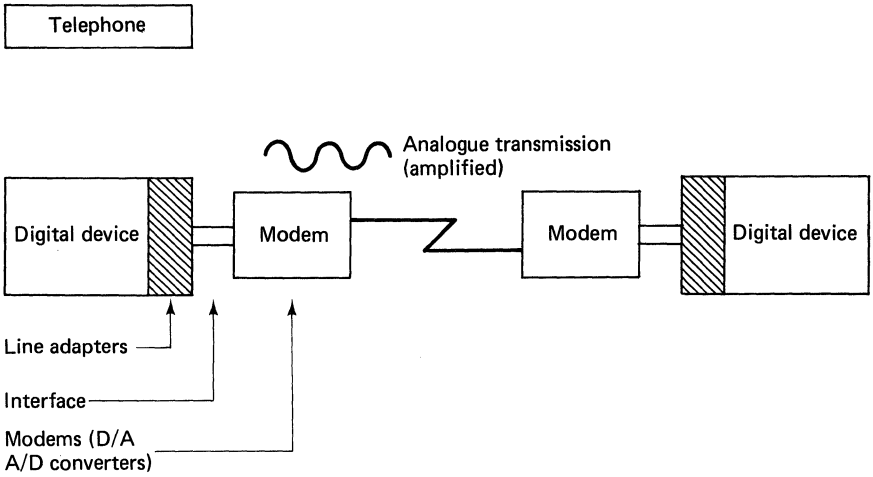
Telephone Circuits

The major difference between telegraph and telephone circuits is that the latter employ analog (AC) signaling techniques. In other words, human speech is transmitted as an electrical analogy of the sound waves made by the voice. Clearly, this method is markedly different from DC digital signaling used in computer components and, as you would expect, the cost and complexity of interfacing increases accordingly. The key to transmitting data on analog telephone lines is a device called a *modem*. In the United States, modems are sometimes called data sets (to distinguish them from speech sets), but as this expression is also used for computer data files, we shall stick to modem. Modems work on the principle of transmitting an AC signal having a certain reference frequency and amplitude (within the frequency range of 300 to 3400 hertz or cycles per second usually carried by speech channels).

Seen graphically (perhaps on an oscilloscope) the signal would be represented as a sine wave. The convention in data communications is that at a given moment in time this carrier signal will represent a bit value of 1. A continuous, unchanging carrier will therefore represent a whole string of 1-bits. The bits are passed to the modem from the transmitting device (perhaps a computer or a terminal) over an interface (figure 3.7). If the transmitter wishes to send a 0-bit, then clearly the modem needs to change the signal in some way so that the receiving modem (which is passing the incoming stream of ones to the sink device) can generate a digital 0 condition at the interface.

This process of changing the carrier signal is known as modulation, and the reverse process demodulation. As most systems require transmission in both directions, one box is designed to perform both functions—modulate/demodulate, hence modem. Three characteristics of the signal can be changed to indicate the occurrence of a 0-bit: amplitude (loudness), frequency (pitch), and phase (timing). A simple modem may send a carrier of 1650 Hz to indicate 1 and increase this to 1850 Hz for a 0. (Clearly, a timing discipline needs to be imposed on this arrangement so that both the transmitting and receiving ends are in step. This will be considered later.) There are about half a dozen modulation techniques in common use and some of them modulate to multiple levels so that 2 or 4 bits can be sent in a single time interval; for the interested reader these techniques are described in more detail in the references.

For most practical purposes, modems come in two types. The precise meaning of the terms asynchronous and synchronous will be explained in due course. Asynchronous modems are relatively low-cost devices suitable for low-speed transmission. The modem may be connected directly to the telephone local end or operate acoustically through a telephone handset. Acoustic couplers work



- Point-to-point/multipoint
- Half duplex/full duplex
- Asynchronous/synchronous timing
- Speeds to 9600 bits/s on 4-wire circuits
- Speeds to 240 Kbits/s on wideband circuits

Figure 3.7
Data transmission on telephone circuits

because the operating frequencies of the modem must be in the 300–3400 Hz range mentioned previously. This means that the signals must be able to go through the microphone and earpiece of a regular handset, thereby giving the terminal a high degree of portability. The coupler could be a freestanding unit or be built into the terminal itself. The modem at the computer end of the (dialed) circuit would normally be directly connected. On analog telephone networks it is unusual to find an acoustic coupler that will work reliably at speeds greater than 300 bps. Directly connected asynchronous modems can operate on the public network at speeds up to 1800 bps, but in most countries the maximum is 1200 bps (see figure 3.8).

Synchronous modems must be hardwired to the line, but for a much higher cost are able to operate at speeds of up to 9600 bps on a single four-wire circuit. The best speed possible on a dialed (two-wire) line is 4800 bps, half duplex. To improve on this it is necessary to dial two lines, which the modem then uses as a four-wire full duplex link. Synchronous modems vary mainly in the way in which they handle line distortion. This can be done automatically, semiautomatically, or not at all. In the latter case, the circuit will need to be preconditioned by the common carrier to a standard suitable for the proposed speed of transmission. These modems can also incorporate test features. The best way of learning about them is to obtain literature from a few suppliers.

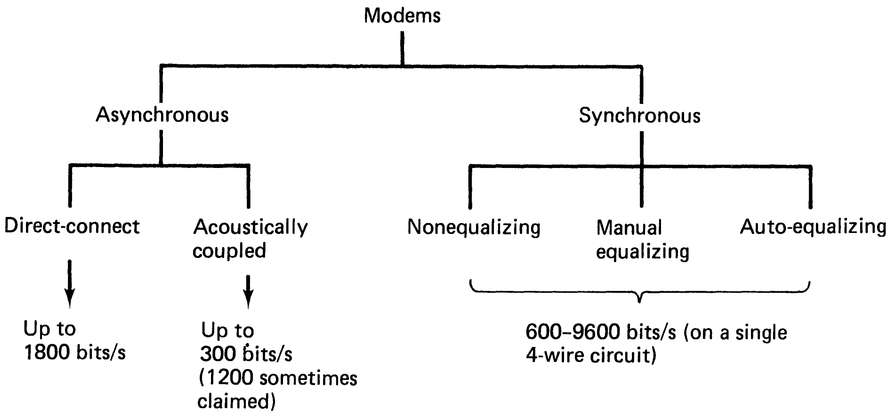


Figure 3.8
Types of modem

Before leaving the subject of data transmission on telephone lines, it should be noted that most PTTs and common carriers in the more developed nations of the world have plans to develop new speech networks using digital transmission technology. In some places this development has already started and entails the use of both the transmission of digitized speech and the use of computer-controlled telephone exchanges. This will ease the problems of data communications, improving the performance of circuits and providing more services and facilities. Indeed, PTTs have generally been careful to allow for the needs of the data user in the design of such new systems.

Data Networks

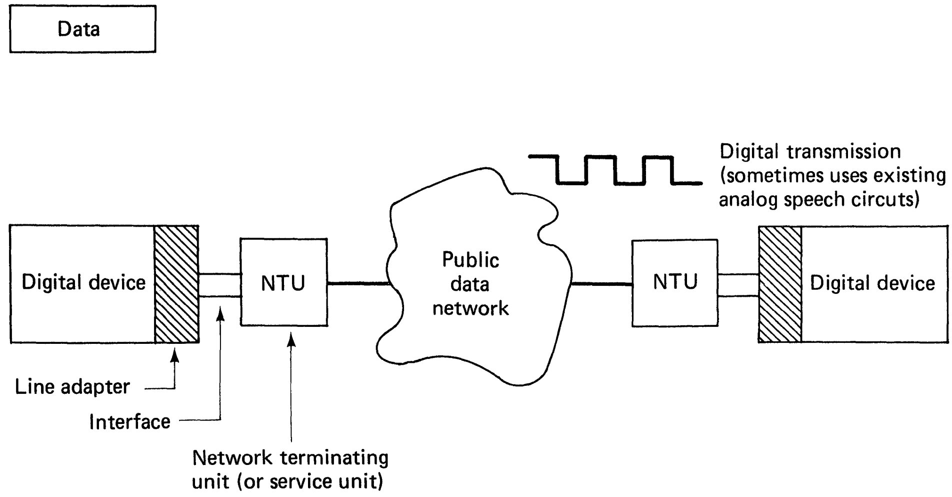
Few countries have been prepared to wait for the enormous task of changing their complete telephone systems to a new technology before attempting to provide commerce and industry with improved facilities for transporting data. In the United States, AT&T already offers leased lines under the Dataphone Digital Service, in the Federal Republic of Germany the Bundespost provides fast circuit-switched services called EDS, and yet other countries have packet-switched system. In spite of attempts by the CCITT to standardize the services and facilities to be offered by telecommunications administrations it does seem likely that, although the interface to data networks will be widely accepted, the features of data networks will vary considerably. The reader is advised to keep abreast of plans and progress in his own country.

Of the services currently being considered, the least likely to succeed as an independent data network in the medium term is circuit switching. This will probably be done (as it is now) on the telephone system. Packet switching seems to be a much more cost-effective approach to data transmission for both the user and the common carrier. There will, however, be a significant body of users, especially those who have developed their own private packet-switched networks, who will resist the move to public service and perpetuate a requirement for leased lines.

At present (early 1980) it is difficult to generalize on the characteristics of data links. With the exception of services such as Bell's DDS most data networks make use of existing speech circuits and internal network links. This means that users interface through orthodox modems. As such networks gradually move over to digital circuits the modem will be replaced by much simple and cheaper network terminating units (NTU—see figure 3.9). The best example of these currently installed is the service unit needed for the DDS. The interface between the terminal equipment and the NTU will be much simpler than the one required for modems. These interfaces will be described and compared later in the chapter.

In-House Networks

The distances involved with in-house interconnections are relatively short so that they have far more scope than public services. Traditionally networks have been very hierarchically structured, with terminals sited around a building linked in a


Data
Digital transmission (sometimes uses existing analog speech circuts)
Digital device
NTU
Public data network
NTU
Digital device
Line adapter
Interface
Network terminating unit (or service unit)

– Switched point-to-point
– Leased point-to-point (possibly multipoint)
– Packet-switched
– Full duplex (half duplex optional)
– Synchronous timing (possible asynchronous option)
– Speeds: 600, 2400, 4800, 9600, 48 Kbits/s, 50 Kbits/s

Figure 3.9
Data transmission on digital data networks

star topology by individual cables to a central computer, with occasional use of multiplexors or cluster controllers. Occasionally, as with IBM, coaxial cables are used, but usually direct V24 signalling will cope with the distances involved using multi-core cable. However, with the increasing desire to integrate computer terminals with word processors, facsimile transmission, photo copiers, and the like, a more flexible general-purpose structure is desired. Preferably a general-purpose network should allow terminals and processors of any mix to plug directly into the network without upsetting other devices already on the network.

Three techniques are evolving: *loops, rings,* and *buses*. All form data into packets and transmit the data to all possible recipients with an address header. The target device detects its own address and accepts the data packet. The loop (e.g. the IBM 8100 system) requires one device to be the controller. The ring (Prime-NET, Cambridge ring) is a variant where all devices can be master. Each node must store at least one bit before retransmitting on the ring to enable the address match to commence and if needed to strip the message off the ring. Since transmission between adjacent nodes is effectively point-to-point, efficient high speed transmission can be used with standard protocols (SDLC, for instance, on the IBM 8100). On the other hand a bus is a common coaxial cable to which *all* devices are directly attached. Any one can become master and place an addressed packet on the bus. All devices are normally in a listen mode so that the addressed devices accept the packet. This is termed a *broadcast*. For a device to determine whether it can become master, it listens to see that the bus is quiet. If the bus is not quiet, it will not transmit. However, contention can occur when two devices wish to transmit simultaneously. To resolve this a sending device reads its own transmitted message off the line and compares it with the original, retrying when it mismatches. Only one packet can be on a bus at once, while multiple packets can be on a ring at the same time, giving an effectively higher bandwidth. However, bit rates on buses are over 4 MHz on coax cable and will go to hundreds of MHz in future with fiber optics.

Noise, Distortion and Error Rates

In our comparison of parallel cable connection with bit-serial telecommunications circuits, one of the major differences was that of error rate. Telephone lines are engineered to a lower level than the cables used to connect (for example) a CPU to a disc drive. This situation is largely acceptable on voice links because of the redundancy of human speech. However, the system designer must consider the communications components to be inherently unreliable. In other words, the system has to be designed on the assumption that failures will occur; data will be lost and circuits will fail for short or long periods of time. The handling of errors on communications lines is a function of the error detection and correction (EDC) features of the link control (protocol) procedures described later. In this section we shall concern ourselves with the source and magnitude of such errors.

Noise is a completely different phenomenon from distortion. Distortion is a predictable property of any communications channel. For a detailed description of the main types of distortion, the interested reader is referred to the books listed at the end of this book. Distortion can be dealt with in one of two ways. The common carrier can "condition" the circuit by inserting components that compensate for the effect of distortion. Conditioning also involves the minimization of noise levels. A switched circuit is always unconditioned. A leased line may be supplied unconditioned or, for a higher rental, conditioned to one of a number of specified levels. AT&T, for example, offers two types of conditioning for speech lines. C conditioning reduces the distortion according to five standards—Cl to C5—on a variety of two-point and multipoint circuit configurations. D conditioning is used to reduce signal-to-noise ratios on circuits to be used for 9600 bps transmission. Lines can have both C and D conditioning as required. As mentioned earlier, the more sophisticated synchronous modems are available with built-in facilities for automatically (or semiautomatically) compensating for the effects of distortion. These can, to an extent, supersede the benefits of C conditioning, and one aspect of network design is to explore the payoffs between lower line rentals and more expensive modems.

Noise, on the other hand, is a less predictable characteristic of the circuit being used and the network of which it is a part. Unlike distortion, noise can only be measured in statistical terms. The three main categories of noise are thermal noise, impulse noise, and induction and crosstalk. Thermal noise is a continuous background hiss that can be minimized but never eliminated. Induced currents are generally picked up from power supply cables, usually in electrically noisy environments (near elevator shafts). Crosstalk is spillage from adjacent circuits in the communications network. But the major problem, particularly on dialed circuits, is impulse noise. Short, random bursts of noise cause corruption of any data blocks being transmitted at the time.

The CCITT has published a recommendation (V53) for the maximum error rates that should be experienced on leased and switched circuits at a variety of signaling rates. In the table below the modulation rate refers to the number of

Modulation Rate	*Circuit*	*Maximum Bit Error Rate*
200	Leased	5–10^{-5}
200	Switched	10^{-4}
600	Leased	5–10^{-5}
600	Switched	10^{-3}
1200	Leased	5–10^{-4}
1200	Switched	10^{-3}

signaling intervals per second. If multilevel modulation techniques are being used, the data rate (bits per second) will be two or four times greater.

Note that this recommendation specifies *maximum* error rates. If these are the rates being experienced in practice, then the user will be having a difficult time! Take, for example, transmission on switched lines at 600 bps and above. An error rate of 10^{-3} represents a loss of one bit per 1000 transmitted. A thousand bits could be a single message of over 100 characters, and an attempt to transmit that message is unlikely to succeed. A bit loss will cause a parity failure and, when the message is retransmitted, it is likely to fail again, causing a further retransmission and so on ad infinitum. In practice, therefore, the quality of lines should be at least an order of magnitude better than the CCITT recommendation. When speech networks have moved completely to digital technology the error rates should be as good as 10^{-6} or even 10^{-7}. The use of packet-switching techniques involving automatic retransmission of bad packets within the network can enhance the reliability of transmission even when analog circuits are being used. On the British Experimental Packet Switched Service, for example, a design objective was to achieve an error rate of one in 10^{-6} *packets* lost (a packet may be up to 2000 bits in length). But even errors as infrequent as this cannot be ignored.

Transmission Timing and Synchronism

In a transmission system using a parallel channel, the transmitting and receiving ends are kept in synchronism by special timing leads that indicate when a character (or word) is available on the data leads. On serial telecommunications circuits the problem is more complex than this. Techniques need to be employed to achieve synchronism first at a bit level and then at a character level.

Transmission Timing

The timing of data transmission (and reception) is accomplished through the use of oscillators or *clocks* located in either the terminal equipment (data transmission equipment, DTE) or in the modem (data circuit-terminating equipment, DCE) as shown in figure 3.10. Some simple (asynchronous) modems do not need timing information and merely copy the bit presented to the line (or vice versa). In this case it is the responsibility of the DTE to ensure that the correct signaling intervals are adhered to. In the case of higher speed (synchronous) modems both the DTE and the DCE need timing information. This is always achieved by having a single clock in either the terminal or the modem and then copying the timing pulses to the other device over the interface. A clock then determines the rate of transmission and

—at the transmission end it tells the modem when to take a bit from the DTE and modulate it onto the line;

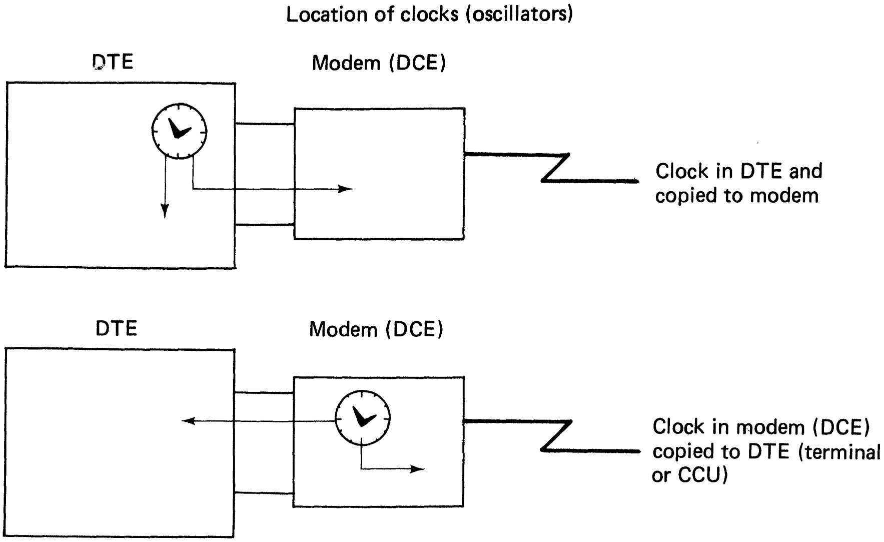

Location of clocks (oscillators)
DTE
Modem (DCE)
Clock in DTE and copied to modem
DTE
Modem (DCE)
Clock in modem (DCE) copied to DTE (terminal or CCU)

Figure 3.10
Transmission timing

—at the receiving end it tells the modem when to sample the line for an incoming bit;

—at both ends it facilitates the orderly passing of bits over the interfaces.

Although it is possible to obtain oscillators that will provide any timing rate, certain conventional speeds have been adopted. These are, in bits per second;

50
75
100
110
134.5
150
200
300
600
1200
1800
2000

2,400
3,600
4,800
7,200
9,600
19,200
48,000
50,000
50,200
240,000

Not all of these will be available on all systems and there may be some devices using unconventional speeds.

The clocks usually provide a pulse for every signaling interval. This is known as the *baud rate*. If there are 2400 signaling intervals in each second, the system is said to have a speed of 2400 baud. This information is not, however, much use to the system designer because he needs to know the *bit rate*. If a four-level modulation technique is being used, then the same link can be better described as 4800 bits per second.

Our next problem is the question of how to ensure that the receiving end is looking for incoming bits at the same time that they are being transmitted. Two techniques are used for this: asynchronous (start/stop) and synchronous transmission.

Asynchronous Transmission

The synchronism problem exists at two levels, the bit and the character. The clocks used in data transmission are on the whole highly accurate, but they do differ fractionally from one another. For example, if one clock varies from another by 1 percent, it will only be necessary to run them for a relatively short period before they become completely out of step. The concept of *asynchronous transmission* is based on the idea of restarting the clocks at the beginning of each character. The likelihood of their getting out of step during such a short period of transmission is slight. When the line is not carrying data it will transmit a continuous carrier. (In the case of half duplex lines, the carrier will have to be specially started by the transmitting modem prior to a period of transmission and then shut off on completion so that the other end may transmit if required.) The start of a character is indicated by the transmission of a start bit, which causes the receiver to activate its clocks. A start bit *must* be a zero, the opposite condition from the carrier.

Immediately following the start will be the data bits (usually five, six, or seven of them), a parity bit and one or more stop bits (see figure 3.11). The stop bits are needed to give the receiving device time to get rid of the incoming character (perhaps by printing it). This enables the next sequential character to follow contiguously. If that next character is not ready for transmission (perhaps it is coming from a keyboard), then the carrier will be sent to hold the line in an idle

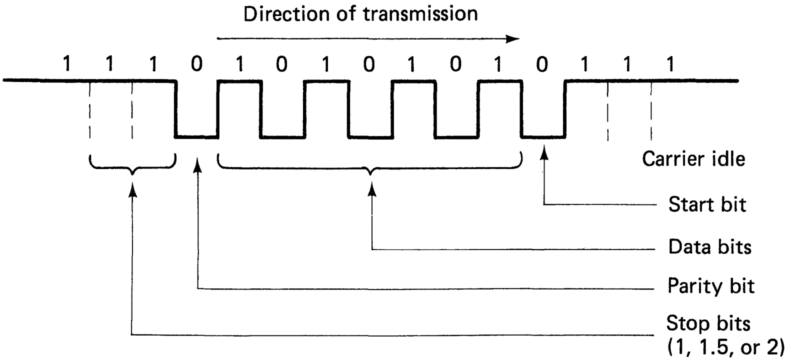


Figure 3.11
Asynchronous (start/stop) transmission

condition. If all the characters have been sent then the carrier will be turned off (on half duplex lines).

This asynchronous method of working (sometimes called start/stop, for obvious reasons) achieves bit timing and character synchronism at the same time. The technique is inexpensive to implement but is relatively inefficient because of the number of redundant bits that must be transmitted with each character. Start/stop transmission is best suited to low-speed, interactive terminals such as keyboard printers. For higher volume, higher speed transmission the overhead of 30 percent or more is generally considered unacceptable. In these cases, synchronous transmission needs to be considered.

Synchronous Transmission

In the case of *synchronous transmission* systems, bit timing is a separate function from character timing. The periodic transmission of a stream of bits enables the receiving device to ensure that its clock is in step with that of the other device prior to the sending of a block of data. In practice the idle sequence may be an alternating pattern of zeros and ones (instead of an unchanging carrier) and the timing is obtained from the average time between transitions from one state to the other.

Even when the clocks have been synchronized, they are unlikely to stay in step for too long. To avoid the danger of this happening during the transmission of a long block of data, the hardware line adapters may inject a short stream of synchronizing bits. This might happen automatically after a predetermined period (say, one second) without synchronism. Bit timing is almost always handled automatically by hardware. Character synchronism and the identification of the beginning of a transmission block is achieved in one operation. This operation introduces the concept of the SYN (synchronizing) character. A SYN character has a prespecified pattern of bits according to the line code being used. The ASCII SYN character, for example, has a bit pattern of 0010110, plus a parity bit. In the line adapter the incoming bit stream is fed bit by bit into an 8-bit shift register. Each consecutive pattern of 8 bits is compared with the SYN pattern. If a SYN character is found, the 8 bits are discarded (because they are not needed in the terminal device).

Because a random occurrence of a SYN character could be caused by noise on the communications channel, it is conventional for a minimum of two and a maximum of six SYNs to be transmitted contiguously at the beginning of each block. The first SYN, therefore, is only accepted as genuine if it is followed by one or more additional SYNs. Subsequent blocks may follow contiguously but must be preceded by further SYNs. Figure 3.12 shows the general format of a data block transmitted using synchronous timing techniques. Problems can occur when random SYNs occur in the data, but this problem will be considered in the section on line protocols.

Synchronous transmission is usually employed for transmission of blocks of data rather than individual characters. Such blocks are usually sent between processors or between a processor and a buffered terminal. Once the block is over

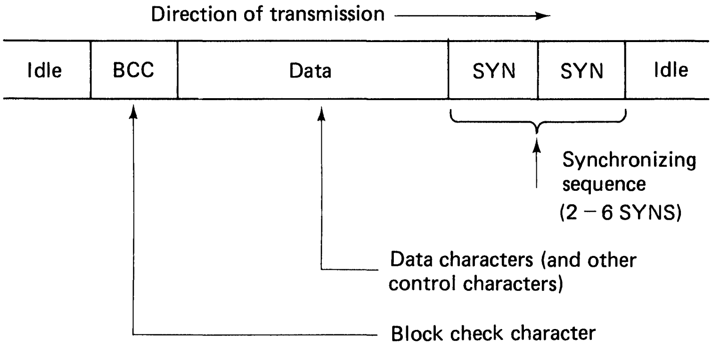

Direction of transmission
Idle
BCC
Data
SYN
SYN
Idle
Synchronizing sequence (2 – 6 SYNS)
Data characters (and other control characters)
Block check character

Figure 3.12
Synchronous transmission

a certain size the efficiency of this technique becomes better than that of a synchronous transmission because it is unnecessary to carry start and stop bits on each character. Faster speeds are also facilitated and data rates are normally in the range of 2400 to 9600 bps. The price of this extra performance and efficiency, however, lies in the higher cost of both modems and line adapters.

Communications Interfaces

The interfaces between the data terminal equipment and the modem or network terminating unit (figure 3.13) are the subject of CCITT recommendations. Two sets of recommendations are currently of interest. The V-series relates to modems and other terminating equipment for telephone and telegraph lines. The more recently published X-series is concerned with data networks. The standard interface circuits most commonly used are listed below. They are taken from CCITT recommendation V24 (the American equivalent is EIA RS-232B):

Data	Transmitted	To modem
	Received	From modem
-Control	Request to send	To modem
1	Clear to send	From modem
	Data set ready	From modem
2	Data terminal ready	To modem
	Ring indicator	From modem
3	Data carrier detector	From modem
	Data modulation detector	From modem
4	Speed selector	To/from modem
-Timing	Transmitter signal element timing	To/from modem
	Receiver signal element timing	To/from modem
-Grounds	Protective ground	
	Signal ground	

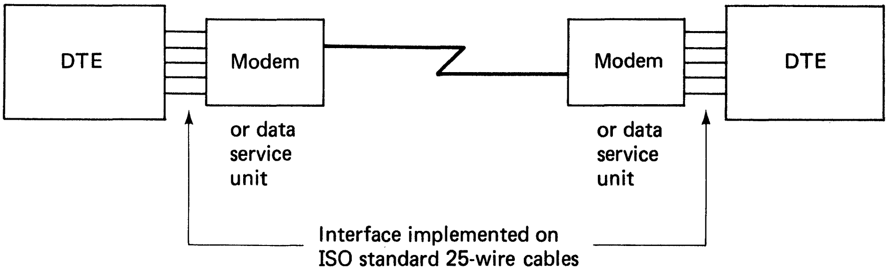


Figure 3.13
Communications interfaces

This fairly comprehensive list of circuits is rarely used in its entirety. For example, asynchronous links do not require timing leads. If a modem is attached to a leased line the control circuits for connecting to calls on the PSTN (public-switched telephone network) are not needed. These subsets of the V24 recommendations are specified by further CCITT standards listed in figure 3.14.

Before proceeding, a more detailed review of the interface would be useful. Most interfaces are implemented on an ISO-standard 25-wire plug/socket and cable, but the full 25 circuits are never used. Although multiple leads are used, it is important to remember that the interface is a serial one, that is, data is passed in a serial-by-bit fashion over the transmitted data lead (to the modem) or the

V24	List of all interchange circuits likely to be used to implement a specific interface. Circuits are numbered—100-series for data transmission and modem control, 200-series for auto dialing
V23	Subset of V24 required for 600/1200 bps transmission on PSTN and/or leased lines. (United Kingdom Post Office Datel 600)
V21	Subset of V24 required for 200 bps transmission on PSTN and/or leased lines (300 bps possible in practice) (United Kingdom Post Office Datel 200)
V26	2400 bps modem for use on leased circuits
V26b	Suitable for 2400-bps transmission on leased lines with fallback to 1200 bps on the PSTN (United Kingdom Post Office Datel 2400)
V27	4800-bps/modem
V35	Recommendation for a 48-Kbps modem
V41	Specification of a polynomial for cyclic redundancy checking
V10	Telegraph transmission at 50 bps on TELEX service or 50/110 bps on leased lines

Figure 3.14
Major CCITT V-series recommendations (existing services)

received data lead (from the modem). Various control circuits are available. Those in the above list marked (1) are obligatory and those marked (2), (3), and (4) are optional. Before a terminal transmits data, it raises a signal on the request to send lead. This tells the modem to start transmitting the carrier, but as the carrier will take some time to stabilize (up to 100 milliseconds) the terminal must wait until the modem signals on the clear to send lead before presenting data. The data set ready circuit indicates to the terminal that the modem is switched on and ready to operate. This lead should be checked prior to issuing request to send/clear to send sequences and is sometimes copied to a display lamp on the terminal.

The next two control leads, those marked (2), enable the terminal to automatically connect to an incoming dialed call. The modem is attached to a local exchange line. When an incoming call is received it puts a signal to the terminal on the ring indicator circuit (if the caller is dialing manually, he will actually hear a ringing tone). The signal on the ring indicator usually raises an interrupt within the processor (or terminal control logic), whereupon it decides whether or not it is able to accept the call. If it can, it signals back to the modem on the data terminal ready lead, which causes the modem to connect to the line (equivalent to picking up the receiver). The circuits shown above are known as the 100-series. An additional set known as the 200-series (V25) can be used for the automatic dialing of calls on the PSTN from the terminal.

The leads marked (3) are used to tell the terminal that a carrier is being received and that the quality of the modulation is acceptable. Two optional speed selector circuits enable multispeed modems and adapters to be implemented.

The timing leads are needed to pass clock pulses between the adapter and the modem (according to the location of the clock). Separate circuits are employed for transmission and reception because different data rates could be used in each direction. The protective ground is an electrical safety feature. The signal ground provides a common reference potential for the other circuits. With the advent of public data networks, the CCITT has published a new set of recommendations known as the X-series. The more important of these are listed in figure 3.15.

As the cost difference between synchronous and asynchronous terminal devices is shrinking, it seems likely that the most popular method of interfacing to public data network terminating units (NTUs) will be through the X21 synchronous recommendation. This is very much simpler than the equivalent V26 arrangement, and the only interchange circuits required are as follows:

Data	Transmitted	To NTU
	Received	From NTU
Control	Control	To NTU
	Indication	From NTU
Timing	Signal element	From NTU
	Byte (optional)	From NTU

The data circuits are used in the same manner as on the V-series interfaces. The control lead is used by the terminal device to indicate to the NTU that a call-

X1	User classes of service: Class 1: Asynchronous up to 300 bps Class 2: Synchronous at 600 bps Class 3: Synchronous at 2400 bps Class 4: Synchronous at 4800 bps Class 5: Synchronous at 9600 bps Class 6 Synchronous at 48000 bps *Also packet-switched classes
X2	User facilities in public data networks. For example: Closed user groups Incoming calls barred Auto-calling and auto-answer Abbreviated calling Remote line identification Call transfer/redirection Multiple address calls Packet delivery confirmation
X20	Asynchronous transmission on public data networks
X21	Synchronous transmission on public data networks
X21b	Access for devices using V-series interfaces
X24	Definition of interchange circuits
X25	Interface for packet-mode terminals

Figure 3.15
Major CCITT X-series recommendations (public data networks)

establishment sequence or data is being sent. It will be off at all other times. The indication circuit is used by the NTU to tell the terminal that an incoming call is being set up or data is being received. On digital data networks bit timing is obtained from the incoming bit stream. The derived timing pulses are passed to the terminal over the signal element timing circuit. Some digital networks will operate on a byte-synchronous basis. In these cases, the byte timing lead will indicate to the terminal the first bit on any received bytes (and when the first bit is transmitted on any outgoing bytes). The other important CCITT recommendation relating to public data networks is X25, which defines an interface for a packet-mode terminal (to be considered later).

Data Link Controls

So far we have considered the means by which we can transmit bits of data from one digital device to another over a telecommunications channel. The practical use of such links, however, requires additional, higher-level procedures known as *data link controls* (DLCs), sometimes called *line protocols*. DLCs are implemented in a combination of hardware, software and, increasingly, programmable hardware interfaces. The objectives of a line protocol are to control the orderly flow of data from one DTE to another at maximum efficiency (i.e., minimum overhead), and with the minimum number of undetected errors.

The obstacles to achieving those objectives may be summarized as follows:

Serial-by-bit transmission requires that control and addressing functions be sent in-stream with data;

Inherently unreliable circuits cause a relatively high frequency of bit-loss and circuit failure;

Nonpermanent connects often require the frequent establishment of actual or conceptual circuits between the DTEs; and

Shared (multipoint) circuits complicate the management of data flow on the links.

Four main types of data are used in data link controls. These are:

The information the user wishes to transmit

Addressing data used to select the device to receive the information (or the device that may transmit next)

Control information consisting of commands and responses (get ready to send, ready to receive)

Checking information such as longitudinal and cyclic redundancy check bits used to detect bit loss or inversion

The organization and use of these fields is best considered in the context of specific link controls. As we see it, there have been three generations of line protocol: Telegraphy, Basic Mode, and High-level Data Link Controls. Each of these will be considered separately.

Telegraph Message-Switching Protocols

Perhaps the earliest form of protocol is one still in extensive (and expanding) use. Telegraph message-switching systems have been installed mostly by large organizations for internal use or by groups of organizations such as airlines, banks, and stockbrokers. Telegraph systems are, technically, relatively unsophisticated. Larger networks may appear complex, but the basic principles of telegraph communications are very simple. (The references at the end of this book provide detailed explanations of telegraph systems and equipment.) This very lack of sophistication (for example, there is no hardware error detection) transfers complexity to the operator, who has to use his teletypewriter to enter control information as well as the message. The Teletype (or teleprinter) may be connected to the switch by dialed or, more usually, switched lines. The switch itself may be completely manual in operation, the receiving station punching paper tape which is then transmitted onward to the destination according to the address in a header.

A number of semiautomatic systems are available, but most message switches currently being installed are computer-based. However, because of the basic nature of telegraph equipment (still used for reasons of cost) the protocols used have developed little since the "torn tape" days. The implementation of message formats varies enormously; some systems manage with a header of only a few characters, others have headers greater than the average size of the message text. On simple Teletypes everything has to be keyed by the operator. On some systems where more sophisticated terminals can be attached, some control sequences may be entered automatically.

Figure 3.16 shows a typical message format. The start pattern is used to "wake up" the switch. The start-of-header sequence is not found on all systems

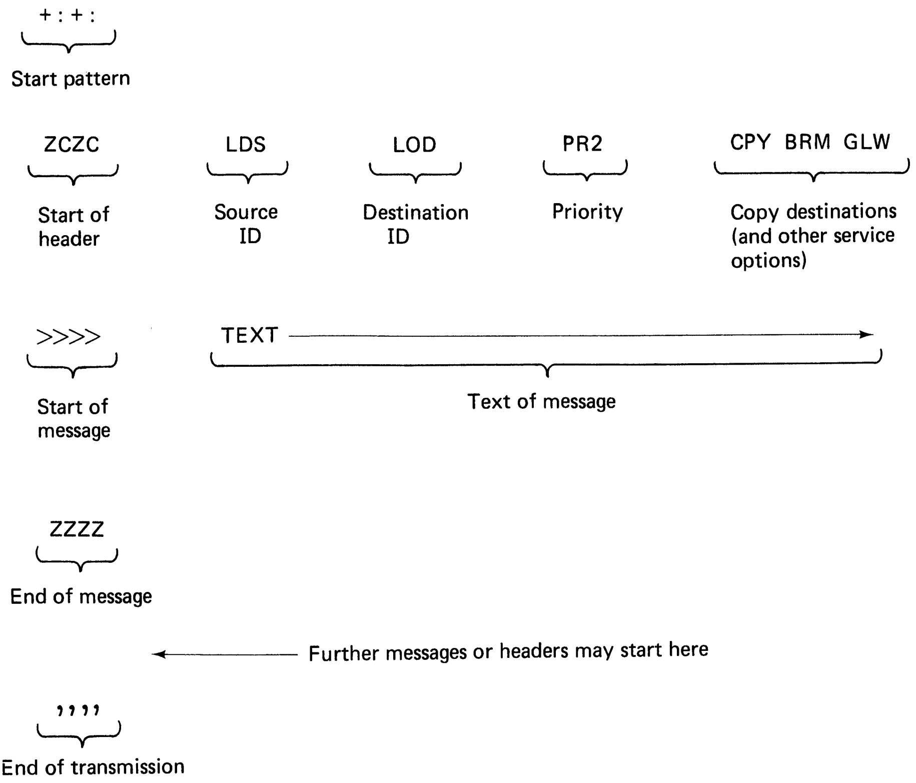


Figure 3.16
Typical format of input message in automatic/semiautomatic switching system

but may be needed if multiple messages have to be sent in one transmission session. The header usually comprises identifying codes for the source and destination. Most message-switching systems are subject to congestion, and therefore a message priority code is useful. The priorities normally used are "life and death," "urgent," "normal," and "deferred." The header will also include a number of service options according to the sophistication of the switch. Our example in figure 3.16 shows the message being copied to two other destinations.

As the header can be of variable length, a start of message sequence is needed before the beginning of the message text. Similarly, an end of message sequence is needed at the end. More messages to the same location may be entered, but it is more usual for the header to be reentered. The end-of-transmission sequences closes down the transmitting station and makes it available to receive incoming messages. A priority incoming message may interrupt the keying of a message.

What are the major problems of a protocol of this kind? First of all, it is inefficient; there is a very high ratio of control characters to text characters. There is no automatic error detection and correction. Transmission errors affecting the header could cause misrouting and errors in the text could cause the message to be misunderstood unless the Teletype operator takes care to respect crucial figures and words. Misrouting or partial loss of the message could occur because of the lack of transparency in the text. Because the switch needs to scan all incoming characters for control sequences, if the message contains ZZZZ, then the switch will assume that the message has ended. The protocol has very limited functions; it is not possible, for example, to poll shared lines. The code used in telegraphy systems is usually the 5-bit International Alphabet no. 2, which has no lowercase and few special characters. Clearly, message-switching protocols are not really suitable for data applications—they were not designed for the job. But they do give us a reference point against which we can compare the more sophisticated link controls.

Basic Mode Link Controls

If the DTE-DCE interface is an area where an agreeable level of standardization has been achieved, only the opposite can be said of data link controls. The second generation of protocols were developed at a time when the cost of "intelligence" was high and simple control procedures were needed for links between relatively powerful mainframe systems and "idiot" terminals. In spite of work by standardization bodies on international norms for such link controls (ISO 2628 and BS 4505), variations have proliferated. Some suppliers managed to standardize on one protocol, but others sometimes developed protocols for specific terminals. The result has been that it is difficult to attach one supplier's terminal to another's processor unless the former was developed to emulate an acceptable link control.

Although *basic mode control procedures* is the expression used by the ISO to describe its standard link control, we shall use *basic mode* to describe all second-generation protocols. These fall readily into two categories with some extensions: (a) contention (+conversational), and (b) polling/selecting (+extended basic modes). The two most widely used codes in basic mode protocols are CCITT International Alphabet no. 5 (and its American equivalent, ASCII, the American Standard Code for Information Interchange, 7 bits) and IBM's 8-bit Extended

				b_1	0	0	0	0	1	1	1	1
				b_2	0	0	1	1	0	0	1	1
				b_3	0	1	0	1	0	1	0	1
b_4	b_3	b_2	b_1		0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	TC_7 (DLE)	SP	0		P	`	p
0	0	0	1	1	TC_1 (SOH)	DC_1	!	1	A	Q	a	q
0	0	1	0	2	TC_2 (STX)	DC_2	"	2	B	R	b	r
0	0	1	1	3	TC_3 (ETX)	DC_3	£(#)	3	C	S	c	s
0	1	0	0	4	TC_4 (EOT)	DC_4	\$(¤)	4	D	T	d	t
0	1	0	1	5	TC_5 (ENQ)	TC_8 (NAK)	%	5	E	U	e	u
0	1	1	0	6	TC_6 (ACK)	TC_9 (SYN)	&	6	F	V	f	v
0	1	1	1	7	BEL	TC_{10} (ETB)	'	7	G	W	g	w
1	0	0	0	8	FE_0 (BS)	CAN	(	8	H	X	h	x
1	0	0	1	9	FE_1 (HT)	EM	)	9	I	Y	i	y
1	0	1	0	10	FE_2 (LF)	SUB	*	:	J	Z	j	z
1	0	1	1	11	FE_3 (VT)	ESC	+	;	K		k	
1	1	0	0	12	FE_4 (FF)	IS_4 (FS)	,	<	L		l	
1	1	0	1	13	FE_5 (CR)	IS_3 (GS)	-	=	M		m	
1	1	1	0	14	SO	IS_2 (RS)	.	>	N		n	
1	1	1	1	15	SI	IS_1 (US)	/	?	O	—	o	DEL

Figure 3.17
CCITT International Alphabet 5 (recommendation V3)—ASCII

Binary Coded Decimal Interchange Code. The CCITT code known as IA5, is shown in figure 3.17. EBCDIC, the IBM standard, is shown in figure 3.18. Each code comprises the following groups of characters: alphabetic (upper- and lowercase), numeric (0 through 9), special characters (punctuation, etc.), and control characters.

The implementation of basic mode protocols involves the use of special control characters at certain points in the data flow. These control characters and the functions they perform are described in detail in the references. For the purposes of this chapter they will be described in the context of their use in specific link controls.

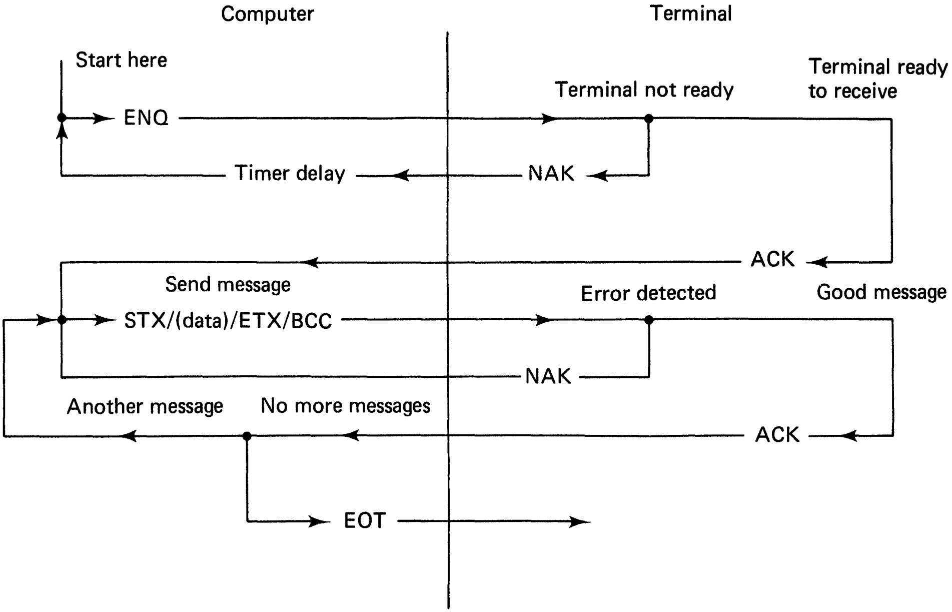
The simplest of link controls is known as the *contention* protocol. This can only be used on point-to-point lines and is widely used for Teletype dial-up to time-sharing services. The code used is almost always IA5 (ASCII-7). A simple

		00				01				10				11				← Bit 0.1
	Hex 1	00	01	10	11	00	01	10	11	00	01	10	11	00	01	10	11	← 2, 3
Bits 4567	↓	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	← Hex 0
0000	0	NUL	DLE			SP	&	-									0	
0001	1	SOH	SBA					/		a	j			A	J		1	
0010	2	STX	EUA		SYN					b	k	s		B	K	S	2	
0011	3	ETX	IC							c	l	t		C	L	T	3	
0100	4		·							d	m	u		D	M	U	4	
0101	5	PT	NL							e	n	v		E	N	V	5	
0110	6			ETB						f	o	w		F	O	W	6	
0111	7			ESC	EOT					g	p	x		G	P	X	7	
1000	8									h	q	y		H	Q	Y	8	
1001	9		EM							i	r	z		I	R	Z	9	
1010	A					c	!	¦	:									
1011	B					.	$	;	#									
1100	C		DUP		RA	<	*	%	@									
1101	D		SF	ENQ	NAK	(	)	—	,									
1110	E		FM			+	;	>	=									
1111	F		ITB		SUB	|	¬	?	"									

Figure 3.18
IBM Extended Binary Coded Decimal Interchange Code (EBCDIC), version used for IBM 3270 display system

representation of the protocol is shown in figure 3.19. The two devices on the line act as equal partners. If one wishes to transmit to the other it must first send an ENQ control character. This character is generally used to solicit a response from a device. It may be generated by pressing a key (usually marked *attention)* on the terminal keyboard or may be transmitted by the computer software. After sending an ENQ, the device must wait for a response. If the other unit cannot accept a message, it will reply with a NAK character (negative acknowledge) and the would-be sender will have to try again later. If the reply is an ACK (positive acknowledgement) then the message can be transmitted. If the terminal is unbuffered, the keyboard will unlock and the user will have to start retyping.

In our example in figure 3.19 the data is prefixed with STX (start of text). This may not always be necessary. ETX tells the receiver that the message has finished



KEY

ENQ	Enquire
ACK	Positive acknowledgement
NAK	Negative acknowledgement
STX	Start of text
ETX	End of text
BCC	Block checking character(s)
EOT	End of transmission

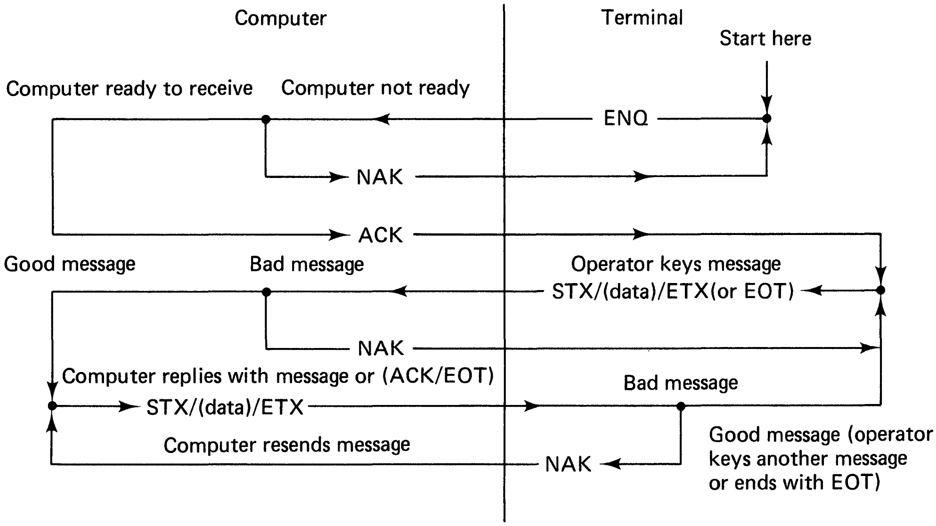
Note: Same sequence may be started by the terminal

Figure 3.19
A typical contention protocol

and, in this case, that a block check character (LRC or CRC) follows. In this type of basic mode protocol block parity checking is optional, as indeed is character parity. If, however, some form of checking is being carried out, then the receiver must tell the transmitter that the message arrived with good parity or not, as the case may be. If the parity was bad, then the receiver responds with the NAK character and this will cause retransmission. An ACK will enable the sending device to transmit another message or, if it has finished, an EOT (end-of-transmission) which will free the line. Either device will then be able to transmit ENQs in order to seize the line.

This type of link control is sometimes streamlined by replacing the ACKs with a data block that needs to go in that direction anyway (if one is available). This is particularly relevant in interactive systems where a terminal operator receives data from the computer for each input message (or vice versa, where the computer is soliciting data from the user). Such protocols are called *conversational,* and a typical example is included in figure 3.20.

The other main type of basic mode link control is known loosely as *polling/selecting protocols*. This kind of protocol is obligatory where multipoint circuits are used but may optionally be employed on point-to-point lines. The use of polling protocols implies a master/slave relationship between the computer and the terminal. This is caused by the additional logic required. Polling requires that each terminal device attached to a circuit has a unique address, enabling the computer to poll each device in turn to give it an opportunity to transmit data to

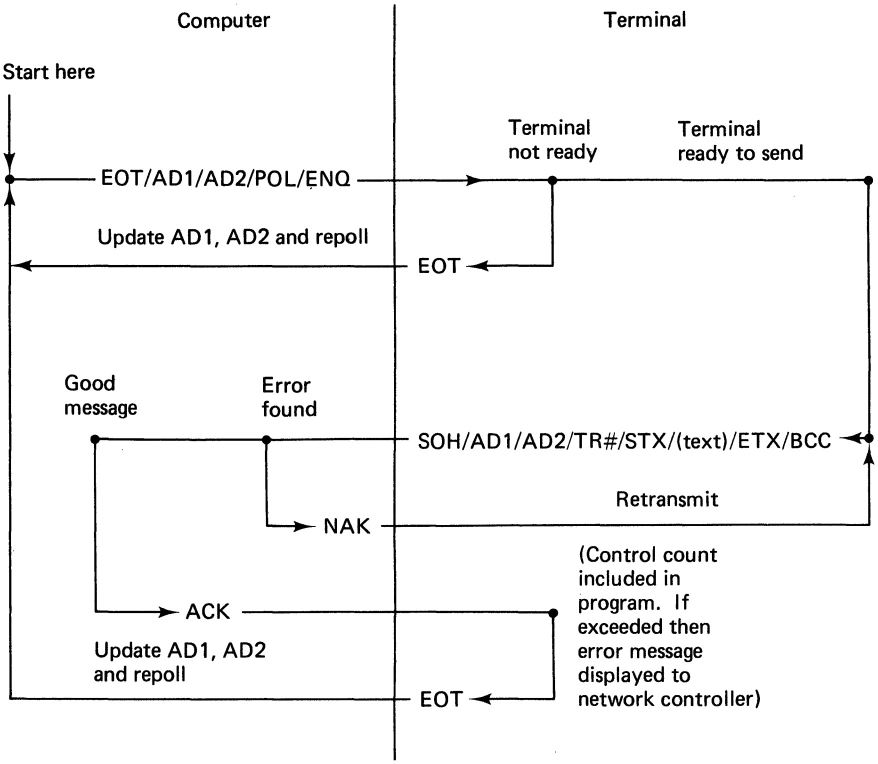


Note: Computer or terminal may start a sequence.

Figure 3.20
A typical conversational protocol

the central site. Clearly, only one device may transmit at any one time and the computer acts like a (good) chairman of a meeting who organizes a coherent flow of data. Polling enables terminals to send messages to the computer. The opposite arrangement, used by the computer to send a message to a specific terminal (or group of terminals), is called selecting.

Let us consider some examples. Figure 3.21 shows the protocol used by


Computer
Terminal
Start here
Terminal not ready
Terminal ready to send
EOT/AD1/AD2/POL/ENQ
Update AD1, AD2 and repoll
EOT
Good message
Error found
SOH/AD1/AD2/TR#/STX/(text)/ETX/BCC
Retransmit
NAK
(Control count included in program. If exceeded then error message displayed to network controller)
ACK
Update AD1, AD2 and repoll
EOT

EOT	End of transmission
AD1 AD2	Two-character terminal address
POL	ASCII lowercase 'p': indicates poll operation
ENQ	Used to indicate that a response is required
SOH	Start of heading
TR#	Optional transmission number, up to three ASCII numerals
STX	Start of text
ETX	End of text
BCC	Block check character (longitudinal parity)
ACK	Acknowledgement
NAK	Negative acknowledgement

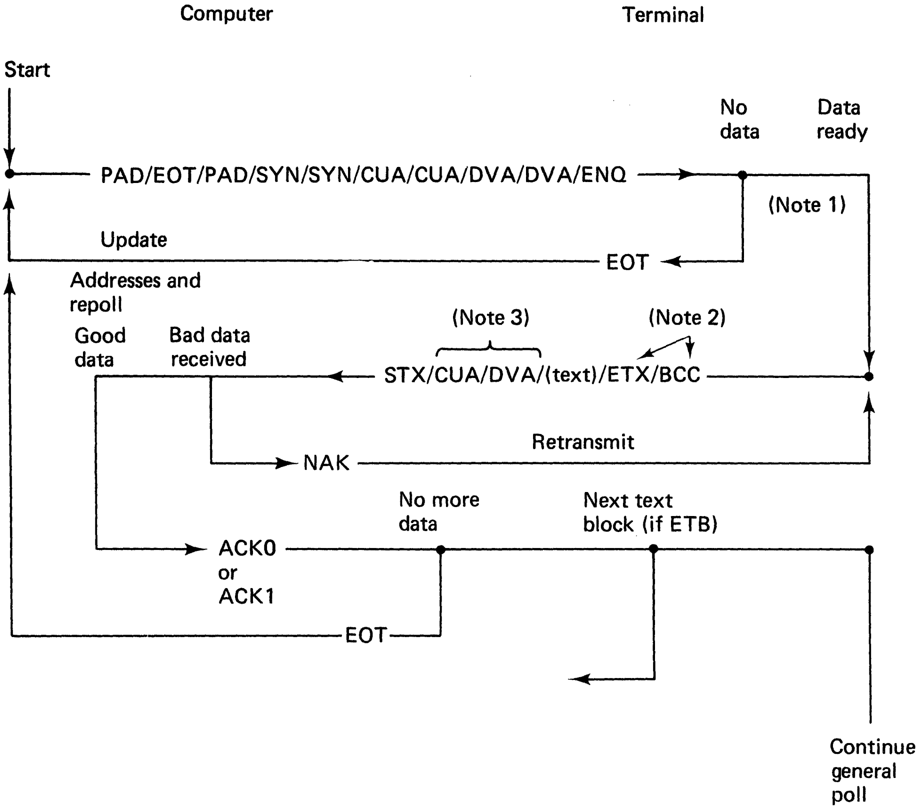
Figure 3.21
Asynchronous polling example, Burroughs TC500

Burroughs for its TC500 programmable terminal (and other devices). The computer starts by sending a polling sequence. The EOT clears the line, two address characters indicate which terminal is to respond, the poll character indicates the operation being carried out, and ENQ solicits a response. Note that POL is not included as a control character in the IA5/ASCII-7 code and, therefore, Burroughs has elected to use lowercase "p" for this purpose. If the terminal has no data to transmit it sends EOT and the computer initiates a poll sequence for another terminal on the line. Otherwise it sends the message currently awaiting transmission. As a checking mechanism, the terminal repeats its address in the header of the message and also includes a sequential transmission number. The ACK/NAK procedure is used to confirm (or otherwise) the receipt of a good message.

The IBM 3270 display system uses a standard IBM link control called binary synchronous communications (BSC). In principle it is very similar to our TC500 example but varies in detail. Figure 3.22 shows a general schematic of BSC. The first difference to note is the use of SYN characters (the TC500 protocol was asynchronous). Ignore the PAD characters—these are needed because of the timing requirements of the control units. The polling sequence is also made longer by the use of separate addresses for the cluster controller and the attached devices (displays or printers). This arrangement enables a general poll to be carried out; instead of sending a special poll sequence to each operator station, the controller is asked whether any of its attached devices has data ready for transmission. This reduces the overhead on the circuit associated with the polling function. The polling function, then, is a technique used to solicit data from a specific terminal or from any terminal that happens to be ready to transmit. When the computer has to send data in the opposite direction to a terminal, a technique usually known as *selecting* is used. Figure 3.23 shows the selecting procedures used by the Burroughs TC500. This starts with the computer sending a selecting sequence identical in format with the polling sequence (figure 3.21), but SEL (lowercase "q") is used at the command. If the terminal is busy it can reply with NAK. If ACK is sent, then the computer will go ahead and transmit the message.

In summary, basic mode link controls operate on the system that terminals can only transmit when invited to do so (polling) and the computer can only transmit when it has asked the terminal if it is ready to receive (selecting). Error checking is based upon the use of character parity plus a longitudinal redundancy check or on a cyclic redundancy check for the whole block. ACKs and NAKs are used to indicate good or bad transmissions. Although this simple mechanism enables error blocks to be retransmitted, it does impose a significant overhead on data transmission (especially on half duplex circuits where it could take up to 100 milliseconds merely to switch a modem from reception to transmission). Even when a full duplex channel is used, the nature of the protocol is such that transmission only takes place in one direction at any one time.

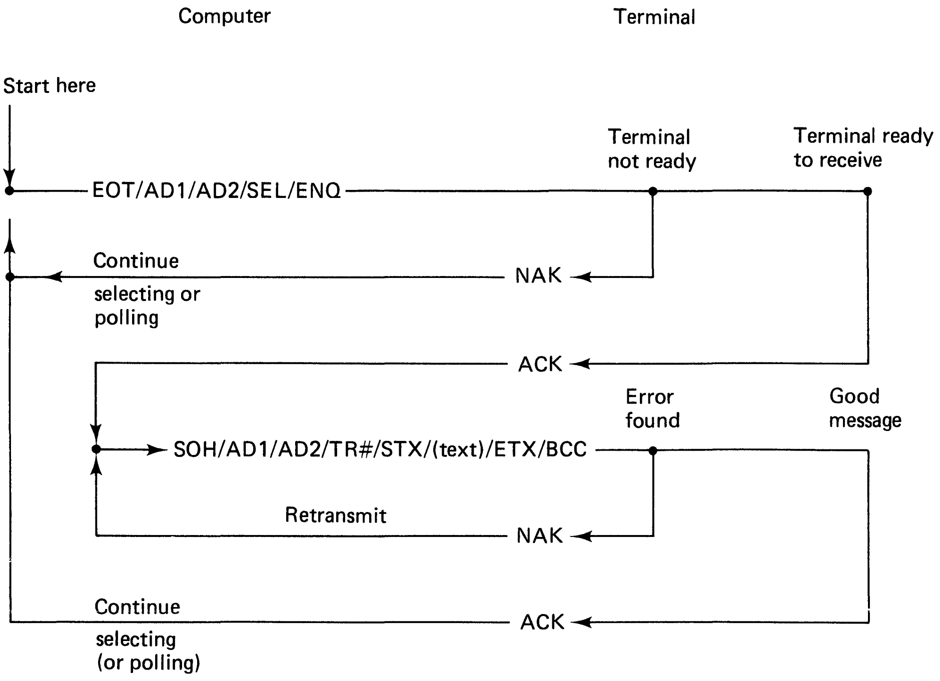
A further problem is caused by the fact that control characters are transmitted in sequence with data and are only recognizable by their value. Unfortunately it is



PAD	Hardware-generated one-character time delay
EOT	End of transmission
SYN	Synchronous idle
CUA	Control unit address
DVA	Device address
ENQ	Response required
STX	Start of text
ETX	End of text
ETB	End of transmission block
BCC	Block checking character (3)
NAK	Negative acknowledgement
ACK0	Acknowledgement of even-numbered text blocks
ACK1	Acknowledgement of odd-numbered text blocks

Note 1: See relevant IBM manuals for details of alternative responses
Note 2: ETB is used instead of ETX when more blocks follow
Note 3: Address need not be inserted after the first block

Figure 3.22
Synchronous polling example, IBM 3270 display system


Computer
Terminal
Start here
Terminal not ready
Terminal ready to receive
EOT/AD1/AD2/SEL/ENQ
Continue selecting or polling
NAK
ACK
Error found
Good message
SOH/AD1/AD2/TR#/STX/(text)/ETX/BCC
Retransmit
NAK
Continue selecting (or polling)
ACK

Select character (ASCII lowercase 'q')
Other characters as in figure 3.21

Figure 3.23
Asynchronous selecting example, Burroughs TC500

not always possible to guarantee that the information being transmitted does not contain spurious control characters. This problem is most acute when program object code is being sent from one computer to another. For example, ETX is used to indicate the end of data and that the next one (or two) characters are block-checking bits. If ETX occurred in the data, the end of the block would be detected prematurely and the block parity would almost certainly fail. The basic mode technique used to prevent this is known as *escape logic*. The objective of escape logic is to achieve *transparency* of the data being transmitted, and two approaches are employed. The first requires the insertion by the transmitting device of an escape (ESC) character in front of each spurious control character (including other ESCs) in the data. This tells the receiver to ignore the following character. The second option (used by IBM) entails putting ESCs before each *real* control character. The method does work, but it is cumbersome to implement and increases overheads.

It was to resolve these and other problems that a new generation of protocols (known usually as high-level data link controls) were developed. These will be considered in the next section. It should be noted that some equipment suppliers have attempted to improve the performance of basic mode protocols, especially when used for interprocessor communication. These developments usually improved error handling procedures, facilitated full duplex working, and added other features such as data compression. Examples of these protocols are Univac's Remote 9000 Interface and ICL's Extended Basic Mode. Another "intermediate" protocol is DEC's DDCMP, but because this relates more in structure to high-level data link controls we will consider it separately.

High-Level Data Link Controls (HDLC)

The inefficiencies and constraints of basic mode protocols have long been recognized. As far back as 1968 the British Standards Institute (the United Kingdom equivalent of ANSI) proposed a new link control architecture based on a unit of transmission now known as the *frame*. Since that time both standardization bodies and the data processing industry have made significant (albeit slow) progress toward a flexible and open-ended link control that can be used in a wide variety of systems. The new protocol has emerged in a number of slightly varying guises, some of which are listed below:

Source	*Name*
International Standards Organization	High-level data link Control (ISO 3309)
British Standards Institute	HDLC (BS 5397)
American National Standards Institute	Advanced Data Communications Control Procedure(ADCCP)
IBM Corporation	Synchronous Data Link Control(SDLC)
Burroughs Corporation	Burroughs Data Link Control(BDLC)

It is not possible within the scope of this chapter to attempt a full description of HDLC, nor would such an attempt be wise in view of the ongoing development of the standard. We shall, therefore, restrict ourselves to consideration of the basic concepts of HDLC, the format of HDLC frames, and the way in which the protocol might work in a simple point-to-point link. The major features of HDLC are:

Support for HDX and FDX circuits
Support for two-point and multipoint lines
Complete code transparency
Bit-synchronous transmission
Positionally significant control bits
Flexible error handling
Open-ended structure
Suitability for all types of message traffic

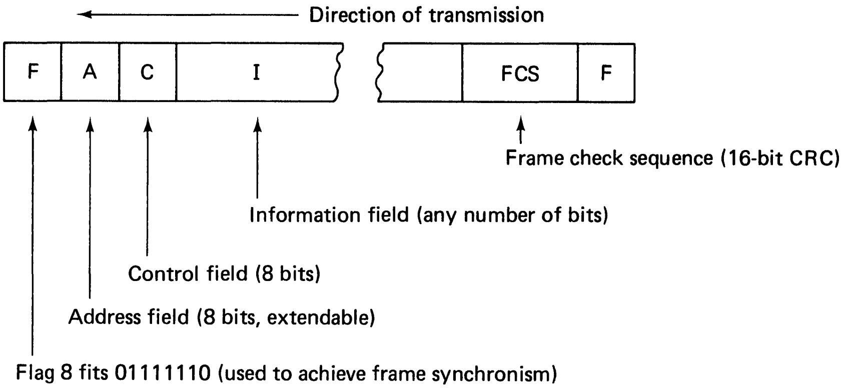

Direction of transmission
F
A
C
I
FCS
F
Frame check sequence (16-bit CRC)
Information field (any number of bits)
Control field (8 bits)
Address field (8 bits, extendable)
Flag 8 fits 01111110 (used to achieve frame synchronism)

Figure 3.24
Format of the HDLC frame

The concept of the frame is fundamental to HDLC and is illustrated in figure 3.24. A frame is delimited by unique *flag* sequences of eight bits—01111110. The flag that ends a frame may be the first flag of the next consecutive frame. By examining each incoming sequence of 8 bits the receiving device can get into synchronism with the frame when it finds a flag. What happens, you may ask, if there is a spurious flag sequence in the middle of the data field? This problem of transparency is solved through a simple hardware mechanism called *bit stuffing* and *bit stripping*. As the address, control, information, and FCS (frame check sequence) fields are being transmitted, the line adapter counts the number of 1-bits following any 0-bits. If the count reaches 5, the adapter automatically inserts a 0-bit. Similarly, at the receiving end, whenever 5 contiguous 1-bits are encountered, the following bit is removed. This simple arrangement solves the problem of transparency and enables any combination of bits to be sent in a frame.

Within a frame there are four fields. The address field is always the first 8 bits after the starting flag and the control field is always the first 8 bits after the address. The FCS field is always the 16 bits preceding the terminating flag. The information field may be any number of bits (including none) between the control and FCS fields. The FCS is generated using the CCITT V41 standard 16-bit polynomial and includes address and control fields but not flags. Figure 3.25 shows that each HDLC link comprises a primary (controlling) station and one or more secondary stations. In some cases, a secondary station may also be primary.

The address field in a frame is used for addressing secondary stations. The control field is used for sending commands from the primary to the secondary and responses from the secondary to the primary. The format of the control field varies according to the type of frame being transmitted. There are three types of frames:

I-frame: Information transfer

S-frame: Supervisory

U-frame: Unnumbered

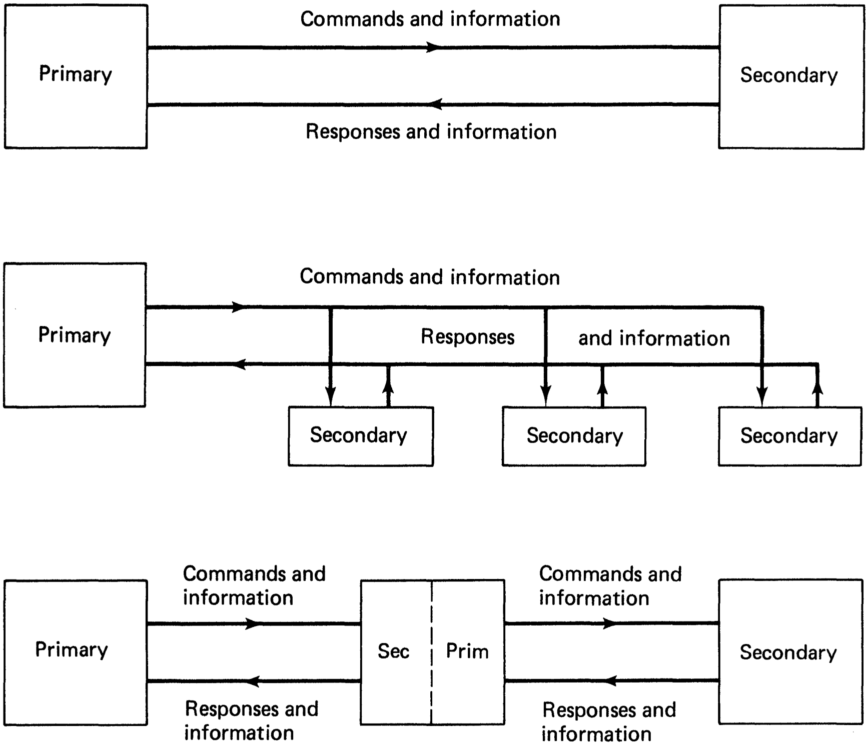


Figure 3.25
Primary and secondary stations in HDLC links

The control field format for each frame is shown in figure 3.26. I-frames are used to transfer information between stations and are the only frames that can contain information fields. Each frame transmitted is numbered on a cyclic basis 0 through 7 (N(S)). In addition, the control field contains the number (N(R)) of the next frame it expects to receive at that station, thus implying that all frames up to and including N(R) − 1 have been received successfully. This technique obviates the needs for separate ACKing of each transmission block and thus improves circuit utilization. Bit 5 of the control field is used in one of two ways. If a primary station sets it in any frame, it polls the receiving secondary (invites it to transmit). If it is set to 1 by a secondary it indicates to the receiving primary that the frame being transmitted is the last and no more will be sent until the next poll is received.

S-frames are used to perform supervisory functions and do not contain an information field. An S-frame can be used to poll a secondary, stop transmission of I-frames, request retransmission of I-frames, and acknowledge I-frames if there have been no other transmissions in the opposite direction for seven received

Frame type	Control field bits				
	1	2 3 4		5	6 7 8
Information transfer	0	N(S)		P/F	N(R)
Supervisory	1	0	S	P/F	N(R)
Unnumbered	1	1	M	P/F	M

N(S) Transmitting send sequence count (3 bits)
N(R) Transmitting receive sequence count (3 bits)
S Supervisory function bits
M Modifier function bits
P/F "Poll" bit on primary transmission
"Final" bit on secondary transmission

Figure 3.26
Formats of control field in HDLC frames

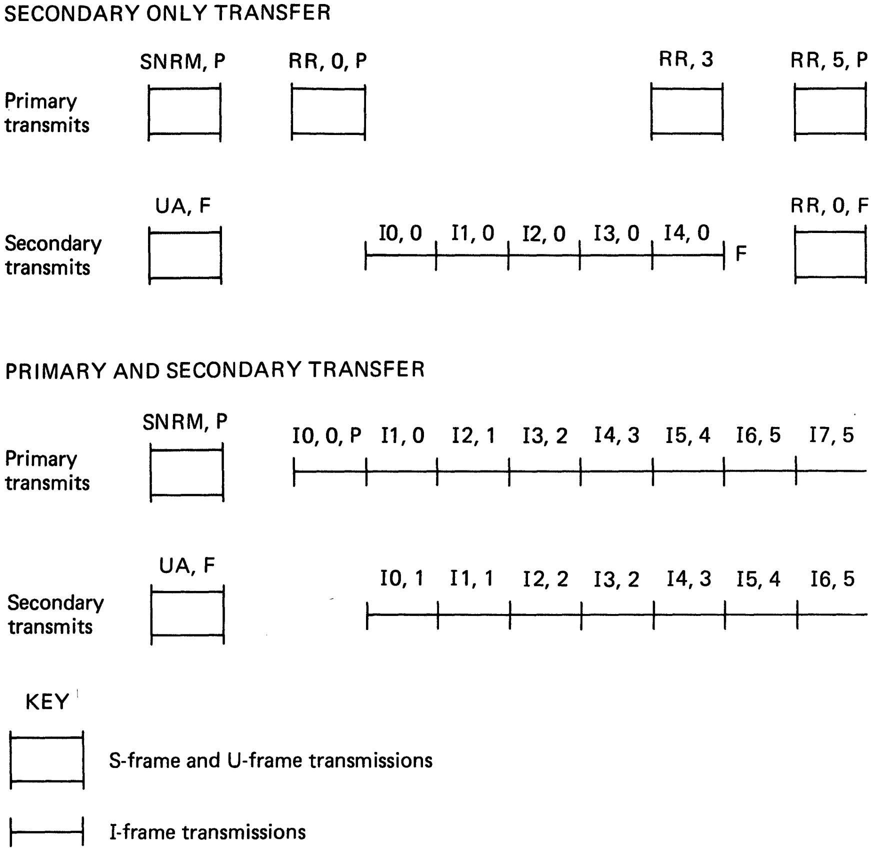
frames. The two S-bits in the control field indicate the supervisory function being carried out.

U-frames are used to provide additional link control functions (up to 32 commands and 32 responses).

The draft international HDLC standard specifies two operational modes: normal response mode (NRM) and asynchronous response mode (ARM). In the simplest of terms, NRM is a polled mode whereby the secondary must receive permission to start transmitting (one or more I-frames). In the case of ARM, the secondary may transmit at any time.

Figure 3.27 illustrates simple exchanges of data over a point-to-point FDX link using normal response mode. In the first example only the secondary is transmitting; in the second example both stations are transmitting. If a transmission error occurs on a line, two commands can be used to cause retransmission: REJ (retransmit from the numbered frame onward) and SREJ (retransmit the error frame only). Examples are shown in figure 3.28.

HDLC represents a significant breakthrough in the technology of data link controls. The protocol is highly efficient and enables good use to be made of full duplex circuits. It seems likely that HDLC will be widely accepted by equipment manufacturers as a good general-purpose, open-ended standard. HDLC does, of course, require the development of special line adapters to perform functions such as bit stuffing and stripping. One company (Digital Equipment Corporation)



COMMANDS USED

SNRM	Set normal response mode
P	Poll
UA	Unnumbered acknowledge
RR	Receive ready
F	Final
x, y	N(S), N(R)
I	I-frame

Figure 3.27
Examples of NRM FDX transmission

decided to develop a link control that would employ many of the concepts of HDLC but that could be implemented on existing asynchronous and character-synchronous line adapters. The DDCMP (Digital Data Communications Message Protocol) frame format is shown in figure 3.29. SYN characters are used to achieve block synchronism. The header includes a count of the information field

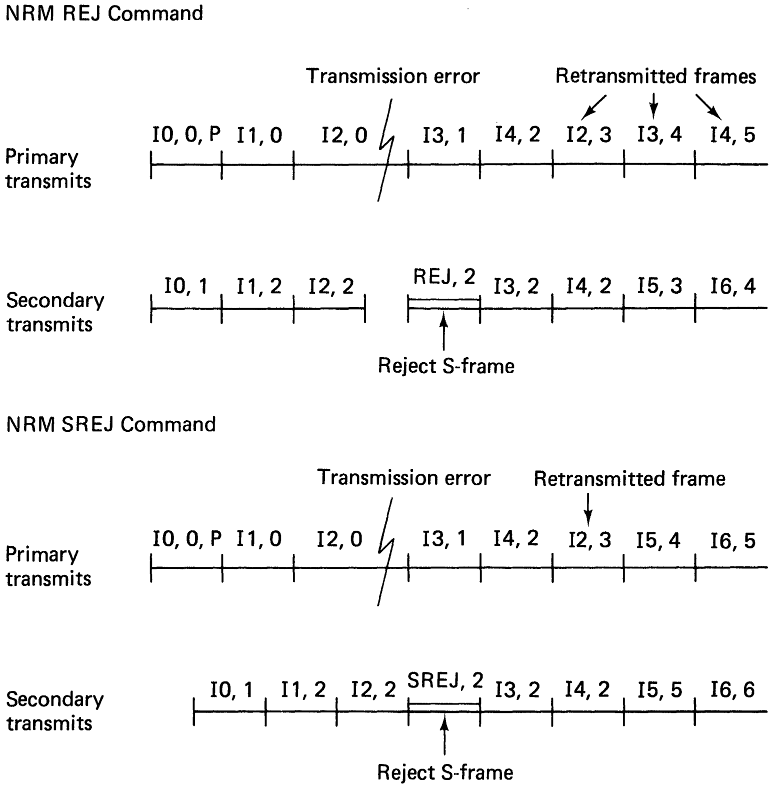


Figure 3.28
Examples of HDLC error handling (NRM FDX)

and this is used to implement transparency, because the receiving station does not need to look for control characters to locate the end of the block. Separate CRC checks are carried out on the header and the information field (if any). DDCMP can transmit on HDX or FDX, two-point or multipoint circuits. It can also be used on parallel transmission links for interprocessor communication. DDCMP was originally released as part of DECNET, which provides application programmers with a high-level protocol for interprocessor communication. High-level protocols will be discussed in a later section.

CCITT Recommendation X25

We have postulated that there is likely to be a significant expansion in the United States and Europe throughout the 1980s of public packet-switched networks. In anticipation of these developments, the CCITT has published a recommendation that defines an interface between a packet-mode terminal (or computer) and a public packet-switched network. Some equipment suppliers have

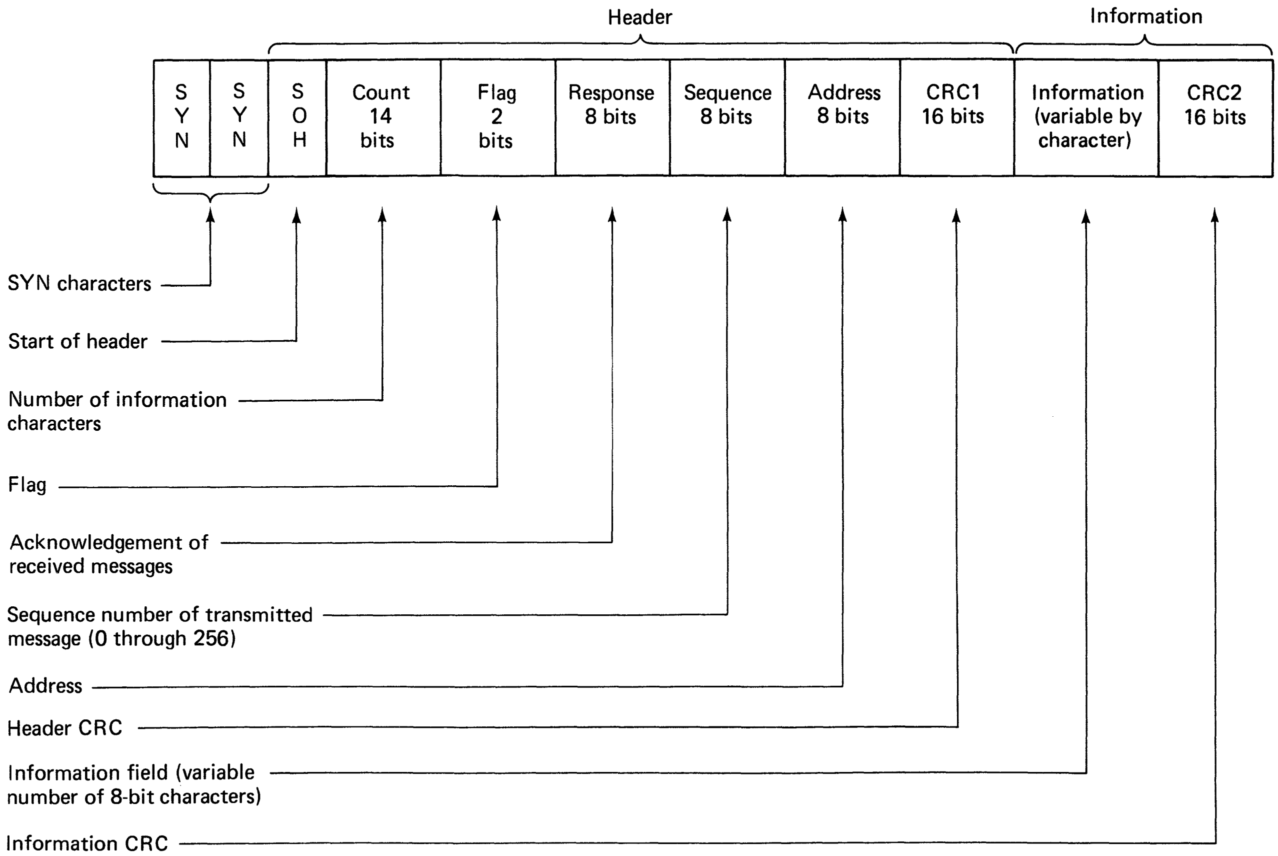

Header
Information
S Y N
S Y N
S O H
Count 14 bits
Flag 2 bits
Response 8 bits
Sequence 8 bits
Address 8 bits
CRC1 16 bits
Information (variable by character)
CRC2 16 bits
SYN characters
Start of header
Number of information characters
Flag
Acknowledgement of received messages
Sequence number of transmitted message (0 through 256)
Address
Header CRC
Information field (variable number of 8-bit characters)
Information CRC

Figure 3.29
Format of DEC DDCMP messages

already announced plans to produce terminals with X25 interfaces, and many prototypes have already been installed. Networks that will operate to this standard include TELENET (United States), TRANSPAC (France), DATAPAC (Canada), and EURONET (European Economic Community), as well as a number of private networks. The X25 recommendation is specified as three independent levels of operation:

Level 1: Bit level (based on CCITT Recommendation X21)

Level 2: Frame level (based on HDLC ISO3309 and ISO4335)

Level 3: Packet level (defined in X25)

Level 1 provides an interface for passing bits of data between the network and the terminal. Level 2 uses HDLC procedures for the exchange of data (control, addressing, and information) between terminal and network. Level 3 specifies a packet format for the transmission of data end-to-end (see figure 3.30). The X25 recommendation is likely to be subject to further development, and the reader is advised to monitor the progress of this combined interface and data link control.

High-Level Protocols and Network Architectures

Terminal-based computer systems are developing in such a manner that more and more networks, both large and small, are incorporating more than one processor. The need to improve the means whereby application programs resident in different

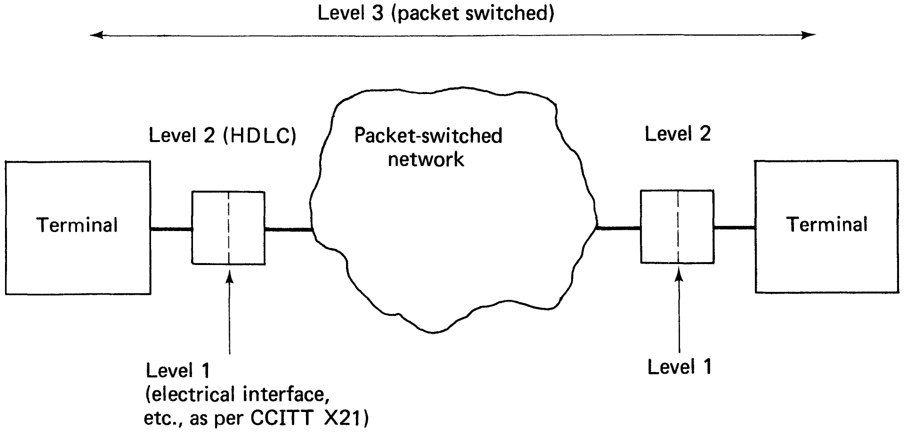

Level 3 (packet switched)
Level 2 (HDLC)
Packet-switched network
Level 2
Terminal
Terminal
Level 1 (electrical interface, etc., as per CCITT X21)
Level 1

Figure 3.30
Three levels of the CCITT X25 specification of interface to packet-switched network

computers can "talk" to one another has given rise to the development of *high-level protocols* (HLPs). HLPs allow the programmer, among other things, to

Send data to a program in another processor
Interrogate or update a file in another processor
Cause a program in another processor to be initiated
Transmit some data to be printed at a remote site

The British Standards Institute has again undertaken a pioneering role in developing proposals for an international standard HLP. This work is now being developed further by ISO.

The role of the HLP is best seen in the context of an overall network architecture, as shown in figure 3.31. The lowest two levels of DTE/DCE interface and network protocols we have already examined. The network management level includes such functions as end-to-end path establishment and message routing.

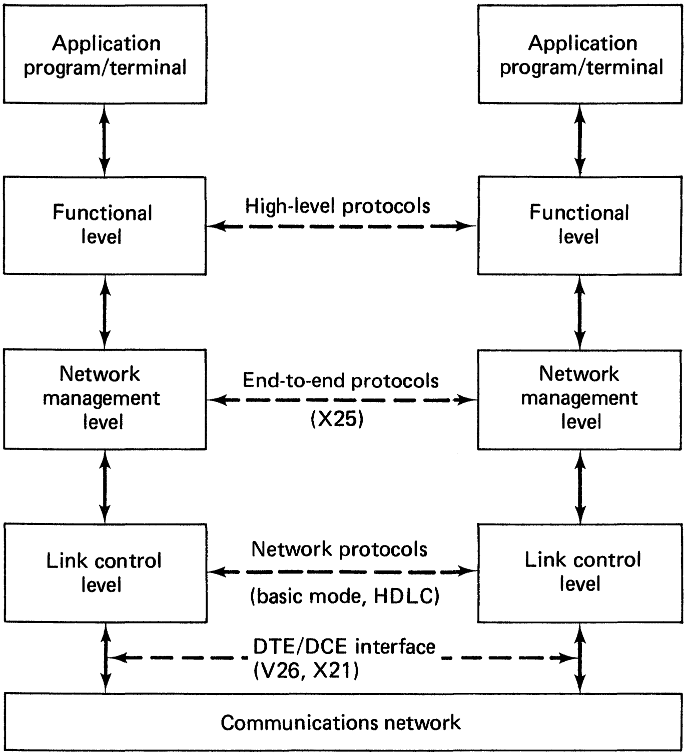


Figure 3.31
Typical network architecture

The functional level is the one at which high-level protocols operate. Commands are available to enable application-level components (such as programs or terminals) to establish links with other such components in remote processors connected by telecommunications facilities. Network architectures of this type are commonly implemented using a hierarchy of headers in the messages being transmitted (see figure 3.32). Three main classes of communication may take place:

Interactive (program-to-program or program-to-file exchange of data)
Remote job entry (remote task/program activation)
File (or program) transfer

It will clearly take some time before the ISO achieves an internationally accepted standard for HLPs, if ever. Even if it does succeed, the new standard will need to compete with network architectures that have already been implemented either by manufacturers or on private packet-switched networks. IBM's System Network Architecture (SNA) and DEC's Digital Network Architecture (DNA) are examples of this structured approach to data communications.

Terminals

The terminal is arguably the most important single component of a teleprocessing system. It is true, of course, that teleprocessing would not be possible without the computer. However, the system exists to serve the needs of the user department

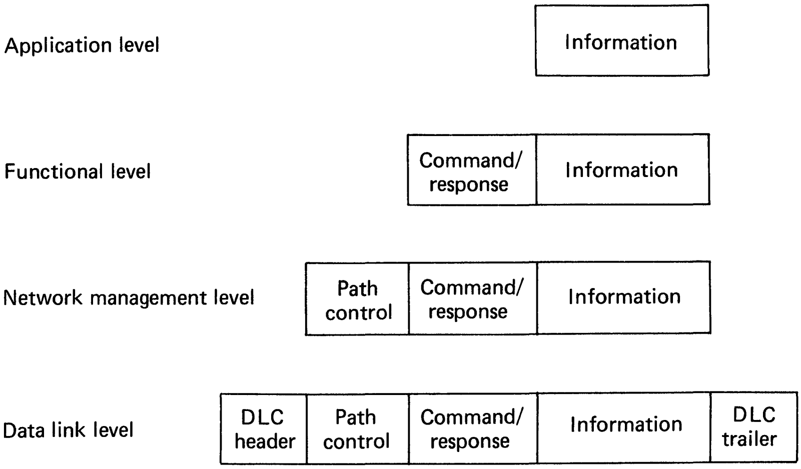


Figure 3.32
Use of headers in network architectures

and the only part of it the user sees is the terminal. This device could play a more important role in his working life than his telephone, his adding machine, his typewriter, or the photocopier. Choosing a terminal, and using it properly, is all-important. In this section we will look at the main types of terminals currently available.

In the early days of teleprocessing (not long ago), the only device that was readily available and could easily be used as a terminal was the teleprinter, or at least the American version known as the teletypewriter or Teletype. Throughout the 1970s, the Teletype was the most popular single terminal device in use, probably because few suppliers have ever been able to match its extremely low price. The standard Teletype is basically a keyboard with a character printer which will work at a maximum speed of 10 characters per second. Print quality is poor and there is no lowercase alpha. The next generation of keyboard printers (figure 3.33) was based on electric typewriters, usually IBM's Selectric, which uses the "golfball" printing mechanism. Print quality was much improved, but maximum speed was only 15 cps.

The current generation of keyboard printers mostly use a *dot matrix* print mechanism. This has not improved print quality (in our opinion the quality has deteriorated). However, this technique has greatly improved the speed of hardcopy terminals with rates in excess of 120 cps easily attainable. Recently a new printing device (using a plastic or metal wheel, with the characters at the end of the spokes) has been developed which gives print quality very comparable with IBM's golfball but speeds (at present up to 55 cps) unlikely to match the matrix mechanism. Other techniques (such as the use of ink jets) have been developed, but these have had only moderate success, especially if they will not produce copies.

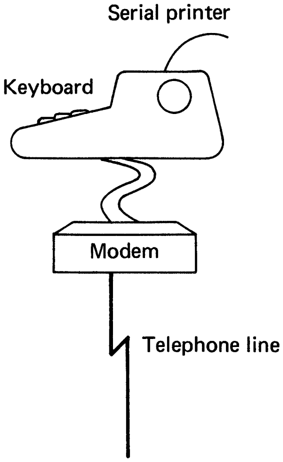


Figure 3.33
Keyboard printer terminal

The keyboard printer is classed as an interactive terminal because of the direct nature of the conversation between the user and the computer. As the user types, each character goes down the telephone line to the computer. When the message is finished (this is indicated by the user pressing a control key), the computer can process it, perhaps transmitting a response of some kind back to the terminal. The other main type of interactive terminal is the visual display unit or VDU. In North America this is usually called a CRT (cathode-ray tube) or a video terminal, but we are all talking about the same thing (figure 3.34). Instead of a printer the VDU uses a television-type display screen on which the characters being entered on the keyboard or received from the computer will appear. There are usually between 16 and 25 lines of 80 positions in which a character or a blank space may appear.

After each operation, all or part of the screen may be cleared by the operator or the computer to make way for the next. A good VDU is usually quiet and pleasant to operate. It can have a number of features that make it particularly suitable for commercial use. These include the *forms mode* method of operation, which enables the computer to display screen layouts that look like original input documents. The operator then fills the blanks in the layout with the details on the source document currently being entered. He cannot overwrite the screen form (which is protected), and when he has finished, the "variable" items just entered are transmitted to the computer. These are then cleared to blanks, leaving the original form so that the next document can be typed in.

This type of operation is the main reason why VDUs have overtaken the keyboard printer in popularity, certainly for commercial applications. The fact that the visual display unit does not use paper is both an advantage and a disadvantage

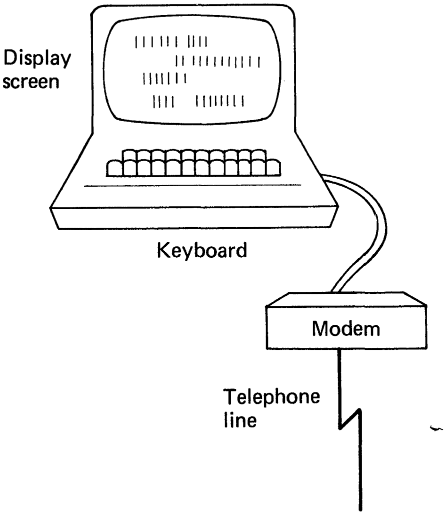


Figure 3.34
Visual display terminal

at the same time. Although the benefits of the speedy and silent method of working may be clear for one area of a company's operations, we have not yet reached the stage where the rest of the world can manage without paper. Invoices need to be sent to customers, checks and remittance notices to creditors, reports to senior management, and so on. An on-line system may facilitate the use of an efficient high-speed line printer at the central site, but it is still often the case that documents must be produced at the terminal location. Many firms do this by attaching serial printers directly to the display unit (figure 3.35). This enables the contents of the screen to be copied onto paper (which may, of course, be preprinted) or, in some cases, for the print details to be sent directly to the printer from the computer.

Most minicomputers use *Teletype-compatible* VDUs as their standard terminal. As their name implies, these displays were originally developed as plug-compatible replacements for the teletypewriter. These were originally very simple devices, but with the recent introduction of microprocessors as control mechanisms in terminals, many of the characteristics of commercial VDUs (as supplied by computer manufacturers) have been incorporated. Three main transmission modes are usually available on *glass teletypes,* each employing special (and confusing) meanings of the terminology *full duplex* and *half duplex*. These are illustrated in figure 3.36. Note that with this type of block mode, the block is transmitted as a contiguous string of asynchronous characters.

If you need a number of VDUs at one location, it is possible to obtain a cluster system (figure 3.37). Under this scheme the displays are attached as peripherals to a controller which converses with the computer on their behalf over the telephone

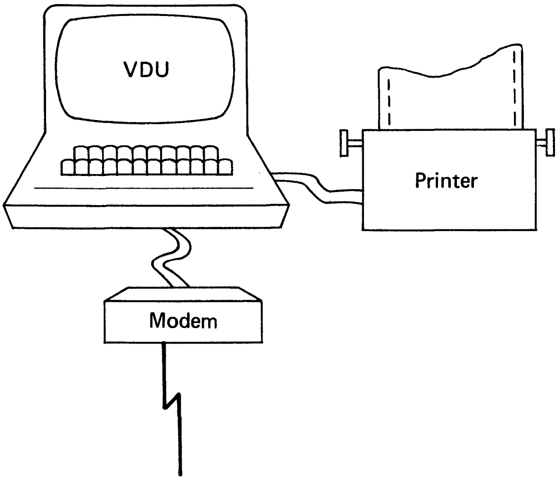
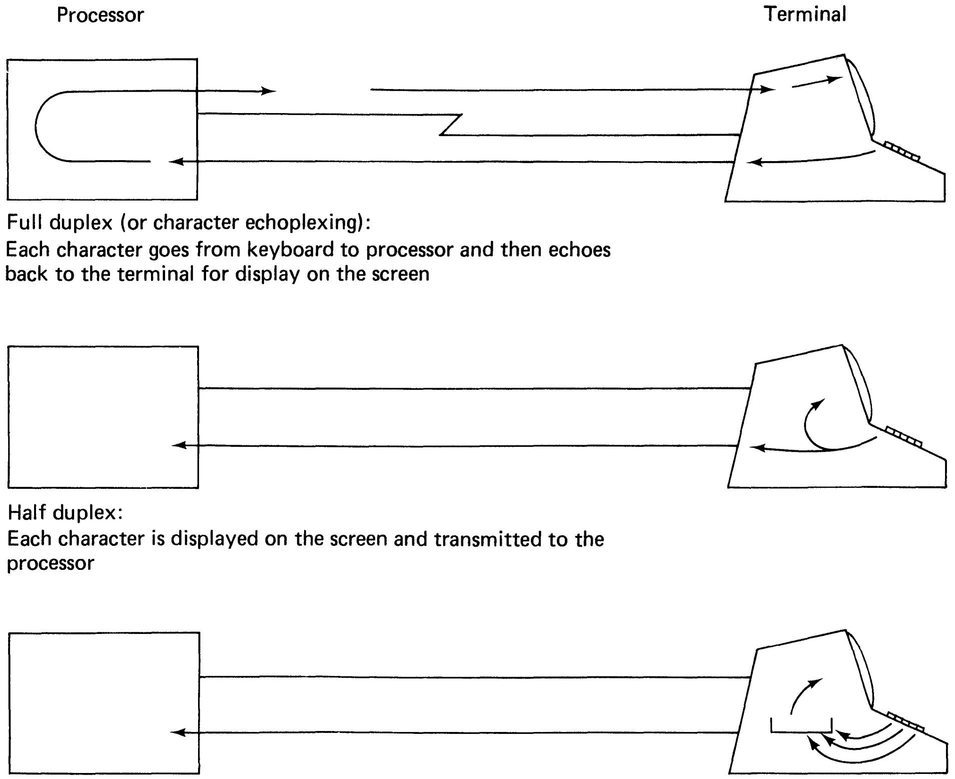


Figure 3.35
VDU with attached serial printer



Full duplex (or character echoplexing):
Each character goes from keyboard to processor and then echoes back to the terminal for display on the screen

Half duplex:
Each character is displayed on the screen and transmitted to the processor

Block mode:
Message is buffered at display and then transmitted as a block (part or whole of screen)

Figure 3.36
Teletype VDU transmission modes

line. A proportion of the electronics previously in each display is now part of the controller and shared between them. This means that once you have more than a certain number of displays in one place, it is cheaper to have a cluster than the individual freestanding units. With the cluster approach it is usually possible to attach up to 32 displays. One or more printers (up to 300 lines per minute each) may also be connected to the controller, but this might reduce the number of VDUs that could be attached. The controller is usually polled.

It is possible to attach peripherals to interactive terminals (both keyboard printers and VDUs). These may be paper-tape readers, cassette tape recorders, discette units, and so on (figures 3.38 and 3.39). These are generally used when the connection to the central computer is via a dialed telephone line. Data can be keyed onto the storage medium and, if necessary, corrections made off-line

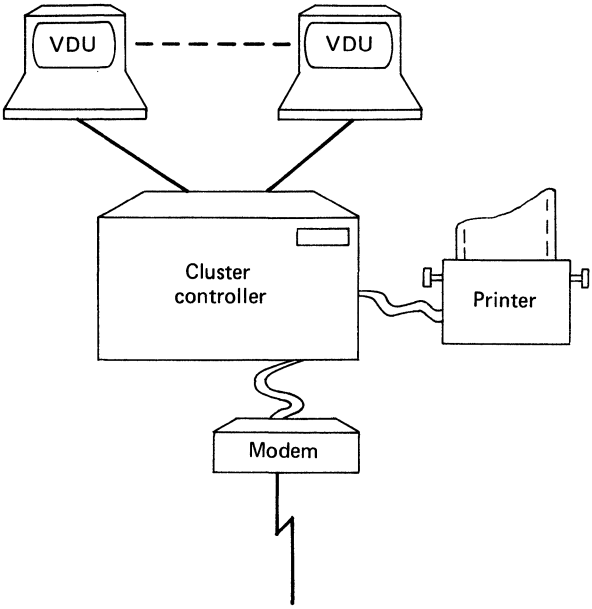


Figure 3.37
VDU cluster

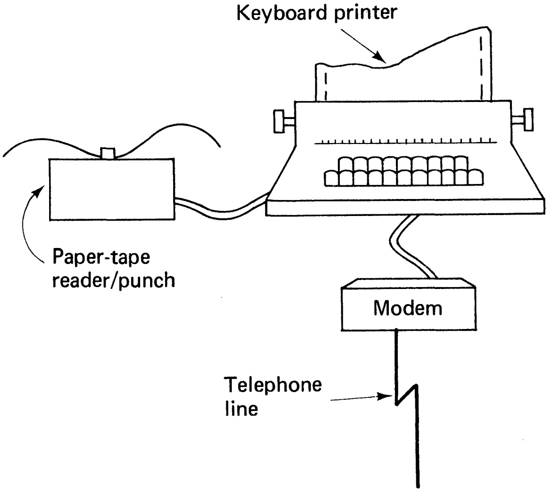


Figure 3.38
Paper-tape reader/punch on keyboard printer

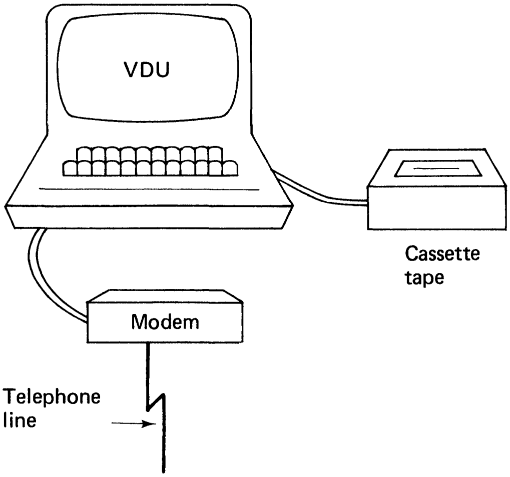


Figure 3.39
Cassette tape drive attached to VDU

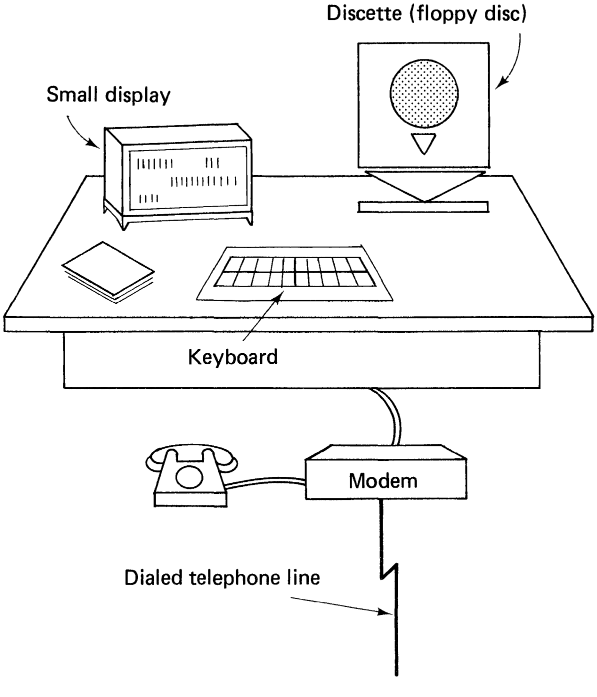


Figure 3.40
Data entry station (key-to-discette)

without incurring telephone charges. When the input is ready the user can call the computer and transmit the data in one continuous stream. Some terminals are designed to work in this fashion alone (figure 3.40). They generally consist of a keyboard, a small display for visually checking the input, and the device for recording the data on the magnetic medium. After off-line entry, the information stored on the cassettes, discettes, or whatever can be sent to the computer over a dialed telephone line. We prefer to call these units *Interactive batch terminals* because the input is interactive but the transmission is in batches.

Another category of terminal is the *remote batch entry* (or *remote job entry) terminal* (figure 3.41). This typically consists of a small controller to which are attached a card reader, a line printer, and sometimes other devices such as card punches and paper-tape readers. Remote batch terminals (RBTs) are generally used for the input of fairly lengthy reports, documents, and so forth. On a typical RBT the card reader might work at 300 cpm and the printer at 350 lpm. Faster speeds are available, but these are constrained by the speed of the communications link being used (2400, 4800, 7200, or 9600 bps are fairly common). The thing to watch about RBTs is that someone somewhere has to punch the cards (or whatever medium is being used), and keypunches can be as expensive as interactive terminals—in which case why bother with the RBT at all? These terminals are certainly declining in popularity in favor of the interactive and interactive batch

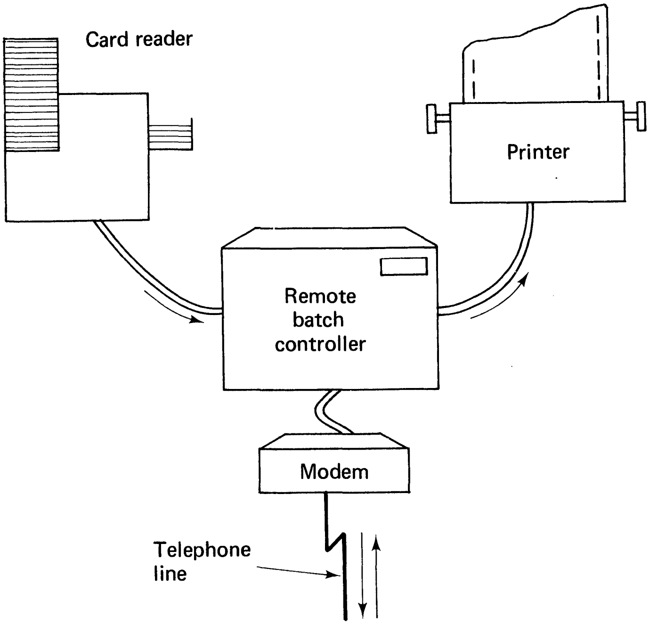


Figure 3.41
Remote batch terminal

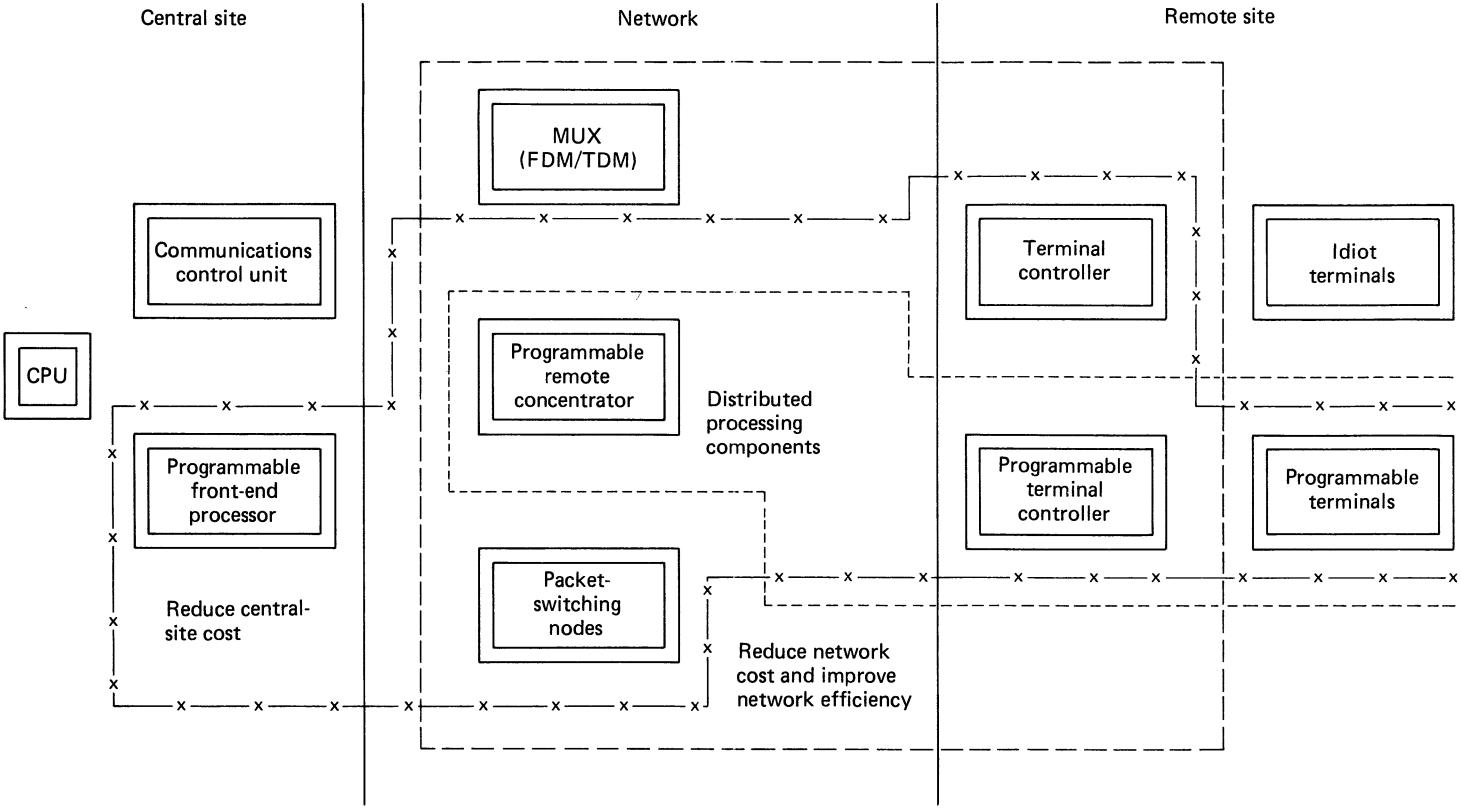

Central site
Network
Remote site
MUX (FDM/TDM)
Communications control unit
Terminal controller
Idiot terminals
CPU
Programmable remote concentrator
Distributed processing components
Programmable front-end processor
Programmable terminal controller
Programmable terminals
Packet-switching nodes
Reduce central-site cost
Reduce network cost and improve network efficiency

Figure 3.42
Data communications systems components

devices described above. However, the IBM 2780/3780 class of terminal remains important because the datalink control and device protocol for this has become a de facto standard for device interconnection.

The final category of terminals that need concern us here is the range of special-purpose systems designed to meet the needs of particular application areas. We will not describe them in detail, but terminals are available with special features for use in areas such as

Banking (cash dispensers)
Retail trade (point-of-sale terminals)
Production control (shop-floor data collection)
Credit control (credit card readers)
Stock exchange (market price displays)
Mobile radio terminals (for police cars, ships, etc.)

One increasingly important fact relating to all the terminals described in this section is that they can be either hardwired "unintelligent" devices or programmable. Some of them incorporate specially developed programmable units, but others make use of minicomputers and/or microcomputers. Intelligent terminals can be programmed to perform some low-level data processing functions such as input verification, display screen formatting, editing, and so on. This takes much of the workload off the central computer and provides faster response times, especially reaction to operator errors. Figure 3.42 is a schematic of the systems components described so far.

4

Transaction Processing and a Review of Minicomputer Software

The material in this chapter covers two areas; an introduction to the specific requirements of terminal-based systems and a general review of minicomputer software. In the first half of the chapter the software needed for terminal handling is discussed with passing references to minicomputer-specific solutions. The second part assumes that none of the general-system software requirements are new to the reader and reviews the state of the art with respect to minis, serving as a comparison with more traditional mainframe techniques. While we attempt to establish definitions of the more commonly used terms, we must state at the outset that there are almost as many different meanings of standard phrases as there are computer manufacturers!

Modes of Operation

With our interest in the characteristics of a computer system using terminals, we imply a particular mode of operation that contrasts with the long-established batch concepts. Before continuing with this theme, then, we shall review the various modes of operation of a computer system. In essence we are considering the characteristics presented by the combined hardware/operating system from the user's point of view.

Off-Line

Quite simply, we mean the use of devices not directly connected to the computer. Examples are off-line printers or computer-output microfilm to which data is transferred via a magnetic tape. Data preparation equipment such as card punches and key-to-discette stations are off-line input devices; an interactive key-to-tape system is off-line to the central computer but is in itself a simple on-line system.

On-Line

Equally simply, the use of directly connected devices. These may be simply "dumb" devices like a card reader or printer, or may be quite "smart" devices like VDUs. There is a common tendency to use the phrase "on-line" to mean interactive or transaction processing, which more strictly are subsets of all possible on-line systems.

Batch Processing

All input data is made available in sequential batches before processing commences. When the program is eventually started it continues until all data has been processed or an error has been detected. Production of output does not usually commence until processing has been completed. Once the program has been initiated there is no useful communication with the user until it has finished. Thus erroneous input data will not be detected until the batch has been completed, often requiring that the complete program be rerun after correction of the data. Often the batch job consists of a number of stages (transfer a copy of a deck of cards to sequential disc file, process the data and write to another sequential output file, and finally queue the output file for printing).

Transaction Processing

Data is presented (and accepted by the computer) as discrete data items, usually via a terminal, as requested by the user. In effect the terminal is the master device and the computer the slave, in contrast to batch processing, where the computer is master and the peripherals slaves. Processing of the input message must thus be initiated immediately, independent of any previous or subsequent activity on this or any other terminal. There is wide variety in the facilities available to implement transaction processing, ranging from simple asynchronous Teletype support to dedicated transaction processing monitors, as discussed in a later section. The term transaction processing is most commonly employed to suggest an environment in which operations are business transactions, (order processing or stock movement), where a special monitor is used to add to existing batch processing facilities of a computer.

Interactive or Conversational

Almost all transaction processing systems are interactive in nature. This also covers the broader class of sceintific systems used for computer-aided design, where each "transaction" can require a lot of processing compared to a commercial transaction. Nevertheless minicomputer systems are successfully using BASIC, designed for scientific work, for commercial transaction processing. There are other dedicated applications of interactive computing, such as program development using editors, debuggers, and so on, and information retrieval systems. In essence these latter systems are specially developed transaction processing application programs.

Direct Data Entry

The use of terminals to enter and validate data, but *not* to process it, is referred to as direct data entry (DDE). This is a very constrained use of transaction processing, common on the foreground/background types of operating system to be discussed later. Data is entered, validated (often assisted by a read-only reference file), and stored on a sequential file. When this data file is closed it can be used by a batch program for processing. There are some excellent examples of dedicated key-to-disc systems using this technique, but many small commercial systems have such poor transaction processing facilities, with inefficient single threading, that DDE is the total extent of their terminal handling ability. Given proper transaction processing facilities the choice of DDE or full processing of an input item is in the hands of the system analyst, not a limitation of the computer facilities.

Real-Time Processing

Strictly speaking, real-time processing is the realm of the process control engineer. It involves completely integrated support for event-driven (peripheral-initiated) inputs and time-related events (for instance, starting at 10.00 hours read an analog-to-digital converter every 20 milliseconds). In many ways the handling of an interactive terminal with fast response to input data is a simplified real-time job, so that the phrases real-time, on-line, and transaction processing are variously being used synonymously. In fact it is largely because of the interrupt-driven real-time operating systems used on minis that they have proved so effective in terminal handling.

Time-Sharing Systems

Time-slicing (discussed later) is a common method of achieving multiprogramming while preventing any one program from hogging the processor. It has been very successfully employed in a number of large-scale commercial and scientific installations where remote users attach their terminals via telephone lines, each user appearing to have exclusive use of the computer and its resources.

Teleprocessing

Teleprocessing implies the use of remote devices coupled to the computer via telecommunications facilities. On many machines there are quite distinct differences between the handling of local and remote terminals, as discussed below. IBM tends to use the expression to describe any system that involves the use of terminals.

Operator Communications

In the foregoing we have considered computer operation from the viewpoint of input data transactions. However, all computers need certain commands to define the required workload and the immediately required resources. In many systems a master console is used to supply the first line of command, while the individual users' requirements are communicated by a set of job control instructions that develop their data and/or programs. With a terminal-oriented system it is possible to enter job control information as well as data if the necessary command analyzer is provided with the system software. Thus a system may support terminals to allow multiple interactive access to the job control system to enter source programs, batches of data, and job commands even though actual processing occurs in batches. Alternatively, a system—typically a DDE system—will allow interactive data entry but the program must be initiated at a central console. The former would support interactive program development, the latter would not. Good operator control functions from multiple user terminals are major factors in the increased efficiency of a modern transaction processing system.

Teleprocessing Systems

Electronic devices can be interconnected in a number of ways, as described in chapter 3. The major components of the system (the CPU, memory, discs, and so on) are, for reasons of speed, interconnected by parallel wires and are all in close proximity. Slower-speed devices can be coupled more cheaply by using a single serial line, which can also employ special techniques to enable them to be situated a greater distance from the processor. Devices within about 1000 metres of the processor can be directly coupled; modulation techniques are used for longer distances. In general longer distances imply leaving the site containing the CPU so that PTT-supplied services are employed. With a variety of methods of connecting devices to a computer system, there is obviously a need for various software products to enable the user, through his application program, to access the device. Ideally the device-handling software should be so designed that the user programs have a common interface. This device independence is achieved by the user addressing a logical device. The system uses a set of tables to allocate this to the required handler for the specific physical device; these tables are updated as a user program is allocated to a physical terminal.

Strictly speaking, teleprocessing is concerned with support for remote devices, which are connected via telecommunications or data communications facilities. Since local terminals use serial techniques, however, their support requires a large subset of the remote terminal support system, so it is common to include both local and remote terminals under the heading of teleprocessing. The essential part of the teleprocessing software is that part of the system that handles the terminals and associated communications control to provide an interface to the transaction processing system.

On mainframe computers it is fundamentally difficult to handle character-at-a-time input so that VDUs are buffered. Data is stored and edited at the terminal and transmitted synchronously in blocks. Control of which terminal is currently transmitting or receiving data is determined by a polling routine maintained by the terminal handlers. The potential economies of relieving the CPU of the responsibility of checking and storing each character are offset by the polling overheads and high buffer requirements in processor memory. In many cases this is unnecessarily compounded by poor data packing techniques (the message FRED is transmitted as F,R,E,D,End of text, and 1915 nulls!).

The fast-interrupt handling of minis makes it possible to use asynchronous Teletype-compatible VDUs on a character-at-a-time basis, which is significantly cheaper. However, the use of minis with a large number of terminals and the desire for alternative screen formatting are leading to increasing use of buffered VDUs and DMA controllers, as will be discussed in chapter 5. Thus while the terminal-handling software on a mini is essentially simple, there is little inherent support for polled and multidrop terminals at present.

On a mainframe computer the data from terminals is routed through a data channel. To multiplex multiple terminals and to handle the communications functions a terminal controller connected to the data channel is required. This can be a *local terminal controller* or a *communications control unit* (CCU) which can support mixed local and remote terminals. These devices perform a number of functions described later, but their logical ability is predefined—that is, they are hardwired devices. The same facilities can be provided with much improved flexibility by using a *front-end processor* (FEP). Device commands from the channel control are interpreted by a program in the FEP, in the same way as a CCU, but they can be tailored to suit. It is also possible that the control programs in the FEP could be downloaded from the main processor on start-up. There are some very sophisticated single-line programmable controllers available now on single LSI chips.

On a mini the I/O bus effectively acts as a multiplexor so that simple terminals need only a cheap interface card as the equivalent of a CCU. A 16-channel device can cost as low as $2,000.

Mainframe or mini, any supported physical terminal interface requires a software driver to match its specific characteristics and to present a device-independent interface to the user programs. For want of a better phrase we have called this combination of terminal drivers, communications line controllers, and other components the terminal subsystem (TSS), as shown in figure 4.1.

Transaction Processing

Transaction processing implies an ability to support terminals. It also implies an ability to accept an input message, to process that message, and to return an answer in the minimum possible time. In chapter 6 the specific requirements of a software system for handling transactions in an efficient manner are explained. It must be stressed that with transaction processing the objective must be to pass the transaction through the system as quickly as possible. Minimum residency is the design objective. Contrast this with a batch system, which must be designed for optimum system efficiency in terms of total throughput. To achieve this, a specific job may be held back for quite long periods. It must also be remembered that there will be multiple transactions in the system at any one time, possibly of differing lengths. One transaction should not be held up while another is awaiting resources but should be able to overtake a slower one. This leads to the concepts of *multitasking* or *multithreading* mentioned later in this chapter and expanded in chapter 6. Note that multitasking is common in minis but is equally uncommon on small mainframes, for which reason transaction processing is constrained to simple direct data entry, a most unsatisfactory situation.

Figure 4.1 shows a schematic diagram of the software structure for a system with both batch and transaction processing facilities. The transaction processing modules may be written in COBOL or a similar language, and are controlled by the transaction processing monitor (TPM). The TPM ideally provides a device-independent interface to the physical terminal via the TSS. Essentially the TPM and all transaction processing programs share one partition, the batch programs another. The TPM provides the multitasking facilities unavailable in the basic batch operating system (OS). The TPM further provides multithreaded access to the disc file system, queueing requests to be handled by standard-access software. Note that terminal handlers and user programs could be run in a batch partition, but the response would be governed by the normal system priorities and queueing, so that one transaction awaiting response to a file request, say, will hold up all other transactions. For reference, readers with IBM experience can recognize CICS as a TPM and VTAM as a TSS. Figure 4.2 stresses the transaction processing software structure.

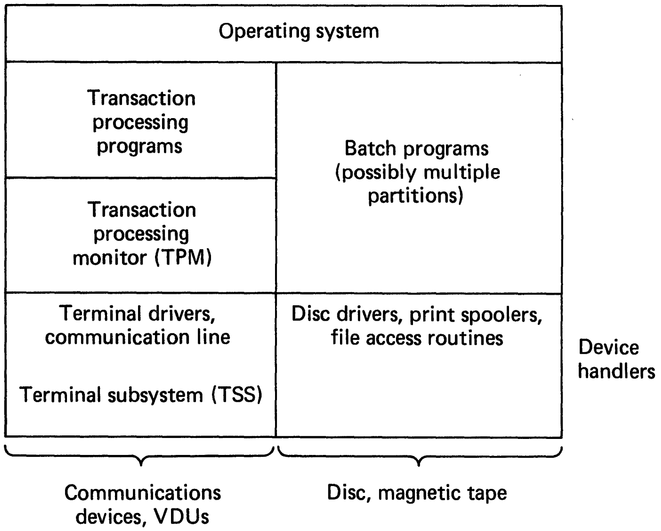
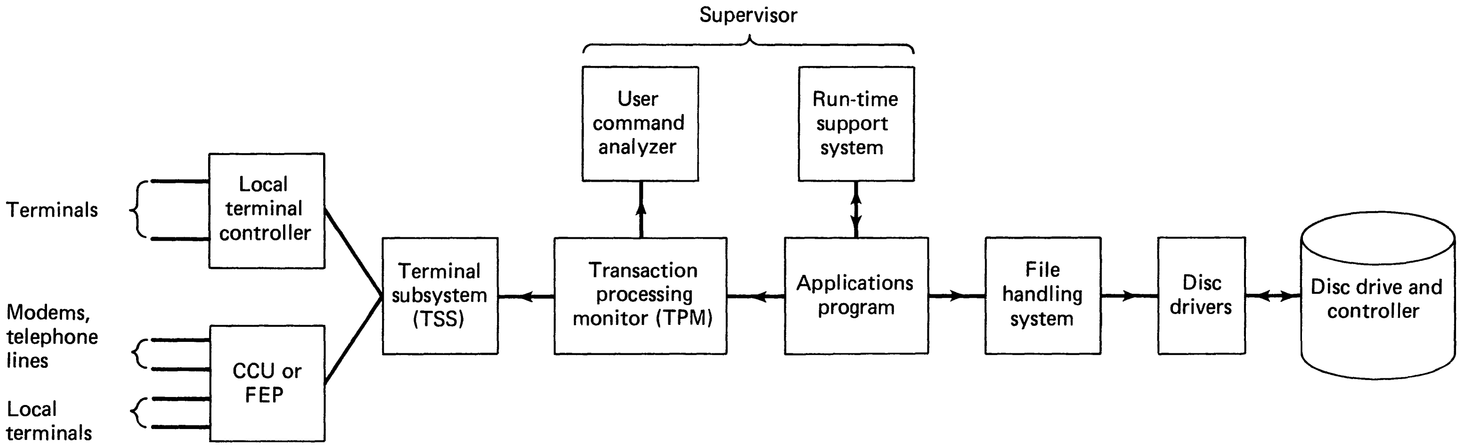

Operating system
Transaction processing programs
Transaction processing monitor (TPM)
Batch programs (possibly multiple partitions)
Terminal drivers, communication line
Terminal subsystem (TSS)
Disc drivers, print spoolers, file access routines
Device handlers
Communications devices, VDUs
Disc, magnetic tape

Figure 4.1
Generalized software system for a combined batch and transaction processing system



CCU Communications control unit
FEP Front-end processor

Figure 4.2
Structure of transaction processing software. With a standard minicomputer the TSS and TPM, with limited facilities, will be integral parts of the operating system

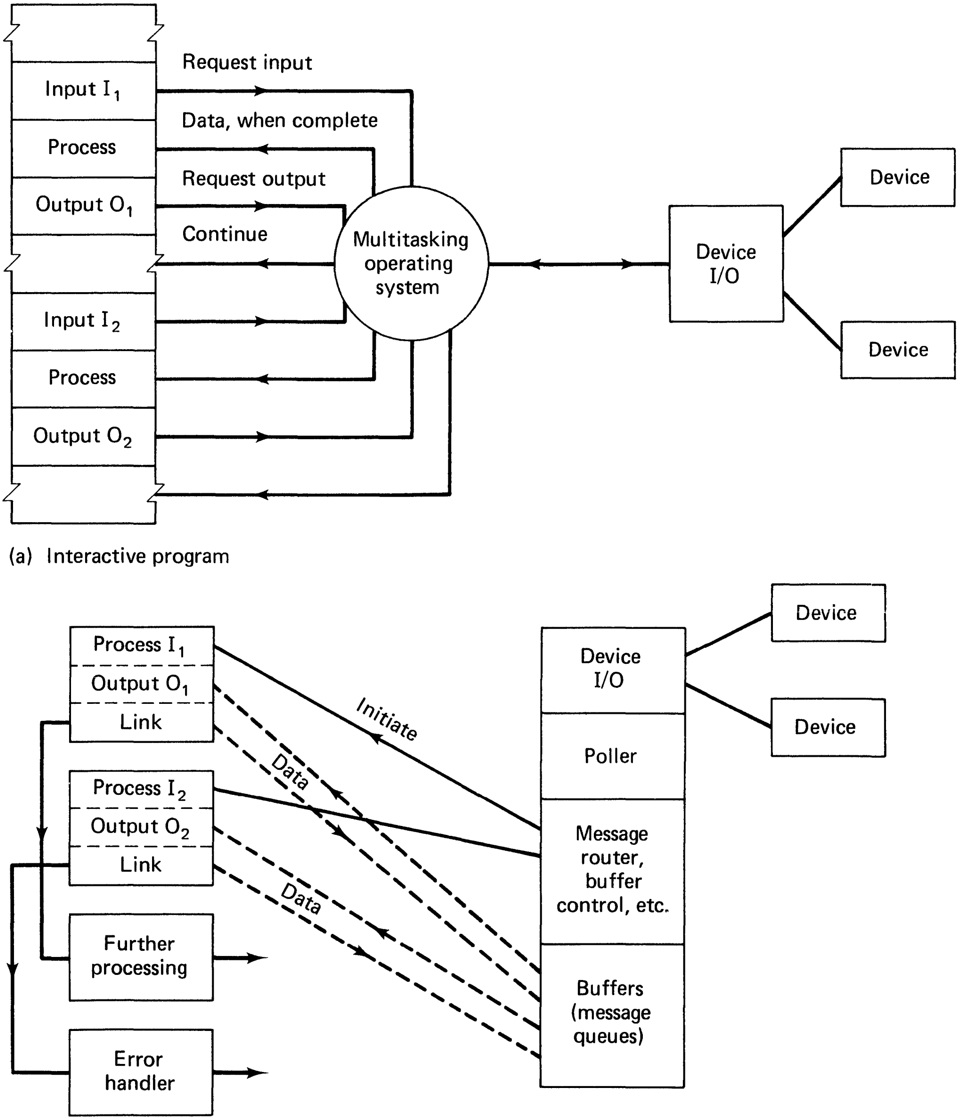
With the process control history of the minicomputer, support for multiple character-at-a-time asynchronous terminals is a standard operating system feature. They are interrupt driven and can take advantage of the inherent multitasking features of such operating systems. The CPU undertakes the execution of routines for character checking, buffering, and routing as well as the processing work. User programs interface directly with the terminal handlers via supervisor calls to the standard operating system, without the controlling intermediary of a TPM. This concept is shown schematically in figure 4.3(a).

In figure 4.3(a) the whole application program, which includes both processing and I/O commands, is loaded. An I/O command is serviced by the operating system, which deschedules the user program until the request is completed, at which time the continuation of the user program is rescheduled. The user program is effectively master of the I/O system; data will only be accepted from a keyboard when it has been requested. BASIC, Interactive-COBOL and its variants are the common language processors used to generate such applications programs.

By contrast, in figure 4.3(b) processing activity is initiated by the terminal. The TPM interrogates the message arriving from the terminal and associates it with a specific processing module. The modules themselves process specific units of data and create output messages; the TPM routes messages from queues to and from the appropriate terminals. Since there is no direct I/O the processing modules can be written in a batch language, typically COBOL, with specialized languages for defining TPM functions, formatting screens, and so on. In figure 4.3(a) the operating system provides the essential multitasking, while the TPM is required to meet these requirements in figure 4.3(b).

With the Interactive-COBOL/BASIC technique the complete user program must be loaded, including infrequently accessed modules. With a TPM only the module immediately required need be in memory; code sharing is much easier in principle. A TPM should therefore reduce memory requirements. Further, the high modularity of the TPM concept allows easier modification (say, a screen layout can be changed by modifying the screen formatting module only, whereas an interactive program needs modifying and the whole recompiling). Thus the interactive program technique is commonly encountered only on minis, while TPMs are found commonly on mainframes but increasingly on minis. Indeed, a standard mini operating system is a good vehicle for supporting a more specific TPM.

A number of specialized "on-line" systems do not support the batch partition. Pseudo batch programs can be run by treating them as on-line programs with no I/O and relying on a time-slicing control to stop hogging of the processor. This technique locks up a VDU not needed by the batch program. Improved systems allow the batch program, once initiated, to be transferred to a pseudo terminal (a disc file for reporting errors, completion, and so on), freeing the VDU. Multiple batch programs can thus be active simultaneously although there are limited batch job-control stream facilities. The problem with these systems, examples of which are given in chapter 8, is that the file access system is designed for on-line work, with extremely inefficient batch processing as a result (no sort/merge, for instance).



(a) Interactive program

(b) Transaction processing monitor

Figure 4.3
Alternative techniques for terminal handling

Two fundamentally different approaches to terminal dialogs appear. With the standard mini systems with asynchronous VDUs the question-and-answer technique is used. The one-user program handles multiple transactions, normally one per line on the screen, which are initiated and hence controlled and routed by the calling program; the program clearly identifies an incoming message as a response to a sepcific request for input. With a TPM and buffered VDUs, the polling system controls input of messages after they have been prepared and edited at the terminal. Thus a received message is associated with a particular screen format to help the TPM isolate the fields and route the data to the appropriate processing subprogram.

The advantages of the standard mini system are low cost, simplicity, and the need for one programming language with integrated screen handling and processing. It offers a minimum of message-passing overheads. The advantages of the TPM are that separation of the screen handling and the processing into sections improves portability in the sense of modifying screen layouts for different languages. With buffered VDUs programming aids are available for generating the screen dialog interactively and rapidly. As mentioned, only the immediate subset of the program required need be in memory; the potential for reducing memory requirements by designing shareable code segments should also be easier. Many mini systems load separate copies of programs for each terminal, even if they are the same job, a factor offset, however, by vastly cheaper memory. With the increasing use of microprocessors intelligent VDUs can be devised to take advantage of the simple asynchronous mini interfaces and handling software for the best of both worlds. Buffered (block mode) asynchronous VDUs can cause overload problems in the handlers, however, since with no polling to control transmissions, multiple blocks could arrive simultaneously.

Transaction Processing and Terminal Handling Functions

The introduction of transaction and teleprocessing techniques in a computer system requires a number of facilities not normally found in a batch processing environment. The software requirements identified in the previous sections are summarized in figure 4.2; the facilities required are listed in figure 4.4. Because any function can in fact be implemented in a combination of hardwired logic, software, or firmware (microprogramming), there is no clear definition regarding which part of the system handles which function; figure 4.4 therefore includes an indication of how each function should be implemented. Each item is explained below.

Data Communications Input/Output

Compared with I/O between a central processor and computer peripherals such as disc and tape, the transmission and reception of data over telecommunications facilities is relatively crude. The operations to be performed include:

—Conversion of outgoing data from parallel form to serial form and the reverse for data being received

—Handling of the interface with the data communications facilities (CCITT V-series, RS-232), including auto-calling and auto-connection

Function	*Desirably performed by*	*Less desirably performed by*
Data communications I/O	CCU	TSS
Terminal drivers	TSS	OS
Code conversion	CCU	TSS
Error handling	CCU/TSS	TPM/AP
Message buffering	CCU/TSS/TPM	TPM/AP
Message queueing	TSS	AP
Network management	TSS	AP
Message preprocessing	TPM	AP
Message postprocessing	TPM	AP
Message routing	TSS/TPM	AP
Task management	TPM	AP/OS
Program management	TPM	OS
Storage management	TPM	OS
Timer control	TPM	OS
Start-up/shut-down	TSS/TPM	AP/OS
Terminal user command monitor	TPM	AP/OS
Terminal I/O control	TPM/AP	AP
File management	TPM	AP
Transaction logging	TPM	AP
Failsafety/recovery	TPM	AP
Debugging aids	TPM	AP/OS
Statistics	TPM	AP

TPM: Transaction processing monitor
TSS: Terminal subsystem
CCU: Communication control unit (or front-end processor)
AP: Applications program
OS: Operating system

Figure 4.4
Transaction processing and communications functions

—Framing of characters with start bits and stop bits in asynchronous communications

—Insertion and recognition/deletion of SYN characters in synchronous communications

—Identification of (and acting upon) control characters in incoming data streams (ETX, EOT)

Traditionally these functions have been performed in a combination of hardware and software. There is now increasing use of microprocessors and special LSI chips in this interface area.

Terminal Drivers

These reentrant routines, one per type of terminal supported, act as an interface between the communications I/O and the TPM. They are table-driven to allow a user program to be attached to any terminal at run time, with a common interface to support device independence wherever possible.

Code Conversion

This is the conversion between the code used for data transmission and that used within the computer. It may be a complete conversion (ASCII-7 to EBCDIC) or a partial one (transposing lowercase characters). Conversion is usually done in software although some processors have special instructions to speed up the operation (a translate instruction on the GEC 4000 and move byte and translate on some MODCOMP minicomputers).

Error Handling

This involves a combination of low-level and high-level functions. At a low level, hardware is used to detect errors through parity (redundancy) checking techniques. At a higher level, software is generally used to determine and initiate the action to be taken when an error is detected. Software may also be used for more sophisticated error handling techniques such as transmission block numbering. Error service routines can sometimes be hardware interrupt initiated.

Message Buffering

One of the major characteristics of on-line systems is the discrepancy between the speed of the lines and the speed of the CPU. One implication of this is that terminal operations must be overlapped with processing (and, indeed, with the other terminals). Furthermore, the input of messages to a transaction processing system is unpredictable. This means that the system needs to have some mechanism that makes available main storage buffers for the output (or input) of messages as and when required.

Message Queueing

Situations can often occur where messages are arriving at the CPU faster than the application programs are able to process them. Similarly, a number of messages may need to be sent to a terminal that is already busy. These situations cause queues to form in the system and a facility needs to be provided to manage these queues, either in main storage or a disc, according to predetermined procedures and priorities.

Network Management

A teleprocessing network is rarely a static system. Terminal users join and leave the service, lines fail, and bugs can be discovered in the software. In order that these occurrences have the minimum effect on the total system, the computer operator (or network manager) needs facilities to shut down lines, suspend applications tasks, send messages to operators, and so on.

Message Preprocessing

Messages received from terminals are often in a format difficult for the application programs to handle. They may contain embedded control characters, have variable-length fields, and even variable occurrence of fields within the message. Although programs written in assembler-level code can be developed to work at this level, the problem is much more acute for high-level languages such as COBOL and PL/1. For these reasons some systems incorporate a message preprocessor, which converts incoming messages into a fixed format prior to handing them over to the application program. The most obvious example is conversion from an ASCII character string to an internal binary number representation.

Message Postprocessing

The problem that dictates a need for message preprocessing also dictates a need for a postprocessor to convert from an easy-to-program internal format into a character string suitable for transmission to a terminal.

Message Routing

When a message arrives in the system, it is necessary to determine which application program will process it. This can be done in a number of ways, some of which are described in chapter 6.

Task Management

In simple, low-volume transaction processing systems, it is possible to use single-thread programming. Single threading occurs when processing for a particular message dominates the CPU from the beginning to the end of the processing cycle for that message (even when I/O activity causes the CPU to idle). A more efficient arrangement employs multithreading (or multitasking) techniques whereby, if the processing of a particular message has to be suspended, another task can be activated. In this way the CPU is used more efficiently and the average turnaround time for each message can be reduced. The job of scheduling tasks according to I/O activity, task priority, and so on, is known as task management, also discussed in chapter 6.

Program Management

A program (or subprogram) may be initiated by any of a number of events. These include a message arriving from a terminal, a request from another program, a command from the system console, an error condition, and a time-related event. The program management routine will determine which program needs to be initiated. If the program is already resident in main storage, then the request will be queued or a task initiated (see Task Management above). If the required program is not memory-resident, then the management routine will locate the program in the relevant program library and when adequate main store is available, load it and start execution.

Storage Management

In spite of claims by some computer suppliers that main storage limitations have become nonexistent and prices so low as to be neglible, in most real-world situations applications programmers have to work within some storage restrictions, if only for reasons of efficiency and economy (reason enough). The random nature of transaction processing message traffic can often lead to the inefficient use of storage. For example, let us take the case of a system which has connected to it twenty terminals, each of which requires a 1 KB I/O area when active. If, however, the mean terminal occupancy is only 50 percent, then the actual storage requirement is 10 KB rather than 20 KB. (The optimum space can be estimated using the analytical techniques based on a queuing theory.) In order to share storage in this fashion, the programmer must be able to obtain working storage as it is needed and then return it to the common pool when no longer required. Teletype-compatible VDUs are less demanding (100 character buffers), so fixed buffers are common on minis. Pooling of disc buffers is rare; they are often included in the application program. Storage management techniques will be referred to again in chapter 6.

Time Control

As a generalization, it could be said that the only timing problem in batch systems is getting the job completed in the smallest possible elapsed time. The timing problem in on-line environments is much more varied. The fact that the processor operates at speeds so much in excess of the human terminal operators means that it is necessary to be able to initiate tasks (for example) after a specified elapsed time or at a particular time of day. Time slicing is also used in task control.

Start-Up/Shut-Down

The start-up of a transaction processing system is a relatively simple affair, usually including such operations as loading system programs and broadcasting "good morning" messages to the terminals. Shut-down is somewhat more complex because terminal operators may be in the middle of interactive operations. In order, therefore, to make the shut-down as orderly as possible and without loss of data, the control program may transmit warning messages at predetermined intervals and then, when close-down is inevitable, stop accepting input messages but allow output messages through until the system is clear of traffic. Start-up and shut-down operations are normally initiated via commands from the system console, but they may be time-of-day activated.

Terminal-User Command Monitor

It is desirable that the terminal user should be able to issue commands to the system as well as to interact with a user program. Thus a terminal is initially attached to a command analyzer which enables a monitor to respond to what is effectively a terminal-oriented job control language (JCL). Commands can be issued to sign on, sign off, check passwords, and run user programs. More sophisticated systems also allow access to systems utilities, such as the editor and compiler, to enable interactive program development.

Terminal I/O Control

Low-level data communications functions (as described earlier in this section) should be of no concern to the application programmer. The programmer does, however, need to be able to initiate terminal I/O operations at a logical level (for example, write a message to a terminal, read a message, and so on). These requests for system services are usually implemented as macros and/or subprogram calls. Many of the operations will be carried out by the relevant control program, and this will include such operations as the handling of queues, automatically polling terminals for new input messages, undertaking broadcast functions, and initiating pre- and postprocessing. It should also enable user reference to channel numbers to be associated with specific physical terminals at run time.

File Management

The reader might ask why file management need be considered in a discussion on communications functions. The importance of multitasking concepts in achieving efficient utilization of the processor in a transaction processing system will become increasingly clear. If a task is suspended for the duration of a disc file read, then control must be transferred to another task. But if a number of tasks are accessing the same file, then the file management routine itself needs to be shared. In the case of some control programs (for instance IBM's CICS) task rescheduling is only carried out when a macro is used to return to the TPM for some service. For this reason file I/O operations may be requested by issuing TPM calls rather than a direct invocation of the file manager itself, a complication avoided by the more integrated mini executives. The file handling routines, part of the OS, map user-generated logical requests into physical disc sectors. The routines include buffers and tables to aid the mapping and to check access attributes, and to queue and schedule requests. The interrupt-driven drivers handle the disc controllers.

Transaction Logging

To aid failure recovery a file is maintained to sequentially log each transaction. This can be used by automatic recovery programs to rebuild files from archive copies after a crash.

Failsafety/Recovery

Failsafety is concerned primarily with protection against the failure of system components, including programs, data, and lines. Should a failure occur some action must be taken to minimize its effect and to facilitate subsequent recovery with minimum loss. Failsafety and recovery are very difficult to generalize without incurring large overheads, and although some control programs such as IBM's IMS do offer such facilities, more often they have to be designed into the system and implemented by the application programmer. This is particularly true with minis.

Debugging and Program Development Aids

Many control programs incorporate special facilities to assist in the development and debugging of application programs. These include the familiar trace and dump services but also routines for feeding input messages to individual programs from disc files or unit record equipment as though they had come from a terminal. Output messages are routed to a spool file or a printer. This enables the programmer to test his programs prior to delivery of the terminal and also enables him to test more efficiently by getting through higher volumes of test data without having to key it at a terminal. Other facilities enable the programs to run without multitasking and in some cases to checkpoint tasks during execution so that intermediate data may be examined.

Statistics

The regular production of meaningful statistics is fundamental to the performance monitoring and quality control of transaction processing systems. It is necessary to record for subsequent analysis (perhaps not continually) the time and source of both good and bad transmission blocks, the frequency of access to various files, and the rate of initiation of subprograms. For tuning purposes it may be useful to be able to obtain the average execution time of each software component of the system (if this can be done without distorting the results). Depending on the degree of sophistication of the TPM many of these functions may be provided as standard, but this is relatively unusual in minis.

Overview of System Software

It is stating the obvious, but a glance at figure 4.5 shows that there are no superficial differences between minicomputer and mainframe software. We will assume that the reader has a general knowledge of all the items mentioned on the figure and will concentrate on current and future mini technology.

Binary Code

All software is simply an ordered sequence of binary machine instructions and data. A specific piece of code may be classified as:

Absolute. Must be loaded into specific memory locations

Relocatable. Can be loaded anywhere into virtual memory. It is created with an assumed base address of zero and embedded addresses are corrected by the loader at the time of loading. Not to be confused with dynamic relocation of virtual addresses into physical addresses using mapping registers.

Position-independent (PIC). All address references are program counter relative so that it can be loaded anywhere without address modification.

Linking

A complete program may consist of a combination of a number of object code modules created independently. These must be precombined into a loadable image, with all "global" addresses resolved when the program is *installed* or *bid*. It can then be loaded into memory and executed by a request to run the program without linking delays—most important if the program may be swapped during its execution. More sophisticated linkers, known as *program builders* or *task builders,* are required to generate overlays.

Reentrancy

In a multiterminal system there will always be some degree of commonality between programs. Memory requirements can therefore be materially reduced if the instruction code can be shared by common routines. This can be implemented at a number of levels, as shown in figure 4.6. Figure 4.6(a) shows a typical software layout as previously illustrated in figure 4.1. The user program is shown separated into three nominal parts, the user screen formats (including field specification and data validation), the data processing, and the user file requirements. Figure 4.6(b) shows a nonreentrant system where two complete software sets are employed. It is unacceptable that the general-purpose system part of the software should not be reentrant; figure 4.6(c) shows the more typical arrangement. The actual user-specific code is not reentrant.

Figure 4.6(d) shows a fully reentrant system; of course if the two programs are dissimilar, then system (c) must be used. Here only the data areas are user-

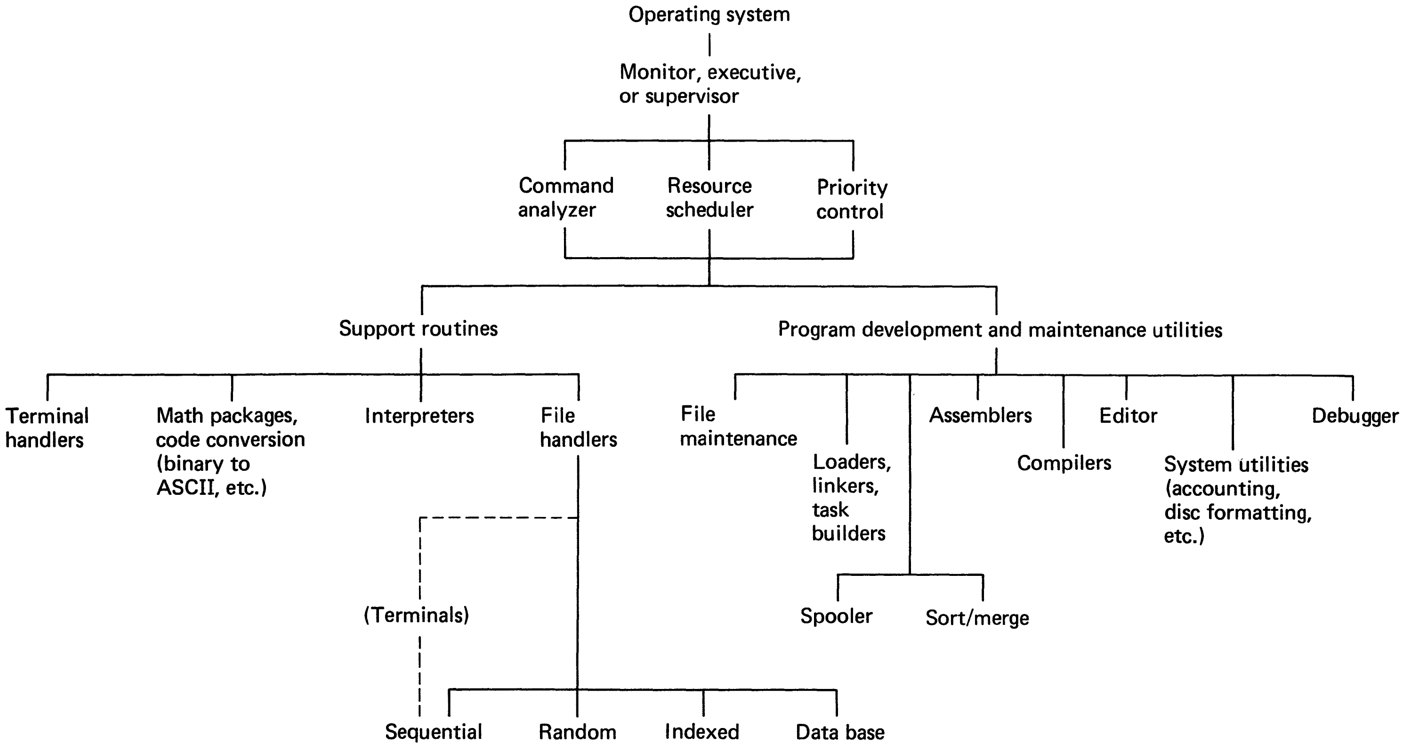


Figure 4.5
System software modules

specific. Buffers for both terminal and disc I/O can be allocated as shown in figure 4.6(d), loaded directly by the handling routines. Alternatively, the system may use separate buffers in the terminal and file handling routines, copying to the user data areas when they are ready for processing. To avoid excessive storage the buffers in the handlers can form a pool, dynamically allocated as requested.

Minicomputer architecture should readily lend itself to full reentrancy. This is readily achieved using macro assemblers for user programming, but seldom when high-level language compilers are employed; there is distinct room for improvement in this area. A halfway solution is achieved by using an interpreter as shown in figure 4.6(e). Multiple copies of the user code are still carried but these are relatively small, as they are in the high-level intermediate code that calls the interpreter. The interpreter itself is reentrant. Other basic support software such as mathematical routines should also be reentrant.

Some nonreentrant code modules can be shared by multiple users provided they are not interrupted. Typically they have embedded data work areas. Such modules, once initiated, must be run to completion before other users can gain control; they are commonly called *serially reuseable*.

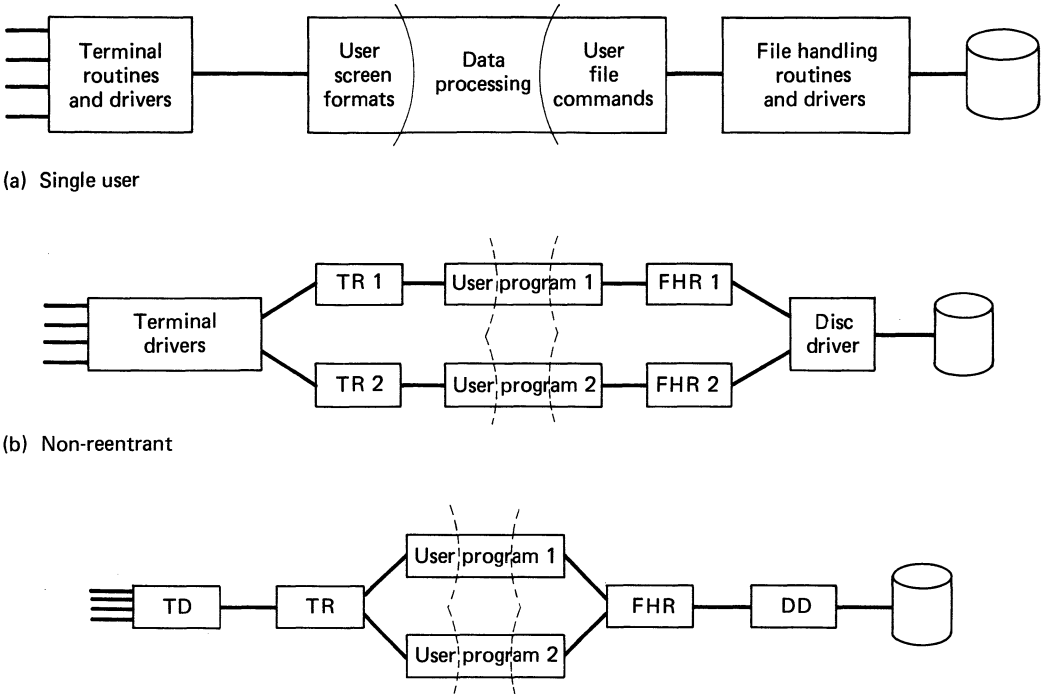
Interpreters and Microcode

Let us now consider figure 4.7. All programs start off with some form of mnemonic coding. In general the source code is compiled (or assembled) into a machine-executable object code, which is then independently executed. Some run-time debugging aids are available to allow the executing code to pause and display the current values of variables; the debugging traps are then purged from the checked code.

The alternative is to precompile the source code into an intermediate code, usually using push-pop code, and then to execute that code with the aid of an *interpreter.* Since it is far easier to relate the variable back to the source code, this technique was pioneered for interactive languages, particularly BASIC. It also has advantages in time-sharing systems when one copy of the interpreter is shared among multiple users.

With the advent of microprogramming the definition of "machine code" becomes less rigid, and thus the boundaries A and B shown in figure 4.7 can vary widely. The earlier BASIC interpreters worked on the source code direct; modern ones are far more efficient. Interpreters for COBOL and other high-level language precompilers are now in use. If more of the interpreter can be shifted into microcode then run-time execution time will be reduced and memory otherwise occupied by the interpreter will be freed. A more diverse application of interpreter/microcode is to emulate the instruction set of another machine.

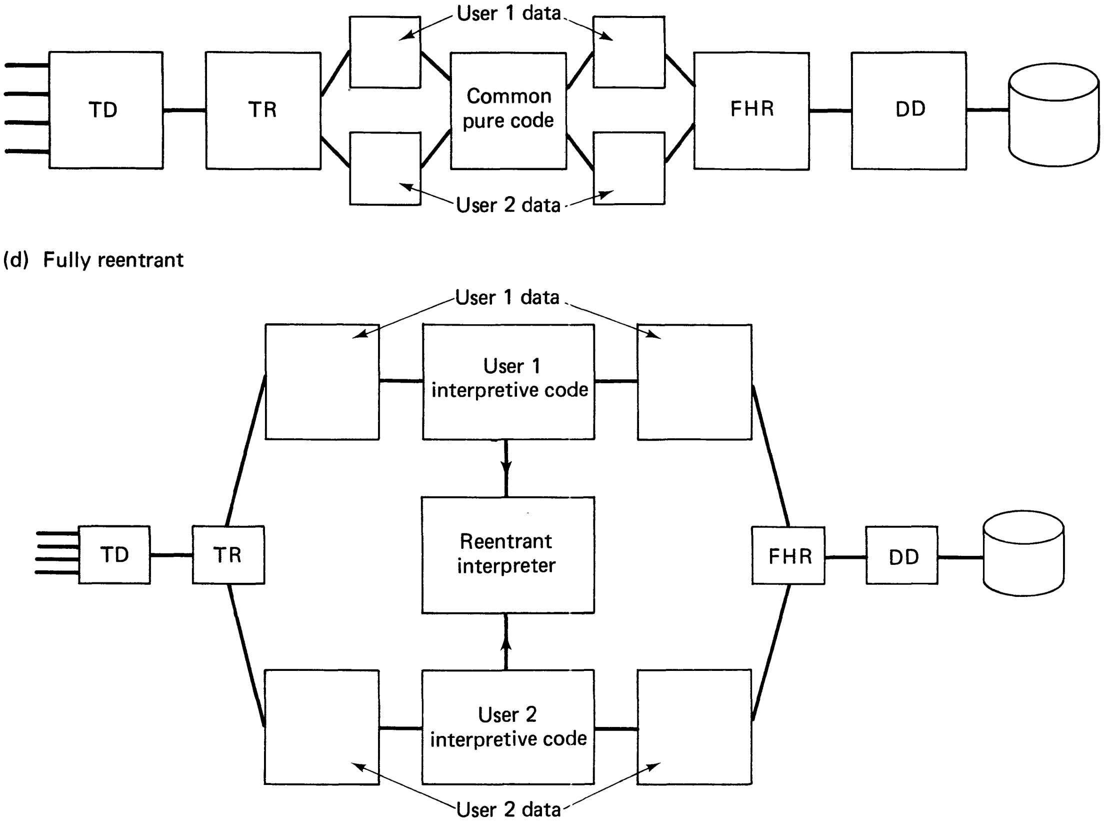
One possible future application of the interpreter concept is shown in figure 4.8. Ideally, all source languages compile to one target code. This target code is fully implemented in microcode on a large machine but by an interpreter and microcode on a smaller machine. Thus all application programs could run on all machines in the family.



(a) Single user

(b) Non-reentrant

(c) Reentrant system, separate user programs



(d) Fully reentrant

(e) Interpretive system

Figure 4.6
Sharing of software modules.

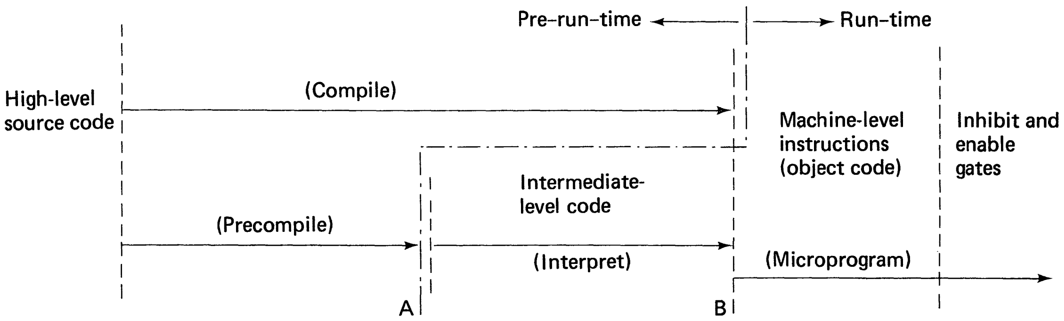


Figure 4.7
Compilers versus interpreters

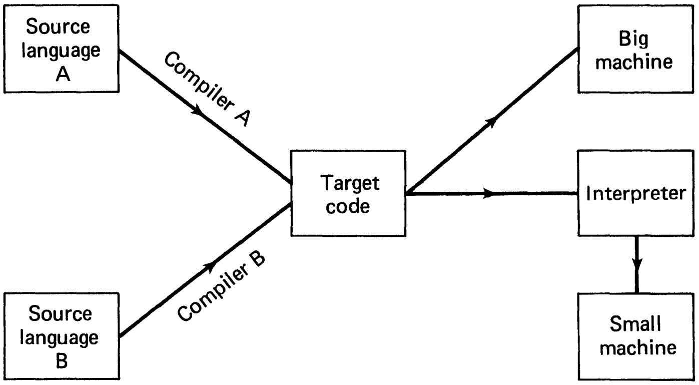


Figure 4.8
Use of common intermediate code on various machines

Tasks, Programs, and Procedures

Before continuing with our discussion of system software, we should define some of the terms used, particularly since there are distinct differences between manufacturers. The actual words don't matter, as long as we establish a consistent base for this text.

A *task* is a complete asynchronous logical path used during the execution of a program. Two tasks may in fact share the same piece of code, each with its own priority. Thus a task is a dynamic entity, allowing *allocation* of resources *during* execution, avoiding preassignment as in a batch system. A task is a complete logical routine in itself, which normally terminates with a request for supervisor services. There are various definitions of a task, some others of which are discussed in chapter 6.

A *program* is a piece of code that performs a given function. It has a distinct start and end. A program can be written as a self-contained entity; however, such routine operations as reading data from terminals are part of the operating system, so that a program normally consists of a set of tasks with the implied use of OS services. The program defines the logic with which the OS can keep the individual tasks in synchronism.

Note from the above definitions that a program must be initiated by an operator, while tasks are initiated by significant run-time events, controlled by the OS. A program is a static entity, a task a dynamic one.

A *procedure* is a piece of code that defines a sequence of operations to be performed on specified data (cf. a procedure division in a COBOL program). Working data is separated from the procedure. If local variables are also separate so that the procedure contains only instructions and constants, then it is reentrant and can be shared by multiple programs. Note that on some mainframes the word procedure is used to define a prespecified sequence of job control statements.

An interrelated sequence of programs, together with identification of files, requests for use of compilers, and so on, is loosely termed a *job*. A *job control language* is provided to communicate related information to the operating system monitor. In a batch system sequences of JCL statements are stored and accessed by the monitor to control the machine workload; in an interactive system JCL commands (although seldom so called) can be entered through the same terminal to be used for data entry. The best systems will support batch streams and interactive JCL. Note, however, that support for terminal I/O devices does not necessarily imply interactive JCL (for example the Burroughs B700 and Data General RDOS). Such systems still need a dedicated console. Further, many minis support excellent interactive JCL but limited batch facilities; often a terminal is tied up by a batch job.

Operating Systems

In general we will here consider the operating system to include basic transaction processing features. Operating systems are discussed in detail in chapters 6, 7, and 8; in this section we will simply establish a few definitions. Figure 4.9 shows

- Job scheduling
- Priority control
- Task management
- Job control language interpretation, command analysis, operator interface
- Error detection and handling
- I/O handling
- Interrupt handling
- Resource control
  - peripherals
  - memory
- Protection
  - access and file ownership control
  - memory mapping
- Message routing, interprocess communication
- Buffer allocation
  - fixed
  - dynamic
- Accounting of resources
  - error logging and device utilization
  - transaction logs and recovery utilities

Figure 4.9
Functions of an operating system

a summary of the functions of the OS required to implement the communications concepts discussed in the Transaction Processing section, as well as batch processing. Note that the primary objective is to remove direct control of the system resources, both peripherals and memory, away from the user's program. In this way the chore of writing drivers is avoided and the standard ones can be shared by multiple users. Equally important is the fact that conflicting requirements for resources can be resolved. The use of an operating system has unavoidable overheads in looking up tables, switching to supervisor space, and so on, but the control and logical independence are essential in practice.

Access to the supervisor services provided by the operating system is of two kinds, that initiated by the operator from the console and that called by the user program at run-time. The command string analyzer interprets console requests; regular sequences of commands can be stored and executed from disc files which can extend to more conventional batch control streams. On many minis in a multiuser environment any terminal can initiate a background batch job and then *detach* from it to run a foreground job giving multiple concurrent streams of batch programs along with on-line jobs. Tasks with interactive terminal work should be given a high priority. Run-time access to the OS services is achieved by *supervisor calls* (SVC). These are usually macros in the user code which expand to jumps to the OS with appropriate parameter tables defining the service required and any data such as buffer addresses and word counts. The supervisor responds to the SVC by initiating the appropriate service routine and updating the task status tables; there is always an overhead on a supervisor call compared to a procedure or subroutine call. SVCs are commonly implemented on minis by interrupt-emulating instructions.

In general we are interested only in disc operating systems, that is for computer systems that use a fast-access bulk store to store programs and data not currently required for processing. For readers with a mainframe computer background this may seem a statement of the obvious, but simple minicomputer and microcomputer systems still use paper tape and cassettes as bulk storage. While these are really low-cost laboratory systems we have a definite interest in memory-only operating systems for dedicated applications. Thus in developing, say, a data concentrator, a disc-based system will be used for program development and testing but the final system only requires a subset of the full OS to act as a run-time monitor, the program being "burnt" into a ROM or loaded from a cassette or tape. Returning to disc-based systems, a variety of characteristics serve to differentiate one system from another and in many senses make a system more suitable for one type of work than another. The more common characteristics, listed in figure 4.10, can now be reviewed.

Single user versus multiuser. A single-user system can run one program at a time from a single console terminal. A multiuser system (or multiaccess) allows multiple terminals to initiate jobs concurrently. A multiuser system must have far more secure file access and security features such as passwords and user identification codes (UIC). One terminal is always required as a console, which is the prime means of communication with the command analyzer and monitor; with

- Single user versus multiuser (multiaccess)
- Foreground/background
- Multitasking
  - single threading versus multithreading
- Multiprogramming
  - Job mix: core resident versus swapping
  - Swapping by program, overlaying, paging and segmentation
  - Shared code and data
  - Virtual systems
  - Multipartitions
  - Multilanguage
  - Concurrent program development
- Time sharing and/or event driven
- Resource sharing

Figure 4.10
Operating system terminology

a multiuser or multiaccess system, selected (possibly all) terminals can be initially attached to the command analysis program, although a few commands will be defined for exclusive use of the console or master terminal. Striking a control character to terminate execution of a program reattaches the terminal to the command analyzer.

Foreground/background. Still basically a single-user system, but with the ability to initiate two jobs concurrently. Typically the command analyzer runs in the background; the foreground job will use peripherals other than the console and has a higher priority than the background job. Most often the foreground job is an on-line program, say, controlling direct data entry stations, while the background job is batch oriented (compilation, file update, sorting, and so on), usually—but not essentially—not requiring any terminal interaction until the job has been completed.

Multitasking. The ability of the system to maintain multiple tasks in an active state at the same time. Each task is given a priority that is dynamically altered depending upon its relation to the rest of the system. The highest-priority task is given control of the processor. When a task terminates, say, by initiating a disc read request, then its priority is marked down and the next highest priority task started. When the disc read is complete the task required to process the data is raised in priority so that it will be activated when possible. The actual priority system contains information about importance as well as current status (to be discussed in more detail in chapter 6). Multitasking (or multithreading) should be contrasted with *single threading,* whereby a routine once started retains control of the processor even when it is awaiting completion of a requested I/O transfer.

Single threading is common in pseudominis but unusual in minis, since the interrupt-driven hardware concepts make multitasking a natural method of operation.

Multiprogramming. The ability to maintain concurrently multiple independent programs in the system. Given a multitasking system the OS must maintain schedules of which tasks are part of which program and maintain the correct logical linkage; this in effect is a high-level control of the task priority system. Program control can be achieved by one of two methods, *event-driven* or *time-sliced.* In time slicing the processor is allocated to each program in turn for a fixed interval of time on a round-robin basis. Most mini data processing systems are basically event-driven, a natural concept with multitasking, but with an overriding clock control to time-out processor-bound routines. The effect is a time-shared system in which any program can terminate its slice whenever it cannot continue, with all the supervisor functions such as I/O handlers being interrupt-driven. Thus a program requesting input of a binary number is descheduled immediately; the terminal handler independently collects the key-driven characters, converts to binary, and only when ready reschedules the program. Note that multiprogramming can be achieved on a batch system and with single threading by using time slicing; multithreading is essential for communications or interactive work.

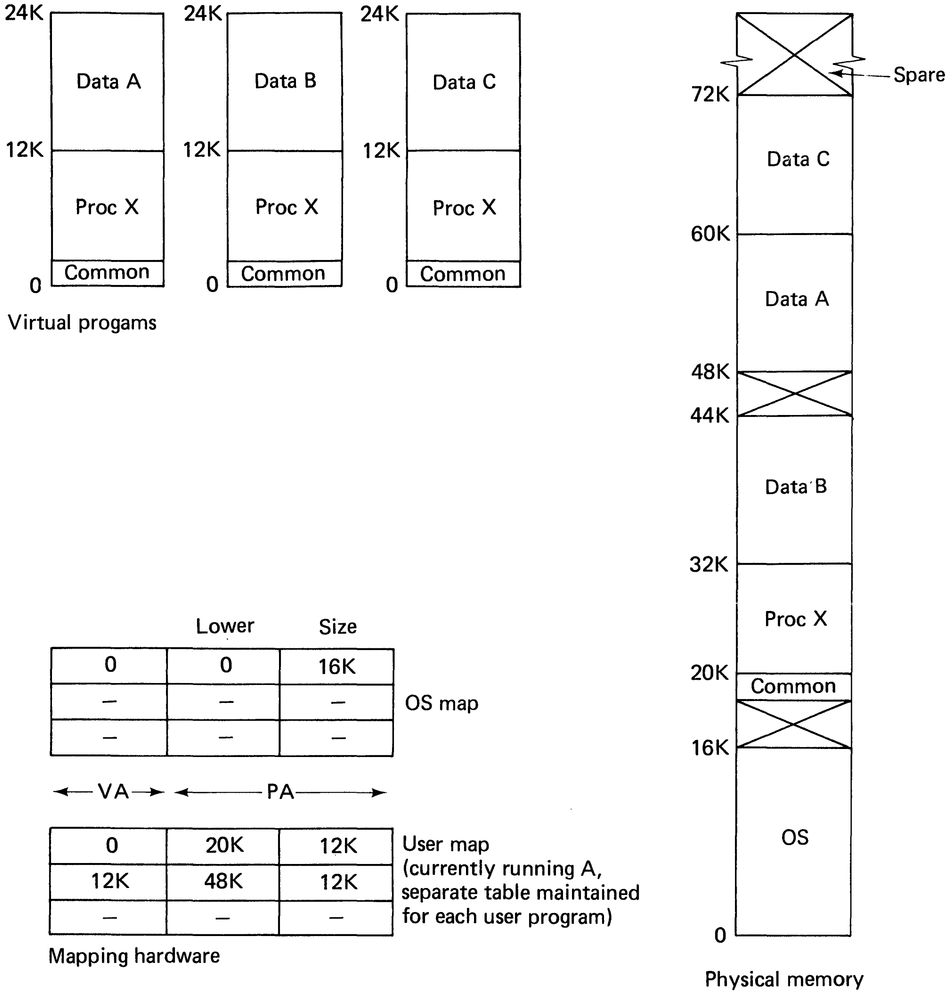
Mapping. The 16-bit word length constrains any one program to an address space of, say, 64 KB. With no mapping hardware the OS and user programs must share this space. If two or more programs are active then errors in one (writing more elements to an array than were allocated) can corrupt the other. Some checks can be made when the programs are loaded, but not at run-time. With mapping hardware multiple programs can be allocated addresses in the range 0 to 64 KB and loaded into differing physical locations; the program address is termed the virtual address. Thus the OS can use one virtual space and each program its own virtual space, mapped into separate physical memory locations. Protection of one program from another is provided by the mapping hardware. All I/O is performed in the OS so that user programs can use the full 64-KB range. The user program communicates with the OS by issuing SVCs. The processor must provide facilities for rapid context switching, for example, when a user program executes a SVC, the maps must be changed from the user set to the OS set (kernel or supervisor mode) and register sets changed or their contents stored. A further problem area is that the supervisor function initiated will need data (parameters) from the calling program, which means that instructions must be provided to move data from A to B, where A uses the kernel map and B uses the user map. A common alternative is to set the mapping up so that, say, virtual addresses 1 to 2 KB of the OS are the same physical locations as 1 to 2 KB of the current user program. With mapping hardware it is also common to inhibit the execution of certain privileged instructions, when not in the kernel mode, such as Halt. With virtual I/O systems (DEC PDP-11, TI 990/10), only in the kernel mode are the top addresses mapped to physical I/O; in the user mode they can be used for normal program space, effectively constraining I/O to the OS.

Multipartitioning and code sharing. With memory mapping the concept of foreground/background operation can be drastically enhanced. Multiple partitions can be provided, each with full facilities (access to the command interpreter and all peripherals through the OS). Partitions can be of any size and in some cases can be dynamically located. A high degree of security can be achieved by allowing programs to run only in the partition in which they are installed. With a partitioned system there are distinct limits on the ability to share memory or data between programs; this is usually overcome by defining a block of memory as a global or common area mapped into multiple partitions.

The alternative approach for concurrently loading multiple programs is to ignore any partition constraints and to allocate *any* available memory to a requesting task. This requires rather more activity in maintaining the current mapping registers and possibly provides less security than the partitioned concept, which stems from the need to run fully protected real-time process control programs concurrent with program development. This mapping concept allows easy use of shared code, as shown in figure 4.11 with the user program divided into logical *segments*.

In practice each segment (procedure or data) may be mapped into smaller noncontiguous subunits, depending on the mapping hardware—the more pages the more efficient the memory utilization but the higher the map bookkeeping overhead. Given three mapping registers, say, the OS could be designed to keep each segment contiguous but allow multiple levels of sharing as shown in figure 4.12. Thus, for example, Procedure A could be an interpreter and Procedure B and Procedure C user code. Procedure B however, say, a stock inquiry program, is being used by two terminals, each of which have their own local areas, Data B1 and Data B2. Note that because mapping is used, addressing between, say, PROC A, PROC B, and DATA B1 is within the program's virtual address space (provided the map is correctly set up). Contrast this with communication between the user program and the OS (SVCs), which are using alternative mapped spaces. Thus a mapped system implies far more procedure-level calls than supervisor calls, with a resultant minimization of run-time overheads. If shared code is to be employed, references inside the procedures to peripherals and files must be to logical channel numbers and so forth. The actual physical devices allocated must be defined in tables shown as attributes in figure 4.11, although the OS must also maintain some tables to avoid double allocation of unshareable resources.

Swapping. Memory utilization can be greatly increased if segments of code of an activated program not currently in use are rolled out to a disc file and rolled back in again when they are actually needed. Typical situations occur when a user program issues a request, via a SVC, to read an input data item from a keyboard. The OS collects the input characters in a local buffer, checks for special characters, converts to the desired internal form, at which point the user program can be rescheduled to accept the data and to continue processing. Thus all the time the OS was active on behalf of the user program, it could have been rolled out to disc and rolled back in when rescheduled. Obviously, some code cannot be rolled



	Lower	Size
0	0	16K
–	–	–
–	–	–

VA	PA	
0	20K	12K
12K	48K	12K
–	–	–

Figure 4.11
Mapped operating system

out, such as the monitor or a shared procedure. A task that is to be allowed to swap is declared at task-build time as "checkpointable". Less frequently used modules of the OS are often checkpointable. If a checkpointable task is awaiting completion of a disc transfer, swapping must be inhibited; DMA transfers data to physical memory locations and there is no guarantee that the task will be swapped back into the same physical location.

Overlays and chaining. The concept of breaking a program into segments has been discussed under mapping, above, where segments were combined into a program by setting up mapping registers (possibly also involving swapping). The

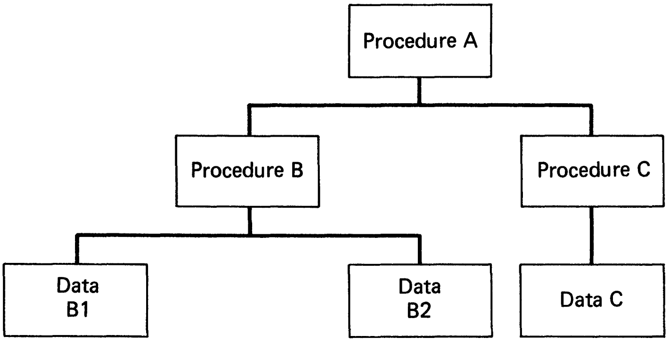


Figure 4.12
Shared code

OS was in charge so that the segmentation was asynchronous as far as the user program was concerned. However, a programmer in developing his program has distinct control over the logic and can break his program down into sequentially executable segments. Thus he can request that only specific segments be in memory, replacing one segment with another when required. This is termed *overlaying,* the point being that swapping is synchronous and under user control, with reduced overheads compared to OS-controlled swapping. Overlays must also be defined at task-build time to ensure that maximum required space is allocated and the overlays are valid. A typical system is shown in figure 4.13.

Chaining is a simple variant whereby one program, on completion, initiates another program. Unlike overlaying, however, special facilities must be made to transfer data from the first program to the next, usually through a disc file. There is significant overhead in closing files at the end of one program just to reopen

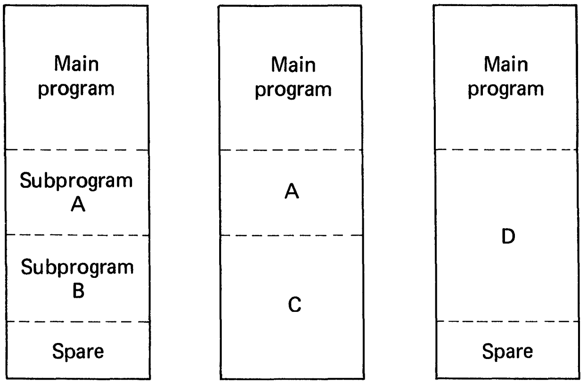


Figure 4.13
Overlaying

them in the chained program, but the technique is very simple to use and avoids the overlay build process.

Virtual systems. A virtual system attempts to provide more memory space for the user programs than is physically available. This is achieved by keeping only parts of the active programs in memory, the rest being on a special area of disc. Each user program is broken down into a number of fixed-size pages, within the virtual address space. Pages are loaded into memory until memory is full. During execution the program will access a virtual address which is not in physical memory and a page fault occurs. At this stage the operating system identifies the least recently used page, which is rolled out to disc and the new required page rolled in in its place. This is called *demand paging* or simply *paging*. If the old page has not been modified it need not be rolled out since a true copy already exists on disc. Note that the active copy on disc must be in loadable form, not the basic program filed in the normal storage system. With this technique it is not only possible to run programs which in total exceed the available memory, but any one program may have a virtual address space in excess of physical memory. The use of such a technique for handling big data arrays is inherent in the concepts of APL and to an extent in an ideal PASCAL.

Virtual systems imply a large virtual address space so that an excessive number of relocation pointers on a one per virtual page basis would be impractical. Thus some form of content-addressable mapping is required, one pointer per *physical* page, which can say whether the desired page is in memory or not with no time overhead. With a few exceptions (Prime 500, VAX11/780) minicomputers support neither virtual addresses in excess of 64K, nor page fault detection hardware, both features of modern mainframe machines. In fact, with a 64K virtual address, minicomputer memory mapping is totally differently organized, translating from virtual to physical address by retaining pointers for each *virtual* page, since the required translation hardware is not excessive. Thus, with the exceptions mentioned, virtual operating systems (with their high overheads) are not used on minis. Note that in all cases, too small a physical memory, implying a limited number of resident pages, can cause "thrashing," whereby the system spends too much time in swapping rather than in useful processing.

A most interesting variant of paging was introduced by Datapoint and adapted by Computer Automation's SyFA (see chapter 8) and BLIS/COBOL for Data General machines, which do not rely on memory management hardware at all. In these systems the *compiler* generates reentrant position-independent code in modules which do not exceed a fixed page size, say 512 bytes (related to a disc sector). Address references are created as a page number + in-page offset. Thus the size of program is governed by the number of pages, not the machine's virtual address, and in the systems mentioned can be as big as a megabyte. At run time part of the memory is allocated to code, in fixed page increments. The first page of a program is loaded into the first available page of physical memory and executed. Data is allocated in separate memory areas. The operating system maintains a table of physical page allocation, so that address references outside one page to another can be translated to the physical page in which that virtual page is stored. When physical memory is full the OS overlays least recently used pages with new ones.

The most common technique used on minicomputers is to swap whole programs. This allows the total active programs to exceed physical memory size, but does not allow virtual addresses to exceed the normal 64K limit. The technique is best illustrated in systems like DEC's RSTS/E and TI's DX10, which use a reentrant interpreter for the BASIC or COBOL programs. In this case only the user code need be swapped. Page maintenance overheads at run-time are avoided, but swapping overheads are high since segments around 32KB will have to be rolled out and in. Note, of course, that the system software, including all communications routines, must not be swapped. We have coined the phrase *semi virtual* to describe these systems since they meet many of the requirements of a virtual system with the notable exception of support for programs bigger than the physical memory or bigger than the 64K virtual limit.

Assemblers and Loaders

Because of the process control and the data communications history of the mini, where fast, compact code is important, assembly language has been the major tool for developing applications programs. As a result the available assemblers are very good, with extensive labeling and macro facilities. They are capable of producing absolute or relocatable code, defining tables or global variables for use when linking modules into complete tasks. Linkers and loaders are equally sophisticated, as already implied in the abilities to create overlays, define swapping attributes, define as reentrant, and so on. Usually the assembler (and compiler, for that matter) creates an individual object file for each source program. The linker is then used to build a loadable binary code file, resolving global addresses and adding a header with attributes needed by the operating system when loading the program. Any library routines are also built into this file, unless the OS supports mapping into a shareable library permanently loaded in memory. It should be pointed out that there is a wide variety as to which functions are included as part of the OS, which are in a shareable library, and which are linked in with each user program. Embedded into the user program, the functions execute faster but there are likely to be multiple copies in memory at any one time.

High-Level Languages

It is important in all computer systems, but particularly so with minis and micros, to differentiate between languages designed for user-oriented applications programming and the more specialized system programming languages. For the former class, user friendliness and ease of use is important, while for the latter class flexibility and efficiency are vital. A very simplistic viewpoint is to consider an application-oriented language as one which presents the programmer with a logical input/output interface, the actual physical requirements being transparent. On the other hand a systems programming language allows the programmer direct access to the operating system, allowing control of flags and SVCs; this should include all the facilities of an assembler but with high-level code. A system programming language cannot be truly machine independent since the operating system services available will differ. Such a language must generate relocatable code modules, which can be linked by the programmer, hopefully compatible with code generated by assemblers and other languages.

Obviously there are overlaps in languages such as the modifications providing wait, direct input/output, read and write data to specific memory locations, in BASIC or FORTRAN. The better compilers for languages such as FORTRAN automatically invoke the linking process, but others require the link-edit to be performed after compilation, as with a system program development. Most application languages either need an interpreter or invoke linking to a run-time support library of basic routines. System programming languages may also use a run-time library, but the programmer would need direct control and understanding of what is involved, a concept of no interest to the application programmer.

Usually the syntaxes of application and system programming languages are quite different. However, an interesting trend is the adoption of PASCAL for both, different implementations favoring one or the other use.

On both minicomputers and microcomputers the languages available are rather a mixed bag. Implementation of most languages in general use, plus some more specialized ones, are available, but the quality is varied. The problem is compounded on minicomputers by the wide range of operating systems supported, so that even for a given machine a compiler available on one operating system is either not available or runs very inefficiently under emulation on another operating system. Support for reentrant code is weak, although improving. Historically speaking, BASIC interpreters were the first high-level systems to become available, and were aimed at allowing research labs to use their machines when these were not tied up for the process control work for which they were installed. FORTRAN followed, and other languages have limped into consideration as we shall describe below.

BASIC. Designed as an easy-to-learn, easy-to-use interactive language for scientific work, it is well suited to time sharing with a "semi-virtual machine" concept. Most systems use an interpreter, although BASIC compilers are available. Often multiuser BASIC systems run under a single-user OS; the BASIC monitor, once loaded, providing the multiterminal support, which does not encompass all the facilities one would normally associate with a multiuser OS. In the attempt to make the system easy to use the monitor commands (run a program, list a file directory, and so on) are integrated with the actual user program commands. This provides a superb "friendly" user interface, a standard that other language processors must try to achieve. It also provides interactive program debugging, since the values of current variables can be displayed or changed on request at the terminal. BASIC is common on microcomputers.

Extended BASIC. BASIC has been extended by adding both integer and floating-point arithmetic, string manipulation, and file handling. While there is an ANSI standard for BASIC, each manufacturer's extended version is different; some are significantly more powerful than others. DEC's BASIC-PLUS was probably the first BASIC system to be accepted in commercial data processing. The language is more powerful than, say, COBOL, particularly for on-line applications. This, combined with the superb user and program interface, high level of file security, and availability on a standard mini, more than compensated for the relative slowness of the execution (compared to compiled code) and the limited structure and syntax of the BASIC language. BASIC-PLUS was also unique in that it was supported by its own dedicated swapping time-sharing operating system called RSTS, whereas most BASIC interpreters run as a job under standard operating systems. There are now a number of commercial BASIC systems, Microsoft's microprocessor-based MBASIC being one, but others are trying to use "extended BASIC" systems that are just not powerful enough (single precision arithmetic, poor file handling).

FORTRAN. Well supported on minis; some systems are interpreted, most compiled. A few manufacturers offer alternative compilers; development systems that compile quickly, with extensive diagnostics, but run inefficiently and a "racetrack" compiler with limited diagnostics that is slower to compile but produces optimized code. Essentially a batch language for scientific work, FORTRAN has been extended for real-time work by supporting call statements to timers and peripheral drivers. It has also been applied to commercial data processing by improving string handling and file systems, notably by General Automation. Other commercial FORTRANs are essentially a suite of subroutines available with the standard FORTRAN and are not very good.

ALGOL. Rather poorly supported: it is difficult to see any demand that will change this situation, although the language is good.

COBOL. Every computer manufacturer with an interest in commercial systems must consider a COBOL compiler, since like it or not it is the industry standard language and there are numerous COBOL programs and trained programmers in existence. However COBOL is a batch processing language which is not particularly appealing to minicomputer uses. Data entry languages are used to create files for later processing by COBOL programs, the appeal of which is compatibility with IBM System 3 machines, with access to existing application programs. (In practice, compatibility of one COBOL with another creates very serious problems.) To use COBOL in a transaction processing system requires major differences, largely to the operating system environment. Transaction Processing Monitors are used to support terminal I/O, linking to user processing modules which, since they do not handle the I/O themselves, can be written in COBOL (the compiler should generate reentrant code); a typical system is described in chapter 10. However, the more common technique introduced on minis and micros is to enhance the language to take direct advantage of the operating system's interrupt-driven I/O services, a concept which would be difficult on a mainframe OS. Thus file-like READ and WRITE statements are introduced to handle blocks of terminal data, with ACCEPT and DISPLAY statements for single-character I/O. Since many such interactive COBOL systems use interpreters, the similarity with BASIC (the concept, not the syntax) is obvious. Debug modules are also included in some with standard ANSI 74 level 1 specification and some level 2 features. The Texas Instruments system described in chapter 7 is a good example, while with Microsoft's COBOL-80 and MicroFocus's CIS-COBOL, interactive COBOL has become a microcomputer standard. However, it must be noted that the concept of using COBOL to achieve a means of adopting existing application programs fails with interactive COBOL since the existing programs are batch-oriented and require complete reanalysis for on-line processing.

PL/1. An attempt to blend the arithmetic processing power of FORTRAN with the data handling of COBOL, it is generally felt that PL/1 compilers are large and would therefore not interest mini suppliers. Now that memory has fallen in price and large memories are common on minis, the situation may change; indeed Digital Research have introduced a version for micros. It is interesting to note that IBM's initial offering on Series 1 was PL/1 and not COBOL—that could be, however, because IBM is sure independent software houses will develop COBOL compilers.

PASCAL. PASCAL is a structured programming language which was initially of interest to language specialists in an academic sense. Implementations were developed on minis in universities with relatively little vested interest, and as such were adopted by, rather than thrust upon, the users. The interest in PASCAL coincided with the growth of microcomputers so that the language became widely available at low cost. Ideally the language allows data structures to be defined which would encompass large data arrays; such an implementation would need virtual operating system support and so conventional file-handling characteristics are (grudgingly, by the purists) being added. With widespread availability and low commercial pressure, minicomputer and microcomputer enthusiasts are using PASCAL in a wide range of applications. Most are experimental and many scientific or mathematical, but there is also a strong interest in commercial applications. The other aspect of PASCAL, with its wide scope for handling various data structures, is its use as a system programming or real-time language. Variants, heavily integrated with real-time operating systems, support concurrent execution of modules (Concurrent-PASCAL) for process control, multi-tasking applications; Texas Instruments supports this approach. For microcomputers the University of California at San Diego (UCSD) PASCAL is commercially available on many machines. A pre-compiler generates an intermediate P-code which is interpreted at run-time, an implementation aimed at retaining the friendly user interface of BASIC. On the other hand the Microsoft PASCAL compiler generates code modules compatible with their linker, an implementation aimed at system and real-time programming. The concept of a common language being used as the basis of totally different programming requirements by differing implementations is most appealing.

RPG II. This is another batch processing language, possibly simpler to program than COBOL but much less powerful, introduced by IBM for use on the System 3 with memories too small to support COBOL. As terminals have been introduced on these machines and their replacements the language has been enhanced to suit. Since the IBM System 3 replacement market is very attractive to mini makers, RPG II compilers abound. Frankly an RPG II system is inferior to, say, commercial BASIC unless one is forced to use a small memory.

COBOL-like languages. A number of languages have been developed specifically for on-line data processing. Structurally they are similar to COBOL but greatly simplified. It is a shame that with the mini's positive impact on data processing the opportunity to establish a new internationally standard language along these lines was missed. Probably Datapoint's DATASHARE was the first, followed by DIBOL, SyBOL, and one or two others. CAP's MicroCOBOL is the common micro example.

Data-entry languages. Special-purpose languages have been developed for key-to-disc systems, a well established use of minis. The essential characteristic of these languages is that they are designed for screen formatting, data validating, and so on, rather than data processing. These are superb examples of software systems being developed to perform a dedicated job very efficiently at the justifiable expense of versatility. While BASIC, Interactive-COBOL and the like have facilities for reading and writing on VDUs, all screen formats and so forth have to be programmed from scratch. Thus the example set by the key-to-disc systems is being adapted to standard minis with special data-entry packages. Data once collected can be batch processed using COBOL programs since the files are compatible. An example is Data General's IDEA, a software package that runs as a job under standard operating systems.

An alternative approach, a few examples of which exist, is the program generator. Here a simple-to-use screen formatting and validating procedure is used to generate a module of code for a standard language such as BASIC or COBOL; further processing operations can then be added and the whole compiled as a normal program. DEC's DECFORM generates DIBOL program and FILETAB Services in the United Kingdom has developed a FILETAB (RPL) precompiler that generates BASIC-PLUS programs. CIS-COBOL provides the FORMS utility on micros. Program generators are an enigma; they undoubtedly speed up program writing, but the code they generate is difficult to follow and debug and can be inefficient at run-time. They are extremely difficult to evaluate, the good points being immediately obvious, the bad only materializing when a few production programs have been developed. The transaction processing monitors developed by software houses for pseudominis often include first-class facilities for developing data-entry programs, including transaction log files. Transaction logs are nearly always left as an application program function on minis.

APL. At the moment there are only a few APL systems available. This will change over the next few years as the general interest in APL increases, an interest stimulated by universities, which are now rather firmly mini oriented. It will be interesting to see whether future enhancements to the language will lead to its adoption in commercial data processing, but it is very inefficient. Terminals with APL character sets are readily available at little extra cost.

CORAL and RTL/2. The (loosely phrased) real-time languages; essentially these languages have a syntax similar to ALGOL but with I/O designed to mate directly with specific real-time operating systems supervisor functions. The objective is to produce a high-level language that retains the efficiency and versatility of an assembler, generating linkable object code modules. As such they are not specifically applications oriented and it would be very difficult to write, say, commercial data processing programs with them. These languages were originally designed for process control applications, but there is a good deal of interest in their use for writing special-purpose communications programs. All compilers, must, of course, be designed to run with a specific operating system, since they use the SVCs. In languages such as COBOL this is transparent to the user; with CORAL and RTL/2 the programmer trying to take full advantage should be fully aware of the specific operating system in use.

As previously mentioned, various implementations of PASCAL are growing competitors to CORAL and RTL/2. The concept of integrating the real-time operating system and compiler to create concurrent applications modules without the programmer having to control the SVCs and flags is most important. Concurrent PASCAL is one such language but MODULA and others exist in experimental form. The so called real-time extensions to BASIC and FORTRAN are a much lower-level concept. In general they provide facilities for direct control of I/O and timing, but only for sequential programs; concurrency (which implies multi-tasking) is difficult with these languages. Nevertheless they are simple to use and are therefore very important for less complex applications. Real-time BASIC is an important teaching tool since it stresses the broader applications of computers rather than mere calculating.

System programming languages. Most system programming on minis and micros is done in assembly language. There is, however, a growing use of special-purpose compilers aimed at improving programmer productivity, documentation, etc. Often these languages are very much machine dependent (Hewlett-Packard's SPL) but there are some common ones variously but not widely supported such as LISP and BCPL. Real-time languages such as CORAL and PASCAL are also used. A most interesting variant is C, a language developed for the Bell Labs' UNIX operating system for PDP-11 machines, and now available for 8080/Z80 microprocessors. The microprocessor manufacturers have produced their own languages for system programming and for generating modules for real-time programs. These are ALGOL/PASCAL-like but in fact are not common. It is most unfortunate that PASCAL was not advanced far enough at the time to have become a standard, thus avoiding the unnecessary spread of languages. The leading micro languages are PL/M and PL/M 86 (Intel), PL/Z (Zilog), and MPL (Motorola).

ADA. The latest in the attempt to produce an all-purpose language is ADA. The specification of ADA has been built up by a series of trial projects by U.S. agencies, and as such must have the most comprehensive features of any language. Implementations are not yet common so that little further can be said at this point.

Conclusion

High-level language support on minis still has a long way to go. An operating system as well established as DEC's RSX 11-M for a long time supported only an assembler, FORTRAN, and CORAL. By 1980 this list should be extended to include COBOL, RPG II, DIBOL, and BASIC-PLUS-2, which is a fair reflection on the current state of development across the industry. By far the weakest area, particularly important in multiterminal installations, is a lack of compilers which generate reentrant code.

Program Development Aids

In batch systems, programs are traditionally keyed on cards, read into the machine, compiled as a batch job with all listings, and diagnostics printed out only on completion. With a terminal-based system, programs can be keyed into the machine and stored on disc files with the aid of an *editor.* The same editor can be used to correct source code. Editors still vary in quality, but in general the products offered on minis are good and some are first class. With interpretive systems, particularly BASIC, the first level of syntax checking can be performed as the code is keyed in; this is desirable for all languages, but it does mean that language-specific source entry editors are needed, which are rare. There are endless arguments (nothing to do with minis) as to whether a programmer should be allowed to sit at a terminal, request a compilation, wait for the diagnostic listing, edit the program, and so on, or whether the older batch approach enforces more constructive thought as opposed to impulsive action.

When it comes to testing a program, there is no doubt that *on-line debugging* is a powerful technique. Breakpoints defined in the program cause halts in the natural execution sequence, returning control to the terminal. The programmer can then display the current values of program variables with the assistance of a source listing. The breakpoint traps are eventually removed before the program is released for use. An extremely powerful feature in favor of BASIC is the integrated editing and on-line debugging in such a system.

File Management Systems

Minicomputer file management systems were very slow to develop. Originally introduced for disc-based program development systems, they were designed to load and save contiguous sequential files. Run-time access to user data files was poorly supported. Further, since they were largely single-user systems, there was little file security and and no ability to extend files. The advent of multiprogramming systems with commercial applications has changed all that, and the latest breed of minis support some very good file systems. Nevertheless this can be an area of weakness in older systems. Industry-standard sequential magnetic tape files, usually IBM-compatible, are standard features; there is little else to say. Here we will concentrate exclusively on disc systems, which we will consider from the physical, logical, and user-access points of view.

Physical file allocation. A disc can be considered as a number of blocks (sectors), usually 512 bytes each, numbered sequentially from 0 to the maximum. Multiple drives are identified by unique codes (D, D1, and so on). A file of user information is stored in a number of sectors and some method of allocation and identification is required. Needless to say each disc can store multiple files; further, multiple users should be provided for, each with multiple files. Each file must have a unique identifier, a user-allocated name, not merely a system-allocated number. Finally, files of differing lengths must be allowed for. Thus each physical disc must maintain a special system-controlled file (or files) called a directory. Some large files are organized as volumes and may reside on different discs. Therefore, while the directory usually applies only to the disc on which it resides, it may have to contain a cross-reference to another disc. Since a removable disc may contain secure information each cartridge is initialized with a code name. This is checked in a "mount" operation and must tally before the directory can be accessed. The directory consists logically of two levels, the master file directory (MFD) and the user file directories (UFDs). Figure 4.14 is a schematic of physical file allocation.

The MFD is accessed when a user logs on to gain access to a specific UFD. Whenever a particular file is opened, details are read from the UFD of the file at run-time. The UFDs are merely ill-ordered sequential files which can spread over a number of blocks; there can, therefore, be a high overhead in opening a file. On better systems, UFDs may be hierarchically subdivided, say by group codes; directory clean-up utilities, eliminating entries for deleted files, are most valuable. Note that deletion of a file means marking the directory entry only; the data blocks can be freed for other use, but are not actually affected by the deletion and for this reason erroneous deletions can sometimes be corrected. The MFD must maintain a table of all unallocated blocks as files are added, deleted, or extended.

There are three basic methods of allocating physical blocks to a file: linked, contiguous, and clustered (see figure 4.15). The last word of a block in a *linked file* points to the next physical block address (not the logical sequence number) except for the last block, which contains an end-of-file character. *Contiguous files* are allocated a fixed number of blocks in physical sequence. A *clustered file* is allocated a number of groups of blocks. Each group, called a cluster, has a fixed number of contiguous blocks, but consecutive clusters can be allocated in any available space large enough. Relative characteristics are shown in figure 4.15.

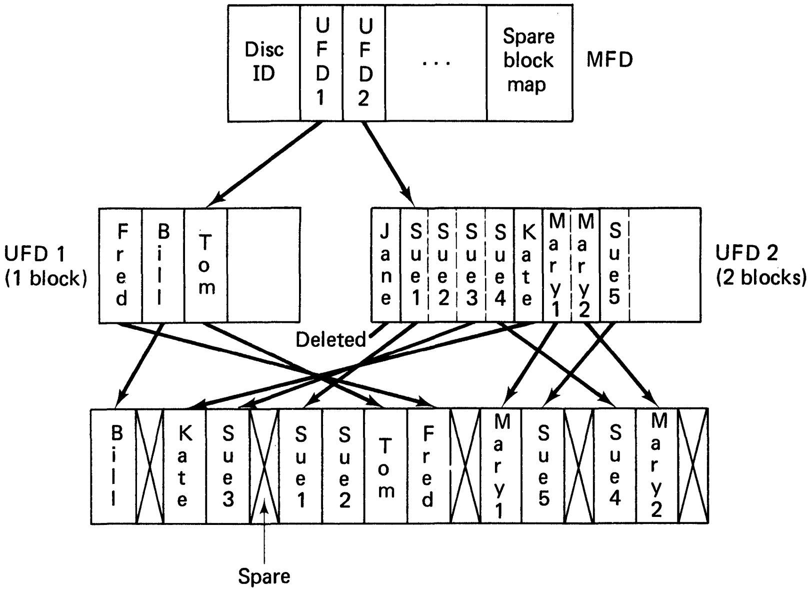


Figure 4.14
Physical file allocation. Note that if contiguous spare blocks are smaller than the cluster size, the disc will be declared full even though there is unused space.

Clustered files are clearly the best, but a word of warning. Taking DEC's RSTS system as an example, the core resident tables maintain pointers to seven clusters; access to data in another cluster causes an additional directory read. A file of 1MB with a cluster size of four blocks would have 500 clusters and virtually every access would require a directory read through a multiblock UFD as well as the data access, possibly tripling the access time compared to the same file created with a cluster size of 256 blocks. Some systems use allocation algorithms which add larger clusters each time the file extends.

Linked	*Contiguous*	*Clustered*
One directory entry	One directory entry	One entry per cluster
Extendable	Not extendable	Extendable
Sequential access only	Any access method	Any access method
All freed space reuseable	Inefficient reuse	Good reuse

Figure 4.15
Physical file block allocation methods

Logical file access. From a user viewpoint a file is accessed by name. As described above a channel in the user's program is associated with a named file when the open function is performed. As well as reading the data from the directories for the logical to physical mapping the file control attributes must be read. These cover read only, read/write, shareable or nonshareable, read by some users, read/write by others, and so on. On most batch processors once a file is opened by one user it cannot be opened by another until it has been closed. For online processing this is unacceptable; lock-out must be applied only at record level, and even then it must induce a wait and try-again response and not an error. Note that file lock-out can be achieved by markers in the directory, but record level lock-out requires run-time control in the executive file control services.

Once the file has been opened the user program must subdivide it into records and gain run-time access at the record level. Each record is transferred to a buffer in the user's program; from there it can be split into fields as required. Records can be either fixed length or variable length. In the latter case part of the record is used as a delimiter or record length counter; in the former, the record length is part of the file attributes. "Blank compression," replacing strings of nulls with a null counter, is supported for some variable-length record systems.

Sequential files. Both fixed- and variable-length records are supported. A reasonably common facility is support for files of ASCII data so that the file can be accessed by terminal or printer oriented instructions, thus providing a spooling feature, since either a printer or a disc file can be assigned to the channel at run-time.

Direct random access files. For fixed-length records only. There are two variants, relative block and relative record. With relative block, data is accessed by the complete physical block. On read, this can be transferred directly to a buffer in the user's program; from there it can be directly processed into records and fields. Similarly data can be assembled in a buffer before writing to disc. This is very efficient but requires processing in applications programs. With relative record files the nth record can be directly passed to the user program. Knowing the record length and physical block size, an algorithm can be established to read the appropriate block into a buffer and then to pass the appropriate record into a defined data space in the user program. Records can be *packed,* using every byte by crossing block boundaries, or *best-fill,* with an integer number of fixed records in each block, may be used. The former is more efficient for space utilization, the latter for accessing. Note that whenever consecutive records are accessed a disc access may be avoided, but writing one record enforces read/modify/write of a block. Both block- and record-relative files can be accessed sequentially or randomly. Note that disc-to-buffer transfers are via DMA, whereas moving data between buffers and user data areas implies a processor overhead.

Key access files. In practical user files, the records are usually identified by alphanumeric keys that differ from direct access in that the keys do not start from 0 and advance in fixed increments. Thus some unique method must be provided to identify each key with a record. If there are a large number of keys, then some "tree-structured" arrangement is necessary, giving a *multilevel key* system. Further, the data may well be accessed by alternative keys, in a *multikey* system. Some of the methods employed in minis for providing keyed access will now be briefly considered.

Hashing. The key is processed by a numeric (hashing) algorithm to produce a block number. The algorithm is chosen so that all keys map into a number within the range of blocks needed to store the data. Two consecutive keys will not map into consecutive block numbers, and if multiple records can be stored in one block, multiple keys will hash to the same block number. The data is then written into the relative block in sequence after the last entered record. The actual key must be written with the data, since on read the block is read into memory and scanned to match the key. When a block becomes full a linkage pointer to an overflow block is recorded, as shown in figure 4.16. Each record is terminated with an end of record (EOR) character.

The problem with hashing is the problem of choosing weighting in the algorithm to spread the records evenly over the physical block range, when the keys in practice will be unevenly spread. Further, if the block number range is

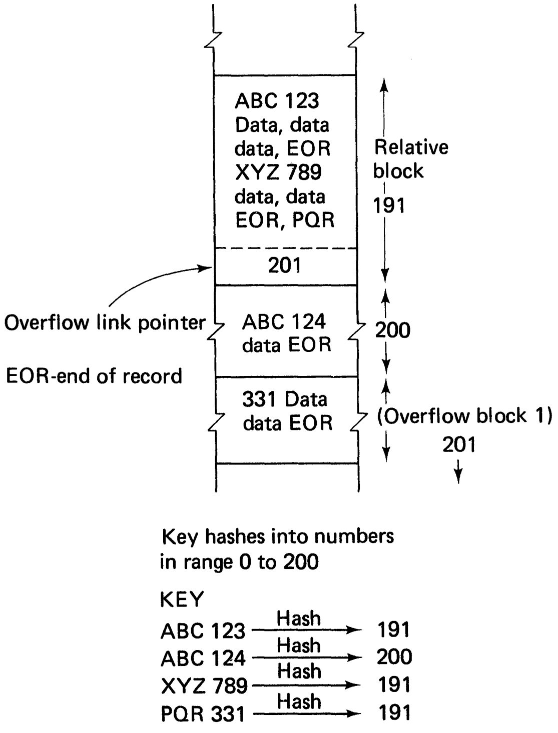


Figure 4.16
Hashed file structure

spread, a lot of disc space is allocated and never used, while if the block number range is tight some blocks will overflow many times; any record accessed from that block will have to work through the links, initiating multiple disc accesses. Variable-length records can be used, but then all additions must be at the end of a block and deletions are simply tagged. The whole data base thus needs regular, extensive reorganization, which carries a heavy overhead. Note also that the key itself must be recorded in the data file.

Indexing. The data is stored in a main file and an index file is built to note where in the file the data for each specific key is stored. The key is then used to access the index file, followed by a second access to the data proper. For complex structures there may be multiple levels of index file. If access to the main data is required by alternative keys, then by maintaining separate index files the same main data file can be used. This is fine for reading and updating records—fixed-length records, anyway—but additions and deletions mean updating *all* index files. Usually the system is designed most efficiently around one key, called the primary key, with the other, secondary keys carrying higher overheads.

There are many ways of organizing indexed files. The superior methods maintain either the data or the index file in sequence. This nearly always means some loss of efficiency for random on-line access, but enormously improves batch processing, sorting, and merging of files. The sequential file can be maintained by linking every block to the next in sequence. When an insert is made that overflows the block, the linkage pointer is changed to one in a pool of spares, the old link now being written into the new block. The keys are still required in the data, and since each block must be accessed the records need not be in strict sequence inside the block. An index can now be maintained to store only the key of the highest record in each block or group of blocks. A similar index can be maintained for the index file, which in most cases will be small enough to be memory resident. This and variants on the theme is referred to as an *indexed sequential access method* (ISAM). A variant on the ISAM theme stores only the keys in the sequential file with a simple pointer to a record in another file which holds the data added in time sequence. A further access is thus required to reach the data after locating the key; however, more keys can be stored in one block, so that subindices are most unlikely to be needed and the file is directly amenable to a tag sort. Alternative key index files can easily be maintained and variable-length records can be supported. A representative system is outlined in figure 4.17.

Data bases. Full general-purpose data bases in the CODASYL or Relational sense carry extremely high disc access overheads. Since most data processing systems are disc bound rather than processor bound, full data bases only make sense if full facilities are genuinely needed; all too often they are used just because they are easy to program. It is probably true to say that with the relatively infrequent use on minis of multiport memory and multiple data channel controllers, a full data base would hit at the machine's weakest point.

There is a far better trend, well established now, in the development of highly versatile multikey, multilevel ISAM packages that support features such as generic

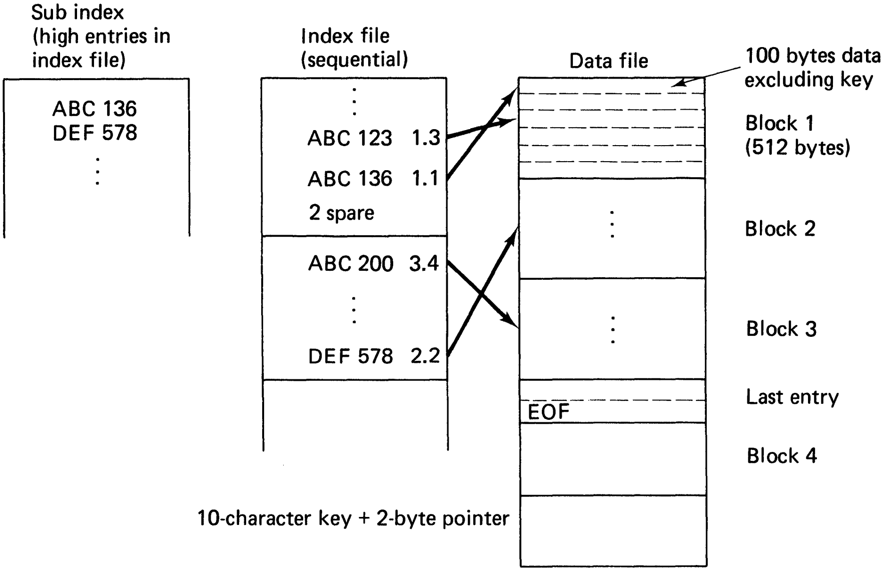


Figure 4.17
An indexed access system with sequential index file

and approximate keys with powerful utilities for file creation, maintenance, and enquiry. The TOTAL data base (initially installed on Varian), Hewlett-Packard's IMAGE 3000, Data General's INFOS, and Digital Equipment's RMS-11 are the prime examples. Nevertheless, note that the versatility of these systems must be paid for in run-time overheads and memory requirements. The simpler hashed routines on Reality and Texas Instruments or the proprietary ISAM packages on DEC's RSTS are most suited to smaller systems. In dedicated applications there is still room for specially tailored file access systems taking advantage of specific characteristics of the job and using the basic relative block access techniques provided with the OS. Finally, remember that a file management system is only as good as the utilities that support it.

File handling routines. All file systems supported by the operating system require suites of routines accessed from the user program. Figure 4.18 shows the overall picture. The file primitives such as read directory, write the nth physical block, and so on, are the basic functions and are required by all file organization techniques. The file control services (FSC) interpret the user request, mapping from logical to physical disc addresses, manipulating index files, and so forth. The primitives must be reentrant and the FCS at least serially reusable. Tables must be maintained to define the various logical channels and the actual physical devices currently associated with them and their attributes. Buffers must also be allocated and queues for services maintained. As shown, file access calls, implemented as predefined macros that set up an array of parameters and load the

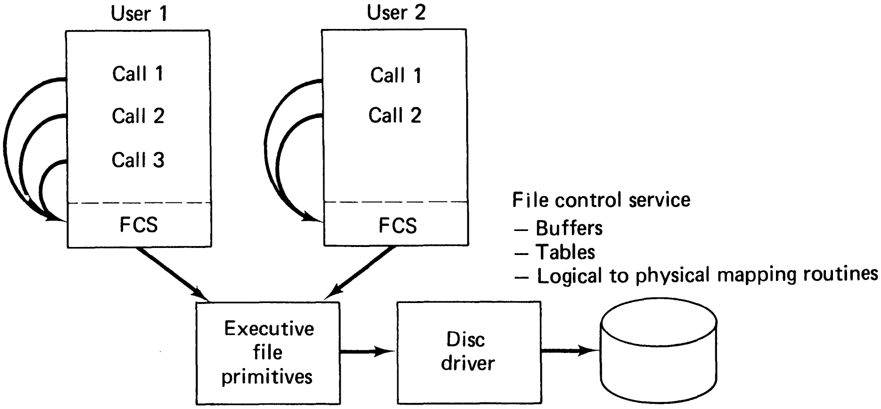


Figure 4.18
File system software

appropriate pointer registers, execute subroutine jumps into the FCS, which in turn make and synchronize SVC calls to the executive. Depending on just what is included in the FCS and what in the executive, simple calls such as sequential or random block access could be direct SVCs, bypassing the FCS. Multiple copies of the FCS are required, possibly minimized by a virtual OS, but the smaller they are the better, since they occupy part of the user address space. In some cases, particularly the more sophisticated multikey ISAM packages, the whole FCS is implemented in the executive. This topic will be discussed in far more detail in chapter 6.

Communications Software

Every peripheral device requires a basic module of software to handle interrupts, load data, test status, report errors, and so on. These we have called device drivers. Terminal-handling software is then required that uses the device drivers to move data from specific terminals to specific buffers and to tie them to specific applications programs. The drivers vary in sophistication depending on the hardware characteristics of the interface; in general, the less sophisticated the hardware the simpler the driver but the more involved the total software.

Thus, say a mini is to be used to communicate with a remote machine using bisynch protocols. A standard synchronous interface can be employed to form all incoming bits into 8-bit characters and pass them to a special program that checks the incoming characters for the defined protocol, causes a response to be transmitted, and then extracts the data characters, passing them to a buffer. Error checks are computed and any appropriate actions taken. In effect the standard interface with the special line-driver software form an *emulator* for the bisynch line protocol. If, however, programmable I/O chips are used then a further special routine is required to initialize the interface. Software emulators can be taken to much higher levels. The most common, employing the bisynch protocol, is an IBM 2780 terminal emulator. At a higher level, HASP remote work station emulators are available. Often these can run as a job in a multiprogramming system, providing full on-line processing with batch communication with a mainframe. Be warned, however; the software overhead in the emulator can consume significant portions of the processor time—up to 30 percent!

The basic 2780 style of communication is aimed at linking two specific devices together. If multiple machines are to be connected, in a network, then far more sophisticated network-control software is required in each machine. Further, if data stored in one machine is to be accessed by another, then a task on the requesting machine must be capable of initializing a file access task on the second machine and communicating data between the two tasks. IBM's SNA is such a concept, as is the more successful DEC DECNET described in chapter 7. It has been suggested that the on-line task-to-task communication is in many applications an overkill and that lower-level network systems will be developed, using minis as nodes, which allow data file transfer from any machine to any other in batch mode, utilizing features such as full duplex protocols and packet switching.

5

Data Communications Handling on Minicomputers

We have reviewed the nature of data communications and discussed in general terms a typical arrangement of the hardware and software components of a teleprocessing system's central site. Inevitably, individual manufacturers will have varying approaches to such configurations. In this chapter we will examine in some detail the techniques used by two minicomputer suppliers, Digital Equipment Corporation and Modular Computer Systems. The selection of these two companies is in no way an endorsement or criticism of the products described.

Digital Equipment Corporation PDP-11 Series

System Architecture

The Digital Equipment Corporation (DEC) of Maynard, Massachusetts, is currently the largest supplier of minicomputers. In general terms DEC has three main series of processors. The smallest is the long-established PDP-8 8-bit word machine which is gradually being replaced by the microprocessor-based low-power machines at the bottom of the PDP-11 (16-bit word) range. The top-end range comprises the closely related DECSYSTEM-10 and DECSYSTEM-20 series of mainframes. The medium-scale general-purpose processors in the PDP-11 range are fairly typical of minicomputers on the market in the late 1970s and are probably the most widely installed. For these reasons we will look at the way communications interfaces are implemented on the PDP-11 and related VAX-11/780.

Fundamental to PDP-11 architecture is the generalized high-speed I/O channel known as the Unibus* (see figure 5.1). Each PDP-11 system includes a single

*Registered trademark of Digital Equipment Corporation. See figure 2.22 for variations on the larger processors.

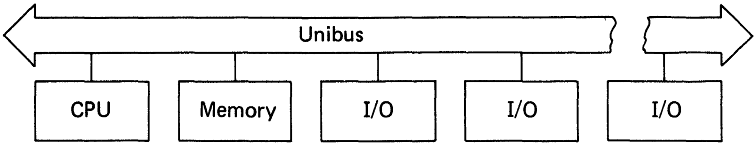


Figure 5.1
DEC PDP-11 system architecture

Unibus to which are attached the CPU, main memory (bipolar MOS, or Core), and peripherals such as paper-tape readers, discs, magnetic tapes, and communications controllers. The Unibus comprises 56 lines for the exchange of addresses, data, and control information between the various devices. Each device (including single-word memory locations) has a unique address on the bus. This arrangement enables peripheral device registers to be examined and manipulated as easily as memory locations, using the same instruction set. There are two common types of peripheral I/O operation. The first involves the transfer of characters on a byte-by-byte basis via data registers associated with the peripheral. Some instruction code must be executed for each character. The technique, usually called *character I/O* or *bus request* (BR) *working,* is only suitable for low-speed peripherals such as card readers and Teletypes.* Faster devices such as discs and magnetic tapes operate too quickly for this arrangement; the CPU would be swamped by the processor overhead needed to handle the characters being transferred into or out of the system.

In order to overcome this, a different arrangement is used on the Unibus whereby the peripheral controller itself increments main storage addresses as it is transferring the data, thus eliminating completely the need for processor intervention except at I/O initiation and completion. This arrangement is known as *direct memory access* (DMA) or *no processor request* (NPR) interfacing. The character I/O interface requires a command and status register (CSR) and a data buffer register (DBR). DMA units use three additional registers: word count (WCR), data address (DAR), and memory address (MAR). Communications adapters that use both methods of I/O are described later in this chapter.

One of the drawbacks of the Unibus approach (compared with the multiport memory technique used in the MODCOMP II described below) is that it offers relatively poor peripheral switching and sharing capabilities. The DTO3 Unibus switch (see J.T. Martin *Telecommunications and the Computer*, and figure 5.2) is available in two versions: DTO3-FP, programmable and manual control (1 microsecond switching time), and DTO3-FM, manual control only. One major limitation is that the switch allocates *all* I/O devices on the shared bus to one processor or the other. In the configuration shown in figure 5.2, if processor A wishes to access I/O device C2 it gets C1 as well, even if it does not require it.

*Registered trademark of AT&T.

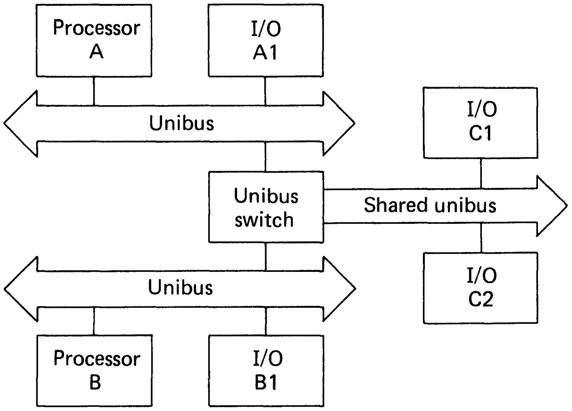


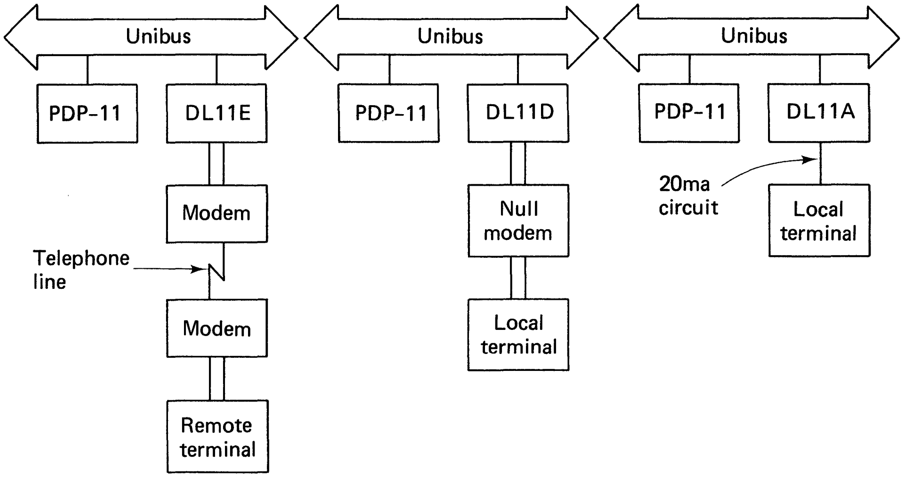
Figure 5.2
Peripheral switching on the DEC PDP-11 Unibus

This will preclude processor B from accessing Cl while processor A is accessing C2. The program-controlled version of the switch helps to ensure that the shared Unibus is held for the minimum period of time, but the systems designer must clearly make sure that it is not overloaded.

The PDP-11 uses a hardware multilevel priority interrupt system. Any number of devices may be attached to each of the eight levels. (This compares well with some minis—the fewer automatic interrupt levels there are, the greater the software overhead in analysing the interrupt and the lower the throughput capacity of the system.) Each device attached to the Unibus features a pointer to the device's interrupt service routine (ISR) and another to the new processor status information. An interrupt thus causes control to be passed automatically (at the end of the current instruction to be executed) to the relevant routine for processing devices on that level. After servicing the interrupt control is passed back to the next sequential instruction in the interrupted program.

DEC PDP-11 DL11 single asynchronous serial line interfaces

The DEC DL11 is one of the simplest of all line interfaces and, for that reason, is a useful point of comparison for other communications adapters on both DEC and other suppliers' systems. The adapter is available in a number of submodels, three of which are shown in figure 5.3. The DL11A is the simplest version, providing 8 data bits, 2 stop bits, no parity checking, and a fixed speed of 110 bits per second. It communicates using a 20-mA current loop technique. The DL11E is much more flexible and offers strappable options for character configuration, parity, and speed. A CCITT V-series (EIA RS-232) interface is provided for communications via modems over telephone lines, and full modem control (through status



CHARACTERISTICS

Number of data bits: 5, 6, 7, or 8
Number of stop bits: 1, 1.5, or 2
Parity: None, odd, or even
Speed: 50 to 9600 bits/s
Data set (modem) control: yes

DL11D has same characteristics, but no data set control.

ASR TELETYPE INTERFACE

Number of data bits: 8
Number of stop bits: 2
Parity: None
Speed: 110 bits/s
Paper-tape reader control

Figure 5.3
DEC DL11 single asynchronous serial-line interfaces (for models E, D, and A)

registers) is possible. The DL11D is similar to the DL11E in all respects but does not have modem control facilities. This makes it particularly suitable for use with a "null modem" (a simple black-box arrangement with standard 25-pin sockets on each side and leads such as transmit data and receive data cross-wired). This is an inexpensive way of attaching local terminals and other PDP-11s.

The heart of a DL11 is a device known as the universal asynchronous receiver-transmitter (UART). The UART (figure 5.4) is a double-buffered MOS/LSI circuit which serializes and deserializes data, handles character formats according to the options set (by on-board switches in the older types of UART) and provides the processor with status information on the line. Each direction of transmission has two registers associated with it; a status register and a data buffer register. The UART inserts an incoming character into the receiver data buffer register (RBUF—see figure 5.5) and raises an interrupt on the processor. This automatically causes the initiation of an interrupt service routine, which checks first of all to see if bit 15 is set. If this bit is 0, then the character (in bits 7–0) is good and can be checked to see if it is a control character, translated, passed to a message buffer, and so on. If bit 15 is set to 1 then one or more of bits 12, 13, and 14 are set.

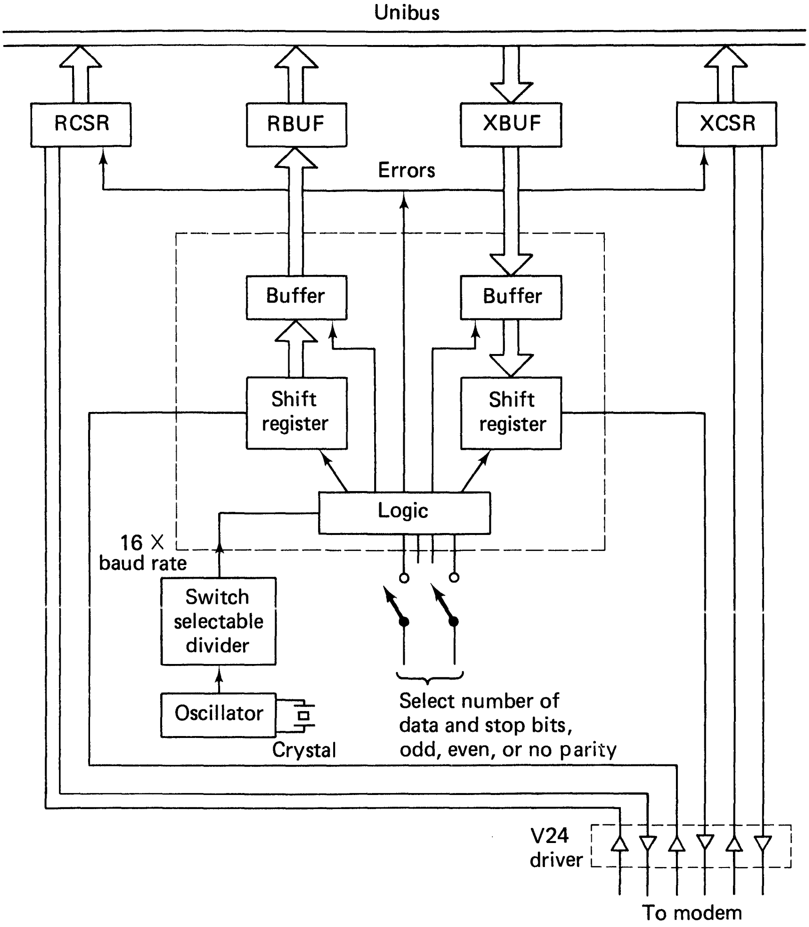


Figure 5.4
Schematic of a DEC asynchronous interface incorporating a single-chip UART

These will need to be examined to determine the cause of the error. The receiver status register (RCSR—see figure 5.6) is used for monitoring and control of DL11 input activity. For example, bit 11 (receiver active) is set on when the UART detects a start bit and off when the stop bit is detected (at which time bit 7—receiver done—is set on). Similarly, the RCSR can be used for modem control (responding to ring indicator, for example).

The transmission side of the DL11 features a transmitter status register (XCSR—see figure 5.7) and a transmitted data buffer register (XBUF—see figure 5.8). For further details of DL11 operation, the reader is referred to *Telecommunications and the Computer.*

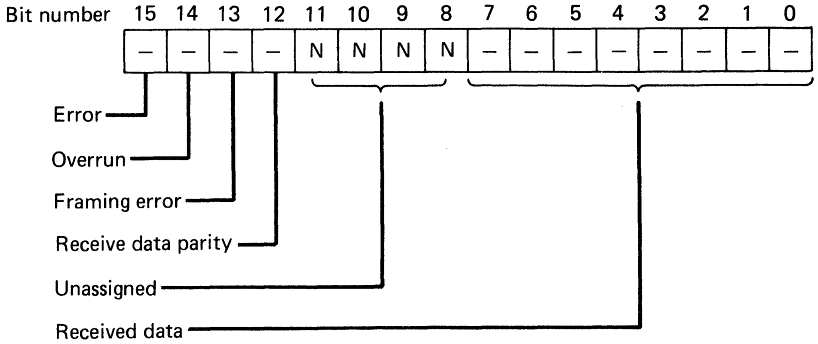

Bit number 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
– – – – N N N N – – – – – – – –
Error
Overrun
Framing error
Receive data parity
Unassigned
Received data

Figure 5.5
DEC DL11 receiver data buffer register (RBUF)

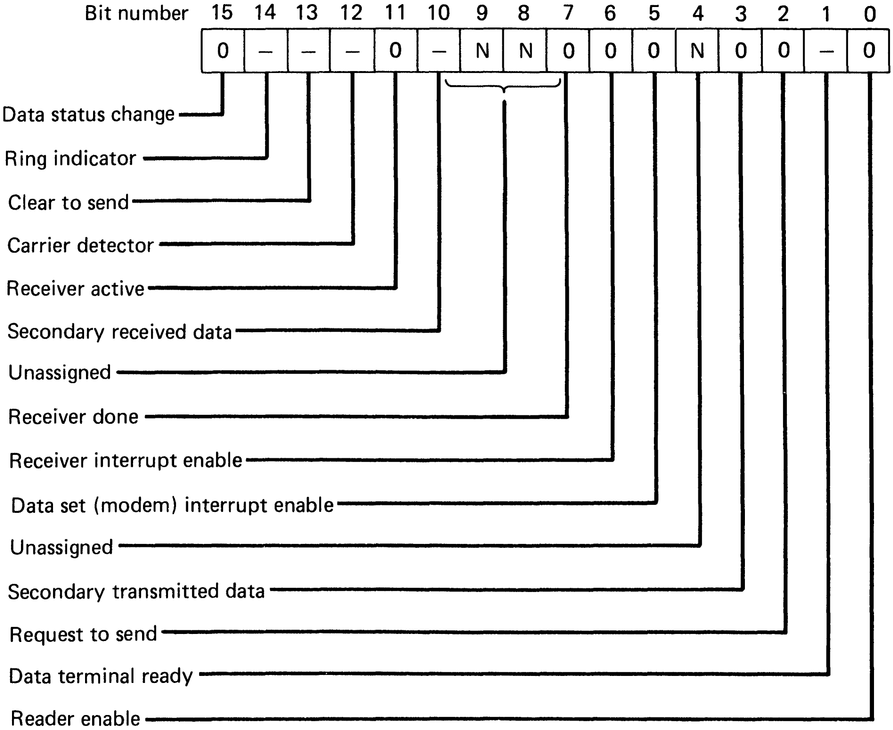


Figure 5.6
DEC DL11 receiver status register (RCSR)

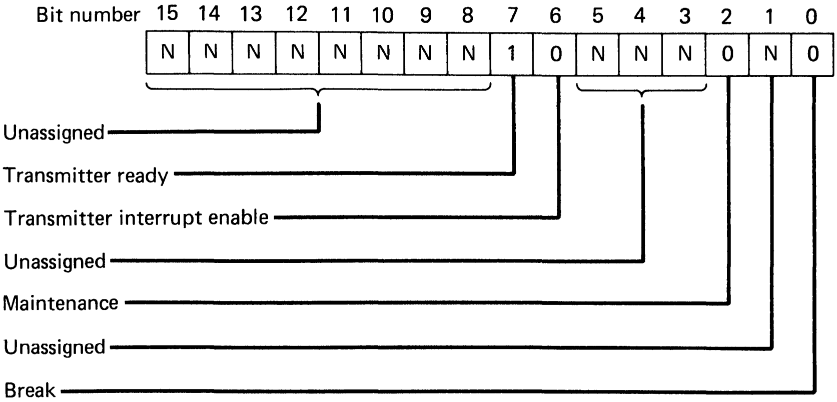

Bit number 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
N N N N N N N N 1 0 N N N 0 N 0
Unassigned
Transmitter ready
Transmitter interrupt enable
Unassigned
Maintenance
Unassigned
Break

Figure 5.7
DEC DL11 transmitter status register (XCSR)

In summary, the DL11 performs the minimum functions required to interface a processor to a communications channel at a character level. (Some obsolete devices performed this task at a *bit* level, but we can safely ignore this technique for all practical purposes.) Although this approach is inexpensive in hardware terms it carries a heavy penalty in processor overhead. The CPU time to service each character can be as much as 300 microseconds. This is of no consequence when the system is supporting only a few low-speed lines, but as the number of interfaced circuits and their speed of operation increases, the processor can be in danger of being saturated.

Let us look at two examples, assuming that our processor overhead is only 100 microseconds per character. If we have a single 110-bit-per-second line attached, then the overhead is

$$\frac{10 \times 100}{1000 \times 1000} = 0.001$$

However, if we have ten lines each operating at 4800 bps (half duplex) we get an overhead of about

$$\frac{480 \times 10 \times 100}{1000 \times 1000} = 0.48$$

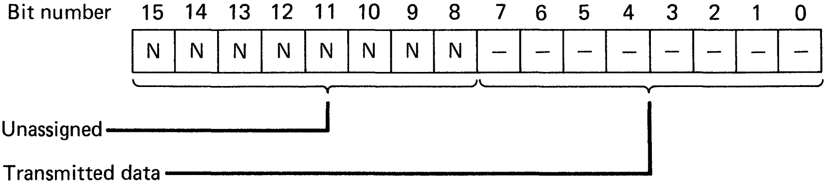
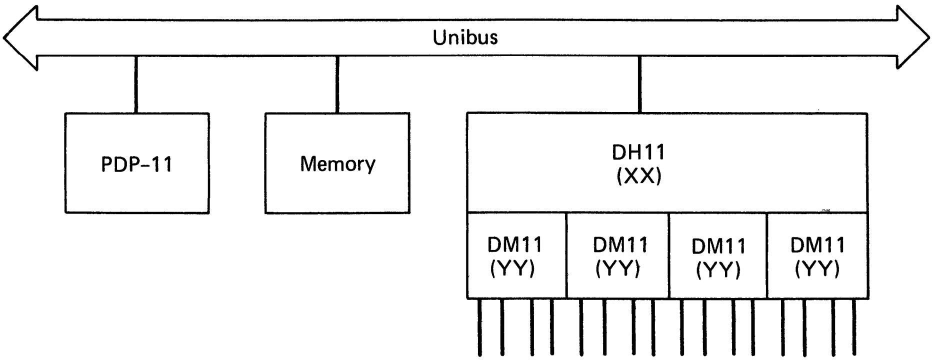

Bit number 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
N N N N N N N N – – – – – – – –
Unassigned
Transmitted data

Figure 5.8
DEC DL11 transmitter data buffer register (XBUF)

Clearly, this level is going to be embarrassing because we have not yet included operating system and file handling overheads. Incoming characters could plausibly be lost.

DEC PDP-11 DH11 16-line programmable asynchronous serial line multiplexor

The DEC DH11 is a programmable multiplexor designed to overcome the processor utilization problems associated with the DL11 single-line interface and also to reduce the cost of interfacing to a PDP-11 when about eight or more lines are to be attached. This is achieved by buffering data on input (thereby reducing the CPU time to as little as 30 microseconds per character) and by using a DMA (note: existing program) arrangement for output data streams. Up to sixteen DH11s may be attached to a single PDP-11, giving a total line capacity of 256. In addition, the DH11 provides program-selectable options such as speed, character format, transmission mode, and parity checking controlled by a Unibus-addressable control register (see figure 5.9).



XX = DH11 model number
YY = DM11 model number

PROGRAMMABLE OPTIONS

Data rates (may be different for input and output): 0, 50, 75, 110, 134.5, 150, 200, 300, 600, 1200, 1800, 2400, 4800, 9600 (plus two external)
Character length: 5, 6, 7, or 8 bits
Number of stop bits: 1, 1.5, or 2
Parity: Odd, even, or none
Mode: Half duplex, full duplex, echoplex
Interfaces: CCITT V-series (EIA RS-232), 20–80 mA telegraph

Figure 5.9
DEC DH11 16-line programmable asynchronous serial-line multiplexor

The method of operation on the receiver side of the DH11 is illustrated in figure 5.10. The objective of the silo is to enable the processor to obtain more than one received data character (up to 64, in fact) each time it is interrupted, thereby minimizing the overhead per character. Each UART performs in its usual fashion. When activated by a start bit it deserializes the incoming character using a shift register. Completed characters have their parity checked and are then passed in parallel to a holding register, thus freeing the shift register for the next character. An automatic scanner looks at each UART in turn for a completed character; on finding one it transfers it along with the relative line number and some control information to the lowest unoccupied word in the silo.

The bottom word in the silo is addressable from the processor and is called the *next received character register.* This is similar to the DL11 receiver data buffer register in figure 5.5. Each time a received character is placed in the silo, an interrupt can be raised so that the program can read it. This is not a very efficient way of using the silo and the programmer may set in a silo status register a "silo alarm level" of 0 to 63 which will cause his program to be interrupted when the specified level (say, 32) is reached. The program can then empty the silo. As each

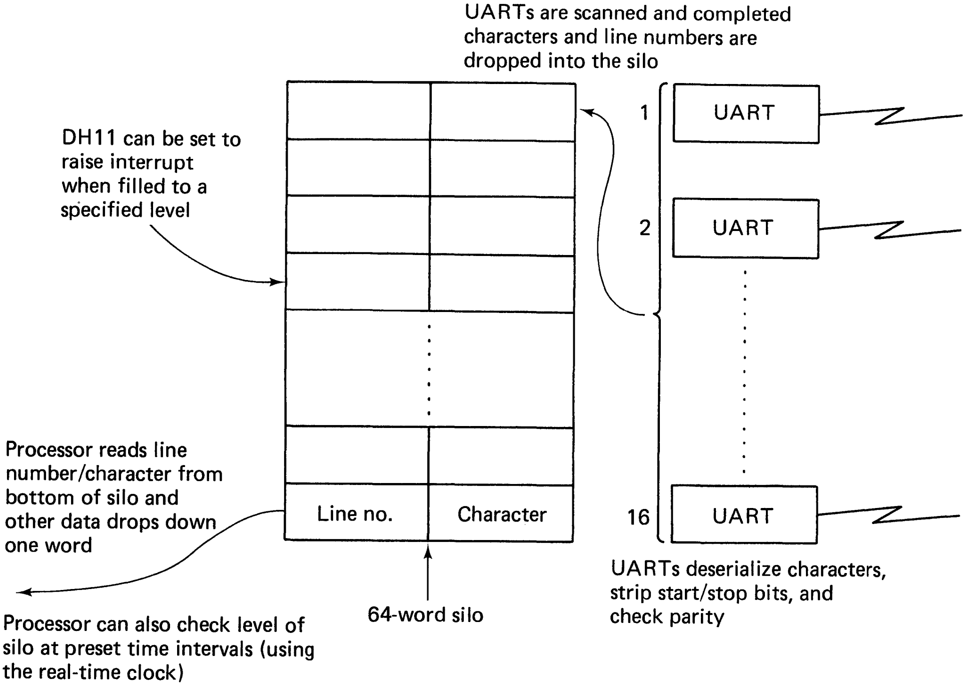

UARTs are scanned and completed characters and line numbers are dropped into the silo
DH11 can be set to raise interrupt when filled to a specified level
1
UART
2
UART
16
UART
Processor reads line number/character from bottom of silo and other data drops down one word
Line no.
Character
64-word silo
UARTs deserialize characters, strip start/stop bits, and check parity
Processor can also check level of silo at preset time intervals (using the real-time clock)

Figure 5.10
DEC DH11 receiver operation using the silo

data character is read the whole silo drops down one word, providing an additional free word at the top (see figure 5.11).

Some care should be taken with this method of operation because if only a few low-speed lines are active there could be some delay before the alarm level is reached. In order to overcome this it is usual for the programmer to set a timer interrupt (using the real-time clock) so that there is a minimum delay between servicings of the DH11. The program can read the status register at any time to determine the number of characters in the silo. Other interrupts that can be raised include one for a "silo full" condition. The transmission side of the DH11 works on a DMA basis. All the programmer need do is select the line via a system control register and set up a current address register with the location of the message to be sent and a byte count register with its length. Setting a register bit associated with the relevant line initiates transmission. The same bit is automatically cleared on completion (see figure 5.12).

The DH11 architecture is highly relevant to interactive working, where characters typically arrive in the system at spasmodic intervals as they are keyed at terminals but outgoing messages are usually complete at the time transmission starts. The DH11 is particularly useful for interfacing low-cost VDUs (perhaps using 20-mA loop circuits) where the PDP-11 is being used as a remote programmable controller in distributed processing systems.

Summary of DEC interface units

In this section we have discussed in some detail where the borderline exists between hardware/firmware and software functions on two of DEC's PDP-11 line interface units. (The reader will find other such devices in the *DEC PDP-11 Peripherals Handbook.)* The way in which DEC software supports these components and fulfills the other communications functions has been examined in chapter 4. The hardware/software borderline, however, differs on other systems; we next consider the approach adopted by another supplier.

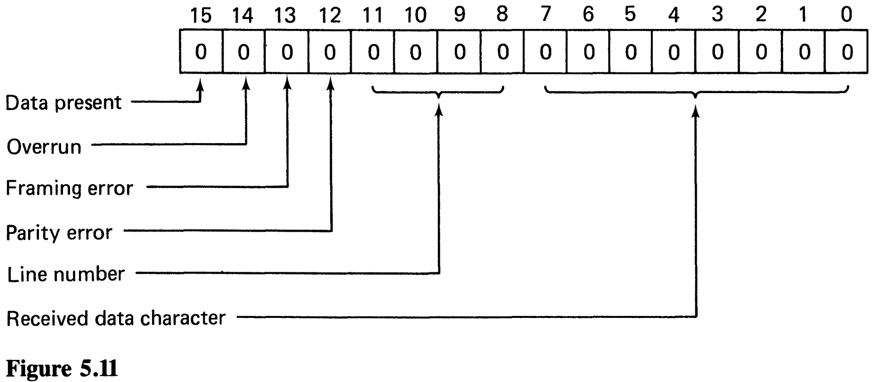


Figure 5.11
DEC DH11 next received character register (bottom word of silo)

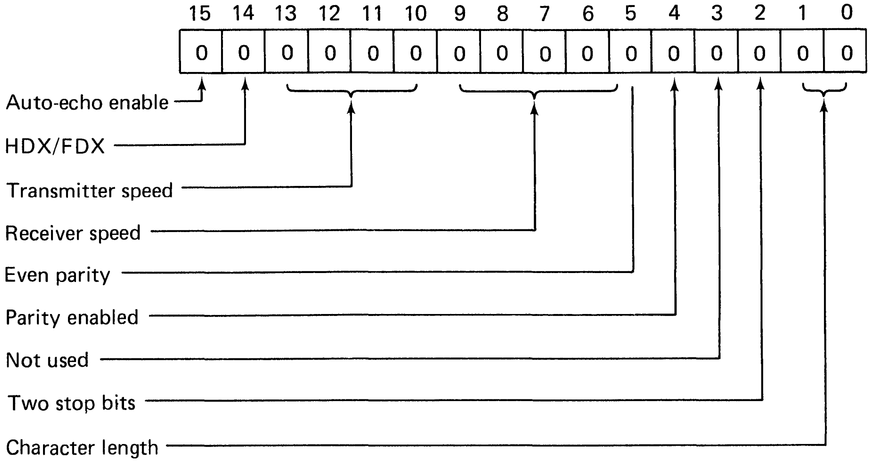


Figure 5.12
DEC DH11 line parameter register

Modular Computer Systems MODCOMP II CP2

System Architecture

Modular Computer Systems (MODCOMP) of Fort Lauderdale, Florida, is a smaller and more recently established minicomputer supplier than is DEC. It has a good track record in the communications field and, in some respects, the way in which it interfaces to data lines is more advanced than the PDP-11 approach. This alternative style is facilitated by a different system architecture and by the incorporation in the processor of specialist communications functions.

At the time of writing, the MODCOMP range comprised three basic models. The small MODCOMP 1 is intended for dedicated acquisition and control applications. At the top end of the range is the 32-bit word MODCOMP IV, which has a memory cycle time of 160 nanoseconds. The mid-range machine is the MODCOMP II, and it is this system which we shall examine here, particularly a special version known as the MODCOMP II CP (communications processor). One essential difference between the MODCOMP II and the PDP-11 is that the former has a multiport memory arrangement in addition to the usual bus facility for I/O. This permits parallel access to different main storage modules. The processor has a 16-bit word and is available with up to 128 Kbytes of 800 to 850 nanosecond cycle-time storage (all of which is addressable) and 15 general-purpose registers. Like the PDP-11 a vectored interrupt system is used, but with 16 priority levels and 128 sublevels. A general schematic of the MODCOMP II Model 45 is shown in figure 5.13.

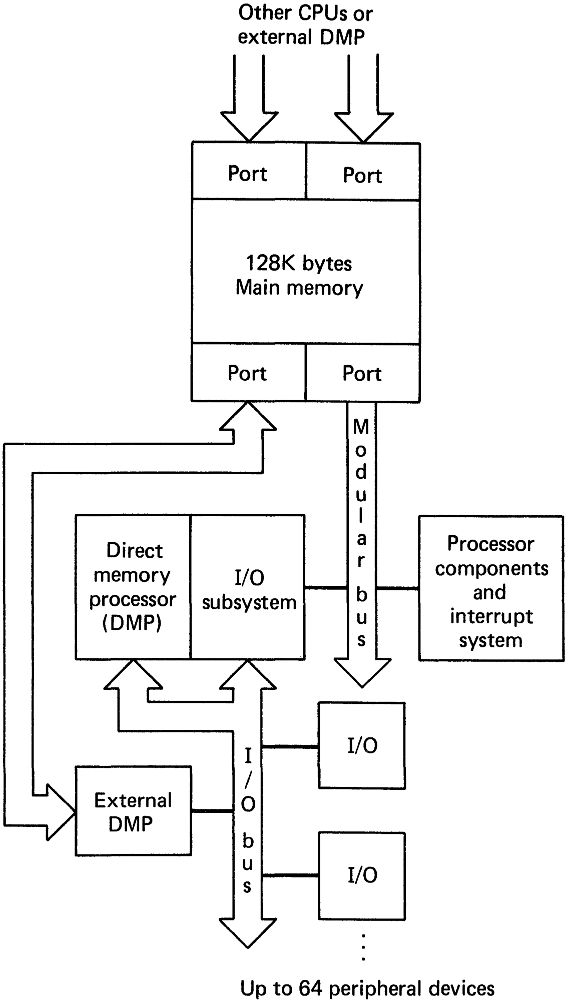

Other CPUs or external DMP
Port
Port
128K bytes Main memory
Port
Port
Modular bus
Direct memory processor (DMP)
I/O subsystem
Processor components and interrupt system
I/O
External DMP
I/O bus
I/O
Up to 64 peripheral devices

Figure 5.13
MODCOMP II/45 architecture

The simplest form of input/output on the MODCOMP II is the transfer of a byte or word between a register and peripheral via the I/O subsystem. Because of the high level of processor support required for such operations, it is only suitable for low-speed devices such as paper-tape readers, printers, and slow communications. The instructions used are input status, input data, output command, and output data. Each instruction contains the following information:

—The device number on the I/O bus (1 to 64)

—The address of the general register

—The direction of transfer

—The type of information (data, command, or status)

I/O instructions of this type may be executed in any sequence. The I/O subsystem can be supplied with a direct memory processor (DMP) for higher-speed peripherals such as magnetic tape, disc and communications multiplexors. Data is transferred along the same bus but at a higher priority, the DMP stealing cycles from the currently executing program as required. Up to eight devices can be supported in this fashion. An alternative DMP arrangement makes use of the multiport memory facilities. An external DMP can be attached to a memory port and effect I/O operations without cycle stealing. Such external DMPs are also connected to the standard I/O bus for program compatibility.

The MODCOMP II can have between 3 and 16 interrupt levels. Two of these levels can have 64 priority sublevels. Each level is assigned two fixed locations in main storage, the first for the return address in the interrupted program (stored automatically) and the second for the address of the routine to service the interrupt. A program can enable and disable interrupt levels as required. The two levels capable of supporting up to 64 sublevels are used by interrupts from the 64 I/O devices which can connect via the I/O bus. Each device is connected to two levels, one for character or word transfer and the other to indicate I/O completion. Figure 5.14 illustrates how peripheral switching can be implemented on the

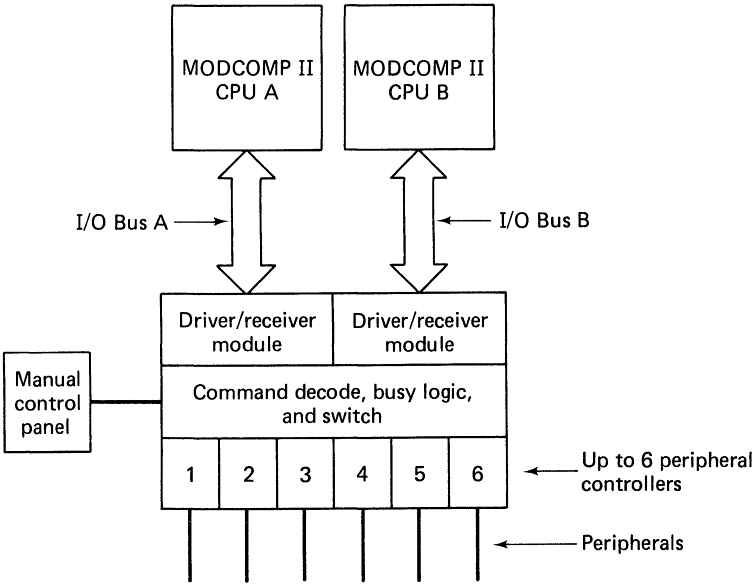


Figure 5.14
MODCOMP Model 4906 peripheral switch

MODCOMP II. This is somewhat more powerful than the PDP-11 system in that up to six peripheral controllers may be switched in any combination between two I/O buses under program or manual control.

Features of the MODCOMP II CP2

The MODCOMP II minicomputer is available in a special version particularly suitable for data communications systems. The CP2 differs from the basic MODCOMP II in that it supports the Universal Communication Subsystem (described below) and has a powerful set of firmware-implemented byte-string manipulation instructions. When miniprocessors are given the roles of remote terminal controllers, concentrators, and front-end processors, much of their workload involves data handling and manipulation rather than computing. However, most minicomputers do not have storage-to-storage instructions (strings being moved by load-register/store-register loops) and those that do usually transfer single bytes or words only. Inevitably, then, any processor that provides fast storage-to-storage movement of strings will have significant performance benefits.

But the communications-oriented instructions provided in the MODCOMP II CP2 can do more than simply move data strings. The complete list is:

—CRC or LRC generation and checking

—Storage-to-storage byte-string moves with compare, edit, and/or translate options

—Pack and unpack byte strings, also with compare, edit, and/or translate options

Although the operation is quite fast (1.867 microseconds per byte for a simple move to 2.66 microseconds for a move, translate, and pack) its use on large messages (say, 500–1000 characters) will inevitably "hold" the processor for a long time compared with a normal instruction. To prevent this from adversely affecting the performance of the system, the move instruction is interruptable after each character cycle. When the interrupt has been serviced, the move operation is automatically reinitiated by hardware at the next character.

The other main feature of the communications processor version of the MODCOMP II is the direct memory interface (DMI) which supports the Universal Communications Subsystem (UCS). The UCS (described later) provides the means to interface up to 256 full duplex channels. Each channel (both in and out) has a DMA path to main storage. The DMI controls the transfer of data over these DMA paths using a microprocessor with parameters held in 2368 words of random access memory (RAM). As well as being connected to the DMI bus, each UCS is also connected to the standard I/O bus for control purposes.

Figure 5.15 shows how each FDX channel has allocated to it nine words of RAM. Three are used for input control, three for output control, and three for

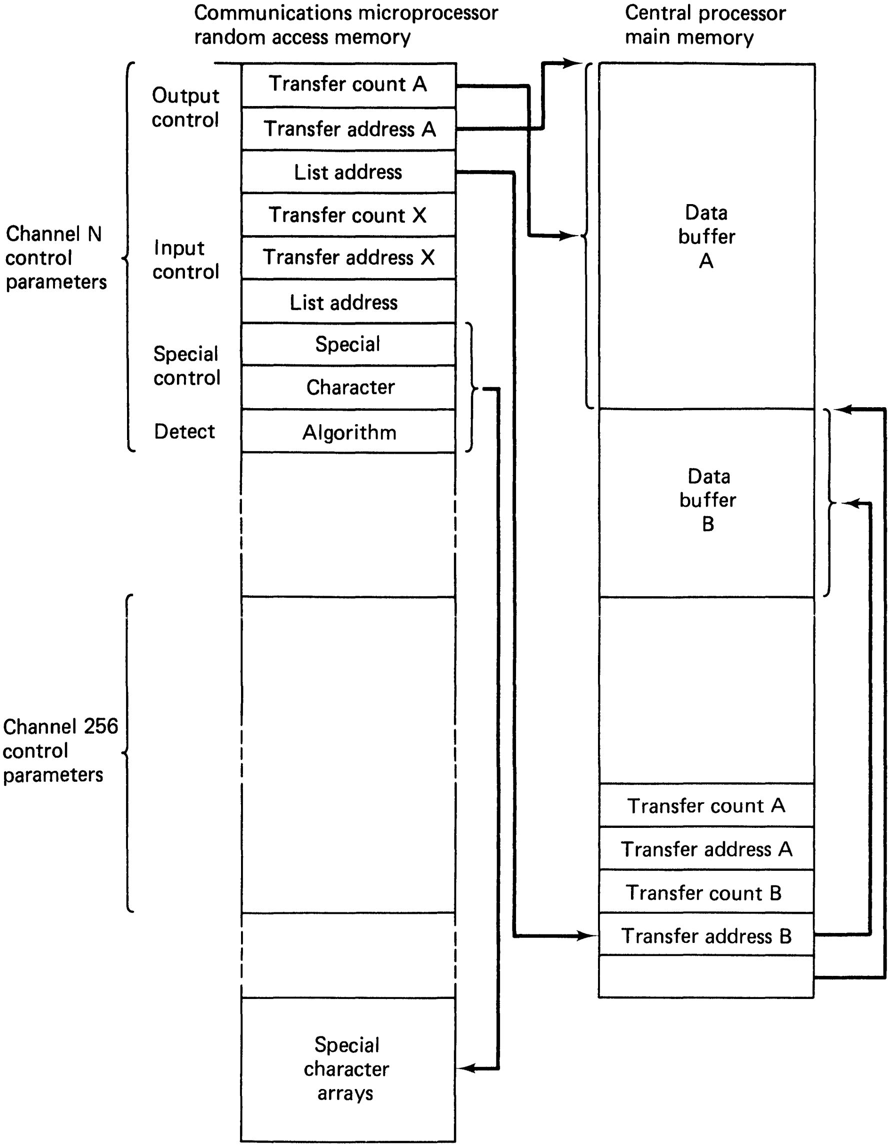


Figure 5.15
MODCOMP II CP2 direct memory interface arrangement

character detection purposes. Of the three words used for I/O control the first two are used for the transfer count and transfer address (in main storage), as would be expected. For most purposes, the use of a single buffer for each message will be satisfactory. However, there are circumstances where a multiple buffer arrangement may be preferable—for example, when a very large message needs to be read or when an adequately sized buffer is not available as a contiguous area of storage. The MODCOMP II Communications Processor supports chained multiple buffers in the following way. When the transfer count for the first buffer has been exhausted, the CP checks the third word of the I/O control parameters in the RAM. If this contains an address it will point to further transfer count and transfer addresses in main storage and will continue to the I/O operation uninterrupted. The buffers used need not be contiguous nor in any particular sequence. The chained buffer technique can be used for both input and output. The final three words in the channel control parameters are used for the detection of single or multiple character sequences in incoming data streams. The characters to be detected are held in two 4 × 16 byte arrays in the RAM (See D.W. Davis and D.L.A. Barber *Communications Networks for Computers* for details). This arrangement assists in the implementation of a variety of line protocols. The control parameters and the special character arrays can be loaded into the RAM or read back into main storage using a set of special DMI instructions.

The MODCOMP II Universal Communications Subsystem

The DMI is designed specifically to support the MODCOMP Universal Communications Subsystem Model 2 which is shown schematically in figure 5.16. The essential features of the UCS may be summarized as follows:

—DMA-type block transfers in both directions

—Multi-buffered transfers possible

—Up to 256 full duplex channels in any mix of synchronous and asynchronous

—Up to 8 communications chassis per controller

—Dual-port chassis available for duplex configurations

—32 FDX channels per chassis

—Programmable character-sequence recognition

—Maintenance commands and wraparound functions for monitoring controller operation

The synchronous channels have the following features:

—Bit rates to 250 Kbits per second

—CCITT V-series (RS-232C) or wideband interfaces

—Program access to and control of modem interface signals

—Software-selectable synchronizing character

—Software-selectable character size and parity

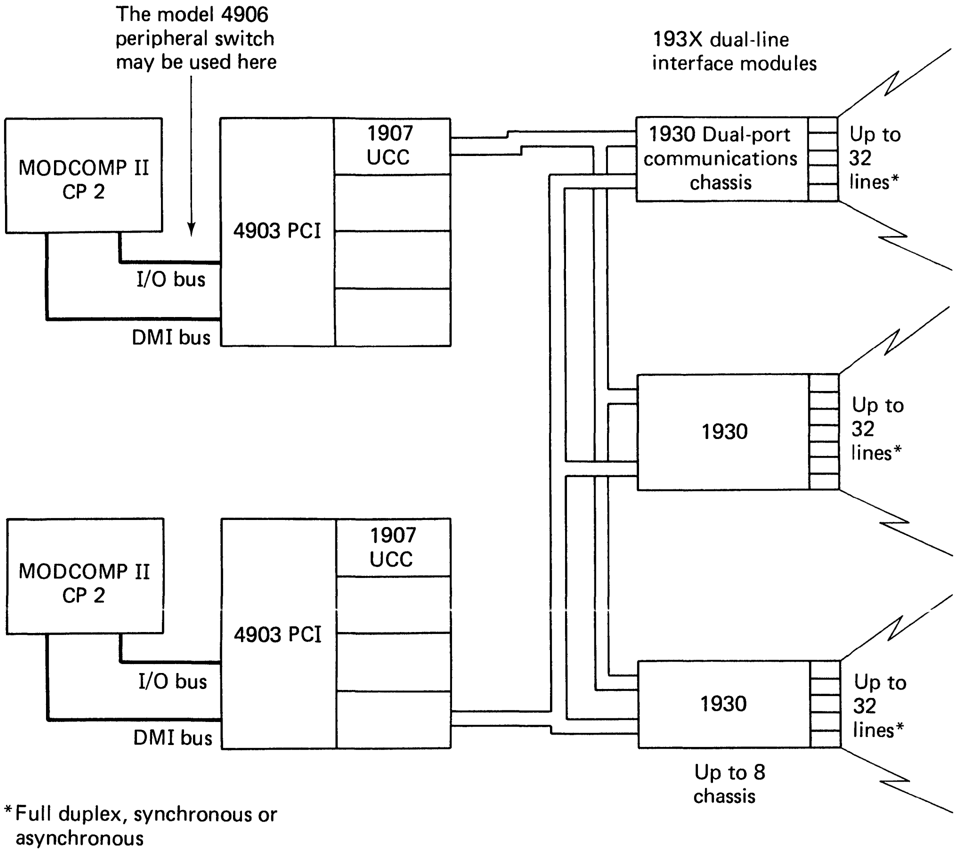

The model 4906 peripheral switch may be used here
193X dual-line interface modules
MODCOMP II CP 2
4903 PCI
1907 UCC
I/O bus
DMI bus
1930 Dual-port communications chassis
Up to 32 lines*
1930
Up to 32 lines*
MODCOMP II CP 2
4903 PCI
1907 UCC
I/O bus
DMI bus
1930
Up to 32 lines*
Up to 8 chassis
*Full duplex, synchronous or asynchronous

Figure 5.16
MODCOMP II CP2 Universal Communications Subsystem

The asynchronous channels have the following features:

—Software-selectable bit rate (any 1 of 16) to 19.2 Kbits per second
—Modem interface or current loop
—Program access to and control of interface signals
—Software-selectable character size, parity, and number of stop bits
—Break detect and break transmit (for Teletype operation)
—Software-controlled echoing (for Teletype operation)
—Split-speed operation (different bit rates for input and output)

Communications circuits are connected via modems (or directly if the current loop type) to 193X dual-line interface modules. In addition to providing the interface and control functions these modules:

—Assemble and disassemble characters
—Strip/insert start/stop bits
—Strip/insert synchronous characters
—Implement timing (from the chassis source)
—Check and generate character parity
—Echo characters to Teletype-compatible terminals
—Wraparound characters for diagnostic purposes

Each channel has a status register and can raise an interrupt on the CPU when a status change occurs so that the program can obtain the register. This status register for an input channel is shown in figure 5.17.

The 1930 communications chassis can support up to 16 dual-line interfaces (32 FDX channels). Its main function is to provide timing to the channels. The 16 asynchronous rates are program selectable; the synchronous rates are also, by means of hardware jumpers. Any mixture of asynchronous or synchronous rates may be selected.

Up to eight 1930 communications chassis may be connected to a 1907 Universal Communications controller, which performs the following tasks:

—Command decode
—Scanning and control of channels
—Interrupt and status control
—Support for the DMI

The controller operates under microprocessor control, permitting a high degree of flexibility in operation. For example, higher-speed lines can be scanned more frequently than the lower-speed ones to ensure the maintenance of throughput.

Summary of MODCOMP Interface Units

The MODCOMP II CP2 is an excellent example of how microprocessors can be applied in both the multiplexor and the CPU in order to provide communications-related functions in an optimum fashion (in terms of processor and main storage utilization). The multiport architecture of the MODCOMP range also makes it particularly suitable for high-throughput teleprocessing applications, either as a remote controller or as the central-site machine itself.

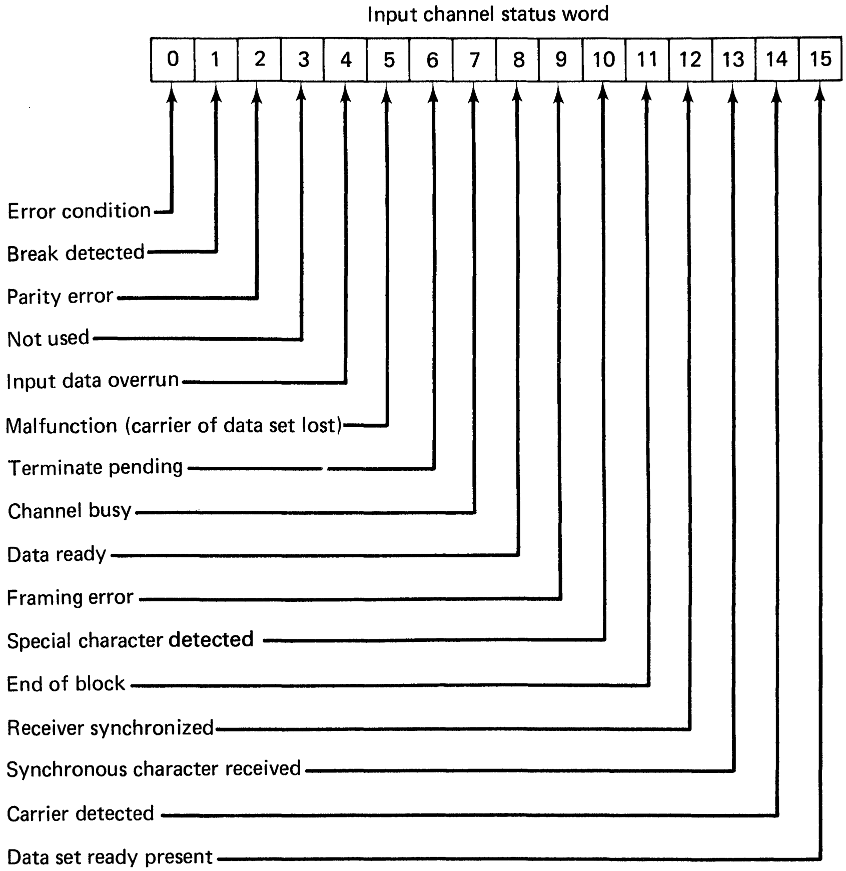


Figure 5.17
MODCOMP II CP2 UCS input channel status

6

Data Communications Software

This chapter examines some of the fundamental problems of control software and applications programming for teleprocessing systems. Practical solutions to these and other functional requirements will be discussed in the context of some operational control programs.

Task Management

Writing programs for teleprocessing applications is very different from writing programs for batch systems. Any software package that is being sold on the basis that it makes teleprocessing programming "easy" needs to be examined with great care. Why is this? Let us start by looking at the components and logical structure of a typical batch program (figure 6.1). Most—if not all—modern operating systems (or executives) provide multiprogramming support; in our example we have six programs running concurrently in six partitions (or regions). If we look at one of those programs in more detail it will probably comprise four main components: the application code, constants, work areas, and I/O buffers.

The application code will be the machine representation of the instructions to be executed by the processor. These instructions may modify themselves (for example, by changing the address in a branch operation), but currently recommended programming practices advise against such techniques. By definition, constants are fields that are not modified during the execution of a program and may include look-up tables, headers for print routines, and edit masks. Constants (especially those generated by compiler "literal" features) are often interspersed with code.

Work areas contain values that do change during the execution of a program, such as register storage areas, loop counts, and accumulators. These are also found interspersed with object code, but sometimes the structure of the program-

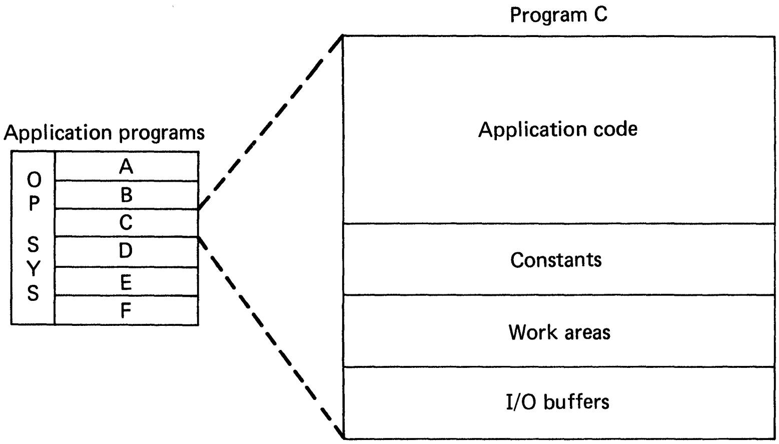


Figure 6.1
Structure of a typical batch program

ming language (say, COBOL) prevents this. Although the practice is generally not recommended, addressing limitations on some computers give the programmer no choice. The fourth part of a program is the buffer, which contains data records obtained from external media or storage or records generated by the program for subsequent output to a file, printer, and so on.

Figure 6.2 shows a simplified flow chart for our specimen program, which is a master file update. This is what the program does:

—A preedited (error-free) transaction is read from a magnetic tape file

—The transaction contains a customer number which is used by the program as the key in a read from the direct-access master file

—The transaction is used to update the master file record

—The record is written back to the master file;

—If the end of the transaction file has been read the program ends, otherwise it loops back to read the next transaction and continue

The basic average timing of the program loop is made up as follows:

READ A TRANS	5 ms (records are blocked)
READ A RECORD	45 ms
PROCESS	5 ms
WRITE THE RECORD	45 ms (assuming the disc read head has moved)
CHECK END OF TRANS FILE	Not significant
TOTAL	= *100 ms*

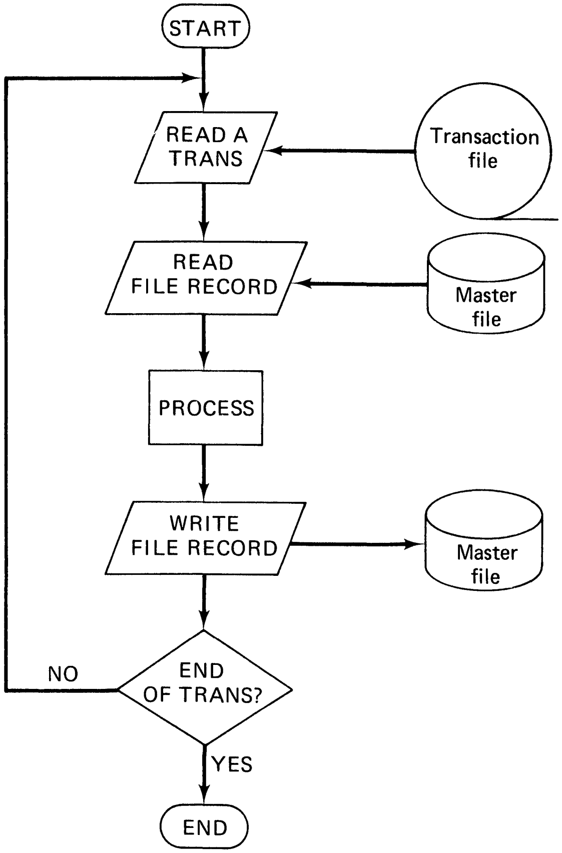


Figure 6.2
A simple batch update program

Because we are working in a multiprogramming environment, control will be passed via the operating system (OS) to another program during the three I/O operations and be returned when they have completed (assuming that our program has highest priority).

Now let us assume that we have to modify the program for a situation where the transactions, instead of being on a sequential tape file, are going to be keyed in at a number of local and remote visual display terminals. From the terminal operator's viewpoint what happens is this:

—Key in a transaction (into the terminal buffer); this takes about 5 seconds

—Press the send key

—Wait for the computer to read the message, process it, and send an acknowledgement when the update has taken place

—Check the acknowledgement and if OK, enter the next transaction

The transaction cycle has been kept abnormally short for the sake of simplicity. A response time of one second or less is considered desirable.

Our update program has now been modified in the following way. Included in the partition is a simple terminal handler which responds to read/write macros. When a read is issued it polls the terminals looking for one with a message ready to send. This is read and placed in a buffer for processing. When a write is issued, the specified terminal is selected and, if ready to receive, the message is transmitted. The revised program logic is shown in figure 6.3:

—The program requests any available message

—The message is then verified (we ignore the procedures for dealing with bad ones)

—The relevant master file record is read

—The transaction and file record are processed

—The master file is updated

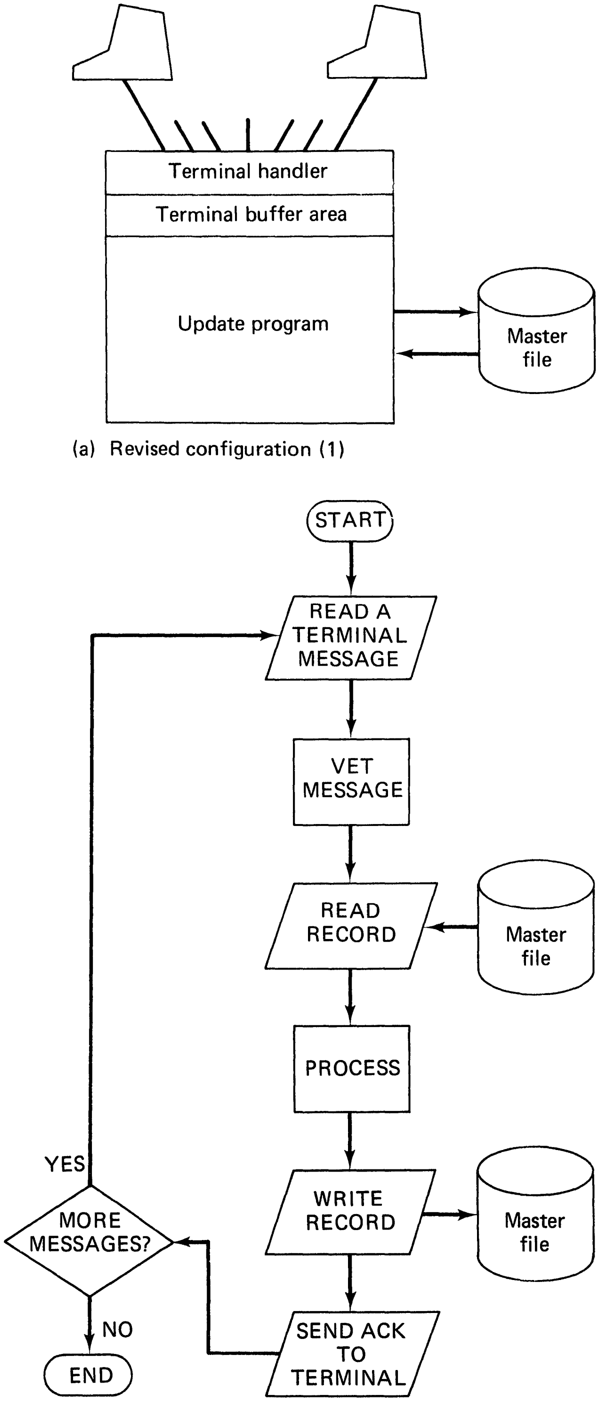
—The program sends the acknowledgement to the terminal

—The program reads the next available message and continues

We stated previously that the above sequence takes about 100 milliseconds to execute. Since we require a response time of less than 1 second, our system should work magnificently—and it would, if we had only one terminal attached, but in fact we have over 100 and they are generating messages at a peak rate of over 20 per second. What are the implications of this?

Let us assume that it takes the terminal handler an average of half a second (500 ms) to accept a read, poll, and receive a message. The program will then begin executing for about 100 ms and the terminal handler might take a further 100 ms to send the reply. This gives a total of about 700 ms per message. Even that might be satisfactory if there were only one terminal, but because the program is processing only one message at a time and the messages are arriving at up to 20 per second, our brand-new on-line system is not going to work. This is particularly galling when one considers that 10–20 ms per message is actual processing time. The rest is taken up by terminal and disc I/O operations (with processor control being transferred by the operating system to other partitions).

One way out of the problem might be to make use of the multiprogramming facilities provided by the operating system. This would mean setting up a separate partition (or region) for each terminal so that a different copy of the same program is provided for each possible source of input (see figure 6.4). When program 1 starts an I/O, control will be transferred to the next highest priority program requiring the processor in order to continue. This approach has two major deficiencies, one of which is glaringly obvious, the other less so. First of all, to have a separate copy of the program for each terminal is extremely inefficient and expensive in terms of storage utilization. Second, the time taken by the OS to reschedule programs when a wait occurs is finite. Furthermore, the rescheduling



(a) Revised configuration (1)

(b) Revised program logic (1)

Figure 6.3
A simple on-line update program

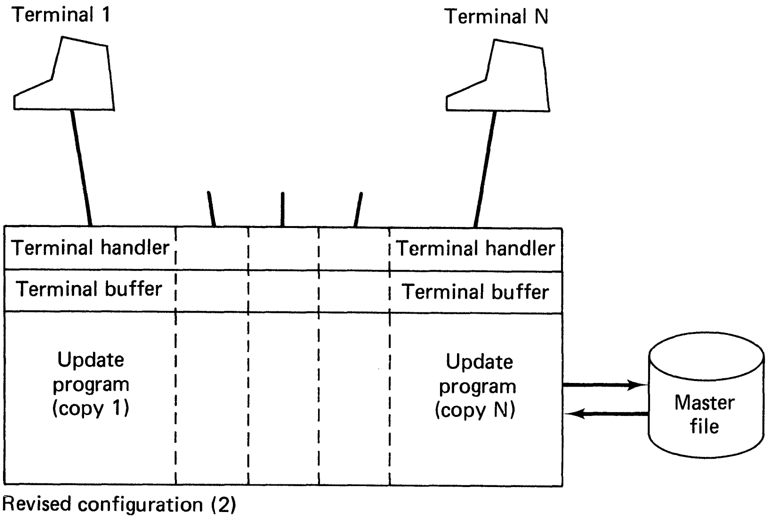


Figure 6.4
Use of multiple program copies

(or context-switching time) generally has an exponential relationship to the number of active partitions in the system. The use of separate partitions for each terminal will clearly cause a large increase in the OS overhead.

What other alternative is there? The first problem we can deal with is that of the terminal I/O operations. Clearly, these should be overlapped with the message processing. The amount of CPU time required to service the lines (start I/Os, route characters to buffers, and so on) should be relatively small. Furthermore, the interrupt structure of the computer serves to tell the terminal handler when some action must be taken. Let us assume that each time a line requires processor intervention, all other operations are interrupted and the terminal handler is activated. The interrupt hardware places the relative line number (RLN) of the terminal causing the interrupt in a register. Because each line will be at a different stage in an I/O operation (negatively acknowledging a poll, sending a message, acknowledging receipt of a message), then the handler will need to access information specific to that line. Such information can be kept in a line control table (LCT) as shown in figure 6.5 and the RLC can be used to find the right table. The LCT might contain information such as:

—Relative line number
—Last operation on this line
—Address of message buffer
—Message character count
—Address of line status word

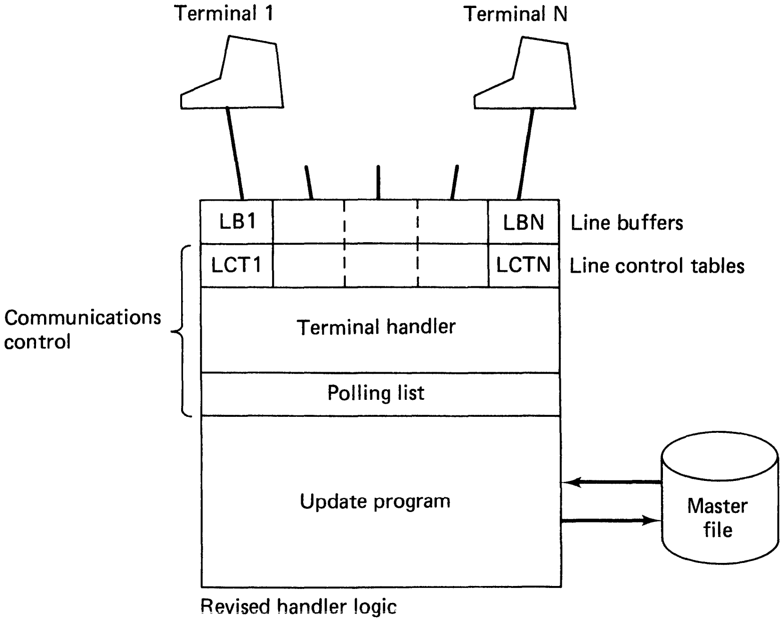


Figure 6.5
Use of an interrupt-driven terminal handler and line control tables

—Other status bits

—Other counts (e.g., bad message rereads)

A typical sequence followed by the drive might be as below (assuming that return addresses and the like are saved by hardware):

—Inhibit further interrupts from the lines

—What was the last operation on the line? (poll sequence sent, say)

—What is status of line? (message received)

—Was parity good? (yes)

—Change status to indicate ACK sent and message ready for application program

—Send ACK

—Deinhibit interrupts and finish

The logic of the routine will obviously vary depending on the last action indicated and the status of the line. In addition to being activated by interrupts from the line adapters (or CCU) the terminal handler may also be activated from the application program and by the CPU's interval timer. In the former case the program may issue a "get-me-a-message" macro and this would cause the handler to scan the LCTs looking for a message ready status. When one is found, control can be returned to the program with the address of the relevant LCT as a parameter. The program can find the message because the LCT contains its address and length in main storage. If the program wishes to send a message it will be to a specific terminal (ignoring the specialized problem of broadcasting) and will need to pass to the handler the RLN, address, and length of the message to be transmitted. If the status of the line is all right, the handler can then go through a select/transmit sequence (but not, of course, waiting for a response from the terminal—the interrupt facility will reactivate the handler when required).

The other source of interrupts to the handler will be the interval timer. We mentioned before that the terminals are being polled and, at the design stage, it may have been decided that a terminal will be polled at least every three seconds. This can be achieved by setting the interval timer to three seconds (or perhaps a little less) and having the resultant timer interrupt activate the polling routine which, working from a polling list, will send polling characters to the inactive terminals and set bits showing the action taken in the LCTs. An alternative method of working would be to use contention mode on the lines and have the incoming ENQ cause the interrupt. In a situation where a line might connect to multidropped or clustered terminals, we may have separate control tables for the line and the terminal (the latter containing a pointer to its LCT.) Our rather stupid line driver now has the makings of a full communications control program (CCP) or, as it is sometimes called, a teleprocessing access method. We shall elaborate on the function it may perform as we continue with this section and the next.

By improving our line handler we now have a CCP which gives us (thanks to the hardware interrupt structure) completely overlapped communications. The terminal operator no longer has to worry about waiting for the other terminals to send and receive. However, even when his input transaction has arrived at the CPU he still has to wait in line for the actual update program to work its way around to him. We have what is known as a *single-threading* system whereby incoming messages are processed serially; one must be completed before the next one starts. In systems where the number of terminals and/or the amount of traffic is low, single threading is a simple and satisfactory approach. But this is not the case in our example, where the message rate is peaking at 20 per second but we are only processing at a rate of 10 per second. If we refer back to our timing of the update program, we can see that most of the transaction time is taken up with the disc read and write. If we can overlap these in some way it may be possible to process messages in parallel. This is known as *multithreading*.

In order to achieve multithreading within a program partition we need to extend our use of the interrupt structure. But first we have to break our program down into *tasks* or *subprograms* that will be activated and deactivated at the right times. An ideal task is a small block of code that *ends* in an I/O operation. Figure 6.6 shows this technique applied to the simplified update program:

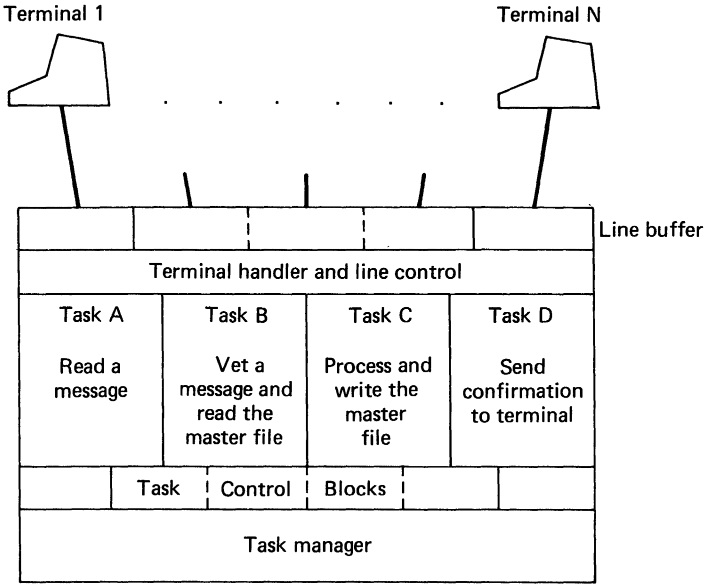

Terminal 1
Terminal N
Line buffer
Terminal handler and line control
Task A
Read a message
Task B
Vet a message and read the master file
Task C
Process and write the master file
Task D
Send confirmation to terminal
Task
Control
Blocks
Task manager

Figure 6.6
The simple update program converted to a multitasking structure

TASK A Merely reads an input message

TASK B Vets an imput message (again ignoring the problem of what is done with the bad ones) and ends by issuing a read for a master file record

TASK C Processes a transaction and corresponding record and ends by issuing a write to return the updated record to the master file

TASK D Sends a confirmation message back to the originating terminal

In order to control the parallel flow of messages through our restructured program we need a *task manager* (or task dispatcher) of some kind. The task manager (TM):

—Provides a task with the message to be processed (or a pointer to the message)

—Initiates the task by transferring control to it

—Accepts control back from a completed task

—Initiates disc I/O operations on behalf of the task (the task branches to the handler, passing over parameters indicating the I/O required along with the key, and so on)

—Accepts interrupts from the disc indicating the completion of an I/O

—Determines the next task to be initiated

The status of each task is maintained in a task control block (TCB). Once activated, a task can only be interrupted by the task manager's receipt of an I/O interrupt or, in more sophisticated systems, by a higher-priority task (usually those later in sequence so that messages can be cleared from the program).

Let us follow a few transactions through the new program. To keep things simple we will assume that when a read is issued for a message, there is always one available. At the start of the day, the program might go through the following sequence:

—Activate *Task A,* which issues a "read message" macro

—Control is transferred back to the TM which in turn calls the terminal handler (TH)

—Control comes back from the TH to the TM which now has the address of a TCB

—The TCB address is passed to *Task B,* which is activated

—*Task B* vets the message found via the TCB and returns the TM when it issues the macro for reading a master file record

—The TM issues the master file read but, as *Task C* cannot start until the read has completed, the processor is inactive. However, *Tasks A and B* have nothing to do, and so

—The TM initiates *Task A* to get another message and passes it to

—*Task B* for vetting and the reading of another master file record

—At this stage the TM will, if possible, issue another master file read and initiate *Tasks A and B* again. It will continue to accept new messages into the system until it runs out of available message buffers. But presumably, before that happens, the first disc read will be completed

—The TM will activate *Task C* to process the first message/record combination. At the end of *Task C* control is returned to the TM with a request to write the record. This will cause a further wait and, by examining the TCBs the TM will determine which task is to be activated next (it could be *Task C* again on the second message)

—The current task will eventually be interrupted by the completion of the disc write and this will enable the TM to activate *Task D* to cause the sending of a reply

—*Task D* will probably not execute immediately (the TM merely indicates in its TCB that it *may* execute) because control will need to be returned to the interrupted task so that it can finish. The TM only initiates new tasks when it knows that they have all completed their current execution (when control is returned via an I/O macro). At that time, the TM will examine all TCBs to determine the best task to initiate next. We have now achieved a situation where processing is overlapped with disc I/O (as well as communications I/O) and the disc I/O operations may overlap themselves (if the master file goes over more than one spindle).

This technique of rescheduling tasks each time the application program returns control to the TM is used by IBM's CICS. This control routine multithreads messages through a program as long as the code between the macros that call CICS is *serially reuseable,* that is, it may modify itself during execution as long as it reinitializes itself prior to completion. Although IBM's approach to task management is to include it in a high-level communications control program, some systems (especially minicomputers) provide the multitasking resource from the operating system itself. It used to be necessary, in some cases, for the customer to have to write his own task manager, but that was not necessary on the computer systems being produced in the late 1970s. One of the deficiencies of the multitasking system described above is that a task cannot be suspended in favor of a higher-priority task. A task can only be suspended to service an I/O interrupt and then control must be returned to it so that it can finish.

An alternative approach to the problem is the use of *reentrant* code. This is code that does not modify itself during execution and can, therefore, be suspended at any time in favor of another task and even enables itself to be reinitiated for another message. Such a technique requires a slightly less sophisticated task manager but even greater throughput is often (but not invariably) achieved. Fundamental to this style of programming is the isolation of variable fields into tables (usually extensions of the LCTs) and the *indirect addressing* of these through the use of registers. Figure 6.7 shows our update program further modified to use this technique.

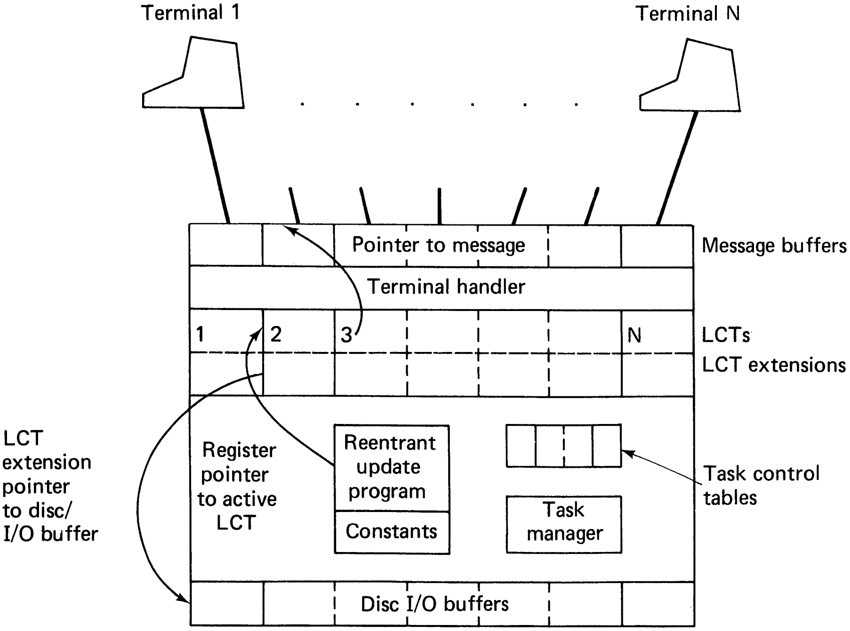


Figure 6.7
Multitasking using reentrant application code

No longer is it necessary to break our program down into tasks. What we are effectively doing is redefining a task as a *transient relationship* between our porgram and the message it is processing. When the message enters the system, the TM creates a task control table specifically for it. When the program begins execution, it is provided with a register pointing to an LCT which has been extended to include those fields that are going to vary during the processing of the message and pointers to I/O buffer areas (for messages, disc records, and so on). Although constants can still be addressed directly these variable fields must be addressed via registers (perhaps with the help of logical overlays like IBM DSECTs). The I/O SVC macros are in-line with the code, and when executed cause control to be transferred back to the TM for initiation and task rescheduling. Unlike the system using serially reuseable code, a task can be suspended so that another task using the same code but processing a different message can be reactivated after a wait for I/O. On suspension, all the TM has to do is store the current registers in the task's control table and load those of the task to be received. Return to the original task is effected in the same way. When the program has finished processing a particular message, the task that that temporary relationship represents is terminated by the TM.

One of the benefits of multitasking using reentrant code is that the efficiency of the multithreading is less dependent upon the cleverness of the task manager. The most efficient method of processing the available message traffic tends to be adopted automatically. Side benefits include the fact that neither the program nor the TM requires modification if the system is expanded. All that is needed are additional line control tables (and/or terminal control tables in a line-sharing situation). Reentrant code can be written by experienced assembler programmers with little difficulty. A few ground rules and a little experience is all that is required. Producing reentrant code from high-level languages is a different proposition, and here the programmer is in the hands of the compiler provided by his computer supplier. For example, IBM's PL/1 compilers produce reentrant code but the COBOL compilers do not (even through the structure of COBOL would seem to lend itself to reentrancy just as much as PL/1).

Program Management

Our discussion of the programs of task management have been based on the assumption that every terminal will be requiring access to a single program (or subprogram). In some systems (say, message switching) this may be the case, but in many—if not most—commercial teleprocessing systems, a number of programs will be required to support a variety of message types arriving from terminals. Figure 6.8 shows a possible software structure for program management in a teleprocessing environment. Task management considerations have been omitted for reasons of clarity. Fundamental to program management is the program manager or message router, which has a number of functions to perform.

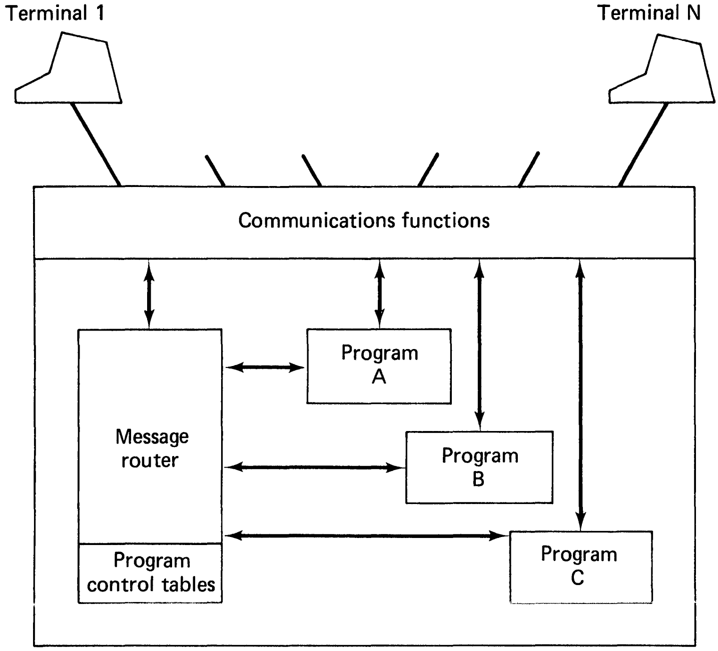


Figure 6.8
A possible program management structure

It must:

—Receive any incoming message for an as-yet undetermined destination

—Decide which program is required to process the message

—Find out whether the program is already in main storage

—If not, determine whether there is space for the program to be loaded into main storage

—If or when the program is ready, pass control to the task manager

Although programs are usually initiated only by incoming messages (in the manner described below), once execution has commenced it is normally possible for them to deal directly with the TM (via the LCT) for all subsequent messages. The simplest form of program management operates at a sign-on level. When the terminal operator first connects to the system, he indicates the program he wishes to use and remains signed-on to that until the end of his session, when he may wish to start a new session with a different program; this is common on minis. The approach that allies itself to the transaction processing concept used by TP monitors permits switching between programs on a transaction-by-transaction level. This requires the allocation of the first four (or six, eight, and so on) characters of a message to an alphanumeric *transaction code* which must be keyed in by the operator (or automatically inserted by the terminal). Some terminals, such as the IBM 3270, have *function keys* which, when pressed, cause the transmission of a message from a terminal preceded by a code unique to that key. This character can then be used to route the message to the relevant program.

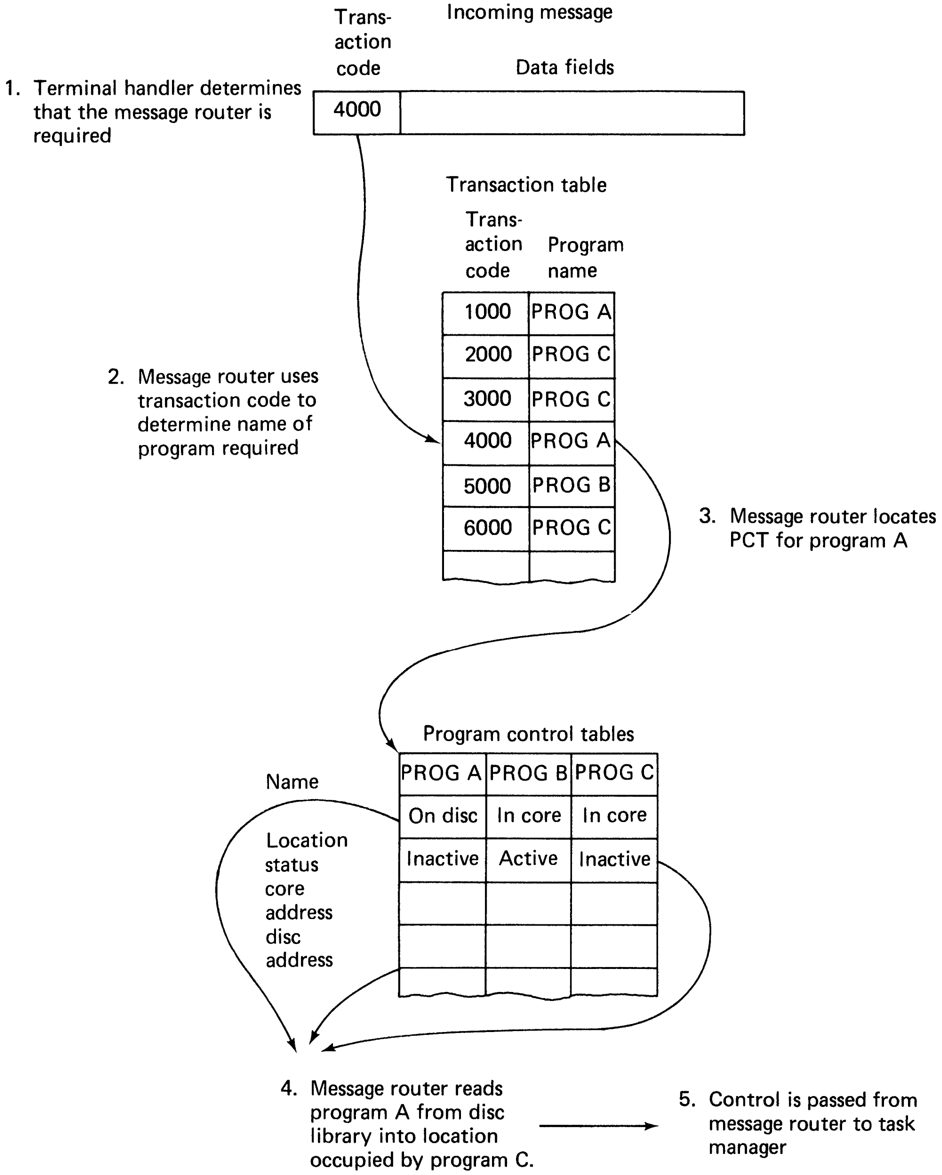
Figure 6.9 illustrates a typical sequence of events. The line control table contains an indicator that shows whether the next incoming message has been specifically read from that terminal by an already active program. If the TH sees that the indicator is off it will pass the message (via a pointer in the LCT) to the message router (MR). The router then examines the first four characters of the message and tries to find an entry for them in a transaction table (TT). If no entry exists, the MR will send an error message back to the terminal operator.

If an entry is found it will contain the name of the program that will receive the message. The MR will then examine the PCTs to determine whether the program is available. In the example shown in figure 6.9, the program required (PROG A) is currently in the disc-resident program library. Is there room for it to be loaded into main storage? The PCTs show that, although PROG B is resident and running, PROG C is inactive and can be overwritten. If the space occupied by PROG C is large enough for PROG A, the message router will read in PROG A and pass control to the task manager.

Storage and Queue Management

Another major programming problem not normally encountered in batch systems is that of storage management. Let us approach our discussion of this by concentrating initially on the problem as it relates to input/output buffers. Figure 6.10 illustrates the queues normally encountered in a fairly typical batch processing program. The program reads transactions from a tape file using a queued sequential access method. This allows the file to be read in advance of the program's requirements, the intermediate records being stored in a prespecified number of buffers. A similar queued technique is used for the output of a print spool file.

Queued access methods cannot be used for the indexed-sequential master file because records are retrieved on a random basis and then rewritten *in situ*. Therefore, single buffers are allocated for input and output to this file. Referring to the schematic of a typical teleprocessing system shown in figure 6.11 we can see that the major difference is the number of buffers required for handling the communications lines. There will also be an increase in the number of buffers required for file handling on direct-access storage devices. This is caused by the fact that a number of parallel tasks could all be issuing reads and writes on a single file.


Trans-
action
code
Incoming message
Data fields
4000
1. Terminal handler determines that the message router is required
Transaction table
2. Message router uses transaction code to determine name of program required
3. Message router locates PCT for program A
Program control tables
Name
Location
status
core
address
disc
address
4. Message router reads program A from disc library into location occupied by program C.
5. Control is passed from message router to task manager

Trans-action code	Program name
1000	PROG A
2000	PROG C
3000	PROG C
4000	PROG A
5000	PROG B
6000	PROG C

PROG A	PROG B	PROG C
On disc	In core	In core
Inactive	Active	Inactive

Figure 6.9
A generalized program management function

It may be useful at this point to clarify the difference between buffering and queuing. *Buffering* is concerned with the provision of an area in main storage for the assembly or disassembly of a message (or record) that is in the process of being read into or written out of the central processor. The problem of queues

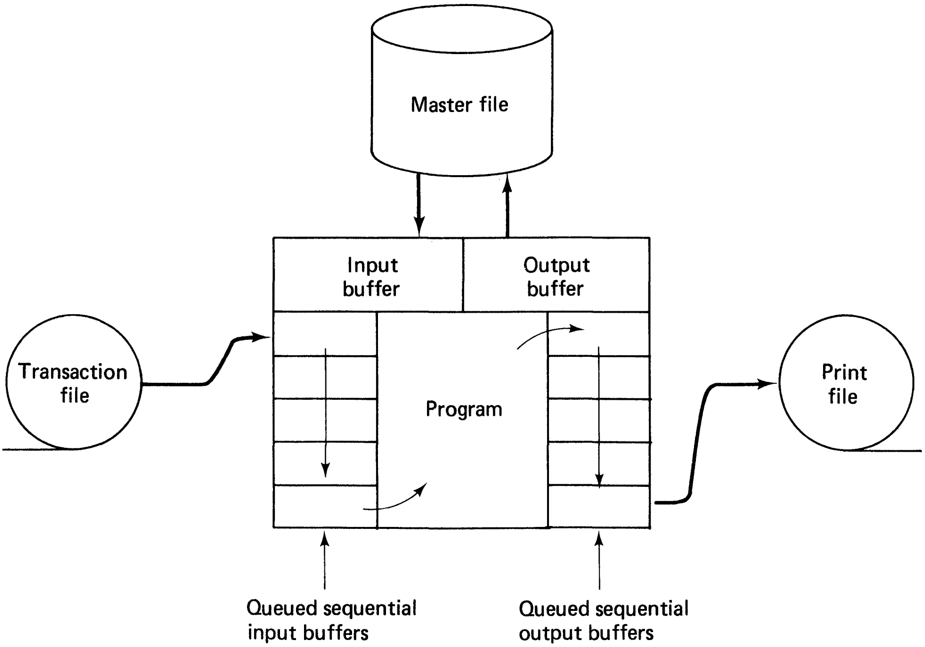


Figure 6.10
Input/output buffers found in a typical batch system

tends not to occur in batch systems because the queues tend to reside outside the system on peripherals, a new record being read in only when the program is able to process it. This is not the case with a terminal-based system, where to refuse to read a message until it can be processed immediately would leave terminals and their operators inactive. This produces a situation where messages for a particular program may be arriving simultaneously and need to be organized in some efficient manner for processing. Let us now examine some of the arrangements used to manage the buffering and queuing problems associated with teleprocessing.

At the very lowest level we often find the use of a rotating character buffer known as a *tumble table* (figure 6.12). This technique is particularly suitable for use in minicomputers when lines are being handled on a character-interrupt basis as with the DEC DL11 described earlier. In order to minimize the time spent in the interrupt service routine, the routine merely places the character and the relative line number in the next free slot in the tumble table. A low-priority task then takes the characters out of the table (in the sequence in which they were entered) and assembles them into complete messages. The tumble table reduces the risk of overrun on the line buffers by getting rid of the incoming characters as quickly as possible. Care must be taken, however, to ensure that the table is large enough to accommodate short bursts of high line activity.

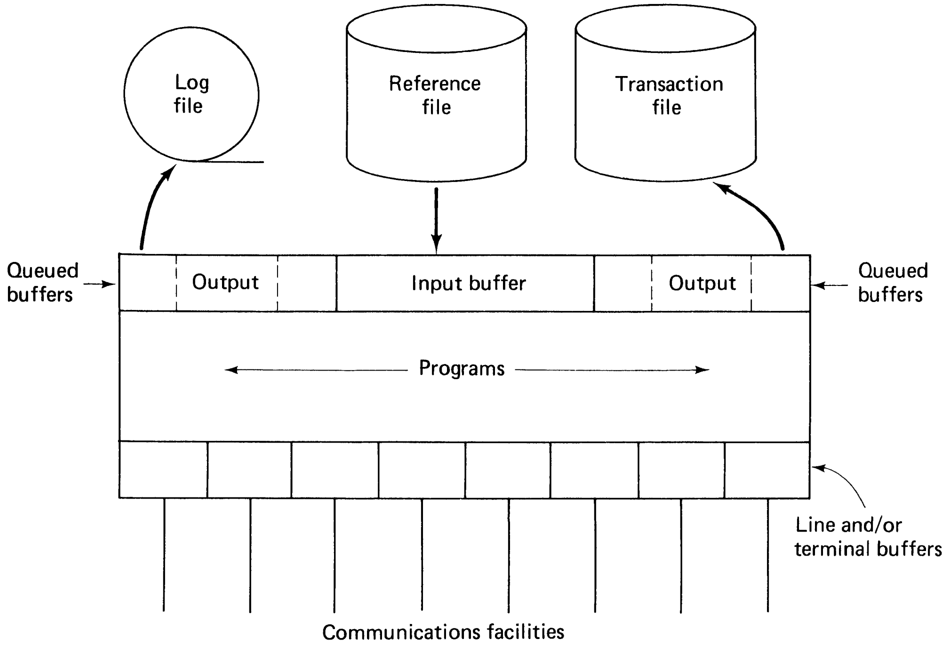


Figure 6.11
Input/output buffers found in a typical teleprocessing system

Where DMA channels are being used for the line I/O, then messages can be assembled directly into line buffers. Once a message has arrived it can be passed to the program that will process it. If the program is busy, a queue will form. Figure 6.13 shows a simple arrangement whereby completed messages are placed in first-in/first-out queues for the processing programs. The movement of the message to the queue may be physical, thereby freeing the line buffer for a subsequent message, or logical, the program being provided with a pointer to the message buffer. A similar arrangement can be used for output, with the control program scanning the output queues and moving the messages to the relevant output buffer for the destination terminal.

Depending upon the nature of the application and the level of message traffic, this approach can be very expensive in its use of main memory. An alternative approach makes use of direct-access storage for the temporary queuing of input and output messages. As messages arrive in the system, the queue manager automatically writes them to disc- (or drum-) resident queue files, one for each process. The application programs then work rather like batch programs, reading each message in turn from the input queue file and writing appropriate responses to the output queue file (figure 6.14). This technique eases the main storage problem at the expense of performance and response times. Some systems get the best of both worlds by using a combination of the two techniques. Each program is

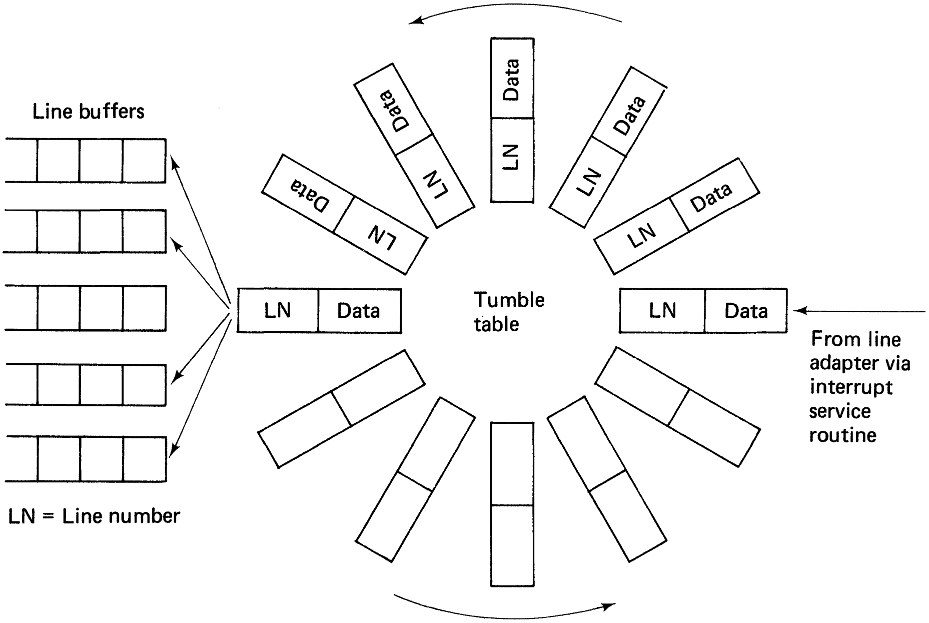


Figure 6.12
"Tumble table" rotating buffer used as an interface between interrupt service routines and buffer handler

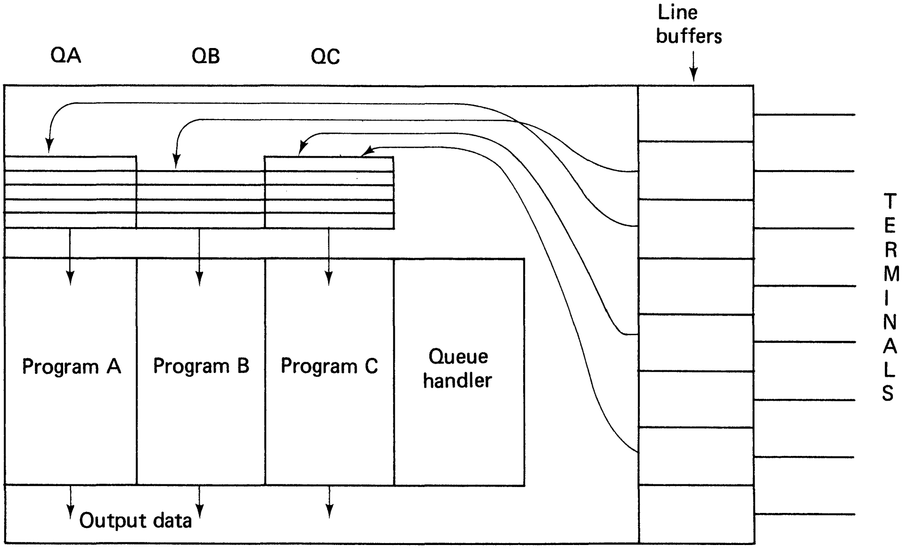


Figure 6.13
A possible main memory queue management system for incoming messages

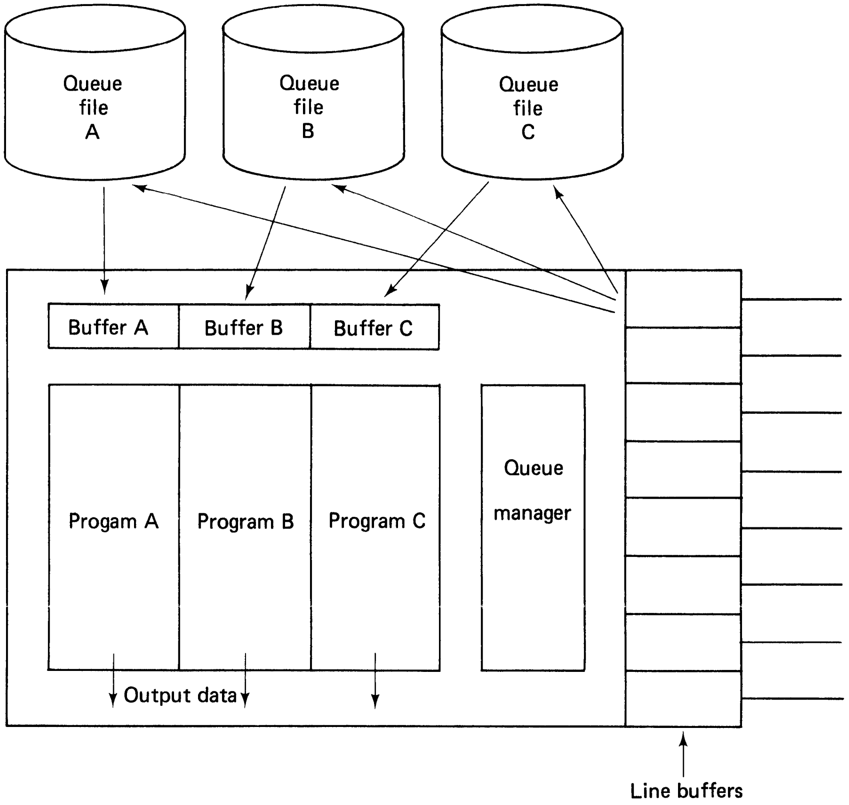
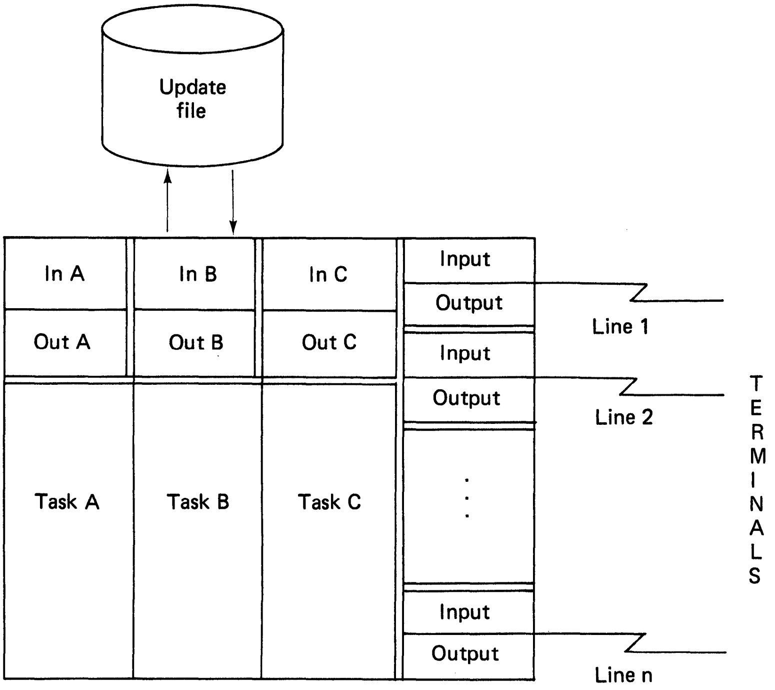


Figure 6.14
A disc-based queue management system for incoming messages

allocated a limited core-resident message queue of (say) five messages. If the queue length exceeds this, the queue manager automatically writes the additional messages to disc and reads them back in when space is available. This arrangement serves to ensure that the system is always able to accept messages regardless of the level of traffic.

Let us return to our basic buffering requirements in order to assess a problem related to—but different from—that of queuing. Figure 6.15 shows again the buffer arrangement in a fairly common data communications system. In this system the line buffer size is determined by the platen width of the remote keyboard printer terminals used. This is, say, 132 but we make a small allowance for additional nonprinting control characters and round the buffer size up to 140. Messages are processed in their input buffers and this prevents the use of the same buffer for the reply. So there are two buffers per line and, since there are 100 line appearances, we have 200 buffers each of 140 bytes, giving a total storage requirement of 28,000 bytes. If the terminals were visual displays that might have



FILE HANDLING
Block size: 512 bytes
Number of buffers: 6
Area required: 3K

TERMINAL HANDLING
Maximum message size: 140 bytes
Number of lines: 100
Number of buffers: 200
Area required: 28,000 bytes

Figure 6.15
A static storage management scheme for buffer handling

local buffers of up to 2000 characters, then the space to be allocated in main storage could be very much worse.

The point that has to be made, however, is that the utilization of these buffers is probably very low. At any given time the majority of the space allocated is probably unused. In order to overcome this problem it is common to use a *dynamic* storage management facility (figure 6.16). This requires an application program (or, indeed, a control program) to request a specified amount of storage from a common storage pool. When the program no longer requires the area, it is returned to the pool for use by other programs. Dynamic storage management may be part of the operating system (executive) or, failing that, the installation may have to develop its own routines.

The use of queuing theory will help the designer determine the length of the queues that will form in the system. In any case, once a dynamic storage facility

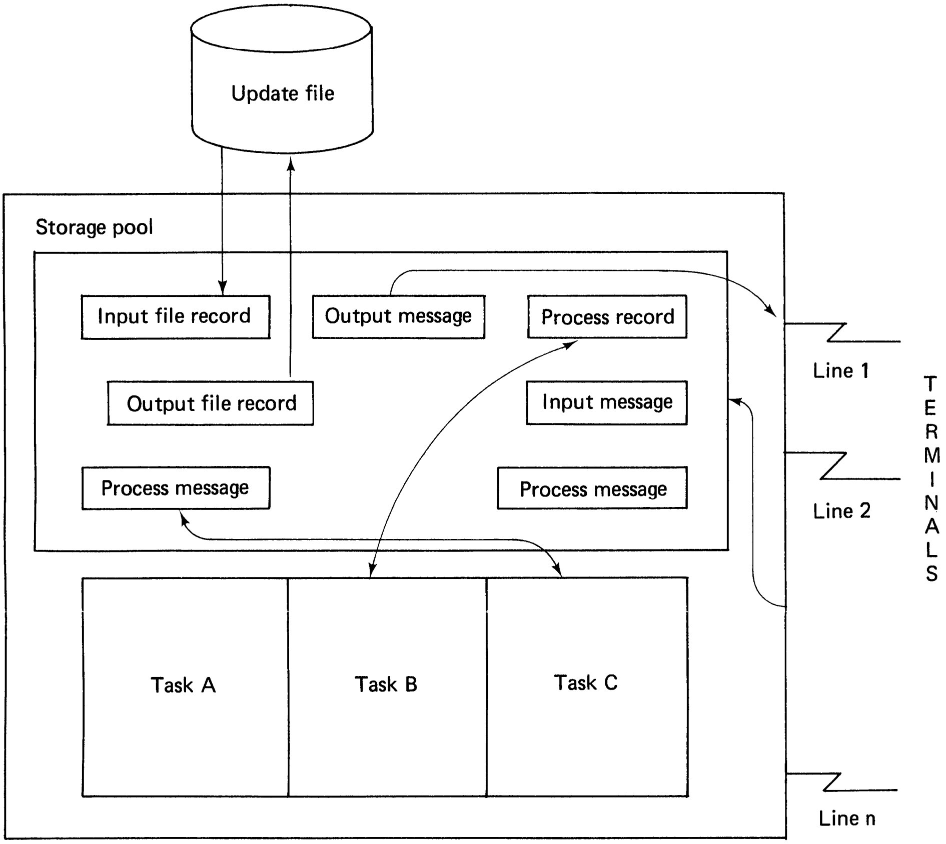


Figure 6.16
A dynamic storage management scheme

has been implemented, the size of the common pool can be adjusted as part of the system tuning.

A variety of techniques can be used to achieve dynamic storage allocation. The segments requested may be fixed length or variable length. Although the latter approach can be more efficient (the variable pool size is smaller) it can give rise to the problem of *fragmentation*. This occurs when, although there is plenty of space available in the pool, there is not a single contiguous area large enough to satisfy the outstanding request. This problem can be overcome by *chaining* areas together (see figure 6.17). Note that the chaining may be implemented using tables and/or buffer headers/trailers and the programs need to be written to allow for the arrangement used. Although the use of variable-length buffers can be more efficient in terms of memory utilization, it is generally less efficient in terms of processor overhead than the use of fixed-length segments.

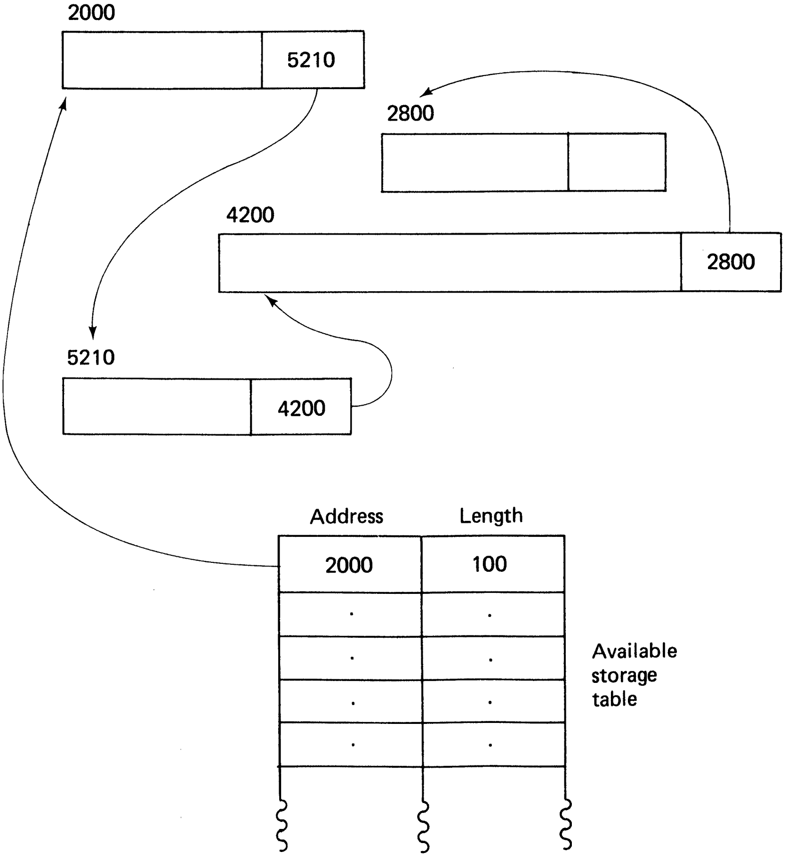


Figure 6.17
Dynamic storage management: chained variable-length buffers

Choosing the length of the segments at the software design stage is clearly crucial. If they are too small it will not be possible to satisfy all requests for storage; if they are too large, the benefits to be gained from the dynamic facility will be reduced. A compromise can be made by enabling the requesting program to indicate the number of segments needed and supplying these in either a contiguous or chained fashion. But as before, such extensions do increase the processor overhead associated with managing the pool.

Design of Teleprocessing Programs

A Checklist of Considerations

—How powerful should the multitasking facilities be? Single thread, multithreading? Reentrant code?

—Should the system be written in macro or a high-level language?

—Should all tasks be core resident? Can some overlaying or swapping be used? Are virtual storage techniques relevant? What about a virtual machine environment?

—What demands will be made on main storage? Will static buffering be possible/economical? Can disc-based queues be used? Will dynamic storage management techniques be required? Will the executive provide such facilities? Or will they need to be user written?

—Should a general-purpose control program be used or would a special-purpose package be appropriate? In other words, what is more important, ease of implementation/maintainability or performance?

7

Minicomputer Operating Systems: Some Examples

Having established an overall picture of minicomputer hardware and software against a background of the requirements of a terminal-oriented system, we will now look at some specific products. We are not trying to do a consumer report; our examples have been chosen to give an idea of the range of products available. Thus, while it will become obvious from the following that DEC's RSX 11-M is more powerful than Data General's RDOS, Data General does have a more powerful OS (AOS) and DEC offers the relatively trivial RT-11. It is very much a case of paying your money and taking your choice.

We have broken our review of operating systems into two chapters. In this one we review five operating systems from the viewpoint of their total capacity. In the next chapter we review other OSs more specifically tied to high-level languages and easy user interface. The former are more general purpose and directly suited to support of specifically tailored applications such as front-end processors or concentrators. The latter are more special purpose commercial data processing systems. Examples are given, however, of packaged data processing systems that use standard operating systems. Also included in this chapter is a look at more specialized networking software, represented here by DECNET.

A Review of General Features

Program Management

Once a program has been compiled and the units correctly linked, the executable code is stored on a disc file. Any program which the system must run can then be *installed,* the OS maintaining a table of all installed programs with appropriate attributes. An installed program can be loaded on command, since all housekeeping chores have already been performed. Most installed programs are physically stored on disc, but there are facilities for keeping special programs that may need rapid activation memory-resident, such as emergency shut-down programs in a process control system.

An installed program can be in one of two states, *dormant* or *active*. A dormant program is placed in the active state by a run or execute command, at which point control is passed from the program manager to the task manager. Note that a run command does not give a program CPU control, it adds it to the list of active tasks under direct control of the task manager. A program cannot be run unless it has previously been installed. Program control commands can be issued in two ways: as an operator command from a terminal or batch job control stream, or as a supervisor call from another active task. The operator command facilities can be activated by a supervisor call so that an executing program can request further program control from the operator's console. The operator command routines may be accessed only by the system console in a single-user system. A multiuser system allows other terminals to be declared as either slave or console-mode, when commands can be entered concurrently from multiple terminals. The command routines may be shared or further copies may be attached to console-mode terminals.

Task Management

The allocation of CPU time among activated tasks is governed by a set of system programs called the task manager. These functions are:

Call processing—a means of interpreting a specific request for supervisor services and initiating the appropriate action

Interrupt handling—routines to service interrupt requests, usually occurring in response to completion of events initiated by supervisor calls; they are typically integral parts of the device handlers

Clock control—setting of time intervals for time of day or user-requested delays and the like

Task monitoring—the current status of any task must be dynamically maintained. The task monitor is activated by the call processor and interrupt handlers

Task scheduler—determines which task is to be given control of the CPU. Task scheduling is fundamentally ordered by one of three functions (or a combination)—priority, time scheduling, and from "events" (as in current task status)

Task Scheduling

Once active a task can be in one of four states:

Executing

Ready—waiting to execute but preempted by a higher-priority task

Suspended—waiting the occurrence of some other event, such as a clock interval, completion of an I/O call, or disc transfer

Interrupted—transiently suspended while a supervisor interrupt service routine executes

In addition, the code for a task in a suspended state (and a low-priority ready task, for that matter) may be swapped out to disc. Certain suspended tasks, however, may not be swapped; this mostly relates to situations where data is being moved into memory areas in the task body or awaiting completion of a disc transfer. Supervisor calls are also available for one task to suspend and reactivate another, including timed actions.

Queue Management

Calls for supervisor services that cannot be honored immediately must be queued. The issuing task may need to be informed of a delayed state and asked to take alternative action if queues are full. In some cases queuing of data messages is also maintained; see section below.

Storage Management

A certain amount of memory (the resident executive, vector addresses and so on), has a fixed allocation. Allocation of the rest is under control of the OS. With an unmapped system physical control is largely related to supervising the relocatable loader, while with a mapped system the OS must maintain the mapping registers. The OS is also responsible for error handling. The rest of memory is allocated between dynamic requirements of the operating system itself and the user program. The physical memory may be divided into independent partitions with either fixed or adjustable limits. Active programs may be constrained to memory or may be swappable (checkpointed).

A particular feature of storage management specific to on-line systems is I/O buffer allocation. In the simplest systems each device handler has a fixed size buffer per terminal; data is then passed to the user program. Certain systems allow some data transfers direct to the user space. Such modules as relative-record file handling require a buffer to hold a complete disc block of data so that multiple records in the user space can possibly come from the same buffer. Similarly write-a-record implies reading a complete block, modifying the block in its buffer, and rewriting.

The support of dynamic I/O buffers, allocated from a pool merely by updating pointers, can greatly reduce the total memory requirements for buffers. This is less applicable to disc transfers than terminals, since buffers are allocated per open file (not physical disc) and records can be updated in buffers rather than reading disc for each read or write.

Device-Independent I/O

The OS maintains a *peripheral device table* (the extended line control table mentioned earlier) for each physical terminal. It defines the status, points to the device service routine, and provides transient data work areas. At the same time OS maintains a set of user *logical device tables* so that the user task refers to I/O devices as logical unit or channel numbers that are mapped to the requested terminals. A different physical terminal can thus be allocated to a program by modifying the logical device tables without modifying the user's code.

If the interface between logical and physical devices is standard, device differences being catered for in the device handler, then device independence is further enhanced. Occasionally the device handler is sophisticated enough to perform protocol functions to a level that, say, a 2780 type terminal or a Teletype-compatible VDU can be directly driven by the same program. Such a complete handler is sometimes called a *symbiont*. More often such a degree of device independence is not supported.

A further variant is to allow the file handling routines access to certain device handlers. Thus print data can be transferred to a logical sequential file channel that can be allocated when the file is opened to a printer or to a disc file for spooling.

The Operating Systems Reviewed

The salient characteristics of the following operating systems will be considered in the following sections:

Data General	RDOS
DEC	RSX-11M
MODCOMP	MAXCOM
Texas Instruments	DX10
Digital Research	CP/M (for 8080/Z80 microcomputers)

In no way dare we imply that these are the best systems. They are merely representative of the wide variety of systems available. Nor can the following notes be taken as a specification for an operating system. First, because it is impossible for us to describe all the features, and second, because the manufacturers continuously upgrade their systems. For our purposes this is irrelevant because we are trying to stress the scope of current minicomputer software and to point to future improvements.

Why have these particular operating systems been singled out? RDOS is a longstanding product typical of the process control type of OS found on earlier minis and still used on the simpler processors. It is a single-user, foreground/ background system with limited file and mapping support. RSX-11M is a multiuser, partitioned system with good mapping and reasonable file support, aimed for medium-sized minis. MAXCOM is a very specialized OS devised for run-time support of dedicated communications systems developed under another MODCOMP operating system. DX10 is similar to RSX-11M, except that it doesn't use partitioning and has more advanced support for sectioned programs, including shared code: it also includes a far more versatile file handling system and multiuser program development. CP/M is the de facto standard used on a multitude of microcomputers; it is a simple single-user batch system, primarily written for discettes.

Data General RDOS

RDOS is a single-user system that can support two partitions, a foreground for an on-line program and a background, usually for a batch program. On simpler NOVA processors the partitions are software controlled as shown in figure 7.1. With mapped NOVAs and ECLIPSEs the system is enhanced as shown in figure

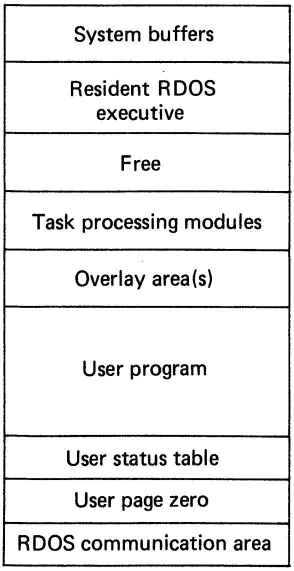


Figure 7.1
A simple unmapped single-ground RDOS system

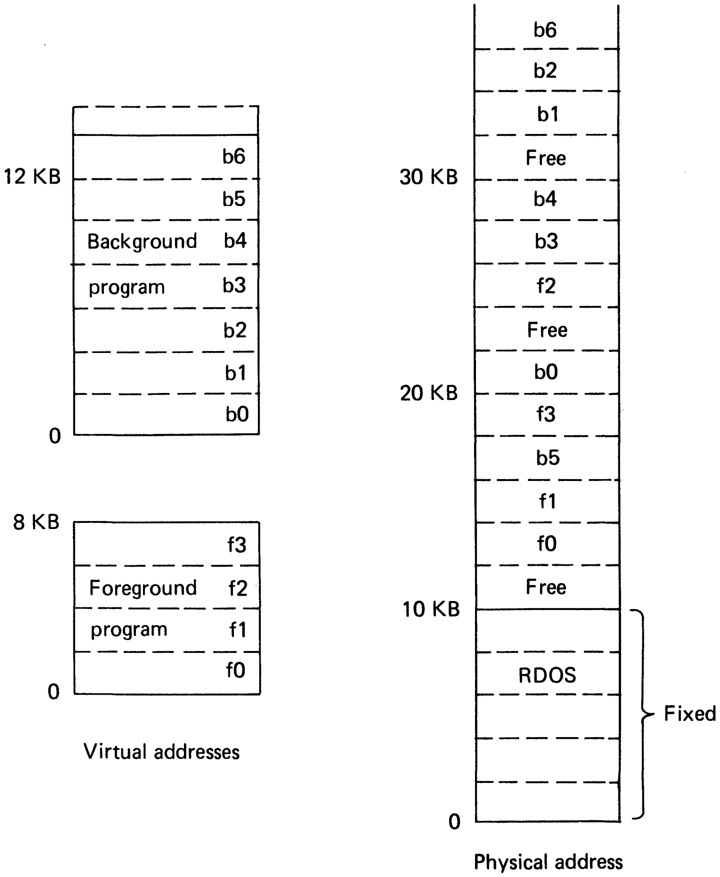


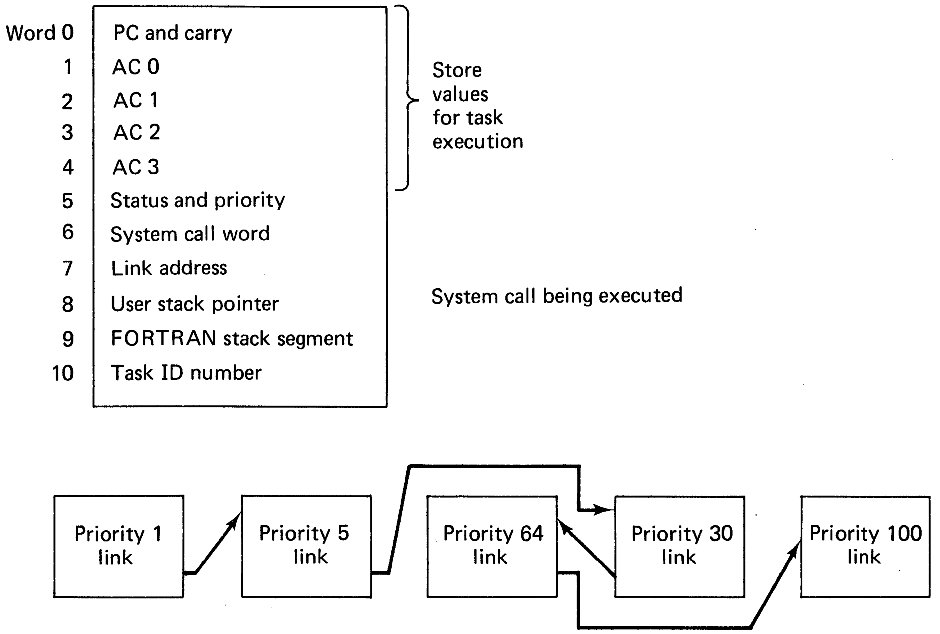
Figure 7.2
A mapped RDOS system

7.2; the operating system and both the foreground and background partitions can be up to 62 KB, with a 2-KB global space. In supervisor mode the processor executes OS functions in physical address space 0 to 62 KB, while in user mode the two separate programs can be mapped into any available noncontiguous memory.

An actual partition is dynamically maintained by the hardware mapping so that code in excess of 64 KB can be stored in physical memory and "swapped" by updating the mapping registers. This is referred to as *window mapping* and *virtual overlaying,* using a hyperspace—who could ask for more? In practice the user program is manipulating the mapping registers rather than the OS, which needs great care in its use. The OS effectively maintains two task schedulers, one for each partition. The task scheduler gives CPU control tó the higher-priority task ready to execute, which retains control until it executes a system call. The background job is at the lowest priority and executes only when no foreground job is in the ready state; the background job is interrupted by any foreground task becoming ready. There are 256 priority levels generally assigned when the task is installed but alterable by other tasks issuing system calls.

Each active task is allocated a task control block as shown in figure 7.3, taken from a pool, that is released when the task completes. TCBs for all active tasks are linked in order of priority. The task scheduler scans the linked TCBs, testing the status word until it finds the next job to run. Changes of priority or completion merely require update of the link pointers. Programs can be overlaid and chained to other disc-resident programs, but there is no system-controlled swapping. Thus there must be sufficient memory to hold all active tasks. The OS itself swaps some less frequently used modules, however. A foreground task can checkpoint a background task by issuing a supervisor call.

Supervisor calls occur at two levels. The normal calls to shared operating system services are implemented by a subroutine jump via a specific memory location in executive space that activates the system call processor after storing the calling task environment in the TCB. Before executing the call, parameters are set up in the accumulators to establish pointers to data areas. The second set of



Word 0	PC and carry	Store values for task execution
1	AC 0	
2	AC 1	
3	AC 2	
4	AC 3	
5	Status and priority	
6	System call word	
7	Link address	
8	User stack pointer	System call being executed
9	FORTRAN stack segment	
10	Task ID number	

Figure 7.3
Task control block

supervisor calls are used to allow one task to control both its own and other task executions. Unlike a system call, the task control calls use code included in the user program from a library (the task processing modules in figure 7.1). Thus while system calls will invoke the task manager, task control calls are direct commands to it. One task control call provides direct communication to the console operator, who can issue any available control function command.

System command calls are:

Clock/calendar
User-defined interrupt servicing
Disc directory maintenance
File maintenance
File I/O
Console I/O
Overlay, chain, and memory control
System and spooling
Foreground/background communication

Task control calls are:

Task initiation
Task execution control
Intertask communication and synchronization
Task control by identification number
User clock and interrupt task control

The operator communicates with RDOS from a console terminal that runs a system program called the command line interpreter (CLI), usually as the background program. Whenever the system is idle RDOS restores CLI to memory. The CLI commands initiate the system utilities (the editor, compiler, BASIC interpreter, and so on) to support program development. They also support similar file and directory functions as accessed from the user program by supervisor calls. Further functions are required to set time and date, boot the system and the like.

A simple batch facility exists whereby a string of CLI commands can be issued from a sequential file rather than directly from the terminal keyboard.

Each terminal is allocated a fixed-length buffer in the OS space, its length dependent upon the speed and type of device. A device control table with reentrant drivers is maintained for each device. RDOS also maintains a set of 512-byte system buffers for intermediate storage of blocks of data in file transfers (except direct block transfer, which takes place to a buffer in user space) to allow records shorter than a full block. The pool size is specified to support the maximum number of files open at any one time.

RDOS supports a simple spooling of output messages to a printer via a disc file to enable messages to be output faster than on the printer. There is no automatic multiuser print spool utility, however. The RDOS disc system is rather simple. A disc is subdivided into partitions, each with a directory, which serve as user file directories. Independent files within the fixed bound of the partition can vary in length. Sequential files are allocated disc blocks on a conventional linked scheme. Random access files can be allocated contiguous (nonexpandable) blocks or via a directory, with one entry per block allocated; there is no facility for multiple block clustering.

The user file access routines support supervisor calls, which can be queued, for the following:

ASCII single character
ASCII line, terminated by CR, LF, or NUL
Sequential (length determined by byte count)
Relative block—512-byte blocks
Relative record—128-byte record only
Free format I/O for magnetic tape records from 4 bytes to 8 KB

Relative-block transfers are direct to the user space; all others are via system-controlled buffers. Once a file has been opened by a program, multiple tasks within the program can access the same file. There is no record level lock-out supported by the system.

On the ECLIPSE range file handing is significantly enhanced by an executive module known as INFOS. INFOS is accessed by system calls from a user program and this in turn makes calls to the conventional file system. INFOS is a multikey, multilevel ISAM package with an excellent specification and a good range of support utilities. It is readily accessible from high-level programs, but it is not reentrant and therefore has limitations in multiterminal on-line applications.

In the standard RDOS system all terminal I/O is handled by the file handling system, a physical device being tied by executing an open supervisor call. While it usually executes batch jobs in the lower-priority background, a terminal can be used by a program in that partition.

Communications Access Manager

The simple and direct terminal handling inherent in RDOS can be enhanced by a separate communications control package called CAM. CAM is not part of the OS but is linked with a user program and run as either a foreground or background program. A simple interactive program allows CAM to be generated to suit a given hardware configuration and to define the quantity and size of buffers and so on. CAM will support Data General's programmable front-end processor, the DC-50, the program for which is downloaded when CAM initializes. User interfaces to CAM are device independent; thus, if a 16-channel multiplexer is replaced by a DC-50, CAM is regenerated and linked to the user program, which need not itself be modified.

Both synchronous and asynchronous lines are supported. Asynchronous data can be transmitted as a direct sequential file with byte count control or in a line mode, controlled by terminating characters such as CR, LF, and others. In line mode received characters are echoed, and rub-out and / characters delete the previous character or whole line respectively. Control can be immediately returned to the user task either on acceptance or on completion. If an immediate return is used, then I/O completion updates a queue which can be tested by a further CAM call. Following an immediate return a second output command can be queued to a correctly executing command on the same line; a third call returns line busy. A time-out can be used to create an error return condition. In the sequential transmission mode, EBCDIC to ASCII translation can be provided by CAM as well as a choice of cyclic redundancy check (CRC) character.

Synchronous transmission does not allow immediate return to the calling task, so one task per line is required. There is full support under CAM for the bisynch protocol, including multidrop terminals. All polling, handshaking, and similar operations are transparent to the user program. The number of retries is specified to CAM at generation. CAM maintains pools of stacks and buffers as well as control tables and queues. One stack is allocated to each active CAM call. Buffers of fixed size set at CAM generation are allocated on demand, one for HDX and two for FDX to each currently active line. Except on the immediate return mode, the call is completed by transferring the data from the CAM buffer to the user program. Calls transferring data from the user program to partially full CAM buffers can accumulate the new data. CAM calls are implemented by a JSR to an indirect page-0 address. A complementary set of calls are supported in FORTRAN.

High-Level Languages and Other Modules

All Data General's compilers are nonreentrant, in keeping with the single-user philosophy of RDOS. One of the FORTRAN compilers can, however, generate reentrant code. A multiterminal editor can be run as one job, allowing multiple users to create source programs simultaneously. Languages supported are:

Assembler and macro assembler
FORTRAN IV with real-time extensions and commercial subroutine package
FORTRAN 5 (optimizing)
ALGOL
BASIC
Extended BASIC (multiuser)
Business BASIC

and for the ECLIPSE only

COBOL
RPG II
IDEA data entry package

A COBOL compiler is available for the NOVA as part of a packaged system called CS40, which is in fact based on RDOS. While BASIC is one job to RDOS it includes its own executive to give multiuser access. Packages are available for 2780/3780 and HASP emulation. The HASP work station is available only on the ECLIPSE. Both run as a program under RDOS. A memory-resident subset of RDOS, called RTOS is available. The 2780/3780 emulators will run under RTOS for a nondisc-based RJE station. CAM is used as an integral part of the emulator.

General Comment on RDOS

RDOS highlights the dilemma facing the older minicomputer suppliers. The large number of users means that the OS is well tried and tested and therefore reliable. User demand has led to the development of a wide selection of utilities, ranging from dedicated programs like the 2780 emulator to the file management system, INFOS. However, use of the superior instruction set of the ECLIPSE has led to the development of utilities like the COBOL compiler, which will not run on the NOVA. The overall impression left by RDOS is that its minimal storage management is strangling the versatile use of the powerful utilities available. Hence Data General has introduced the multiprogramming AOS system, currently only for the ECLIPSE, since it is appreciably larger (of course) than RDOS. The dilemma continues, however, because it will take time for the new system to support the range of utilities available to RDOS and to build up an equally stable user base. The simple partitioned disc management system of RDOS leaves a lot to be desired, particularly in commercial applications, a fact recognized by Data General and overcome in the more sophisticated AOS.

Digital Equipment Corporation RSX-11M

RSX-11M is a multiuser, multiprogramming system for PDP-11 minicomputers. The physical main storage space can be divided into mutually exclusive partitions will full multitasking control within each. In DEC's terminology a task is a complete user program; multiprogramming implies multipartitions. User tasks are installed into specific partitions and must be linked by the task builder beforehand.

RSX-11M is a member of a family of operating systems, the others being RSX-11S and RSX-11D. While a simple RSX-11M system can be used on an unmapped PDP-11 (except the 11/03) it will be considered here with hardware mapping support, as in the 11/34, 11/60, 11/70 and other models.

There are two types of partition, user- and system-controlled. User partitions will accept only one program and are intended for dedicated memory-resident tasks. System-controlled partitions will hold as many user programs as there is space for, with the proviso that they are contiguous within the partition. A supervisor function allows shuffling to regroup noncontiguous blocks. Tasks in the system-controlled partitions may be swapped by the OS but overlaying only is allowed in the highly controlled user partitions. Global areas are defined for communication between partitions and code sharing.

The OS operates in a hardware supervisor mode that is mapped to include the lower physical memory locations (vectors and so forth) and the top physical 8-KB page, which contains the peripheral registers and the like. All user programs can be a full 64 KB, but must then use supervisor calls for all I/O. SVCs are implemented as software interrupts (EMT) which vector to service routines and automatically switch to supervisor mode. A typical memory layout for an RSX-11M mapped system is shown in figure 7.4.

The partition sizes and physical location are fixed at system generation. User-controlled partitions, since they contain only one program at a time, are usually less than 64 KB; system-controlled partitions can be any size, up to the available limit. Thus a system configured with just the one system-controlled partition is a conventional multiuser, multitasking system with swapping. Alternatively and unnecessarily restrictive, a system configured with just two user-controlled partitions is a foreground/background system similar to RDOS or DEC's own RT-11F/B. With multiple system-controlled partitions, specific tasks will be installed in particular partitions; thus while all are executed under the umbrella of the task manager, some hierarchical management of the workload can be maintained.

In principle any one program is constrained to a maximum of 64 KB by the 16-bit word length. However, memory management directives available to the user program manipulate the mapping hardware to reallocate a different physical page to the same virtual address space. This technique is referred to as window mapping and effectively allows single programs to exceed 64 KB. This feature, along with other attributes such as overlays, checkpointability, ownership, partition control and priority, is specified when modules of code are linked by the task builder. Libraries of routines can be maintained, some comprising reentrant code linked with the task code. Alternatively, appropriate routines are loaded in a shared global area, the physical address of which is mapped to the virtual space of the user-specific code by the OS, using the references supplied when the task was built.

There is relatively little inherent support for shared code at present. If two users ask to use a compiler, two copies are loaded. Similarly, some DEC processors include dual hardware mapping for I (instruction) or D (data) space which is not employed.

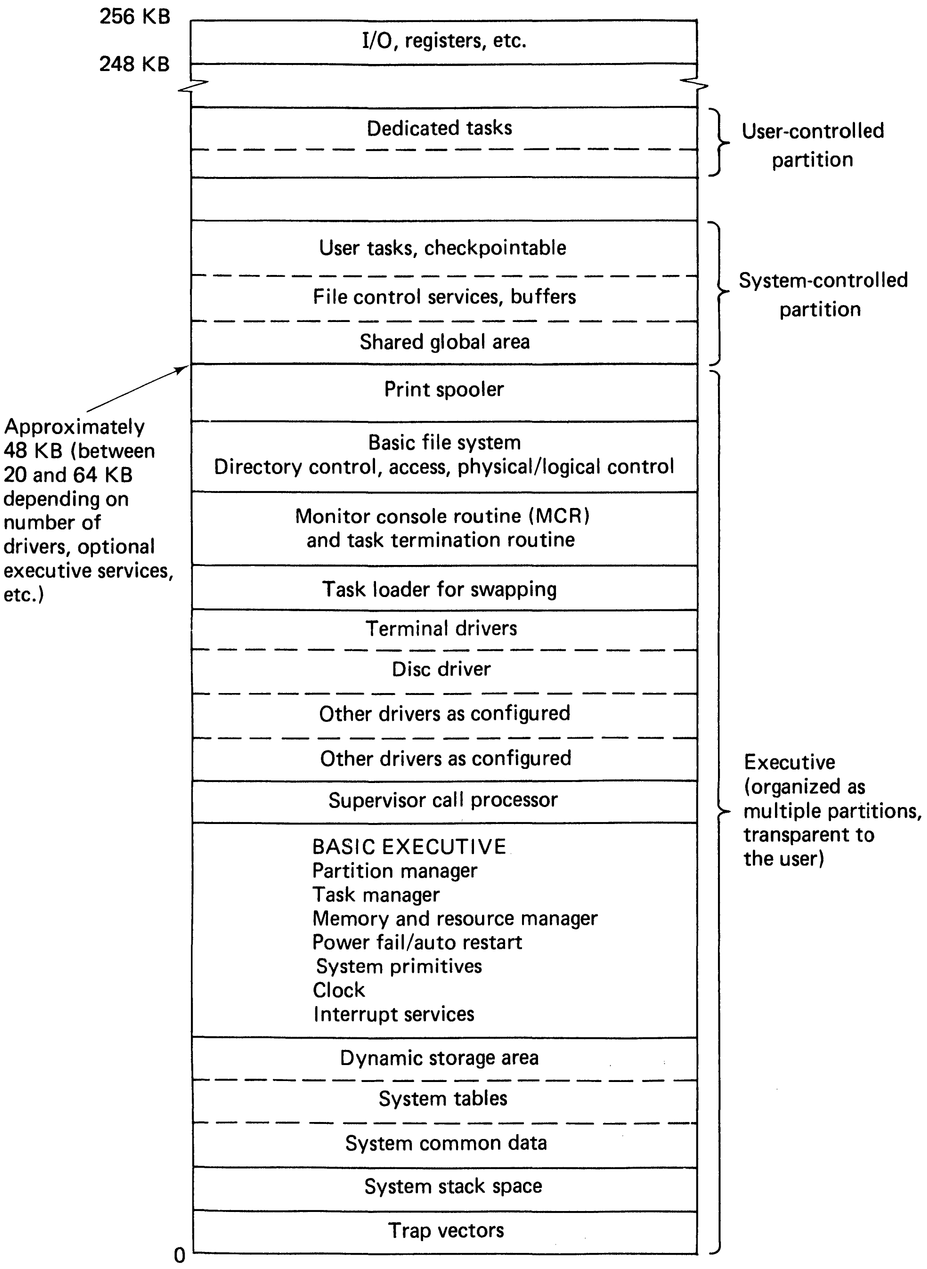

256 KB
248 KB
I/O, registers, etc.
Dedicated tasks
User-controlled partition
User tasks, checkpointable
File control services, buffers
Shared global area
System-controlled partition
Approximately 48 KB (between 20 and 64 KB depending on number of drivers, optional executive services, etc.)
Print spooler
Basic file system
Directory control, access, physical/logical control
Monitor console routine (MCR) and task termination routine
Task loader for swapping
Terminal drivers
Disc driver
Other drivers as configured
Other drivers as configured
Supervisor call processor
BASIC EXECUTIVE
Partition manager
Task manager
Memory and resource manager
Power fail/auto restart
System primitives
Clock
Interrupt services
Dynamic storage area
System tables
System common data
System stack space
Trap vectors
Executive (organized as multiple partitions, transparent to the user)
0

Figure 7.4
RSX-11M memory allocation

Allocation of memory to multiple programs within a system-controlled partition is dynamic, although each program must be contiguous. A shuffling routine allows released memory to be grouped as a contiguous space. Relocation of programs is performed by setting up the mapping registers, tables of current allocations being maintained by the OS, but only for programs installed to run in a particular partition.

Let us now refer to figure 7.4 in more detail. System parameters for a specific generation are held in a common data area. The basic tables (data descriptors) used by the executive are also initialized at generation but are maintained by the system. A number of data descriptors are allocated on an as-required basis from a dynamic storage area, the size of which is set at generation time. Extensive use is made of linked blocks for task control, partition windows, and attachment descriptors (peripheral control).

Areas for intermediate buffering of terminal I/O can also be taken from the dynamic storage area. Examples of control blocks are shown in figure 7.5, although far greater knowledge of RSX-11M is required to appreciate the significance of the details. RSX-11M maintains a number of other tables, such as:

Physical unit directory—used to point logical requests to physical devices

Task partition directory—mapping data

Global common directory—enables virtual addresses in programs using shared code or data to map to the same physical locations

System task directory—a list of all installed tasks; an installed task can respond immediately to a run command without needing linking

Active task list—priority-ordered list of all active tasks; linked TCBs dynamically maintained to give CPU control to the next task to be executed

Fixed task list—list of inactive but memory-resident tasks; linked to ATL when activated and back to FTL when task exits

Checkpointable task list—priority-ordered list of checkpointable tasks for each system-controlled partition

Memory required list—priority-ordered list of swapped-out tasks waiting for memory, one for each partition. The executive maintains a table of blocks of memory in use and released

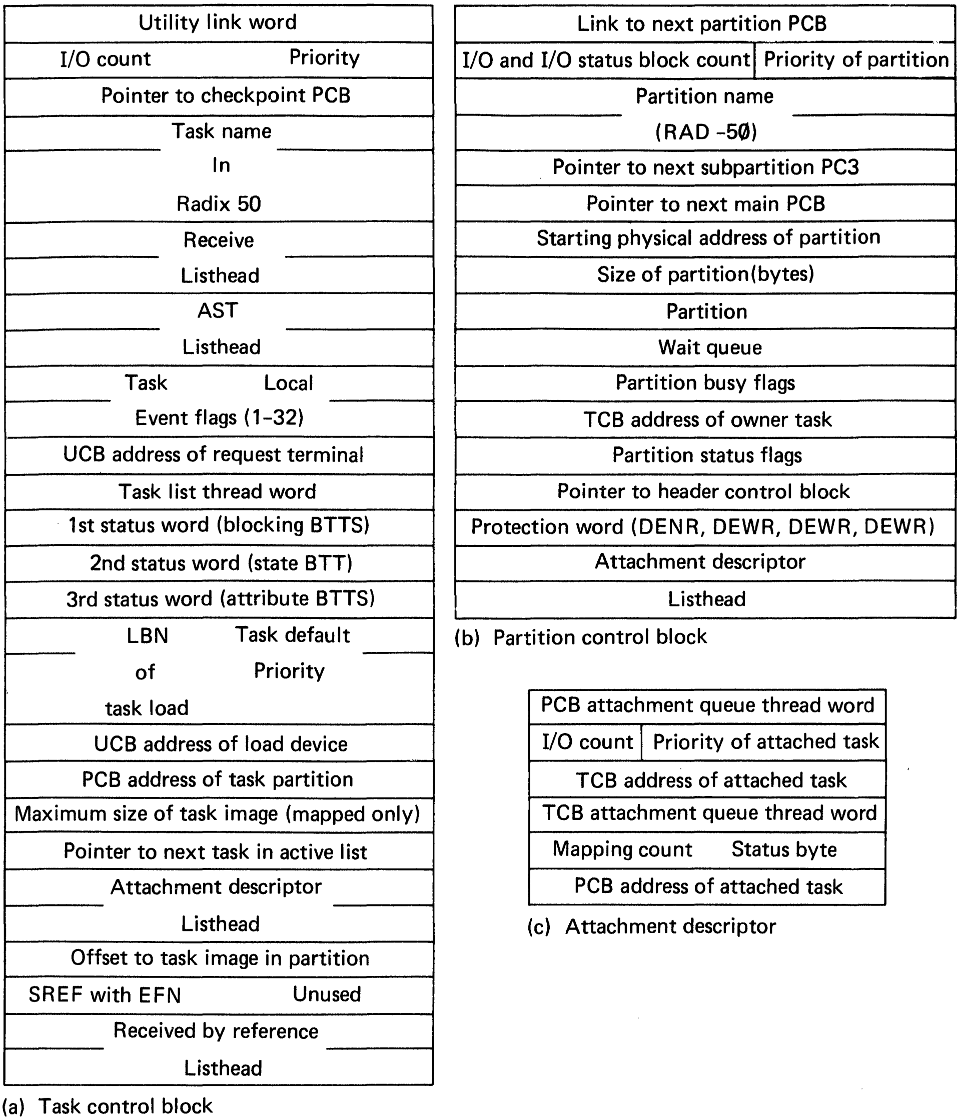
Asynchronous system trap queue—list of ASTs waiting to be serviced; identifies which task is waiting for return after completion of a supervisor call

Clock queue—stores data defining events to be initiated at some specified time; the scheduler periodically runs a program to service this queue

MCR command buffer—temporarily stores commands to be serviced by monitor console routine requests

Batch command buffer—stores batch commands to be serviced

RSX-11M is essentially an event-driven system. The highest-priority task capable of running (independent of partition) is given control of the CPU, which it



Utility link word

I/O count | Priority

Pointer to checkpoint PCB

Task name
In
Radix 50

Receive
Listhead

AST
Listhead

Task Local
Event flags (1-32)

UCB address of request terminal

Task list thread word

1st status word (blocking BTTS)

2nd status word (state BTT)

3rd status word (attribute BTTS)

LBN of task load | Task default Priority

UCB address of load device

PCB address of task partition

Maximum size of task image (mapped only)

Pointer to next task in active list

Attachment descriptor
Listhead

Offset to task image in partition

SREF with EFN | Unused

Received by reference
Listhead

(a) Task control block

Link to next partition PCB

I/O and I/O status block count | Priority of partition

Partition name
(RAD -5Ø)

Pointer to next subpartition PC3

Pointer to next main PCB

Starting physical address of partition

Size of partition(bytes)

Partition
Wait queue

Partition busy flags

TCB address of owner task

Partition status flags

Pointer to header control block

Protection word (DENR, DEWR, DEWR, DEWR)

Attachment descriptor
Listhead

(b) Partition control block

PCB attachment queue thread word

I/O count | Priority of attached task

TCB address of attached task

TCB attachment queue thread word

Mapping count | Status byte

PCB address of attached task

(c) Attachment descriptor

Figure 7.5
Typical RSX-11M control blocks. *(a)* Task control block *(b)* Partition control block *(c)* Attachment descriptor

maintains until it executes a system directive (SVC) or an external event creates an interrupt, the servicing of which modifies the priority of the active task. Such an occurrence is termed a *significant event.* A set of event flags, 32 common to all tasks and a further 32 per task, are affected by significant events. These can be tested by the executive or the user task, either individually or in logical combinations, thus providing extremely versatile control of task scheduling.

Task execution can also be controlled by system traps, which are effectively software interrupts local to the task rather than system-wide. A synchronous system trap (SST) occurs at fixed points in the program (for example, illegal instruction execution), while asynchronous system traps (AST) occur when some external event, usually a significant event, is declared. The task must provide service routines for system traps. The executive maintains a first-in/first-out list of ASTs pending execution.

While the task manager is essentially event driven, a facility is available for time-slicing control of checkpointable tasks. In effect, the time slicer manipulates the priority of tasks on the memory required list. The algorithm is effective but not to be compared with the sophistication of dedicated time-sharing systems such as DECs RSTS/E. Tasks on the RSX-11M are allocated default priorities (not to be confused with hardware interrupt priority levels) in the range 1 (low) to 250 (high) when the task is built. Priorities for an installed task can be changed from a system console.

Since RSX-11M is a multiuser system, multiple terminals can attach to the monitor console routines (MCR). These provide the usual range of interface facilities to the executive services, including a password log-on system. Any terminal not in use by an executing task can gain access to MCR by typing the appropriate control character; such an unsolicited AST is routed to the MCR. MCR includes commands for:

Initialization—log on/off, password, install task, time, mount, dismount, set parameters

Information—print active task list, system characteristics and statistics, terminal usage, log comments

Task control—allocate logical references to physical devices, check and alter priority, run programs, abort, cancel, remote, and continue; QUEUE passes parameters and files to a print spooler

System maintenance—break to an executive debugging tool, examine words of memory, save memory images.

A sequence of MCR commands can be saved in a file. Execution of these commands constitutes a batch stream. An extension allows command sequences to be given to specific tasks such as the assembler. Extensions to the direct MCR commands are allowed in the file to logically control the stream.

The user communicates with the executive through a comprehensive set of system directives (SVCs). Excellent macros are provided to define the associated machine code. The system directives include:

Task execution control—run (with time control), suspend, abort, cancel, exit

Task status control—enable and disable checkpointing, fix a task in memory

Informational—get task, partition, logical unit, common, time- and console-switch parameters

Event-associated—declare significant event, control, test, and act on event flags

Trap-associated—set and cancel time delays, control AST and SST vectors

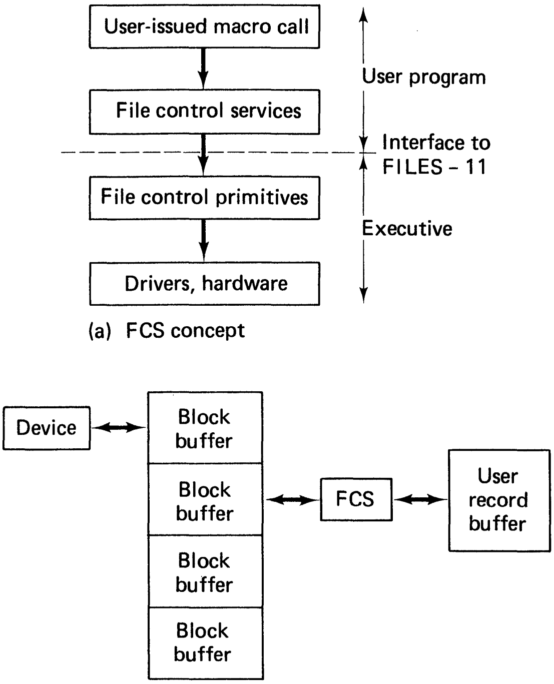
I/O-related—queue I/O with various modes of return to issuing task, intertask data send and receive, get MCR commands

I/O is performed by the queue I/O directive (higher-level file control will be discussed below). Queueing of all I/O requests is a powerful feature of RSX-11M; they are taken off the list and executed in a priority sequence. Drivers for common peripherals are reentrant and are, of course, interrupt driven. Buffers are normally allocated in the user space, accessed by special instructions that allow moves from a location in one mapped space to another. Intermediate buffers can, however, be taken from dynamic memory space, when the buffer transfer to the issuing task is performed by window-mapping the user's physical space temporarily to the executive space and executing normal move instructions. DMA transfers are to direct 18-bit addresses in the user space, initiated by the current map setting but autonomous from then on. (RSX-11M loads all programs into contiguous locations, thus avoiding problems of noncontiguous pages for the DMA transfer; in contrast Data General's RDOS uses noncontiguous pages but maintains a completely separate DMA hardware map.)

Multiple priority-ordered I/O request queues are maintained, one for each logical unit. Control can be returned to the issuing task immediately after the directive is queued or the task can be suspended to await rescheduling on completion of the request. An event flag, an AST, or an in-task status block may be optionally specified as an I/O completion indication. If intermediate buffering is used, a descheduled task can be swapped to disc while it awaits I/O completion.

File handling utilizes the basic executive services but extends these with a set of file control primitives, which manage the file directory data required for the more complex logical to physical mapping of a disc system. These primitives are accessed by a set of routines called the file control services (FCS). FCS services are invoked by macro calls in the user program. The calls to FCS and FCS itself are part of the user program linked at task-build time; FCS issues supervisor-level calls and provides buffer space. The overall system is shown in figure 7.6.

Before discussing FCS, the user interface level, it must be pointed out that the physical file structure of RSX-11M, called Files-11, is very good. It is a hierarchical volume/MFD/UFD system. A volume is a collection of files which, together with MFD and UFD attributes, provides a high degree of security. Physical block allocation is completely dynamic, single files building up with noncontiguous groups of blocks. The FCS routines are reentrant and could possibly be included in a shared library. It is common, however, to include one



(a) FCS concept

(b) Data flow

Figure 7.6
File control system in RSX-11M

copy with each program. A pool of block buffers is maintained in each user program, dynamically allocated as a file is opened; thus only sufficient buffers are needed to meet the maximum number of files open at any instant. Tables required by FCS to maintain the logical channel relationship with the physical files are also maintained with the buffers.

FCS supports sequential, block, and record I/O. Sequential access is applicable to record-structured devices (printers, card readers, as well as disc and tape). Data is transferred to the buffers in blocks. For block I/O, the data is passed direct to the user buffers. For record I/O, data is stored in the block buffer pool and the appropriate subblock transferred to the user record by FCS; records are not physically read or written if they can be manipulated in the block buffer (reading two or three consecutive records may require only one disc block read). Sequential files may carry fixed- or variable-length records; direct access files support fixed-length records only. A user program has an option for processing records directly in the block buffer.

With record I/O, control is only returned to the user program after the operation is completed; with block I/O, a return immediately after the operation is initiated is possible by means of an event flag to coordinate interdependent operations. Block I/O is aimed at maximum efficiency, record I/O at ease of user interface. A special form of the open macro allows shared access to files.

FCS macros fall into four classes:

Initialization—establish a file data block of execution-time characteristics (record length, type, access privileges); a data set descriptor to define file name, type, location, plus a default information block. Some data is provided at assembly, other data at run-time

File processing—OPEN—normal, shared, and temporary; CLOSE; GET—read logical, fixed-length random and sequential records; PUT; READ and WRITE for block access; DELETE; WAIT—suspend until completion of block I/O; PRINT—queues a file for printing

Command line processing—allows access to special routines available in the system object library as though issuing requests from a terminal

Call—gives access to file control routines to interrogate update and add to file directories

RSX-11D, RSX-11S, and IAS

RSX-11M is a member of a family of operating systems. RSX-11S is a run-only subset of RSX-11M. Programs are developed on an RSX-11M system, linked, and the complete RSX-11S plus user-program system is transported. A peripheral device (discette or tape) must be included for program transportation. Even if a disc is included for program loading it is not used by RSX-11S; the special subset of FCS incorporated does not include support for file-structured devices. A front-end processor is an obvious application for RSX-11S.

RSX-11D was introduced as the top of the RSX family. It included a few extra directives and a single-stream batch processor. The major difference is that terminal handlers, drivers, and so on are supervised as conventional tasks in cooperation with user tasks, rather than as functions of the executive. This adds some run-time overheads but greatly enhances flexibility, since the drivers can be changed without system regeneration. As RSX-11M has been enhanced and relatively few systems actually need response to a changing environment, the benefits of RSX-11D have been less in demand. Digital Equipment has now released IAS, which in simple terms is RSX-11M running the sophisticated RSTS/E time-sharing executive as one RSX-11M task. This gives full batch, time-shared and real-time support on the one, albeit rather large, executive. (RSX-11D as such will probably be squeezed out.) A full data base system, DBMS-11, is also offered under IAS.

Communications Software

RSX-11M supports a 2780 emulator package and an IBM HASP work-station emulator. Only point-to-point communication is possible. There is no standard support for multidrop terminals. DEC, of course, presses the use of DECNET for computer-to-computer communications. DECNET is supported by RSX-11M and is described later in this chapter. DECNET is a promising product but is only applicable to interconnect DEC machines, not general networks of other suppliers' machines.

High-Level Lanugages

The situation with RSX-11M compilers was very disappointing both in variety and restrictiveness. One hesitates to say quality, because with the notable exception of the early COBOL, the compilers actually perform well what they are claimed to do. The languages available are:

—MACRO, an excellent macro assembler
—FORTRAN IV
—FORTRAN IV-PLUS, an optimizing compiler
—CORAL
—COBOL
—BASIC-PLUS II
—DIBOL

None of the compilers generates reentrant code, except by creating library modules for linking through shared global areas. CORAL, of course, allows a much more direct control of system directives and as such could possibly be used for communications program developments; otherwise MACRO must be used. The compilers themselves are not reentrant, although MCR is intelligent enough to control multiple copies simultaneously.

The COBOL compiler uses an interpreter at run-time and with a relatively small area available for user code employs a virtual file system. Pages of object code are rolled in as execution proceeds, with buffers and data divisions fixed in core. With this technique large programs can be run, but with high overhead because of disc accessing. (This technique has been extended to multiprogramming using the simple SyBOL interactive language by CAI, as described in the next chapter.) Although sound this compiler has a bad reputation for slow compilation and inefficient execution and has now been replaced with an improved ANSI COBOL.

General Comments on RSX-11M

Digital Equipment clearly intends RSX-11M as their prime OS. It is highly versatile, with excellent I/O and a good basic file system. It is rather surprising to find programs in a mapped system constrained to contiguous physical memory locations; this involves a program shuffling overhead that could be high in an installation with a lot of swapping and large partitions.

The partition concept is intriguing. It really seems like a hangover from the early days of multiprogramming on minis, where a real-time process control program and a program development system were run concurrently and there was great concern about possible corruption of the real-time program. Hardware memory mapping and protection schemes with well debugged system software have somewhat removed the constraint, and one is left wondering whether a system like Texas Instruments' DX10 or Prime's PRIMOS IV isn't more user oriented. While RSX-11M provides many facilities for development of special-purpose programs such as communication systems, it does not have a simple user interface (limited high-level language support for writing, say, a multiterminal stock enquiry system). It is still a specialist programmer's system, in which environment it has few rivals.

DEC intends to upgrade support for RSX-11M, particularly to meet the above criticism of poor user/programmer interface. The powerful multikey, multilevel, ISAM system RMS-11 will be supported, accessed by high-level calls as well as macros.

MODCOMP MAXCOM Communications Executive

Modular Computer Systems produces disc-based real-time OSs with program development facilities. A networking system called MAXNET is also available. As previously discussed MODCOMP makes a very sophisticated multiplexing I/O hardware subsystem (DMP) which is ideally suited to a multiterminal on-line system. Here we will discuss MAXCOM and not the full OS (MAX II), which is a special-purpose communications executive. MAXCOM is the essential parts of a CCP, but unlike IBM's CICS, is self-supporting. It is similar in a way to RDOS with CAM but with limited file support. RSX-11S also is similar, but several integrated facilities of MAXCOM would need to be written as application programs for RSX-11S.

Programs to run under MAXCOM are developed on another MODCOMP system. A run-time memory image is then created by linking the user program code to the required MAXCOM modules, extracted from a disc file. The memory image is then dumped to some medium for transportation to the run-time hardware system, which supports a simple loader to boot the new system up. Run-time access to disc is supported, but at a very low level. Figure 7.7 shows a memory layout of a system running under MAXCOM.

MAXCOM is an event-driven real-time operating system controlled by a task manager called Taskmaster. This is a no-frills system; nonessential functions such as sign-on/off, failsafety, and accounting must be implemented by the user (as indeed is the case with most minicomputer operating systems). Taskmaster uses a set of conventional TCBs to manage tasks, linked on a priority basis. A typical TCB is shown in figure 7.8. Unlike RDOS, which uses the TCB, or RSX-11M, which uses stacks, a save area is provided for context switching at the beginning of

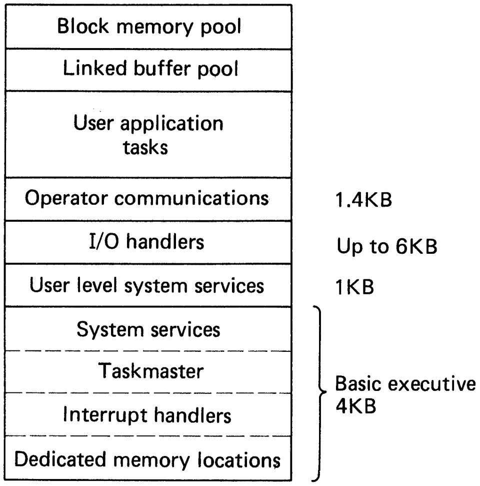
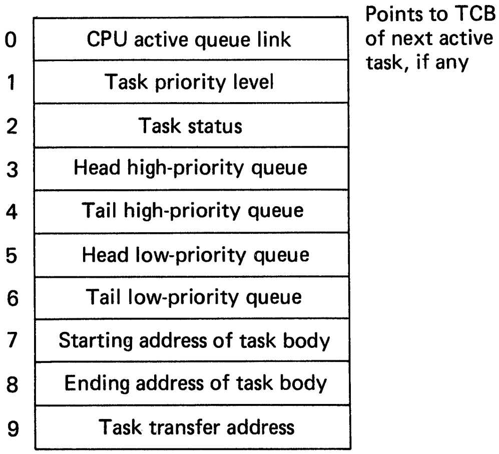


Figure 7.7
The MAXCOM system

each task, as shown in figure 7.9. Taskmaster's scheduler runs the highest-priority task in the queue. When an interrupt occurs it is immediately serviced; return to a task issuing a request for service is made by the interrupt routine updating the task status word in the TCB so that it can be scheduled again.

MAXCOM uses an unusual method of activating tasks. All work for a specific task is placed on a work queue for that task by the issuing task or service routine. Thus parameters or messages are attached to the work queue of a task rather than



0	CPU active queue link	Points to TCB of next active task, if any
1	Task priority level	
2	Task status	
3	Head high-priority queue	
4	Tail high-priority queue	
5	Head low-priority queue	
6	Tail low-priority queue	
7	Starting address of task body	
8	Ending address of task body	
9	Task transfer address	

Figure 7.8
MAXCOM task control block

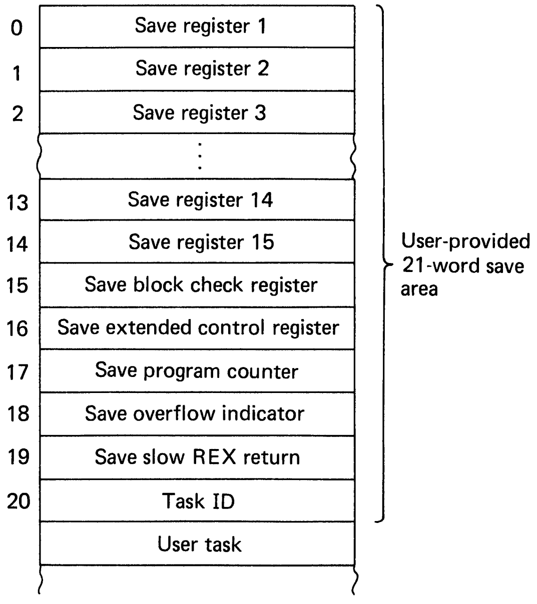


Figure 7.9
MAXCOM task body

in-line parameters. Each task maintains two queues, a high-priority and a low-priority, each serviced sequentially. Thus multithreading is achieved by causing events to queue onto the user tasks needed to process them, rather than conventional reentrant coding. The actual work queues are allocated from a block memory pool when requested by an issuing task and can be of any length. Unused space from the memory pool should be freed by the user task. Again a linked scheme is used, as shown in figure 7.10. Thus the queuing vehicle is serially reuseable; a packet placed on a queue cannot be reused until it has been acted upon.

A linked fixed-size buffer pool scheme is also maintained that is compatible with the MODCOMP DMP, so that multibuffer I/O is provided for with minimum software intervention (see chapter 5). MAXCOM employs the following hardware interrupt levels, in order of descending priority:

—Power fail/auto restart

—Unimplemented instruction trap, used for supervisor calls

—Real-time clock

—I/O data terminals can have selective priorities at this level

—I/O service; when an I/O operation completes, the device handler dequeues the associated I/O parameter list and links it to the task scheduler for further processing

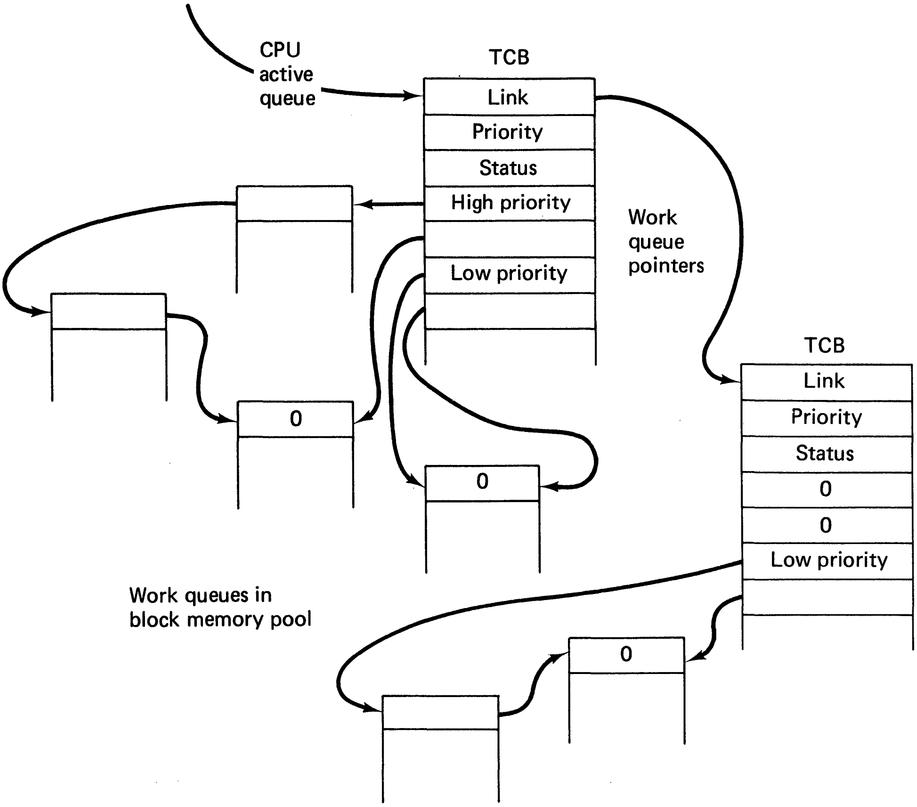


Figure 7.10
TCB linked to work queues

—Console interrupt

—Task scheduler; when all other high-level interrupts and services have been cleared, Taskmaster can run to check for and initiate further work, if any

Bearing in mind MAXCOM's dedicated role, operator communications are minimal. Assuming that a console terminal is included, operator communication is initiated by using the console interrupt switch on the processor. Commands include:

ACTIVATE	initiate a task
HOLD	suspend a task

RESUME	reactivate after suspension
DATE or TIME	enter or display
CHANGE	priority
DISPLAY or TYPE	memory dumps to printer or console
MODIFY	memory locations

Calls to the system services provided by MAXCOM are implemented by software interrupts called REX (Request Executive Service). A REX call emulates an unimplemented instruction, causing a high-level hardware interrupt. The REX call specifies the name of the service function and an optional "quick" return to the calling task (a return when the call is initiated rather than the normal reschedule on completion). A set of macros are defined for REX calls that encompass the following functions:

Task Handling

ACTIVATE	schedule the execution of a user task
QUEUE HIGH	assign a piece of work to the high-priority queue of a user task
QUEUE LOW	assign a piece of work to the low-priority queue of a user task
DISABLE LOW	prohibit a task from being activated due to the presence of work in its low-priority queue
ENABLE LOW	permit work on the low-priority queue to cause task activation
WAIT	release control to Taskmaster
CHANGE	alter the priority of the requesting task

An example of task activation is:

```
•
•
•
LDI, R1,*TSKID (load Reg. 1 with task ID)
REX,ACTIVATE
•
•
•
```

Return occurs to the instruction following the REX, as though for a "quick" return, with no options, for this particular call.

Timer Management

MAXCOM provides timer services based upon the use of the hardware real-time clock. The three services are:

DELAY schedule an event to occur at a specific time of day
ELAPSE schedule an event to occur after a specified period of time
STRIKE cancel a time-dependent event initiated by a DELAY or ELAPSE operation

An example of the use of DELAY is shown below. In time-based operations, register 1 must be set with the address of a timer parameter list which contains:

—The task to be activated at the expiration of the delay
—The entry address of the task to be activated
—The time of day in hours, minutes, and seconds

```
•
•
•
LDI,R1,*PARLIST
REX,DELAY( + QUICK)
•
•
•
```

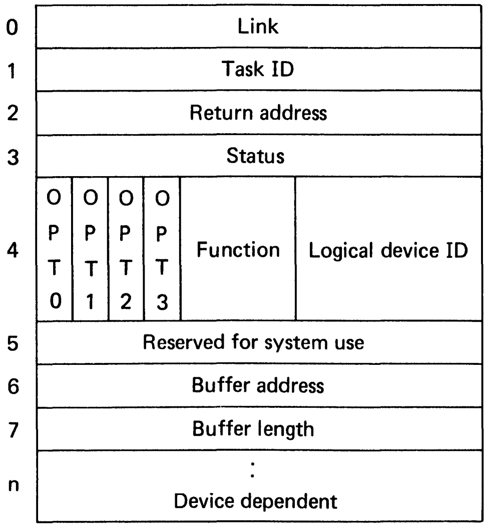
The quick return option specified above returns control to the next instruction immediately the delay has been queued.

I/O Management

MAXCOM I/O control routines enable the user task to submit multiple I/O operations on single devices. Such operations are then executed concurrently with ongoing user task processing. Two REX calls are used in relation to I/O handling:

I/O CALL Read from or write to a peripheral device or communications line
TERMINATE Cancel an I/O call previously issued but not completed

When an I/O CALL REX is issued, register 1 points to an I/O parameter list, the format of which is shown in figure 7.11. The IOPL contains:



0	Link					
1	Task ID					
2	Return address					
3	Status					
4	OPT0	OPT1	OPT2	OPT3	Function	Logical device ID
5	Reserved for system use					
6	Buffer address					
7	Buffer length					
n	: Device dependent					

Figure 7.11
General I/O parameter list (IOPL)

—The task to be initiated on completion and the address at which the task is to be entered

—A status word which should be all zeros on entry (and is analyzed on completion by the user task)

—Various options, most of which are device dependent

—The function to be performed (say, read or write)

—The logical device ID

—The address and length of the buffer

The list may be extended to include device-dependent information.

Storage Management

MAXCOM storage services provide dynamic facilities for handling main storage as single contiguous strings of words and as chained buffers (see earlier description of MODCOMP II DMP). The REX calls are:

ALLOCATE	request a block of contiguous memory for a specified number of words
DEALLOCATE	return storage to the pool
GETBUF	get one or more linked buffers
RELBUF	return linked buffers to the pool

User Support

These services are executed at the priority of the user task; all other calls are executed as Taskmaster priority.

DUMP	print portions of memory during execution
ASCII TIME	
CAN-ASCII	convert from ASCII to Compressed Alphanumeric format (CAN), a radix 40-decimal form that allows 3 characters per 16-bit word
BINARY-ASCII	

I/O Handlers

MODCOM's I/O handlers range from simple Teletype-compatible asynchronous devices to full bisynch protocol. Handlers for magnetic tape, floppy disc, and cartridge disc are also supported but only at direct physical block level, not through a file handler. The more complex handlers, such as the bisynch emulators, are implemented slightly differently with a package referred to as a symbiont.

General comments on MAXCOM

This is a very specialized executive that complements the excellent hardware communications features of the MODCOMP processors. The designers have concentrated on the most important areas of multitasking, queued I/O handling and dynamic storage management, as the framework on which a dedicated communications system can be built. For more general-purpose teleprocessing applications (for instance with batch background work, program development, and so on), the other MODCOMP operating systems can be used.

The Texas Instruments DX10

DX10 is the operating system for the Texas Instruments 990/10 computer. The 990/10 instruction set is somewhat similar to the DEC PDP-11 rather than the more conventional mini. It also uses vectored interrupts and virtual I/O through the top 2 KB of memory space; 1 KB for a ROM loader and 1KB for peripherals, referred to as a TI LINE.

The processor has two distinctive features.First, the 16 general purpose registers, referred to as a work space, are in fact in main memory, pointed to by a CPU register. Thus registers can be changed in a context switch simply by updating the work space pointer register; against this all register accesses are performed at memory speed rather than CPU speed. Second, in addition to the virtual I/O system the processor supports a high-speed (4 MHz) bit-serial I/O port

(CRU). A single instruction can transfer from 1 to 16 bits from or to a memory location starting at any of the possible 4K CRU addresses. TI's own video display terminals (VDTs) utilize this line so that further I/O support is required in the software. The memory refresh buffer and controller for each VDT is implemented on a board that plugs directly into the CRU chassis on the processor; the screen and keyboard are linked as a video monitor. Thus memory to VDT transfers take place at full CRU speed, taking only around 20 milliseconds to completely fill a screen. Remote VDU and other terminals use the TI LINE with conventional software support, providing device independence.

DX10 is a multiuser, multiprogramming OS designed for both commercial and industrial applications. It supports a completely free mix of application programs and program development. Extensive use is made of mapping to run reentrant programs. The mapping hardware allows a program to be divided into three sections, each contiguous but of any length. When loaded, any section can be placed in any physical block of memory that is large enough. Since a section may be quite large this does not permit such efficient use of memory as RDOS's 2-KB pages, but it is more versatile than RSX-11M's contiguous programs. DX10 makes extensive use of mapping concepts to implement reentrancy. The three sections can be established as pure code (procedures), local data, and global data. Thus when the first user of a compiler logs on, he gets the whole lot. The next user to request the compiler gets his own data space but is mapped to the procedure section automatically by DX10. Procedures and data spaces (most confusingly referred to as "tasks") are installed separately if the user wishes to take advantage of shared code. The technique used by DX10 was explained earlier.

Memory management is further enhanced by overlaying and swapping. DX10's scheduling algorithm will roll programs in and out of memory on a priority basis. However, a task requesting an I/O transfer cannot be swapped, since terminal buffers are in user space. Overlays can be called by supervisor functions or the overlay mechanism can be embedded in the user program when it is linked. DX10 maintains a dynamic pool of buffers for intermediate storage in line communication and disc file transfers. The pool is also used for intertask message passing.

DX10 executive requires between 40 and 64 KB, depending on the number of drivers and buffers configured. The task scheduler uses a priority scheme with four levels, defined when the program is installed. A separate linked queue is maintained for each level. Tasks are serviced on each queue on a round-robin basis, under control of a time slicer. A task is added to the end of a queue when it is first bid and at the completion of a time slice. An actual time slice allocated to a task ends when one of the following occurs:

—Expiration of the maximum period
—The task issues an SVC that suspends itself
—The system suspends the task to await completion of an I/O operation

Essentially the low-priority queues are not serviced until no task requires service at a higher level. There is, however, a counting algorithm to avoid complete lockout of low-priority tasks. The upper levels 0, 1, and 2 are conventional but the fourth is dynamic. The task with level 4 priority is initialized at level 1 and lowered after a number of time slices; when requesting terminal I/O the level is raised to 1. This gives fast response to actual terminal I/O but deemphasizes the task during heavy processing loads.

Supervisor calls (SVCs) are implemented as XOP instructions, which are software interrupts. The XOP causes an automatic switch to supervisor mode and then uses special move instructions to fetch parameters from a supervisor call block (SCB) defined in the XOP. A typical SCB is shown in figure 7.12. An example of an I/O SVC SCB is shown in figure 7.13. "00" in byte 0 defines an I/O SVC; the specific function in byte 2 (such as OPEN or READ). Byte 1 is used by the system to return a status code (satisfactory completion, time out, device not available, illegal LUNO, and so on). The logical unit number (LUNO) defined by the user program is attached to a physical device at run-time. Byte 4 contains system flags, initially cleared by the user—busy, error, EOF. Byte 5 contains user-defined flags—immediate or deferred return mode flag, reply flag for a read operation following a write, and a blank-adjust flag. If a reply flag is used the reply block address must be supplied in addition to the buffer address, record length, and character count. Each SCB is coded as a byte string in hex; there are no macros.

Extensive supervisor calls are supported as outlined below:

Task control functions

Bid and terminate a task
Suspend a task pending I/O or time period completion
Extend a task time slice
Activate another task
Change priority
Load an overlay
Set condition bit
Get parameters

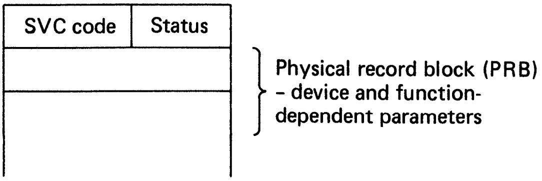


Figure 7.12
DX 10 supervisor call block format

<table>
<tr><td>00</td><td>Status code</td></tr>
<tr><td>I/O Op code</td><td>LUNO</td></tr>
<tr><td>System flags</td><td>User flags</td></tr>
<tr><td colspan="2">Data buffer address</td></tr>
<tr><td colspan="2">Record length</td></tr>
<tr><td colspan="2">Character count</td></tr>
<tr><td colspan="2">Reply block address</td></tr>
</table>

Figure 7.13
DX 10 I/O supervisor call block and physical record block

Service functions
- Date and time
- Binary to decimal or hex in ASCII
- ASCII to binary
- Log a message

Memory management functions
- Access system common memory
- Release system common memory
- Get a block of memory
- Release a block of memory

I/O functions to devices and files
- Open
- Close
- Forward or backward space
- Rewind
- Unload
- Write EOF
- Read device status
- Read (ASCII or direct)
- Write (ASCII or direct)
- Key-indexed file operations

File utility functions
- Create a file
- Delete a file
- Assign and release LUNO to a file or device
- Compress a file
- Modify access privileges

The normal facilities apply, through use of logical unit numbers to write device-independent I/O for compatible devices, (printers or serial files). More extensive file functions will be discussed below, but special mention must be made first of the VDU. These terminals can be accessed as any other record-structured device; however, they have a number of special features that are supported by alternative device-dependent I/O functions. These features are:

—Intensity control
—Beep control; optionally issued on read or write
—Field definition for protected fields
—Default data and fill character
—Scrolling
—Editing within input field
—Special event keys
—Buffered keystrokes; sysgen defined, default 6.

Special keys are used with an editor package which allows a file to be displayed one page at a time. On depressing a key to command the next page, the full screen will be filled within the second, 20 milliseconds for the screen plus a disc access. Since the VDT electronics are directly attached to the CRU line, full status control of the terminal can be applied, rather than to only the interface, giving greater versatility in avoiding peak loading with multiple VDTs in the character mode. The TI microcomputer, the 990/4, is being configured to act as a cluster controller for VDTs.

The DX10 file system is probably the most comprehensive to be built as a standard feature into a minicomputer operating system. It is part of the executive accessed by I/O SVCs, and occupies minimal user virtual space. A multilevel directory structure with full attribute control (read only, write only, shared, exclusive) is used. Files can be contiguous or clustered; they can expand without predefined limits. All file structures are common to all high-level languages. Lock-out at record level is supported. File types are:

Sequential—variable-length records, but they cannot be shared when being written to

Relative record—fixed-length records, buffered by the executive so that with multirecords per block some disc accesses are avoided. There is, however, an option to force a write per record so that buffers can be freed

Multikey indexed—keys can be in any fixed position in the record up to 100 alphanumeric characters long. Multikeys can overlap. Variable-length records are supported. There can be up to 14 keys, one declared as the primary, which must appear in the record. When updating any given record the user may add, delete, or change a secondary key value.

Key values for both primary and secondary keys are stored in indexes within the file. These are hierarchically structured for rapid random access by pointers, and so that they can also be sequentially accessed. The main data is stored in a file in a "bucket" located by a hashing algorithm based on the primary key. DX10 maintains a copy of the initial value of a record during an update to a multikey-indexed file, to assist backup in the event of failure. Variable-length records can be compressed by replacing blank character strings with a marker and a blank count.

DX10 supports a sort/merge package that can be run as a system utility or invoked from a high-level language program. The package features record selection, reformatting of input, and summarizing of output. Sort sequences can be ascending, descending, or other specified collating sequence. Any number of keys can be specified up to a total of 256 characters. Merge supports up to five input files. Sort supports either key sort, summary sort, or address-only sort. There is no automatic line printer spooler under DX10 at present, although one is promised.

The DX10 is a multiuser system. Any terminal not directly attached to a running program can initiate the operator communications program called the command string interpreter (CSI). CSI is invoked by pressing a specific function key on the VDT to bring a copy of the program into memory. There is a separate copy of CSI for each terminal currently using it which is dropped as soon as the services are released.

The range of CSI commands is most impressive, including direct control of program development facilities. The user can define synonyms for regularly used sequences of commands to design simple end-user initiation procedures which can also include access control to specific programs. Each user of CSI can run one background and one foreground job. The background job may be a copy of CSI itself, in which case a sequence of commands can be read from any sequential file or device and evaluated. Thus multiple-stream batch jobs can be supported by DX10, although each must be initiated at a different terminal.

CSI commands are:

—Log in and out, with passwords

—Time and date

—Disc volume control

—Directory control—listing, changing name, protection

—CSI synonym

—File alias name

—Copying files

—Logical unit assignment

—Display task and I/O status

—Program activation and control

—Program installation and deletion

—System log activation

—Program debugging aids—breakpoints, interactive program trace, memory/disc display, decimal/hex arithmetic aids

—Program development support—run text editor, link edit, assembler, COBOL, FORTRAN, or BASIC, and sort/merge

Current support under DX10 for communications packages is limited. There is a Teletype routine and a specific TI machine-to-machine synchronous communications link control package. A 2780/3780 emulator is undergoing field trials which will be directly accessible by SVC I/O calls with limited HASP facilities.

High-Level Languages

As much as any minicomputer OS, DX10 is designed as a high-level-language machine. Languages supported are:

—Assembler, with macro facilities
—FORTRAN
—COBOL
—BASIC
—Business BASIC
—PASCAL

COBOL and BASIC run-time systems are interpretive. All the compilers are reentrant and all generate reentrant code. The COBOL compiler is to ANSI 74 standard, level 1 with nucleus, relative I/O and indexed I/O from level 2. Far more important, however, are the debug and terminal I/O (accept/display) modules implemented for ease of VDT use. TI COBOL, with its multikey-indexed file support, is designed for on-line as well as batch work.

General Comments on DX10

DX10 is a very user-oriented OS. The segmented code concept is very well implemented. Thus when a user wishes to run a specific COBOL program, the COBOL run-time system is loaded as one section of the program, the user's application code as another, and the terminal-specific data as another. Both the run-time and user code are reentrant. Thus when a second user requests the same COBOL program, only a new user data section is loaded and the terminal linked to the already existent code. Thus any terminal can be doing any type of work—one running one COBOL program, two using another COBOL program, one running a BASIC program, another editing and compiling a FORTRAN program, and so on.

Obviously if different terminals are using programs developed in different languages, different run-time support systems are required and memory requirements go up. Also, since there is no dynamic executive-controlled intermediate buffering of terminal data, tasks initiated on I/O SVC cannot be swapped out, reducing the efficiency of the swapping system. Thus memory requirements can be high, although practical experience indicates that the COBOL compiler generates very compact user code. TI, being a manufacturer of semiconductor components, supplies very cost-effective memory. For control of a high number of terminals it would still be necessary to write specific assembly language programs to reduce OS overheads.

A final point regarding the DX10 is the potential of the rather remarkable VDT terminal with its processor I/O port data transfer speeds, rather than a serial line.

Microcomputer Operating Systems—Digital Research CP/M

In general, minicomputers have followed the conventional trend in computing whereby the major software systems are developed by the hardware vendors exclusively for their own products. Alternative software products such as compilers and T.P. monitors are offered by software houses for the leading machines but usually utilizing a standard operating system. Examples of total "foreign" system software packages such as the Bell labs' UNIX operating system for PDP-11's and the BLIS-COBOL system for Data General machines come to mind, but they are rare exceptions.

Now with microcomputers, while there are a variety of microprocessors available, the earlier dominant systems all featured the INTEL 8080 and later the ZILOG Z80, which will execute 8080 code. The PET and APPLE feature the 6502, but these are later products. Thus there exists a large number of suppliers of hardware systems incorporating a similar processor, e.g., ALTAIR, IMSAI, North Star, Cromenco, RAIR, Tandy. To this list can be added a number of bigger names, e.g., INTEL, EXTEL, ADDS, who are also using similar microprocessors in specific products.

With all these systems now supporting floppy discs, the requirements for disc-operating systems is paramount, and with the commonality of CPU one software house, Digital Research, Inc., won the race hands down with its CP/M (Control Program/Monitor) operating system. Probably the cleverest feature of Digital Research's system was the implementation flexibility. If one software system was to be adopted by many hardware vendors, it was important to realize that while the CPUs were similar, the I/O and disc controllers weren't. Thus, as explained later, the I/O routines are supplied as a separate module in *source* code form! Thus the basic CP/M system can be tailored to any hardware system.

The Structure of CP/M

CP/M (Version 1) is a single-user, single-tasking disc operating system. Essentially it provides a user interface to a discette file system and simple I/O routines. It does not provide any form of real-time executive, e.g., no scheduler or task manager, and is thus very different from the minicomputer operating systems previously discussed.

CP/M itself consists of four "sections" plus the user's program as shown in figure 7.14. The I/O and discette handlers (primatives) are called Basic I/O System (BIOS) and a sample is supplied in source code form to any user if required, although, say, PERTEC provides a specific version for their ALTAIR machines. The file handler is included in the Basic Disc Operating System (BDOS). This supports up to four IBM-compatible floppies in the normal form although expanded versions for double-density discettes, etc., are available.

The console commands are processed by the Console Command Processor (CCP), which utilizes the BIOS routines and principally creates calls to BDOS to load programs, etc. User programs are loaded into the Transient Program Area (TPA). Using standard CP/M utilities all programs are loaded and saved only from *tbase* and not from any user-specified locations within TPA. CCP is itself a transient program and can be overlaid, as could BDOS if the user program specifically accessed I/O directly via BIOS.

Most microcomputer systems use a "power-on" reset circuit to cause a jump to a specific memory location. This contains the start of a ROM-based routine which loads the first sector of the CP/M disc into memory. This contains a routine to load the rest of CCP, BDOS and BIOS from the first few sectors of the system disc. A routine in BIOS then sets up certain system parameters in page zero; these

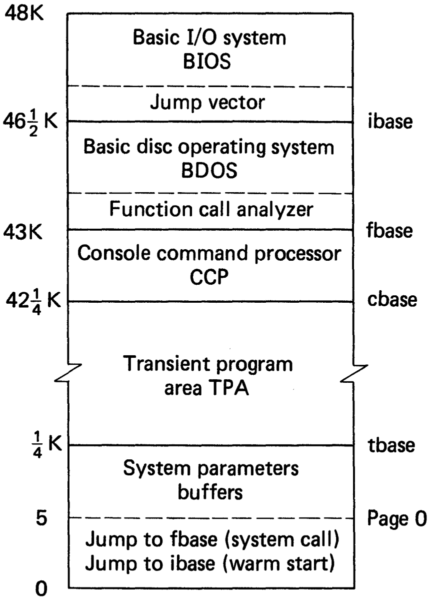


Figure 7.14
Memory map for 48K CP/M system

include a jump instruction for the "warm-start" procedure (typing control C, which is trapped by the BIOS) and the jump to BDOS for entering the call processor. Also included are the remaining 8080 interrupt vectors, some used by CP/M utilities but not CP/M. Buffers are also provided for default disc transfers plus a default file control block. Since CCP and BDOS (if the user program doesn't use files) can be overlaid, the warm-start routine reloads CCP and BDOS from disc before returning control to CCP.

Facilities of CP/M

The functions provided by CP/M are split into two groups, input/output and the file operations. Figure 7.15 lists the functions provided in version 1.4

All functions are accessed by first writing the function number into register C and any input address in register pair D, E. Single byte results are returned in register A and for two byte results (addresses) the high order byte in register B. A jump to subroutine is then executed via absolute location 5 which stores a jump to the location (*fbase* + 6) in BDOS where the function is analyzed and executed.

The BDOS system functions are supported by the I/O primative routines in BIOS. A list of these functions is shown in figure 7.16.

In addition BIOS includes two routines to handle the cold and warm boot procedures.

It must be noted that BDOS is hardware-independent. To tailor CP/M to any hardware environment, only BIOS must be modified. By convention the BIOS routines are accessed by a jump vector at the beginning of the BIOS. Since these are at known memory locations, the BIOS routines can be directly accessed by user programs rather than the preferred access via BDOS.

File Handler

CP/M files are maintained by name in a directory on each discette. A file can be any size from no sectors to a full discette, organized as a logically contiguous sequence of 128-byte records. Physical sectors are not necessarily contiguous and can be dynamically allocated to expand or reduce a file. A 33-byte file control block is maintained for each 16 KB of each file, which is brought into the TPA by an OPEN function. The FCB is updated in memory and written back to disc when the CLOSE function is executed. The form of the FCB is shown in figure 7.17.

The NR field is initialized to zero and sequential access is maintained by incrementing. For a file exceeding 128 records (16 KB) the EX field is incremented and the data for that extent loaded into the FCB. Random access to sectors is simply achieved by precomputing and overwriting NR. A maximum of 16 extents (256 KB) is imposed in version 1.4.

Function	*Number*	*Information*	*Result*
0	System reset		
1	Read console		ASCII character
2	Write console	ASCII character	
3	Read reader		ASCII character
4	Write Punch	ASCII character	
5	Write list	ASCII character	
6	(not used)		
7	Interrogate I/O status		I/O status byte
8	Alter I/O status	I/O status byte	
9	Print console buffer	Buffer address	
10	Read console buffer	Buffer address	
11	Check console status		True if character ready
12	List disc head		
13	Reset disc system		
14	Select disc	Disc number	
15	Open file	FCB address	Completion code
16	Close file	FCB address	Completion code
17	Search first	FCB address	Completion code
18	Search next	FCB address	Completion code
19	Delete file	FCB address	Completion code
20	Read record	FCB address	Completion code
21	Write record	FCB address	Completion code
22	Create file	FCB address	Completion code
23	Rename file	FCB address	Completion code
24	Interrogate login		Login vector
25	Interrogate disc		Selected disc number
26	Set DMA address	DMA address	
27	Interrogate allocation		Address of allocation vector

Figure 7.15
BDOS functions in CP/M, version 1-4

CP/M Utilities

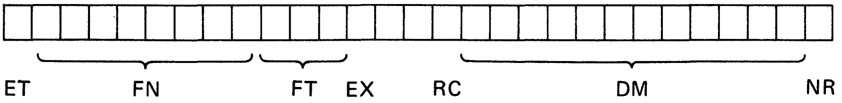
The user accesses CP/M via the console terminal, communicating with CCP. CCP itself contains five built-in functions:

ERA	Erase named file from discette
DIR	Display directory of specified discette

CONST	Return console status in register A
CONIN	Return console character in register A
CONOUT	Output character in register C to console port
LIST	Output character in register C to printer port
PUNCH	Output character in register C to punch port
READER	Return reader character in register A
HOME	Move to track 00
SELDSK	Select disc given in register C
SETTRK	Set track given in register C (0-76 for IBM-compatible)
SETSEC	Set sector given in register C (1-26 for IBM-compatible)
SETDMA	Set memory address for buffer; initialized to 128
READ	Read defined sector to memory buffer
WRITE	Write to defined sector from memory buffer

Figure 7.16
BIOS primative I/O routines in CP/M

REN	Rename file
SAVE	Save *n* 256-byte blocks of memory starting at location 256 onto disc
TYPE	Display the contents of an ASCII file on the console



FIELD	FCB POSITIONS	PURPOSE
ET	0	Entry type (currently not used, but assumed zero)
FN	1-8	File name, padded with ASCII blanks
FT	9-11	File type, padded with ASCII blanks
EX	12	File extent, normally set to zero
	13-14	Not used, but assumed zero
RC	15	Record count is current extent size (0 to 128 records)
DM	16-31	Disc allocation map, filled in and used by CP/M
NR	32	Next record number to read or write

Figure 7.17
File control block format

Any other command issued to CCP is assumed to be the name of a disc file, which is loaded from 256 upwards and executed. The user can thus define any "transient command"; CP/M is supplied as standard with nine utilities initiated by the following commands:

STAT	List statistical details of files on disc; alter and display device assignments.
ASM	Load 8080 assembler and assemble a specified file; produce object file in INTEL hex format.
LOAD	Create disc file in executable code from INTEL hex format file. The file created can then be loaded and run by a CCP transient command of the same name.
DDT	Load and execute CP/M Dynamic Debugging Tool.
PIP	Load the Peripheral Interchange Program for control of disc file and peripheral interchanges.
ED	Load and execute the Text Editor. Used to create new source programs and modify existing source files.
SYSGEN	Create new CP/M system discette.
SUBMIT	Submit a file of commands for sequential batch processing.
DUMP	Dump the contents of a file in hex format.
MOVCPM	Regenerate CP/M system for a given memory size.

Further Software Under CP/M

Because of the acceptance of CP/M on a wide variety of hardware, a standard O.S. interface is in existence on many thousands of systems. The attraction is very obvious, therefore, for any other system software house to produce utilities which utilize the CP/M functions. Leading among such utilities are the language processors developed by Microsoft Corp.

Digital Research produce their own optional utilities, namely:

MAC—an 8080 macro assembler

SID—a symbolic debugger

TEX—a text formatter

DESPOOL—a simple non-interrupt driven routine for simultaneous support of console and printer

They also offer a PL/1 compiler.

Microsoft produce the following language processors and software development aids:

EDIT-80—a text editor

MACRO-80—a 8080 and Z80 macro assembler which generates relocateable code modules. Linkers, loader, library manager and cross-reference list utilities are included

BASIC—a simple 4K, an extended-memory-resident, and a full disc-extended BASIC interpreter (MBASIC) are all available. MBASIC approaches DEC's BASIC-PLUS in standard

BASIC compiler—language compatible with version 5 MBASIC

FORTRAN-80—ANSI '66 compiler, except for COMPLEX

COBOL-80—ANSI '74 compiler with ISAM, COPY and EXTEND. Interactive with ACCEPT/DISPLAY

PASCAL-80—compiler

All compilers generate relocateable code compatible with the MACRO-80 code and linker, etc. PASCAL is not the common UCSD interpreter, and since it can be used to generate code modules it could well become a standard system software development language, competing with INTEL's PL/M, ZILOG's PL/Z, etc. An alternative COBOL compiler, known as CIS-COBOL, produced by Microfocus, includes a screen formatting utility.

A most surprising omission is the ready availability of data communications products. Some good ones are available, however, including a special-purpose CP/M machine-to-machine file transfer package and packages for IBM protocol emulation based on Bisync, that is, 2780/3780 and 3270.

Enhancements to CP/M

With the increasing availability of higher performance microprocessors and cheap discs, particularly eight-inch Winchester drives, there is a need to enhance CP/M. CP/M version 2, released in the last quarter of 1979, is the first step forward. The major enhancements are support for large capacity discs and improved file attributes. This has been achieved by moving *all* disc-dependent parameters into a "disc-parameter block" (DPB) in BIOS. Thus features such as number of tracks, number of sectors per track, maximum number of sectors, etc., can be defined. A DPB is used for each different disc-drive type supported. An optional sector "stagger" translation vector is also included.

With version 1.4 the maximum file size is 16 × 16KB extents (256KB); with version 2 physical sectors can be grouped (clustered) by parameter selection into multiples of 1 to 8, which, coupled with an increase to 64 extents, gives a maximum file size of $64 \times 16 \times 8 = 8$MB. Using a grouping factor of 1 creates a version 1.4-compatible file. At the same time a new random-access facility has been introduced into BDOS which allows physical disc blocks to be allocated dynamically when written.

The directory system has been enhanced so that read, write, and user-number attributes can be appended by file; most discette systems are protected only by a physical tag on the discette itself, so that this is a major advancement.

In addition to version 2 of CP/M, a new system, MP/M, has been released. MP/M is compatible with CP/M but supports a multi-programming nucleus with support for "bank-switched" memory (e.g., two or more memory boards with the same 64KB address space, one of which is selected as active by an I/O instruction). A real-time clock is required and interrupt-driven I/O is preferred. Since the hardware dependency is now becoming severe, MP/M cannot have the universal appeal of CP/M. Since sharing of a cheap 8-bit micro is also somewhat restricting, this system is obviously a precursor for a system based on 16-bit micros. CP/NET has also been introduced to link multiple microcomputers running CP/M and MP/M in a hierarchy. 8086 versions of CP/M and MP/M are now available.

Summary of CP/M

CP/M presents an excellent single-user operating system for program development and single program execution. With FORTRAN, BASIC, and COBOL a CP/M system forms an excellent scientific or small business computer. With the excellent editors, debuggers, assemblers, and probably PASCAL such a system also forms a cost-effective Microprocessor Development System (MDS), *but* as yet there are no equivalents to the In-Circuit-Emulators (ICE) available on the micro-processor manufacturers' own MDS products. INTEL, on their own MDS, also provide a real-time executive, RMX-80. There are few equivalents commonly supported under CP/M, although STOIC is in the CP/M user's group library.

DECNET

While we are essentially considering the connection of terminals to a computer, we have also given consideration to computer-to-computer communication. The general technique with minis is to interconnect two machines by a direct synchronous line and to run 2780 emulators at each end. This technique has much to offer, not the least because it enables communication between different manufacturers' machines. Some manufacturers offer their own point-to-point communication software, but while this is likely to be more efficient than a software emulator the lack of a standard is undesirable. Communication between a number of interconnected machines presents far more problems, and hence there is a demand for network-control software. At the same time such a network of machines offers the potential for one machine to access data stored on another machine, so that software is required not only for machine-to-machine but also *task-to-task* communication, where each task may be one of many running on separate machines. New developments in networking have also provided the opportunity to implement new data communication protocols.

Attempts are being made to produce international standards, particularly the HDLC full duplex protocol and the X25 network control. Nevertheless, two specific systems, totally incompatible, have appeared as the frontrunners—IBM's

SNA (System Network Architecture) with the SDLC protocol and DEC's DNA (DEC Network Architecture) with the DDCMP protocol. There are also strong claims for Hewlett-Packard, MODCOMP, and most other large manufacturers. DNA is not an operating system; it uses a software package called DECNET that runs under a standard operating system and can be loosely considered as a glorified I/O system, accessible by macro calls from any user program. Subsets of DECNET are available under most of DEC's operating systems. DECNET cannot readily communicate with other manufacturers' machines, although an interface to SNA has been announced.

The ideal network software will allow a user task to open a file and access data on one of a number of remote machines using the same commands as are used for local files. Data could then be accessed by the same programs used for local data processing. Logical to physical mapping is then transparent to the user in a global sense. Such ideology doesn't currently exist, and special user programs have to be written. Equally ideally, any machine should be capable of accessing any other, whereas in current practice physical constraints are placed by implemented interconnection. Thus if there are n machines in a network, there should be n-1 input lines into each machine; if there are fewer, some machines cannot talk to some others. It is becoming possible, however, for messages to be passed from one machine to another via an intermediate machine (the network software includes store and forward-message routing). The use of one output line multidropped to a number of receiving stations is also a possibility, but is not currently used in minis. Public switching services such as X25 will allow each machine a single port connection, the network providing the multiplexing and routing.

Objectives of DECNET

The primary objective of DECNET is to pass messages from a task on one machine to a task on another machine and to synchronize communication between them. Thus user programs are written for each node without the user having to check responses and so on. The user programmer can thus code an OPEN statement, DECNET (and operating systems) being responsible for ensuring that processing continues only when the OPEN request is correctly executed. Actual messages will be either control information or data, determined by the executing instructions (for instance, OPEN is all control, GET expects data).

However, various utility-level functions should also be provided in addition to program run-time functions, such as:

—Intersystems file transfer

—Down-line program commands—one program can run or halt a program on another machine

—Down-line system loading—executive software for a machine can be loaded initially from another machine

—Down-line program loading—programs can be stored on one machine and loaded into another

—Software development—the above facilities allow one large machine to be used for program development for all other machines

Another objective of DECNET is to allow user interface via various high-level languages.

The Structure of DECNET

DECNET is modular, comprising a number of layers, each with a specific function. This is in fact an essential concept in network software, since as technology improves certain software functions will be implemented in hardware, which will only affect one module. This may also eventually help in implementing international standards, if such an ideal situation ever can take place. In fact DEC already is taking advantage of the modular approach by implementing subsets on their less common operating systems. The same applies in reverse to enhancements that will be implemented initially on the more common operating systems.

The structure employed by DECNET essentially comprises one hardware and three software layers. These are shown schematically in figure 7.18. The four layers are:

Hardware layer—responsible for the line characteristics (serial or parallel, synchronous or asynchronous, data transmission speed, modem control, character synchronization)

Physical link layer—responsible for handling messages to and from the hardware layer. Concerned with error detection and recovery, also handles identification of messages for sequencing and synchronization. Uses the DDCMP protocol

Logical link layer—multiplexes and demultiplexes error-free messages provided by DDCMP into individual message streams to independent users. The Network Services Protocol (NSP) adds identification to the user message, the combination of which is treated as one unit by DDCMP

Dialogue layer—NSP ensures that each message is correctly routed to the appropriate user program. Whether the message is actual data, a request to access a specific file or device, or whatever, is determined by the data access protocol (DAP).

The user programs can be considered another layer on this onion. The user interface layer allows supervisor-type calls to the services of DAP and NSP to, for example, access a file via DAP or establish connection with a remote task via NSP. These services are available as calls in Assembler and FORTRAN programs under DECNET. Figure 7.19 shows how the various protocols effect the routing of a message from a source program to a destination program.

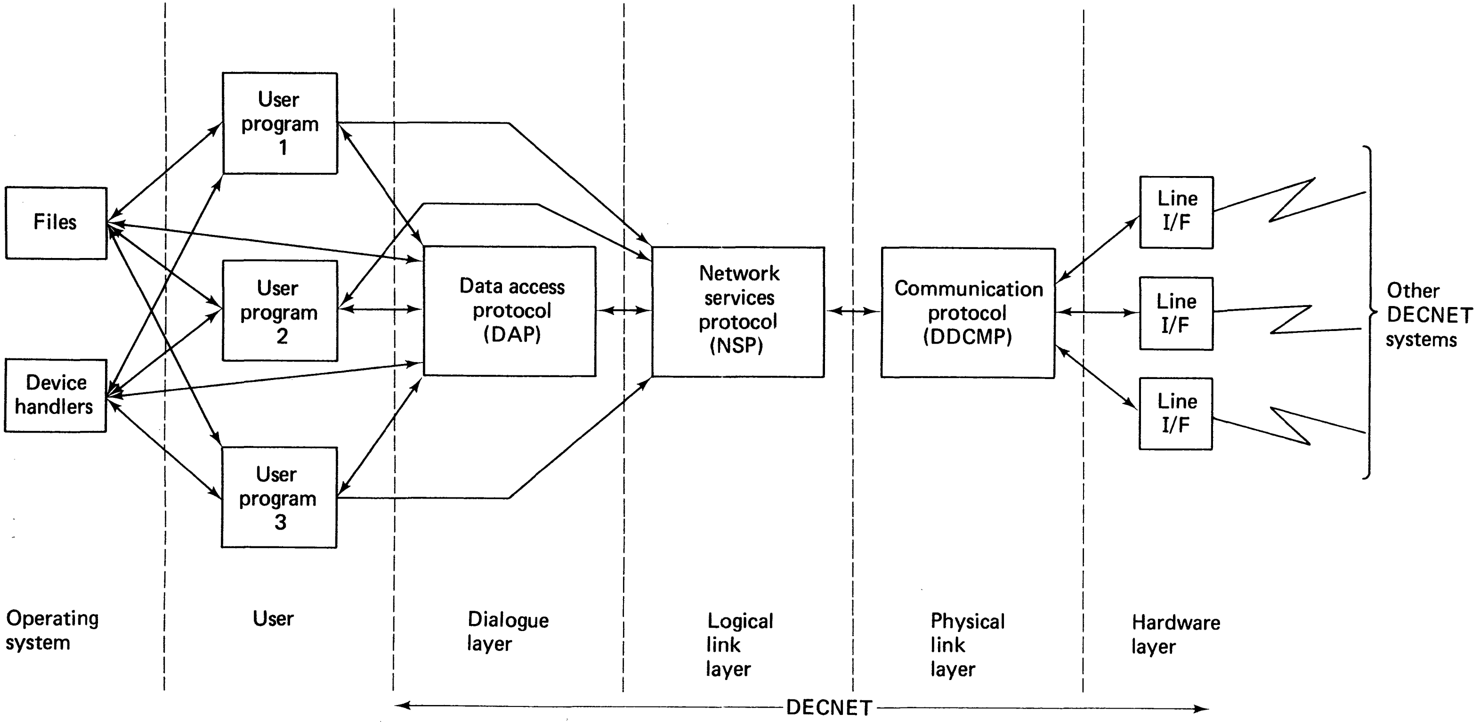


Figure 7.18
Digital network architecture (DNA)

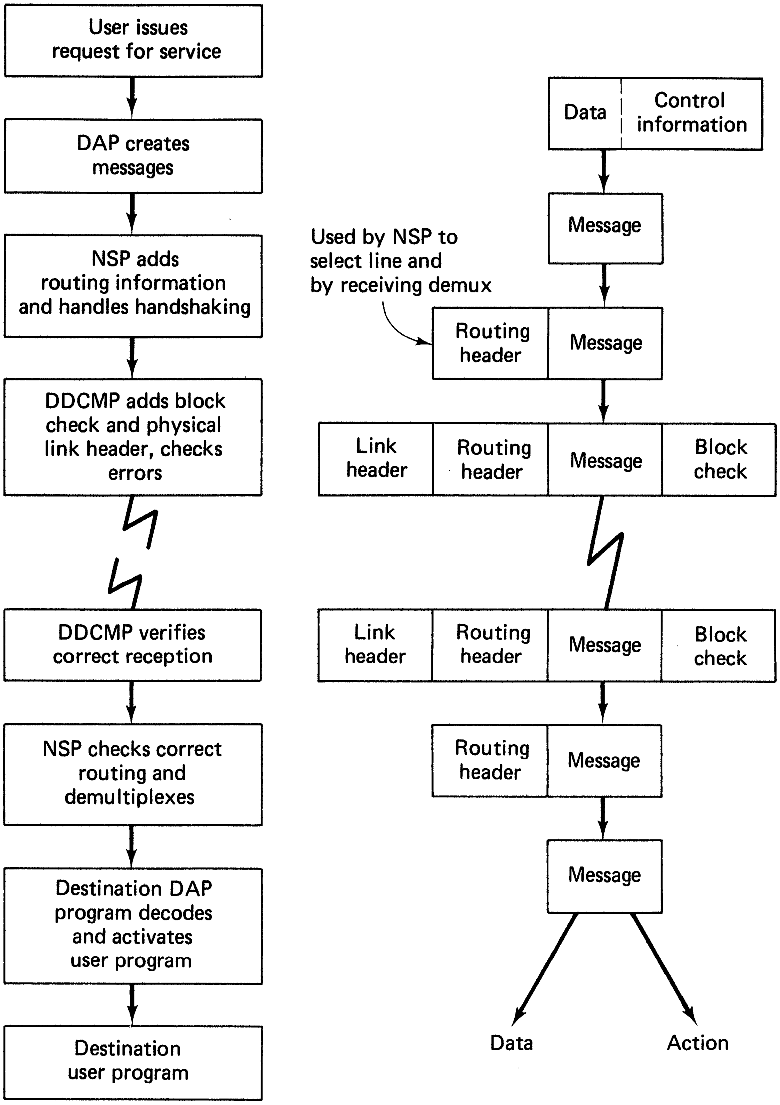

User issues request for service
DAP creates messages
NSP adds routing information and handles handshaking
DDCMP adds block check and physical link header, checks errors
DDCMP verifies correct reception
NSP checks correct routing and demultiplexes
Destination DAP program decodes and activates user program
Destination user program
Data
Control information
Message
Used by NSP to select line and by receiving demux
Routing header
Message
Link header
Routing header
Message
Block check
Link header
Routing header
Message
Block check
Routing header
Message
Message
Data
Action

Figure 7.19
Data flow between programs in DECNET

Hardware Layer

The hardware layer includes a full range of line interfaces with associated line drivers and any interrupt service routines. The objective is to present a line-type independent interface to the physical link layer, thereby enabling DDCMP to be line-type independent. In essence all messages are received and transmitted as though direct to memory viewed by the physical link layer. This implies handling of modem control and synchronization of bit patterns into character strings. In this way the physical link can be a slow-speed asynchronous interrupt-driven line, a high-speed synchronous line, or a parallel line, all similarly handled by the physical link layer.

Physical Link Layer

The physical link layer is essentially required to ensure that any messages transmitted are received error-free and to inform the sender of correct reception. This implies methods of error checking and of acknowledging or requesting retransmission if needed. The physical link layer must also select a specific line for transmission based on a logical routing request.

DECNET uses a full duplex protocol called DDCMP, which is shown in figure 7.20. Recall from chapter 3 that this is a byte-oriented protocol compared to the bit-oriented HDLC and SDLC. This is achieved by using the count to define the beginning and end of the data field. Note also the high number (256) of messages that can be transmitted before an acknowledgement is required. In addition to the protocol shown in figure 7.20, another set exists beginning with

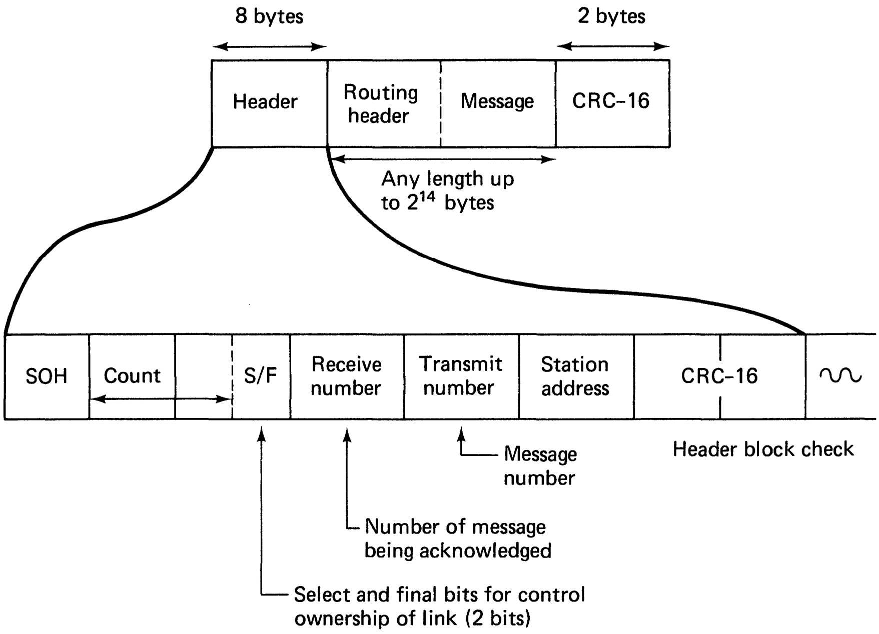


Figure 7.20
DDCMP protocol message format

ENQ rather than SOH and using the count field as types and subtype fields. These are transmitted without data for channel control information as defined by the type and subtype fields, for example, Acknowledge when there is no message to transmit, Boot, Reset, Start, and so on.

DDCMP includes cyclic redundancy check (CRC) characters for error checking on both the header and the data block. Users report this protocol to be very efficient in handling lines that have some probability of error. The 8-bit station address intended to identify multidrop stations is used for route through by DECNET. As technology advances more and more of the functions of DDCMP will be implemented in hardware.

Logical Link Layer (Network Control Services)

The logical link layer effectively creates a line (a "virtual pipe") between one user and another node. The network control software module handles the network services protocol (NSP) designed to control the routing of messages over the network. NSP has two major functions to perform: (a) as the user interface to DECNET for all intercommunications, (b) to generally supervise the network. Intertask communications are to:

—Create the logical link; later releases will allow links to extend through defined intermediate nodes

—Multiplex and demultiplex multiple logical links into physical links

—Control traffic through the logical link, including possible process interruption

—Perform message segmentation and collation

—Ensure end-to-end delivery with correct synchronization and sequencing

—Destroy the logical link

Network supervision functions are to:

—Maintain tables of the network configuration, inform all nodes of new nodes and other changes

—Route messages through the network (automatic store and forward facilities)

—Trace messages through the network

The NSP appends the routing header field indicated in figure 7.20. NSP uses the routing data to instruct the physical link layer which physical line to use, but the information is also transmitted for use in demultiplexing at the receiving station. The header is typically 8 bytes long and contains flags, destination and source link addresses, heading and optional message number (12 bits), and optional mode routing heading. NSP control messages include fields to define the control function and the data required by that function. The control functions include connect and disconnect, set and check link status, (with an allowance for short messages to interrupt long sequences of messages), topology and configuration set and check, and a number of maintenance functions. NSP allows intranode connection between two tasks on the same machine. This of course is less efficient than the conventional intertask communication provided by the operating systems and is provided for development and test purposes.

Data Access Layer

The data access protocol gives meaning at the user level to the messages handled on behalf of the user program by NSP. DAP includes a special module called file access listener (FAL). FAL acts only as a receiving node utility to gain access to any file or device being accessed from the master node. Referring back to figure 7.18, DAP is shown as a reentrant module. In effect each user program can request independent data access via its own effective copy of DAP; NSP multiplexes multiple requests. User programs can of course be responsible for their own files and device access using DAP to pass buffers of data. Any data incompatability between operating systems can be handled by DAP. Ideally DAP would provide all the facilities of a local file control system, but DECNET is much more constrained, being limited to simple GET and PUT record statements. As such, however, DAP allows access to remote devices such as line printers; only on later releases does it allow a terminal on one machine to act as a virtual terminal on another machine.

The user program can also gain direct access to NSP, instead of using DAP, for such functions as establishing links and simple message transmission. NSP also allows user program access to the remote operating systems monitor, in particular to RUN and ABORT tasks on remote nodes. CALL statements are available in FORTRAN and equivalent in Assembly language to gain access to the three basic services described above, message exchange, file access, and task control.

Sequencing

It is most important to realize that DECNET controls the sequencing of events between two communicating tasks. Each program can be written in isolation. Figure 7.21 indicates a simple pair of programs, one to send a message and another to receive it. Each is written independently of the other. When executed, however, DECNET evokes far more functions transparent to the users. Having requested the link, the source task cannot proceed until NSP signals correct completion of the link. To do this the target task must be activated, which will then obtain logical link information and decide whether it accepts or rejects the link. The NSP on the target then informs NSP on the source that it has accepted the link.

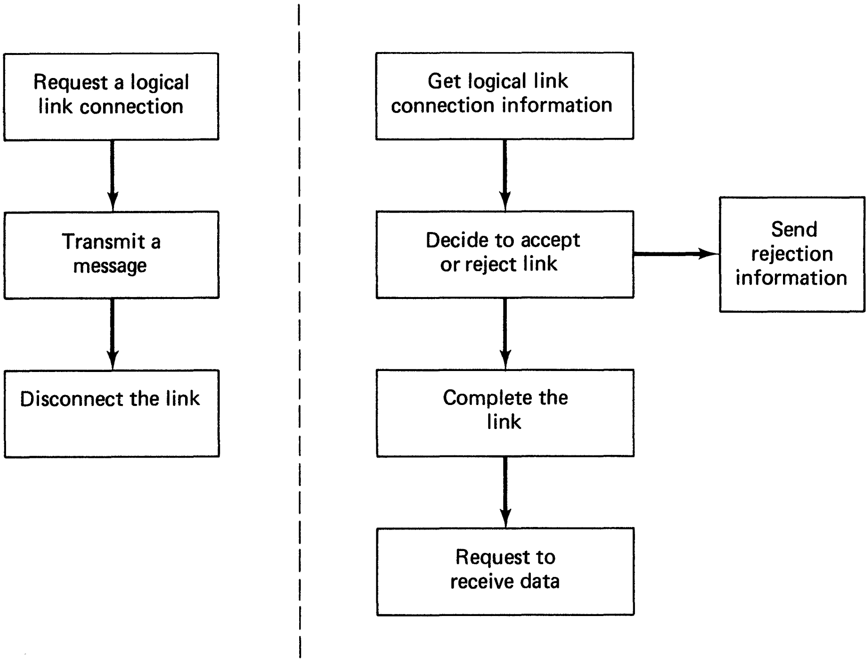

Request a logical link connection
Transmit a message
Disconnect the link
Get logical link connection information
Decide to accept or reject link
Send rejection information
Complete the link
Request to receive data

Figure 7.21
A simple pair of related programs

The target machine can now execute a request to receive a message which will allow NSP on the source machine to execute the send message from the source program. In effect the transmit command in the source program will remain dormant until the target program has issued a request message command. Any transmission errors and recovery will be handled by DDCMP, again transparent to the user. If one considers that multiple tasks using NSP and multiple physical lines must be controlled, it is clear that the network control service program is a complex multitasking software module in itself; it is orders more complex than a direct point-to-point emulator.

Summary of DECNET

In practice DECNET has been quite successful in comparison with other manufacturers' offerings. It can by no means be considered fully developed. There is certainly no attempt yet to allow automatic, or alternative, routing or packet switching. The user interface is reasonable and easy enough to use but has not reached a level of logical compatibility with local programs.

Probably the most successful part of DECNET is DDCMP, although this will probably eventually give way to a bit-oriented protocol such as HDLC when interfaces are suitably developed. The power of DDCMP lies in the fact that it uses standard, readily available interface hardware. This, coupled with the fact that DECNET software occupies roughly 24 KB compared to up to 100 KB for SNA, accounts for the greater acceptance of DECNET over IBM's product.

The availability of a variety of operating systems on DEC equipment is an obvious disadvantage when it comes to networking software, particularly since file structures are incompatible. One cannot help but feel that Hewlett-Packard, with RTE on the 3000, and Data General, with AOS may be in stronger positions to use the now-available SDLC interface components and develop much easier-to-enhance software.

8

High-Level Packages

Minicomputer operating systems are rather more advanced than high-level language compilers. This is particularly pronounced in the area of on-line terminal handling. For dedicated multiterminal systems the powerful macro assembler facilities available can be employed to produce extremely efficient and reliable software. Such a system is also relatively easy to tune, but is inflexible and expensive to program.

At the other end of the spectrum from dedicated terminal handling, the minicomputer can be used as a conventional small business system (SBS). Thus systems using simple on-line data entry and batch processing will abound, based on the existing operation systems, to give some degree of multiprogramming. Such systems, for the benefit of the end user as much as the programmer, must be programmed in a high-level language. Thus COBOL and RPG II compilers (Data General, Texas Instruments, Hewlett-Packard 3000, Varian) are becoming common. Terminal handling is achieved typically by using BASIC or an Interactive-COBOL. Minis are designed as interrupt-driven machines so that the single-threading accepted in small mainframes can be totally avoided.

The pseudominis are supported essentially by batch processing operating systems. To improve these rather limited systems a number of independent software houses have produced transaction processing monitors, which improve not only the terminal handling but also command processing and error handling. Since the technique is well understood in DP circles, it is used on minis as well. One other area in which minicomputers are commonly employed is key-to-disc systems. Conceptually a key-to-disc system is similar to a SBS, but the accent is on dedicated terminal handling routines and simplified file structures. Versatility and processing power are sacrificed for maximum terminal capacity.

Minicomputer On-Line Terminal Handling Packages

While relatively conventional COBOL-based systems with TP monitors will continue to appear, there exist a variety of packaged systems with some interesting features. A few of these will be described in outline in the following sections. The systems described have been chosen because they display original concepts. We make no claim here that they are the "best" in any sense. In most cases the desire to produce an alternative to a batch-COBOL-based system can be summarized as the need for:

—Improved terminal handling statements, including trapping of errors back to the user program

—Simplification of the programming of terminal dialogs

—On-line and interactive program development and debugging

—Direct control of program execution by the terminal operator, with minimal recourse to supervisor control

—Simplification of the syntax of high-level language afforded by the new minicomputer technology

—Password and file attribute security codes

—Accounting of system resources

—Logging of transactions with error recovery routines

No package system will give the total flexibility and efficiency achievable with a multiprogramming operating system with macro assembler. The price paid for the latter system is the difficulty in understanding and making full use of the available facilities; this is not acceptable for a general-purpose, user-oriented data processing system, which should be easy to use even at the expense of efficiency.

Key-to-Disc Systems

A key-to-disc system is used to create machine-readable images of source data documents. Such systems support only simple file structures and allow a minimun of data processing. Multiple terminals linking to one reentrant application program are supported. A simple basic disc-operating system rather than a general-purpose OS is used to minimize overheads both of memory and processor time. By designing the system to handle its own specialized job with no pretensions to full data processing, a simple minicomputer can handle up to 20 terminals. Program swapping is used to support different workloads for each terminal if required.

Apart from demonstrating the importance of designing a system to suit the job, these systems have led to the development of some very good screen formatting and processing languages. These enable easy design of terminal dialogs and simple processing of data fields. The processing is limited compared to a full data processing system, but includes efficient routines for data validation and error checking. A simple file retrieval system is also supported so that, for instance, a name can be returned and displayed against a code number. Once a batch of data has been validated and stored it can be transferred to another machine for processing. A limited version of this technique is used as a foreground program on machines like the ICL 2903 and Burroughs B700, so that the files, once completed, can be batch processed as a background job.

Digital Equipment Corporation COMTEX

COMTEX is an example of a do-it-yourself communications executive. It comprises a system control and interface program to link user-supplied program to the terminal and line-control tasks, including interrupt service routines for specified terminal handling. COMTEX can be configured by the user to include routines to suit his own configuration. I/O handlers are available for most DEC interfaces. More comprehensive control modules are also available to handle line protocols such as Teletype, IBM 2741, bisynchronous, and so on. Applications programs supplied include emulators for 2780 and HASP work stations. In general routines are supplied in source form so that the user can tailor any routine to meet specific circumstances.

COMTEX is supported as a user program under the obsolete DEC batch operating system, DOS, the combination resulting in an on-line disc-based operating system. DOS/COMTEX has now been superseded by RSX-11M, which includes all the above features in a multitasking, multiprogramming environment. It is rather frightening to contemplate an on-line system based on such an elementary and unreliable operating system as DEC's DOS.

Most minis support terminal-oriented routines similar to COMTEX, but they are usually integrated with a real-time OS.

Digital Equipment Corporation COS 350

The COS 350 system is a PDP-11–based member of a range of packaged systems using the language DIBOL. COS 350 supports multiple terminals. DIBOL is one of a new breed of languages structured like COBOL but greatly simplified. Accept and display statements provide single-character handling, but terminals are generally treated as files and accessed with record read or write statements, as are the disc files. A program can support only one terminal, so that even if two terminals are performing the same work, down to sharing common files, two copies of the program must be loaded. DIBOL is a wholly character-oriented language so that numbers are represented and stored as ASCII character strings. BCD support is clearly needed.

The COS 350 executive runs as a task under the single-user RT-11 OS. As such it uses the RT-11 file system (and all other utilities such as edit, and so forth), which allows sequential or random access. Files must occupy contiguous disc blocks, so meaning that maximum sizes must be prespecified. The file handling utility (PIP) can be used to clean up the disc, and they—by copying and deleting—to extend a file. Records can be any length but must be fixed. Thus buffer space for both disc and terminals is allocated in the user program as defined by the data division program statements. The executive is a time slicer, switching between resident programs. With a maximum memory of 56 KB, approximately 36 KB are left for user programs. Programs cannot be swapped but there is an overlay facility, and programs already loaded are shuffled when one exits to leave all available memory as a contiguous block. RT-11 maintains a free memory table to control the shuffling.

The DIBOL compiler is too large to run as a time-shared program, so program development is off-line, the compiler replacing the time sharer. With the mapped PDP-11s a release is available so that the foreground/background version of RT-11 can be employed with, say 128 KB to allow concurrent program development. Since RT-11 is a single-user system and no memory mapping is supported on the COS 350, there is no protection between programs. It is, however, a multitasking system, so that program 2 will execute while program 1 is awaiting, say, a disc access completion (multithreading).

There is a facility for initiating a program at the terminal and then detaching the terminal from that program and using it with another program. Thus two user programs could run concurrently with a one-terminal system, giving a batch facility without tying up a terminal as a console. The number of terminals supported depends upon how large the programs are. With 36 KB of user space about two terminals doing on-line processing or four doing data entry would appear reasonable.

The general-purpose utilities, stemming from the well-established RT-11 OS, are above average. The more specific data processing utilities are average. An indexed sequential package has been introduced, which takes up another 4K of space (see figure 8.1). The file structure is simple and file sizes must be prespecified. The protection facilities are poor. The lack of virtual programs is a serious disadvantage and there is no support for buffered VDUs.

A utility called DECFORM that generates DIBOL programs to create specified screen formats and file structures is available. This is a good example of the trend to introduce improved program development aids.

Digital Equipment Corporation RSTS/E

RSTS/E is a resource-sharing, time-sharing system specifically designed for multi-user terminal-oriented interactive processing using BASIC. It has grown

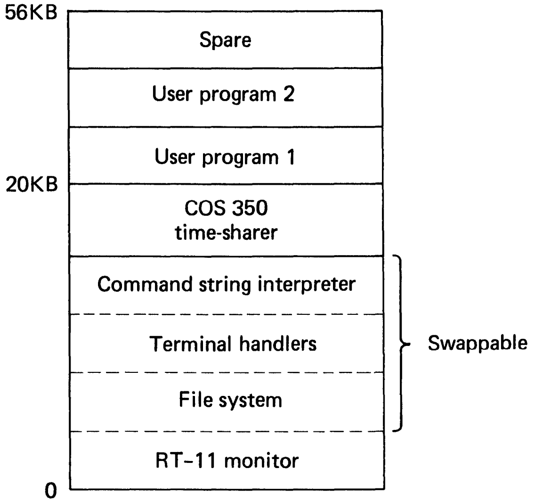


Figure 8.1
The DEC COS 350

through many versions, the early release, Version 4 (RSTS), using an unmapped PDP-11 (no longer supported by DEC) while the extended (/E) versions make extensive use of memory management hardware. Version 5 is the standard system which holds a special place of honor in data processing since more than any other system this accounted for the now common use of minicomputers in commercial applications. Indeed the simple user interface of RSTS/E has set a standard which others must hope to emulate, particularly for first-time user installations. Version 6 can be treated here as an enhancement to Version 5 introducing some compiler support. Version 7, however, is a major restructuring of the executive, providing much wider support for compilers and languages other than BASIC. With the release of RSTS/E Version 7, DEC appears at last to have realized that a sophisticated and complex real-time operating system like RSX-11M is not an ideal user environment for commercial programs which are much more simply served by a time-sharing type of scheduler. Straightforward application programs not using RSX-11M's real-time features developed under RSX-11M using BASIC-PLUS-2, COBOL or FORTRAN will run under RSTS/E Version 7.

Versions 5 and 6. A simple memory map is shown in Figure 8.2(a). The executive controls all I/O and time-slicing; it occupies about 32 KB of memory. The BASIC interpreter resides in a separate virtual space and is also around 32 KB. The interpreter issues service requests to the executive but contains an integrated command analyzer (as in all BASIC interpreters) so that under time-sharing control, each terminal has independent access to all facilities. All I/O is logical, physical terminal addresses being associated with users at log-on. The log-on module has good password and user control features; the monitor also supports terminal status reporting and user statistic logging. User programs, written in BASIC, are entered and edited under control of the interpreter. The

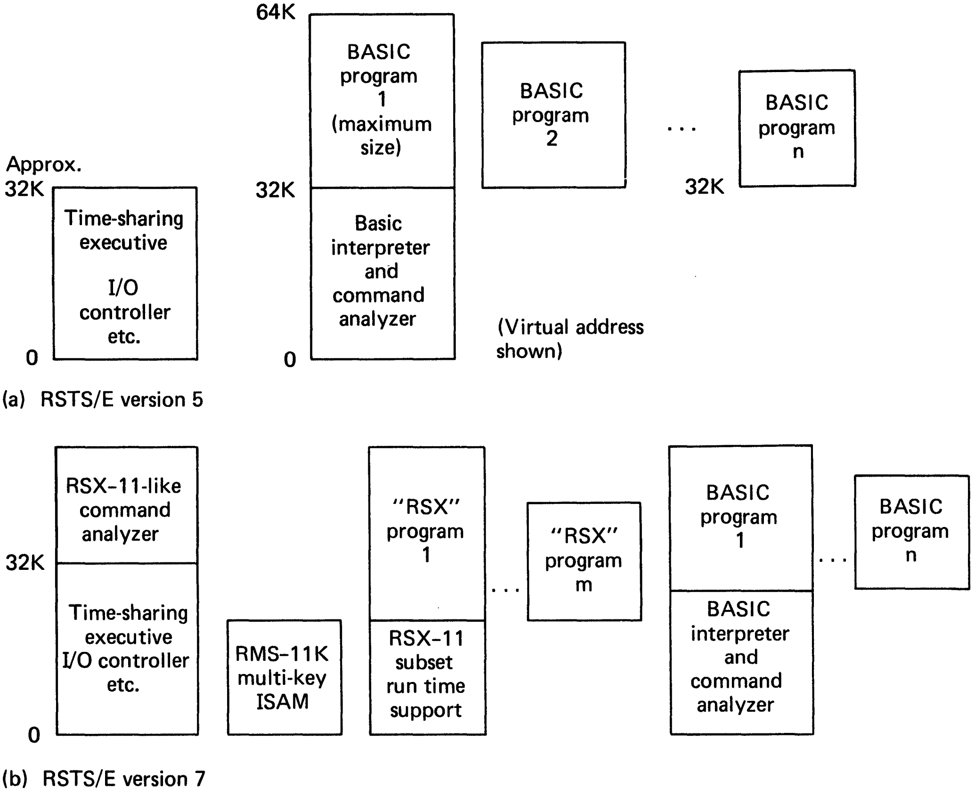


Figure 8.2
DEC RSTS/E memory map

source program is semi-compiled (for run-time efficiency) as it is entered and the syntax checked at each line. Both the ASCII source program and the interpreter level code are filed, the former for listing and editing, the latter for run-time only. The ASCII source program is pre-compiled whenever it is used and therefore takes far longer to load. Each BASIC program has its own space in memory with virtual address 32 to 64 KB; the executive maps the appropriate physical pages into the interpreter space when that program (and terminal) is allocated a time-slice. The time-slicing algorithm is very efficient, taking full advantage of time available from other programs awaiting resources. Thus all keyboard I/O is performed under interrupt by the executive. For example, when a user program executes the statement INPUT A, that program is de-scheduled and takes no further time-slices. The executive collects the keyed-in characters requested, buffers them (from a dynamic buffer pool in the executive address space), checks for validity, code converts to binary, and transfers the binary number to the data area allocated to the variable *A* in the user program. When this is complete, the user program is rescheduled and processing continued.

Each terminal has its own program. Thus if two terminals run the same program, two copies of the program are physically loaded into memory. To minimize physical memory requirements, RSTS swaps user programs out to disc, largely while they are awaiting keyboard input and are inactive in any case. Swapping is transparent to the user. Insufficient memory causes excessive swapping, and while any number of terminals can be logged on, the response will be poor. For a normal commercial system about one in three programs should be in memory. A minimum of 128 KB is required in practice. Programs which do not need interactive I/O can be run from any terminal (usually the console) and the terminal "detached" for running another job. This provides a good batch facility. A line printer spooler is also supported.

The system language is a greatly enhanced version of BASIC called BASIC-PLUS. The language extensions allow floating-point and integer arithmetic and very powerful character string handling. Double-precision floating-point arithmetic (8 bytes) with a user-defined scale factor is used for commercial work (17 digits). True/False logic and AND/OR on bit patterns are both supported so that, for instance, code conversions can be handled in high-level language. Alphanumeric data is normally stored one character per byte, but it can be packed three characters in a 16-bit word, upper case only. BASIC-PLUS suffers from the constraint of variable names being limited to one alpha character or one alpha and one number such as A or Z7; BASIC-PLUS 2 allows multiple-character names. Integer, floating-point and string variables and arrays are allowed.

A feature of an interpretive system like RSTS is its ability to develop and debug programs on line, concurrent with other program execution. Similarly, run-time errors can be trapped and handled by the user program rather than the OS; thus erroneous input data can be detected and the user asked to re-enter.

The file handling is good, but access techniques are limited. Physical files are located by directories as noncontiguous "clusters" of blocks, INPUT and PRINT statements can be used to store ASCII formatted data or GET and PUT statements to access logically numbered blocks, either randomly or sequentially. The system also supports virtual arrays so that an array can be dimensioned up to 32,000 × 32,000, which is accessed from disc via a 512-byte buffer. Lock-out is at block level. With BASIC-PLUS, keyed access routines are included as functions in the user code limiting the virtual space available to user programs.

There is a technique for allocating multiple terminals, effectively single-threaded in one program, but the usual technique is to use one program copy per terminal. The interpreter is reentrant. Strings can be of any length, determined at execution time. They are all saved in a pool with name and length headers. Thus to execute the command A$ = B$ + C$, strings B$ and C$ are concatenated by creating a new string A$ and copying first B$ and then C$ inside the pool. Executing A$ = "NEW" will create a new array and mark the old A$ as dead. Thus the character string pool must be periodically cleaned up by the OS.

Version 7. Version 5 was wholly interpretive. Version 6 introduced support for an alternative compiler system using the enhanced BASIC-PLUS-2. The fundamental idea is to develop and test programs interactively under the interpreter and then to compile them for more efficient run-time execution. Version 7, shown in figure 8.2(b) takes this concept to its logical conclusion. Now there is a separate command analyzer, similar to that of RSX-11M, which provides the initial user interface. Each user can then attach to the log-on routines of either the normal interpreter or the RSX-11M-type run time support system. Only a subset of RSX-11M SVCs are supported, sufficient for the time-sharing environment provided by the scheduler in RSTS. Thus program editors, FORTRAN, BASIC-PLUS-2, and COBOL compilers are all available. Earlier attempts to mount alternative languages were by a run-time emulator of say RT-11, which was grossly inefficient. With the compiled programs it is also possible to support the extended keyed index file package, RMS-11K, moving such code out of the user space into a shared executive space. Version 7 also allows files to be allocated as fixed length contiguous blocks for increased run-time efficiency. Clustered files could still be used during development phases with contiguous files in production once detailed requirements have been established. Probably the most significant feature of Version 7, however, is that DEC appears to have at last produced an efficient COBOL system.

Data General IDEA

Data General's commercial computer, the ECLIPSE C uses the longstanding partitioned real-time OS, RDOS, with the multikey ISAM file system, INFOS. The main processing language is COBOL and without virtual memory support the system is clearly aimed at the largely batch processing domain of the IBM/System 3. As such it has no significant on-line ability. Thus Data General has produced a software package called IDEA based on key-to-disc system techniques. The OS supports three grounds, one of 64 KB for the RDOS/INFOS systems software, another 64 KB ground holds the IDEA monitor, IMON, plus user programs, while the third ground can be used for independent batch processing. The batch processing, given 64 KB, can include IDEA program generation concurrent with execution of other programs. The actual handling of terminal data is performed by the monitor, each IDEA program consisting of a series of high-level calls. Up to 16 terminals can run concurrently.

The programming system consists of two parts, a Format Generator (IFMT) and a processing language (IFPL). IFMT allows screen formats to be generated interactively, defining and naming data fields. Each complete screen definition is stored on disc for future use. The processing system is modular in nature, being driven by input transactions. Using the COBOL-like IFPL, a program module is written to service each named input field in the screen format. Conditional statements are included in IFPL to alter the natural sequence if needed. File handling statements are high level, taking advantage of the powerful INFOS package.

Unusual for a mini, the Data General VDUs use special function keys to notify the monitor of edit, duplicate, erase, and back-tab fields, to end and to log-off functions. The monitor displays the format, controls cursor positioning, and collects data input from the terminal; the operator strikes the enter key when a field is full. The monitor checks for errors and then executes the IFPL module. Control is returned to the monitor to collect the next data item, and so on. When a format is filled the screen is flushed and a new format displayed. A totally different format can be selected under program control, giving program chaining. The monitor supports an automatic transaction file. Data for any field given an output attribute (one of seven possible) is stored on a disc file, together with identification data, before processing by the user IFPL module.

Each IFPL module is small, since most of the features (formats, data conversion and the like) are embedded in the monitor. Each program can be allocated 2, 4, or 6 KB plus 0.5 KB for a screen buffer and 0.75 KB per open file. Since the monitor occupies approximately 50 KB, about three active IFPL programs can share the monitor ground. Further IFPL programs reside in the background space and the currently active modules are mapped into the IDEA ground as required; a program can be mapped out while data is being keyed-in. The system does not support disc swapping, although inactive formats are disc resident. Print control utilizes the same techniques. Data fields are stored on a disc file, which is eventually printed via an IFMT module to format the report. Automatic spooling is not supported.

Data General has recently announced a virtual multiprogramming operating system for the ECLIPSE computers, which could greatly enhance the power of IDEA, since the current system requires a lot of memory. The ECLIPSE processor is one of the few minis with a commercial instruction set.

Computer Automation SyFA

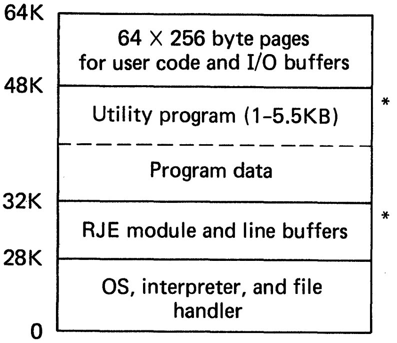
All the systems described except RSTS employ special monitors appended to general-purpose real-time operating systems. RSTS is more specialized but still has a general-purpose applications area. SyFA, however, is an example of a system specifically designed for multiterminal transaction processing, developed by an independent software house and later adopted as a standard CAI product.

SyFA is programmed in a simplified COBOL-like language called SyBOL. The system runs on a 64-KB unmapped CAI LSI-2/60 mini and can support up to 24 terminals. The OS, CyCLOPS, selects any one terminal as the console to control initialization, running of batch jobs, and so on. A RJE program can be supported (2780 or DCT 200) concurrent with the terminal programs; terminal programs can communicate with the RJE system. A 3790 emulator is also available. In addition, one of the utilities can be executed (compiler, sort, ISAM-index-build, for example). The editor is a SyBOL program and can be run from any terminal. An automatic line-printer spooler is embodied which can handle one or two printers; direct printer allocation is, however, possible.

Terminals work in Teletype mode, interrupting by character. The monitor collates data into fields, checking for obvious errors; thus if n + 1 characters are keyed into a field of maximum length n, the last character is not accepted and a bleep is sounded at the terminal. Numeric data may contain a decimal point, automatically padded to fill the specified field. The system is wholly character oriented so that numbers are internally stored as ASCII strings of length specified in the SyBOL program.

The disc system supports sequential (automatically compressed), random or direct and indexed-sequential files. Lock-out is at file or record level. Disc accesses are optimized on a minimum head movement basis. Disc space is allocated as contiguous blocks of nominal length. Further blocks are automatically allocated (updating directories) as required and blocks unused when a file is closed are returned to the available pool. Utilities are available to reorganize the files and list directories. The system only supports 4 × 10mB *or* 4 × 80mB disc drives, with no magnetic tapes.

SyFA is an interpretive system. The SyBOL compiler generates reentrant code which is loaded in 256 byte pages. Some 16 KB of the memory is dedicated to holding pages of user code and I/O buffers. Code pages can be overlaid and buffers rolled-out and overlaid when no memory remains. The OS maintains a priority system linked to the time-slicing executive. In general the least recently used page is overlaid, but shared code and at least one page/program remain in memory. Data areas are not paged and are held in memory. Memory is allocated sequentially as a program is first run and released on completion; the data area is shuffled whenever a program terminates to keep the available data area contiguous. The latest release uses the simple LSI 2/60 memory mapping to swap the 32–48-KB address space between multiple physical memory blocks, reducing problems of limited data space (see figure 8.3). An interesting feature of the SyFA


64K
48K
32K
28K
0
64 X 256 byte pages for user code and I/O buffers
Utility program (1-5.5KB)
*
Program data
RJE module and line buffers
*
OS, interpreter, and file handler

*Usable as program data area when utility or RJE are not being used

Figure 8.3
The CAI SyFA

system is that the terminal log-on routines are themselves written in SyBOL so that they may be tailored to include any password and user code validation and to give access only to selected programs.

The CAI Alpha LSI-2/60 mini is one of the few machines offering a commercial instruction set with character string handling, decimal arithmetic, and multiple stacks. This machine, however, has no hardware memory management to help the paging controller and too few CPU registers. The most unusual feature of this system is its dedicated nature. While it will run multiple terminal-bound jobs, there is no support for other languages, no attempt to provide partitioning to protect foreground from background users, or to support a great variety of peripheral devices. This represents a deliberate trade-off of versatility for efficiency. It is aimed specifically for use as a subsystem, with a mainframe computer for batch programming. The SyFA system is the first to support 3790 emulation to allow attachment to IBM's SNA.

BLIS/COBOL

The BLIS operating system is written for a Data General NOVA by an independent software house (IPI). As in the SyFA system, the use of a general-purpose OS has been abandoned and a system tailored specifically for business data processing has been developed; in this case the high-level language adopted is ANSI COBOL with accept and display statements for on-line processing. A brief specification is:

—32–64 KB NOVA processor, unmapped

—Concurrent multiprogramming of up to 30 terminal or batch jobs

—Maximum program size 200 KB without sectioning

—Concurrent program development, with on-line debugging

—Indexed files with alternate keys

—Operating system occupies 16 KB, the user area being paged, with swapping controlled by an antithrashing algorithm. Page addressing is generated by the compiler, not virtual hardware.

While few details have been given here, the BLIS/COBOL system is mentioned to indicate the existence of specialized system software packages developed to cover holes in manufacturers' software. It uses no Data General software.

Computer Technology TAD

CTL has in the past specialized in the scientific and university market. The well-established MOD 1 processor has been repackaged as the 8000 with an eye on the commercial market. To support the 8000 in the new area further software was required. A reentrant COBOL compiler using the existing MODOS OS gave batch facilities using the standard disc filing system. An indexed-sequential package (IFS) was added and a module called AOF introduced to give JCL control via terminals. To provide run-time support for terminal activities TAD (transaction application driver) has now been released. TAD, AOF, and IFS effectively constitute a transaction processing monitor. The monitor interacts with the processing system through reserved disc files.

Referring to figure 8.4, the system is structured as two autonomous parts, the foreground and the middle ground. Each terminal has its own transaction control program (TCP) which defines screen formats, fields, and so on. A dialog program generator (DPG) is available to create TCPs. Each TCP collects data which is validated with the aid of a library of routines; multiple screens of data can be disc filed to give scrolling. Each TCP has access to a read-only file to give terminal responses. All completed transactions are appended to a sequential transaction log file. As such the foreground routines form a conventional direct data entry system.

In the middle ground, each entry in the log file is processed in sequence; the appropriate transaction processing program (TPP) is identified by the source of the entry. Normally, as soon as the transaction log is entered the TCP can recycle to collect the next input. The TCP can be marked, however, to hesitate until the appropriate TPP has completed, possibly returning information for display via an acknowledgement file. In this way on-line processing can be achieved, although response could be slow, since the TPPs can execute only serially, effectively causing single threading for processing programs, although not for data entry. The transaction log file provides a facility for an error recovery procedure.

Digital Equipment Corporation TRAX

Although its computers have been used with transaction processing monitors for many years, Digital Equipment Corporation only announced its own software for this style of terminal-based processing in 1978. TRAX is a complete package (not a monitor to work with a host operating system as with CICS) aimed at providing the advanced screen-handling facilities, file support, and restart/recovery functions essential to this style of processing. Although designed to run on any memory-mapped PDP-11 (Model 34 and above), TRAX does require the use of a special display terminal (the VT-62) which, by employing a microprocessor, is able to perform most of the detailed screen-handling and low-level data vetting functions. The executive of TRAX is based upon the kernel of Digital's RSX-11M multiuser, multitasking operating system. The RMS-11 data management system is also used. The main features of the system are:

—Up to 64 terminals, but obviously constrained by performance considerations

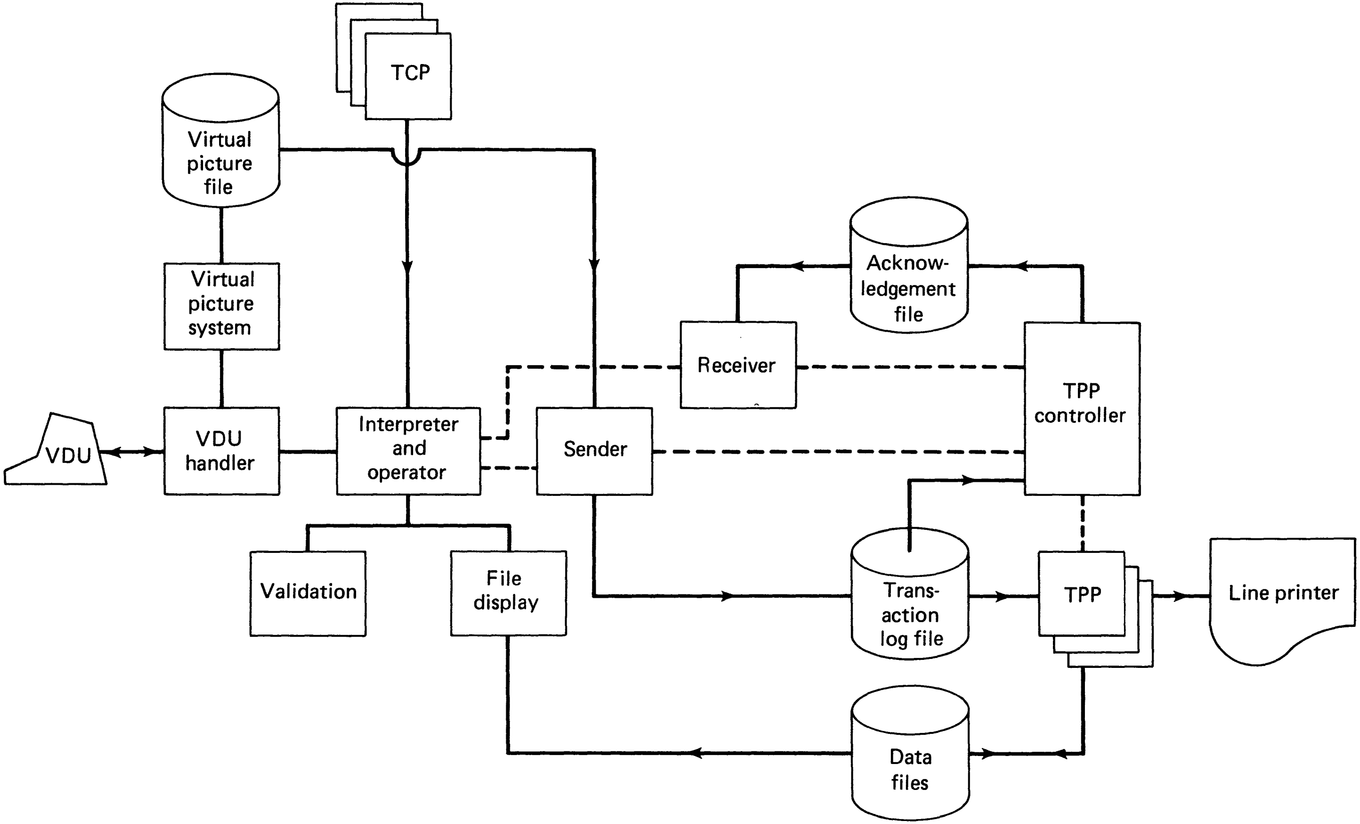


Figure 8.4
The CTL TAD

—Screen handling and data vetting performed in intelligent terminals and specified using a high-level language (ATL)

—ANSI COBOL and BASIC-PLUS-2 language processors

—Programs are written in small modules that are sequenced with terminal I/O according to user-specified transaction description tables

—File support for sequential, relative, and indexed file organizations (with record-level locking)

—Automatic print spooling

—Terminals can be locally or remotely connected, point-to-point or multipoint, using the DDCMP line protocol

—TRAX-to-TRAX communications using a subset of DECNET

—Communications to IBM hosts using a 3271 emulation package

—Sophisticated restart/recovery including journaling and file-update staging

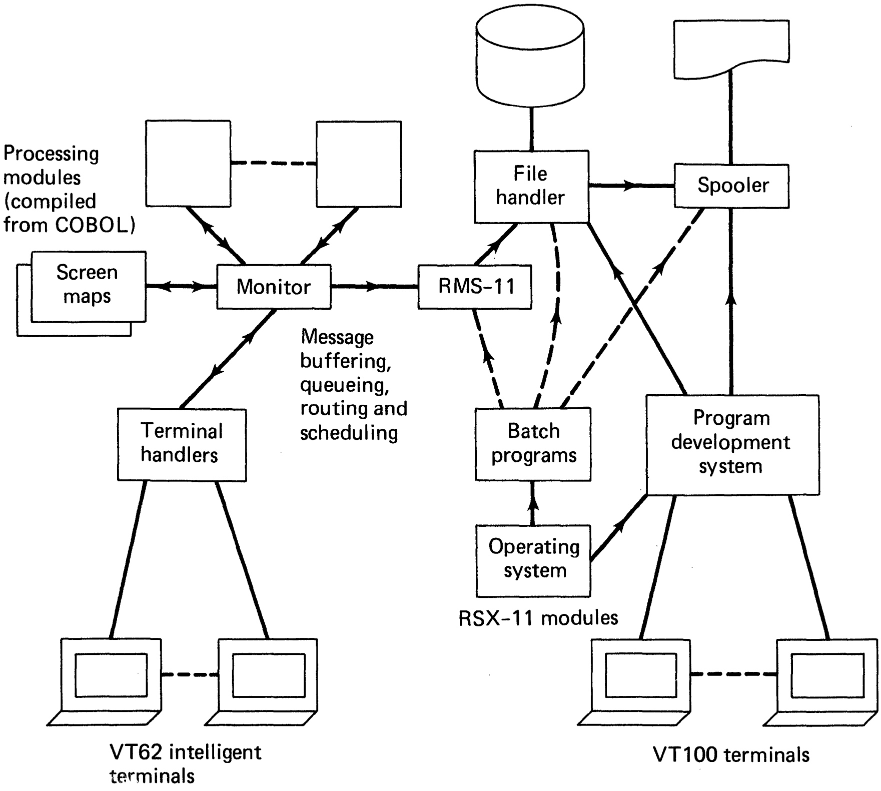

Processing modules (compiled from COBOL)
Screen maps
Monitor
Message buffering, queueing, routing and scheduling
Terminal handlers
VT62 intelligent terminals
File handler
Spooler
RMS-11
Batch programs
Program development system
Operating system
RSX-11 modules
VT100 terminals

Figure 8.5
DEC TRAX 11 system

—User sign-on with password and identity codes

—Interactive development and testing mode

—Batch processing support

TRAX programs are developed under a totally separate partition of the executive and utilize standard VT100 terminals, not the intelligent VT62s. This offers a high degree of protection for the user environment but makes program testing difficult. The system is shown schematically in figure 8.5.

TRAX is a most interesting development in that it diverges from the interactive COBOL or BASIC systems common on minis. While it employs message processing and queuing techniques common on mainframe TP monitors, it is an integrated package. It has many desirable features not built in to systems like RSTS/E (recovery procedure, shared code) and by using small shared processing modules should be more efficient, particularly with multiple users of the same program, thus aiming for support of more terminals. However it loses the simplicity of using and programming a system like RSTS/E, which was a major factor behind the break through of minis into commercial applications. Early Trax systems have suffered severe performance problems and one is left feeling that the whole concept is too late in any case and will now be overtaken by multi-microprocessor based systems.

9

Acquisition and Implementation

It is all too easy to get excited by the prospect of using minicomputers in terminal-based commercial systems. There is many a slip 'twixt the design cup and the working system lip. Our experience has shown that selecting the most appropriate hardware for an on-line system and then making it do the job to the predefined targets of cost, performance, and quality can be fraught with difficulties. Although we do not have space to do justice to system development, this short chapter might help.

Acquisition

It is unusual for professional buyers to be directly involved in the evaluation of computer-related products (probably because of the seemingly complex nature of these products). It is equally unusual for computer people to get any training in good purchasing practices. The key to effective buying is control of the situation. Salesmen are taught that the key to effective *selling* is also control of the situation. Control in this case means who determines the criteria on which the selection decision will be made. Clearly, any salesman worth his salt will stress those criteria best fulfilled by his own product; other parameters will quickly diminish in proportion.

From the customer's point of view the selection parameters need to be worked out well before the salesmen get into his office, and he should stick to them. This can only be done if the buyer has a high level of confidence in his technical capabilities and knows that the specification being issued is reasonable. We find that the best approach is to design a "notional" system that meets the ideal specification in every respect but does not refer to any specific product on the market. Performance and reliability objectives should also be included.

The list of selection criteria can become very large. The problem then becomes one of accounting for the variations in the relative importance of the criteria. For example, one supplier can support 9600 bits per second line speeds but not COBOL; another has the COBOL compiler but can only provide 4800 bps transmission. What is the overall significance of this? A useful and practical technique is to apply a weight to each criteria. This could be in the range 1 to 5, where 1 means "useful feature if available" and 5 means "pretty essential." The weightings should be included in the request-for-proposal (RFP) document. During the evaluation of the bids it should be possible to draw up a large matrix and score each proposed system against the original target system. A method we use involves multiplying the weightings by the following: 0 if the requirement is not met; 1 if the requirement is fully met; and 1.5 if the requirement is *usefully* exceeded. The achieved weighting scores for each proposal can then be accumulated. Although this is a useful technique, your final selection might be made on the basis of criteria that are difficult to score in this manner (say supplier credibility). But it is certainly effective in getting all the criteria in perspective.

Some other factors that might be included in your acquisition procedures are these:

—Some minicomputer suppliers will not *sell* you anything; you have to drag the products out of them. However, do insist that they submit their proposals in the required format. To accept anything else is unfair to the other participants, who are at least trying to give you the information you need.

—To avoid being swamped with boilerplate proposals, format the main body of your RFP as a series of specific questionnaires that require specific answers—can you do it or can't you? Boring or evasive answers will be taken as an admission that the respondent cannot. This approach works wonders.

—Make it clear in the RFP that statements made in the proposal will be taken as contractual. In some countries this is the law anyway.

—Get the supplier to state *exactly* what he will do if his system does not come up to specification in practice.

—Do not accept proposals for anything that is not installed and working at an existing customer site. If it has not been delivered, assume that it does not exist.

—If you are importing a system and your currency is generally on an upward trend, make sure that the price you pay corresponds to fairly current exchange rates. (If your national currency is declining, do not raise the subject!) In either case, make sure that you benefit from any price changes effective between the date of order and the date of delivery (assuming that prices will continue to be reduced).

—It is not at all difficult to find you have over ten companies bidding for your contract and twenty or more is possible—there are a lot of mini suppliers around. The use of independent consultants and product surveys can be a great help in keeping the contenders to a minimum.

—Get all parts of the data processing department involved in the evaluation process. At decision time, the users should also be advised and consulted.

—If you are placing a large order, push your finalist manufacturers hard. Send programmers to their courses, get them to program small, representative applications, and so on.

Implementation

Implementing minicomputer systems successfully is really no different from implementing any other type of system (in the sense that it all comes back to good management of the project). Some factors, however, singly or in combination, may cause problems. These may include:

—Lower levels of minicomputer supplier support than you may be used to with your mainframe company

—Management pressures for early implementation can easily erode time allowed for familiarization with the new technology

—Inadequate time allowances made for all-new system features; minis, transaction processing, data communications, and distributed systems

—In a move from batch (or no computer at all) to on-line working, little or no attention is paid to user needs and no opportunities are presented to the user for participation in the project

—Insufficient staff training directed at providing the new skills needed to develop complex multicomputer, multiuser systems

—Inadequate planning for the introduction of on-line program development

—Not enough input from operators concerning their requirements for facilities to aid the running of multiple computer systems remotely sited from the mainframe center

—Too much wheel inventing and not enough use made of existing software and outside contractors with specialized skills.

These will not be the only problems you encounter. In the belief that problem avoidance is preferable to problem solving we offer the following advice:

1. Do not underestimate the time needed for development. Systems have a natural gestation period and overambiously cutting time on design and programming merely causes the lengthening of the debugging time.

2. Do not cut corners on training costs; your most effective dollar is spent on people rather than hardware. Aim at becoming as independent of your supplier as possible in the area of software support; assume that you will get no help—then when you do, treat it as a bonus.

3. Use outside skills where available to avoid inflating internal staff levels, but maintain strict control of external projects and ensure that you can maintain the resultant software yourselves.

4. Ensure that users get an opportunity to participate in the development project.

5. The operation of on-line systems (particularly where the minicomputers are sited remotely) present particular problems for the operations group; involve them in consultation and ensure the system includes hardware and software mechanisms to aid the day-to-day running of the network.

6. Before the project starts, carefully reassess your data processing department structure and use the introduction of new technology and systems as an excuse to review areas of responsibility, programming team structures, and so on. Finally:

7. Try not to be too ambitious; take things in nice, easy stages and allow time for each stage to settle in before reviewing it and, if appropriate, moving on to the next level of sophistication.

We wish you lots of luck in your new minicomputer-based project, but please don't depend on it.

10

Case Studies

From the wide range of systems with which we are acquainted, we have chosen four case studies intended to demonstrate quite different aspects of terminal-based systems.

Case study 1 is an example of a stand-alone commercial data processing system based on a minicomputer. The operating system used is as supplied by the manufacturer; there are no special-purpose transaction processing monitors employed—indeed, no system programmers are involved at all.

Cast study 2 is based on a system somewhat similar in requirements to the first, although with more remote access facilities. It is also based on a standard mini operating system, but one with limited terminal-handling facilities. Thus the systems house involved has written the transaction processing monitor described as well as the applications programs.

Case study 3 describes a mix of mini and mainframe computers. The system is significantly larger than the other two and involves a number of special software and communications facilities based on a standard real-time operating system. The implications of both mini and host mainframe hardware and software are considered.

Case study 4 addresses the problem of interfacing nonstandard components in small-computer systems using microprocessor-based technology. The available techniques are illustrated by four operational examples.

CASE STUDY 1
Digital Equipment RSTS/E System

Overview

The system described here is installed in a factory complex in Birmingham, England. The company produces a wide range of cooling fans, valves for the petrochemical industry, and aerospace components. The computer system was required mainly for accounting functions in the first phase, replacing Olivetti accounting machines. The engineering department was using an IBM time-sharing terminal for computing data relating to the experimental testing of fans. This terminal was also used to plot results graphically. Similar facilities are now provided on the in-house machine. Since the company had no previous experience with computers, a London-based systems house was employed to provide a detailed specification of system requirements and to select the best hardware available. This system house has since written the applications programs; in fact supplying a full turnkey solution.

Configuration

The system outline was circulated to ten potential suppliers; from the resulting proposals a minicomputer system was selected. The system uses a DEC PDP-11/34 processor, running the RSTS/E operating system with the BASIC-PLUS language described in chapter 8. As such the system was suitable for both accounting and engineering work without alternative language processors. RSTS/E was preferred to the offerings of IBM, ICL, and other mainframe suppliers largely on the grounds of the superiority of the user interface to the time-sharing operating system. The system is so easy to run that it is justifiably classed as a "first-time-user" system. In fact, the superiority of the operating system control language is probably more important than its transaction-processing facilities.

Being a time-sharing BASIC system, RSTS/E offers transaction processing as a standard feature without the necessity for a separate TP monitor. User programs are written to include all screen handling via BASIC INPUT and PRINT statements, and each terminal requires loading of a complete copy of the application program, which can be up to 32 KB in size. Consequently memory and swapping overheads are high, a factor offset by the system's competitive memory prices. While the system uses a DEC processor and software, the hardware, shown in figure 10.1, includes products from other suppliers. The VDUs are asynchronous CDC units with cursor control. The discs are Memorex with the system house's own controller; the printer is also from CDC. The local VDUs are linked via a special eight-channel multiplexor, the modem via a DEC

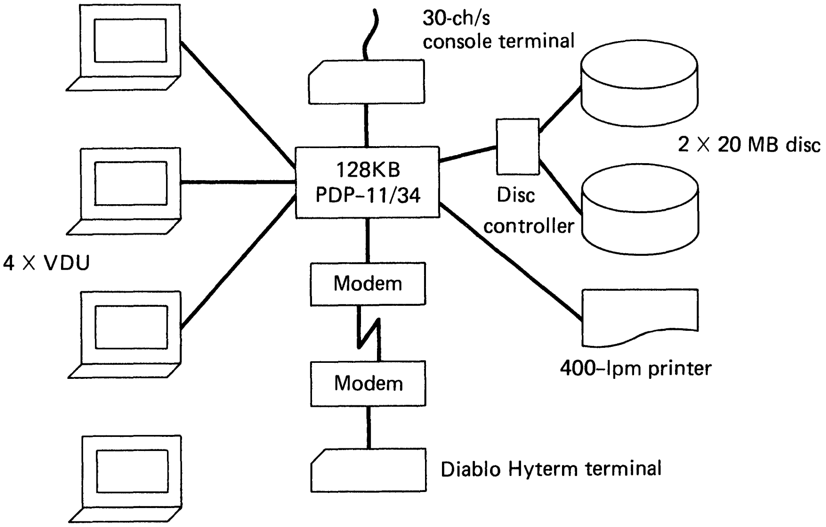


Figure 10.1
Configuration of minicomputer system used in case study 1

DL11E interface. The DEC RSTS/E system has an optional commercial-extensions suite of routines that includes a sort package and an indexed access method routine for file management.

The Applications System

The system currently performs the following functions:

—Sales ledger
—Purchase ledger
—Payroll
—Sales order book evaluation
—Invoicing
—Sales analysis and other management statistics
—Job costing from time sheets

The first phase of a production and stock control system is now underway, essentially the bill of materials breakdown with a check against existing stocks at each level. The computation and graph plotting of fan performance data is also carried out from a remote site, using telephone lines and modems, on a Diablo Hyterm terminal. This is being integrated to develop a library of fan performance characteristics that could be used as a sales aid. It has already been used to help produce catalogs.

The Plotting Terminal

The Diablo Hyterm terminal links to the processor via a dialed line at 300 bits per second, asynchronous. As such it is a conventional ASCII 30-cps KSR printing terminal, although the daisy wheel print head allows the character set and format to be changed. The mechanical movement is effected by stepper motors, which move a fixed number of increments for normal line feed and character spacing. However a set of escape sequences have been defined to switch the system into a mode where direct control of the stepper motors can be achieved. Thus the head position can be accurately controlled so that by printing dots or crosses graphs can be plotted. Both the paper and head can be moved in both directions, which improves the graph plotting facility. Since this terminal is running under the RSTS/E operating system, a BASIC-PLUS program has been written to plot graphs with labels, axis, scales, and so on. User programs write data and commands into a virtual array which is opened when plot is run.

Comments

The remarkable feature of this system is the ease with which programs are written and interactively debugged. Provided a modem is attached, there is no difference between running or developing a program on a local or remote terminal. As a result some programming, particularly maintenance work, can be conducted on the user's machine from a remote office. The software house involved has been required to write very few system software modules beyond those supplied. Modules have been developed to support fill-in-the-blanks screen formatting for specific VDUs with cursor control features. Most programs are, however, written to display a screen background into which the operator keys data which is processed in a question-and-answer style, one field at a time.

In the end the outstanding question to be answered is, How can a minicomputer manufacturer develop such a system, which is competently managed by one person (who also doubles as terminal operator), when similar-sized installations from mainframe manufacturers invariably acquire a data processing staff of three or four people in addition to the terminal operators?

CASE STUDY 2
Data General ECLIPSE System

Overview

Minfo Konsultgrupp is a Swedish software house based in Gothenburg. Minicomputer manufacturers are not generally well established in Sweden, so the choice of system is somewhat more restricted than in the United States or England. One of the companies that is established and offers suitable support is Data General. The immediate requirement facing Minfo Konsultgrupp was for an integrated information system for a container shipping company. This involved a dual processor system with on-line terminals and communications links to an IBM 370/148 for batch processing, an IBM/System 7 used by the stevedores (longshoremen) in Gothenburg, and a Burroughs B3700 situated in Southampton (England) that acts as a central data collection system and a common site to the computer systems used by the company at other European and American ports. The terminals, sited in agents' offices, are the locally produced Alfaskop, which uses a 3270-compatible synchronous polled communication system, an unusual terminal for a minicomputer. Printing terminals use the same protocol. Communication to the three remote computers, including the B3700, is by IBM 2780 emulation. It was estimated that the system took fifteen man-years to complete, with initial installation by the end of 1978 and installation in agents' offices by the end of 1979.

Operational Functions

The applications programs that make up the user system provide four major operational modules, as follows:

Freight Documentation

The documentation requirements in shipping are quite extensive and include documentation to agents, shippers, forwarders, consignees, customers, port authorities, and stevedores, as well as in-house communication. The main functions in the on-line system are:

—Bill of lading registration

—Freight calculation and control

—Printing of documents: (manifest, cargo declaration, unit packing lists, invoices, arrival notices, freight statements)

—Documentation status inquiries

—Correction procedures for bills of lading
—Creation of statistics

Booking

Two months prior to the booking of the cargo, the scheduled voyage of the ship has to be defined for the computer system, giving vessel name, arrival date, deadlines, allotment on ship, and so forth. The booking system consists mainly of the following functions:

—Booking registration
—Automatic check of allotment on vessel
—Automatic check of available containers
—Printing of documents (booking advice, lists, summary)
—Booking forecast
—Booking status inquiries

Equipment Control/Tracking

This module will probably provide greatest financial benefit. The company operates a container fleet of more than 20,000 units, each representing a high capital investment. If the container utilization can be improved by decreasing the turn-around time of the container, much money can be saved. A control and tracking system requires that the status of the container is carefully registered in the computer system. The different functions in this module are:

—Registration of container status changes
—Inquiries as to the container situation at the container pools (depots)
—Trace data on each container or other types of equipment
—Work order; instructions to stevedores
—Transport order; instructions to truck drivers

Marine Operations

Marine operations consist in the main of the creation of a container load plan for the ship, in order to get the right trim and stability. The production of this plan involves a lot of information exchange between ocean ports. Because the system must be available at any time (including outside normal working hours) marine operation information will be Telexed.

Nonoperational Modules

The following data processing functions are also provided:

—Container statistics
—Statistics per commodity, customer, country
—MIS, revenues, costs
—Registration/claims statistics
—Registration/container report and maintenance statistics
—Accounting
—Route code, through transports
—Inquiries

The system modules are shown in figure 10.2.

Hardware

The system is shown in outline in figure 10.3. The required up-time for the system has resulted in a dual processor system. One processor effectively handles the terminal-oriented on-line interactive system, while the other supports the batch facilities, including communication to the remote computers. Each computer has its own 96-MB disc drive with a third drive shared by both processors. This uses a very interesting Data General product that provides high-speed interprocessor coupling with the shared disc controller. The processors used are ECLIPSE C330. The ECLIPSE, while upward-compatible with the NOVAs, is a far more powerful processor. The C version has an instruction set tailored to commercial applications (it includes byte string instructions, for example).

Software

The ECLIPSE uses the standard Data General RDOS operating system, which is a batch foreground/background system not dissimilar in operating characteristics from, say, an IBM System 3. It supports a COBOL compiler with a good reputation and a data management system, INFOS, of some sophistication. However, the transaction processing facilities are rather low-key and certainly not up to the requirements of this system. The IDEA data entry system is interesting, but Minfo Konsultgrupp required a transaction processing monitor to allow the bulk of applications programs to be coded in COBOL. It was also necessary to manipulate files outside the INFOS structure. Thus a transaction processing monitor called TASS (terminal access and scheduling system) was written, employing the Data General communications software modules. (RDOS is described in some detail in chapter 7.)

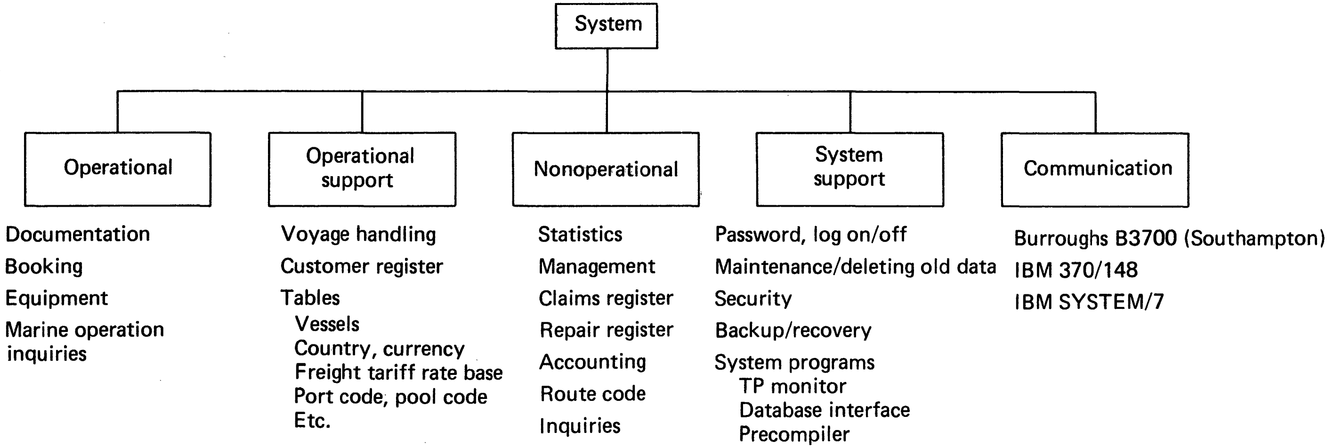


Figure 10.2
Overview of the container shipping information system, case study 2

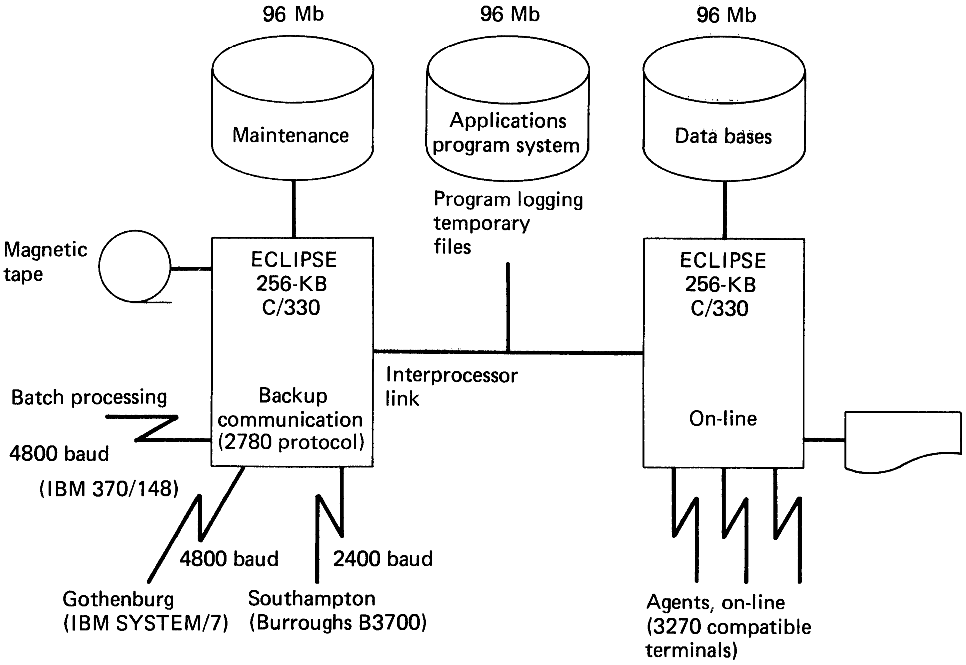

96 Mb
96 Mb
96 Mb
Maintenance
Applications program system
Data bases
Program logging temporary files
Magnetic tape
ECLIPSE 256-KB C/330
ECLIPSE 256-KB C/330
Interprocessor link
Batch processing
Backup communication (2780 protocol)
On-line
4800 baud
(IBM 370/148)
4800 baud
2400 baud
Gothenburg (IBM SYSTEM/7)
Southampton (Burroughs B3700)
Agents, on-line (3270 compatible terminals)

Figure 10.3
The minicomputer configuration, case study 2

TASS (Terminal Access and Scheduling System)

TASS is designed to give concurrent access for a large number of video terminal operators to a Data General commercial ECLIPSE computer. TASS allows messages to be sent from terminals and starts COBOL programs to process these messages. The COBOL programs running under TASS can send messages to video display terminals as well as print on remote printers. TASS uses a simple technique to provide concurrency. To guarantee that all terminals are serviced there is also a time-slicing facility which will, if necessary, swap an overrunning program. Screen layouts are stored in a format library used by the format handler in TASS. This relieves the application programmer of much of the screen handling and gives him the opportunity to concentrate on the data received from or sent to the screen.

TASS appears as a normal program for the Data General RDOS operating system. TASS uses the facilities of Data General software as much as possible, and there is compatibility between TASS temporary files and standard RDOS files, making it possible to use Data General utilities to handle these files. Calls for data base accesses are routed through an interface module called DIPS. DIPS uses Data General INFOS files and adds a number of features not available with

INFOS alone. The structure of TASS is such that it can be transferred to operate under the Advanced Operating System from Data General, when this system is available with support for COBOL programs.

A special feature that will help the programmer develop COBOL programs to run under TASS is a test bed, which makes it possible to test program modules using the Data General debugger to examine and change the values of variables and to insert break points in the program. The application programs request services from TASS by using COBOL-like macros. The macro statements are expanded by a precompiler. TASS itself runs in one partition of RDOS and uses Data General's CAM (communications access manager) package for line support. In particular the SLM (synchronous line modules) are used in this system. In theory 99 terminals, VDUs, or printers working to 3270 specifications can be supported. The computer should be equipped with sufficient main memory to give TASS a 64-KB partition and at least 40 MB in extended memory.

Extended memory is used to hold TASS overlay code and buffers. The more memory available, the less overhead for buffer swaps to disc (see figure 10.4). There is one process for every display in the system, and one for every printer in the system. There are *x* virtual processes in the system (*x* is a system start-up parameter); each virtual process can be either active or free. Each process has its own save area and standard file. If it is a display process it communicates with its own display; if it is a printer process it writes on its own printer (see figure 10.5)

A program can store data for future use by another transaction. This is necessary because each program is terminated once it has produced all messages, and thus no data can be left in the programs' data division. The save area and the standard file are always available, as shown in figure 10.6.

Macro Statements

TASS functions are accessed by calls from the application program. To simplify programming a precompiler is included with TASS. The precompiler will accept macro statements and transform these to the appropriate calls. The macro statements start with the $ sign and are ended by an ! (exclamation).

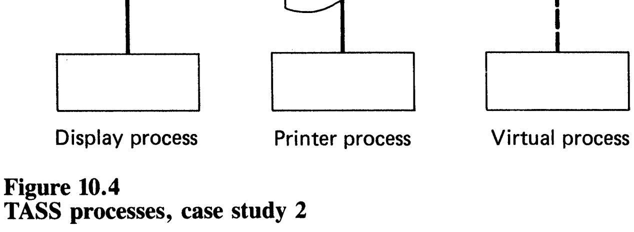
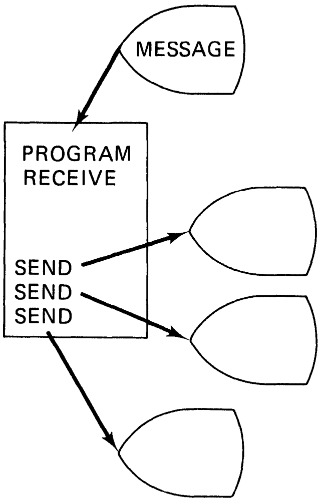


Figure 10.4
TASS processes, case study 2



A message is sent from a terminal to the computer. This causes a program (module) to be started.

The program receives the message.

After processing, the program may send one or more messages to the terminal. These will compose a screen on a display terminal. Only the last may contain input fields.

The program will only receive one message. When it has performed its tasks, the program is stopped.

Figure 10.5
Transaction processing in TASS, case study 2

The Process Save Area (PSA)

Each process has its own PSA, which in turn has two parts; the system part and the user part. The system part is filled by TASS, the user part being available to the user. The PSA contains:

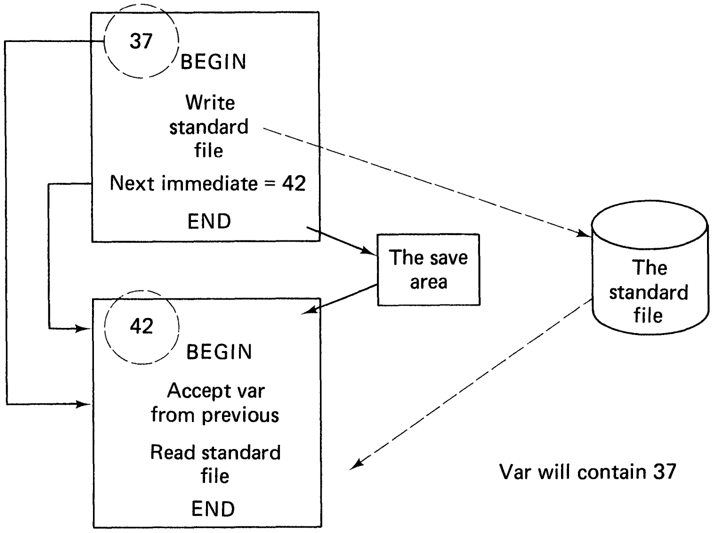


Figure 10.6
Interprocess communication, case study 2

COMA: Communication area, filled by TASS after certain calls

PNUM: Process number: VDU number for display processes (1–99), printer number for processes (101–199), virtual process number for virtual processes (201–299)

USCL: User class—authority classes

USIN: User initials. USCL and USIN in printer/virtual processes are inherited from the display process

TERA: Terminal area code, gives the location of a display or printer. Initialized at system start-up.

PMOD: Previous program module number

PFOR: Format used last in previous modules

KEYU: The key used to end the message from the terminal

AMOD: Currently active program module number

RESERVED: Reserved for future use by TASS

USER: The user part of PSA, 170 bytes

BEGIN and END: $ BEGIN datl!
$ END !

BEGIN copies the complete PSA to the user area datl; END saves the user part of the PSA for future retrieval by another BEGIN.

Program Flow Control

$ NEXT IS datl!
$ DELAYED PROGRAM IS datl AFTER TIME dat2!

With these macros, a program can fix the programs that are to be started next for the same process, under different conditions. User's data field for datl should contain five program numbers vl–v5 indicating the programs to be started under corresponding conditions. The five program numbers to be given are:

vl for ENTER key
v2 for RETURN key
v3 for ATTENTION key
v4 for SPECIAL key
v5 for IMMEDIATE start

Communication with VDUs

$ SEND FROM datl FORMAT IS dat2
STARTING LINE dat3!
$ ERROR IN ITEM datl TEXT IS dat2!
$ RECEIVE INTO datl!

The RECEIVE macro is used to let TASS store the input record in datl. The size and layout of this is as given in the previously executed format. The RECEIVE macro may be executed only once in a program. The ERROR macro is used to display a text of 78 characters given in dat2 on line 24 and to move the cursor to item datl. The keyboard is unlocked and new input may be expected from the terminal. The text is displayed with high intensity. All normal output to the screen will clear line 24. With the SEND macro the variable data given in user's data field for datl is sent to a terminal in the format defined by the format number in user's field for dat2. Only a format can be transmitted to the screen.

File Storage Outside the INFOS Data System

The standard file:

```
$ WRITE STANDARD FILE FROM datl KEY IS dat2!
$ READ STANDARD FILE INTO datl KEY IS dat2!
```

The standard file is an extension to the process save area. There are as many standard files in the system as there are processes.

Temporary files. The temporary files are used to store data for retrieval by the same process, another process or the background.

```
$ ALLOCATE TEMPORARY FILE [SIZE IS datl] ON ERROR statl!
$ FREE TEMPORARY FILE datl ON ERROR statl!
$ READ TEMPORARY FILE dat2 INTO datl
  KEY IS dat3 INVALID KEY statl!
$ WRITE TEMPORARY FILE dat2 FROM datl
  KEY IS dat3 INVALID KEY statl!
```

Data base macros.

```
$ OPEN DATABASE!
$ CLOSE DATABASE!
```

The data base cannot be accessed logically before the OPEN or after the CLOSE. Note that these macros may appear in different modules as long as they are executed in this sequence. At execution of the OPEN and CLOSE macro, an indicator is written in the log file. In this way data-base updates are grouped and the presence of a close indicator means that a logical set of updates is complete. This is used to create a "success unit," which is necessary for the recovery process. The other database macros are:

```
$ UNLOCK segl KEY IS keyl [INVALID KEY statl] !
$ OBTAIN [AND LOCK] segl [APPROXIMATE] KEY IS keyl
                          GENERIC
    [INTO datl] [PARTIAL INTO dat2] [KEY INTO key2]
    [INVALID KEY stat1][LOCKED ALREADY stat2] !
$ OBTAIN [AND LOCK] seg1 NEXT
    [INTO dat1][PARTIAL INTO dat2][KEY INTO key1]
    [AT END stat1] [LOCKED ALREADY stat2] !
$ INSERT seg1 KEY IS key1
    [FROM dat1] [PARTIAL FROM dat2]
    [COPY seg2] [KEY IS key2]
    [INVALID KEY stat1] !
$ REPLACE [AND UNLOCK] seg1 KEY IS key1
    [FROM dat1] [PARTIAL FROM dat2]
    [INVALID KEY stat1] !
$ DELETE Seg1 KEY IS key1 [INVALID KEY stat1] !
$ COPY seg1 KEY IS key1 ON seg2 KEY IS key2
    [INVALID KEY stat1] !
```

Other macros.

```
$ ALERT TERMINAL dat1!
$ POST dat1 TERMINAL dat2!
$ FETCH AND DELETE dat1 TERMINAL dat2!
$ POST dat1 ON CONSOLE!
```

These are used to send and receive messages between processes and to send to the console.

```
$ START PRINTER dat1 PROGRAM IS dat2 IF BUSY stat1!
$ STOP PRINTER dat1!
```

The START macro is used to start the printer process dat1 with dat2 as the first program.

```
$ START VIRTUAL PROCESS PROGRAM
IS dat1 ON ERROR stat1!
$ STOP VIRTUAL dat1!
```

It is possible to start a virtual process with the START macro.

```
$ SWAP!
$ NOSWAP!
```

The intention is that most programs will be of such short duration that it is advantageous to let them run to the end without swapping. If, however, a program exceeds a system generation time parameter, it will be swapped under certain conditions. The programmer may stop a program from being swapped by issuing the macro NOSWAP. SWAP will again permit TASS to swap.

```
$ PRIORITY IS datl!
```

This is used to set the priority.

Other Features of Tass

Format Handler

TASS isolates the application program from the actual layout of the terminal screens. The application programs only send and receive records consisting of the variable data items that are to be displayed on the screen or that have been entered on the screen by the terminal operator. The cursor position may be controlled from the application program.

The formats normally contain all editing information. There is also a numeric de-edit feature. The format handler and the way formats are described are based on the use of 3270-compatible terminals in the network. With this concept the terminal screen can be seen as consisting of various fields. A field can contain either fixed information (from the format) or variable information (from the program). Fields can also be used for entering information from the terminal keyboard.

Error Handling

TASS simplifies error handling in the application programs with a special error statement that causes the cursor to be moved to any erroneous input field and an error text to be displayed. The error text will always be displayed on line 24, which is reserved for use by the system. This line is always cleared by the system each time a message is sent to the terminal unless the error statement is used.

The error text is given with the ERROR statement and should be 78 bytes long, including blanks. If no error text is given in the error statement TASS will simply display the text "ERROR" on the 24th line. After executing the error statement the application programmer should restart the input-handling program from the application program.

Start Up and Close Down

All parameters for starting TASS are stored in the file START UP. The parameters concern the network configuration and TASS options. The installation may have several copies of the START UP file with different information under different names. Before starting the system any of these files may be renamed.

Start TASS by typing "TASS" at the console. During the execution it is possible for the computer operator to intervene with the normal running of TASS in several ways. Using the background console it is possible for the operator to insert and remove program module overlays. By this technique it is possible to remove an erroneous program module and replace it with a copy of the "nonexistent module" program. Using the foreground console it is possible to interrupt any process, whatever state it is in. The process will return to the state it had at start-up.

It is also possible for the operator to immediately stop TASS. A "soft touchdown" closing procedure is provided whereby the processes are stopped when they return to system level. After this command has been given TASS will indicate which processes are still on the air.

Comment

TASS is an example of how an enterprising software house has taken full advantage of the price/performance benefits of a conventional minicomputer system and enhanced the system software to meet its own specific requirements. As a result the system is ideally suited to developing applications programs, mainly in COBOL. The various communications problems involved and the desire to use a relatively large number of 3270-compatible terminals means that a straightforward system like the DEC RSTS/E system previously described was unsuitable.

(The authors would like to thank Jan Jägerström of Minfo Konsultgrupp for his permission to use the material in case study 2.)

CASE STUDY 3
A Distributed System

Overview

This case study reports on the feasibility study, system design, and acquisition and implementation phases of a commercial terminal-based system. In the early phases of the project no commitment was made to any style of networking. The development team did not set out to design a distributed network; a systematic evaluation of the alternatives led to the conclusion that such an approach would provide a most beneficial combination of cost/performance, flexibility, and failsafety. However, the conceptual end product of the design exercise is not without its problems when it comes to translating it to a real-life, operational system. Such problems are likely to be encountered by other data processing systems designers when they get involved in the distributed processing approach. We do not believe we have all the answers, but we can certainly indicate where the pitfalls lie.

Background

The company concerned is a large retailing organization that needed to service some 600,000 customer accounts by the time the new system was fully operational. The accounts were previously handled by clerical staff working in four regional administration centers. For various reasons, it was desirable for the regionalized structure to continue. The offices were organized on a work-group basis, each group looking after some 7500 accounts.

The functions that required a computer system (regardless of its configuration) were:

—High-volume data input (about 60,000 transactions per day)
—Inquiries
—Input exception handling
—Account exception handling

Each of the 80 work groups required three terminals for the last three functions. An initial objective was for the data input function to be carried out by the work-group staff. However, since the input documents arrive singly, by mail, the cost of sorting them into groups and the time it would have taken dictated that a specialized team of terminal operators would be required. This would add 30 terminals to each site, giving a total of 360 for the whole system.

The dialog design work suggested that the terminals should be visual units with:

—1600–2000-character screens
—Forms mode capability
—Alpha keyboards with a numeric keypad
—A price of about $1500 per unit (1979)

The dialogs are highly interactive. In the case of the input transactions, much of the data to be entered will already be held within the system and displayed on the VDU screen for the operator's confirmation. This dictates that response times need to be fast (1–2 seconds in most cases) and the displays must have a minimum screen-fill rate of 480 characters per second. When an error occurs in the input, all relevant details are passed across to the appropriate work group for correction. The posting of the account file (600,000 × 900 byte records) takes place overnight and includes input from both the terminal system and an associated batch processing system (already operational). After posting, each account is examined for irregularities. Reports on these irregularities must be sent to the controlling work groups, who will then use terminals to make adjustments and initiate remedial action (automatic production of a letter to the customer, for instance). Printers installed at the remote sites must have the following characteristics:

—1000 lines per minute (total net rate for each office)

—Upper and lowercase printing

—High print quality (for letters)

For failsafety reasons at least two printers are needed in each location.

System Design Alternatives

The system schematics shown in figure 10.7 are examples of the many alternatives considered (only one regional office is included for reasons of clarity). Option 1 was designed and costed to provide a base system to which the other (more serious) options could be compared. The system required substantial duplexed mainframes if the target throughput and response times were to be achieved. Option 2 was designed to minimize costs by using proprietary clustered VDUs and key-to-tape systems. Although the approach was the cheapest evaluated, there were still performance problems and the key-to-tape arrangement was not an attractive way of getting data into the system (because it failed to make use of the data already in there).

Option 3 involved the use of either general-purpose or special-purpose programmable remote controllers in the regional offices. It was a requirement of this configuration that the local system should have direct-access storage facilities and should be able to perform a substantial amount of the processing (if necessary on a freestanding basis). This made it possible to reduce the central site to one less-powerful mainframe system.

Option 4 was the ultimate in decentralization; a separate medium-scale computer installation in each regional office. The road transport of magnetic tapes would be used for interchange of data with the present system. This was by far the most expensive system and was ruled out for reasons of cost, flexibility, and failsafety.

Although option 3 was not as cheap as option 2, it did offer significant overall operational and performance benefits. It was decided, therefore, to develop the system along those lines.

The Regional-Office Configuration

Once the overall approach had been agreed upon, the team then looked at the ways in which the remote-site systems could be configured. A set of design objectives were specified:

1. The processors needed to have the right capacity. Too little capacity and the number of processors required gives configuring problems and additional expense on the overheads; too much capacity means an excess of unused power in the minimum duplexed configuration
2. High degree of failsafety. The hardware and software must be intrinsically reliable, but it should be possible to configure a duplexed system
3. Each terminal must have access to all file records held locally
4. The 9600 bits-per-second link to the CPU should be functionally full duplex (FDX). Otherwise two such links would be needed from each location, thus doubling communications costs at a rate of over $30,000 per link
5. The display terminal specifications should be met
6. The printer specifications should be met

Although at first sight these requirements may not seem too unreasonable, difficulties were encountered. In figures 10.8 and 10.9 you will see two alternative ways in which a duplexed configuration may be organized. System A (figure 10.8) involves the connection of half the displays to each processor. The processors are both programmed to handle all transaction types. Should one of them fail, the other could handle all the messages but at a somewhat reduced rate. The major problem with this configuration is that the processors must share files. Sharing disc storage is not possible on most of the special-purpose controllers currently available, and although this can be done on most minicomputers, it is a function not normally supported by the software.

One way around this problem is illustrated as system B (figure 10.9). In this case the split is not by volume but by function; one processor looks after file handling and the other handles the terminals. Some form of processor-to-processor connector is needed to implement this. If one processor should fail, the other will have to undertake its functions at a reduced rate of throughput. Once more, interprocessor links are unusual on special-purpose remote processors but fairly common on minicomputers.

On the problem of capacity, it was found that many of the special-purpose controllers were designed to handle far fewer terminals (16–32) than would need to be attached in this case. Once you get above two processors at each site the problems of interconnection get significantly worse. At the moment we are in the ridiculous situation where full duplex communications facilities (lines, modems, and adapters) are easy to obtain, but there is hardly any software/protocol to

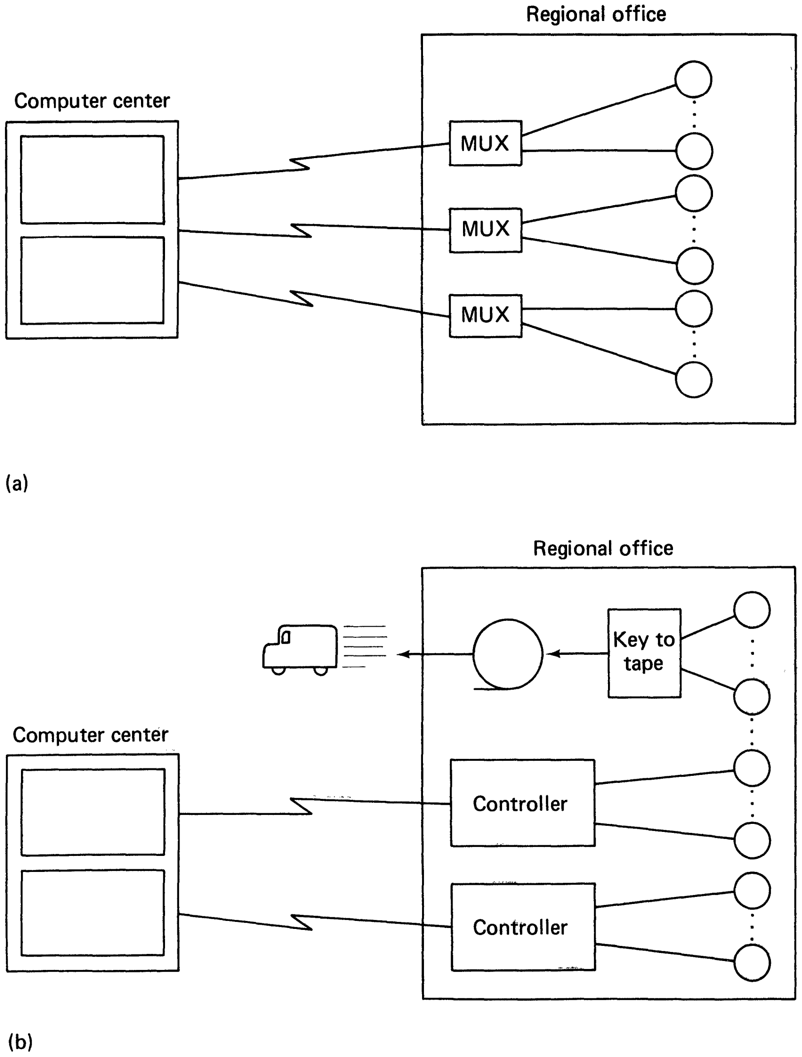
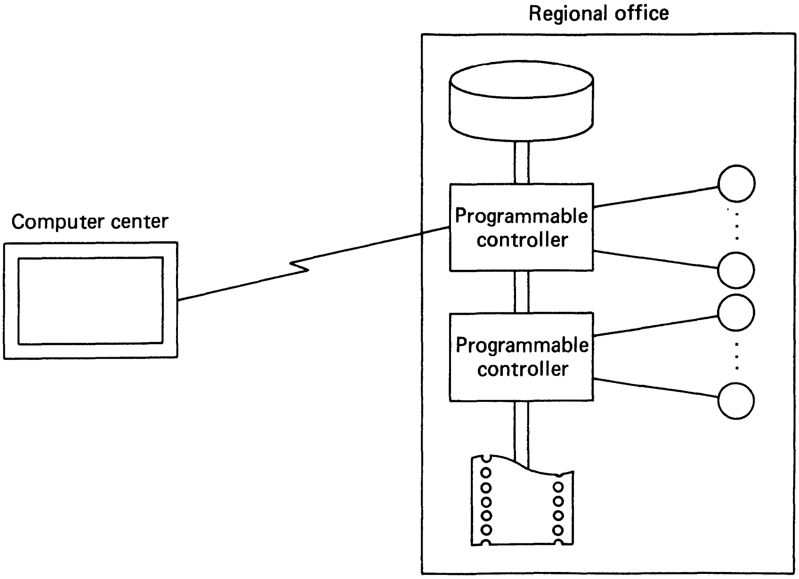
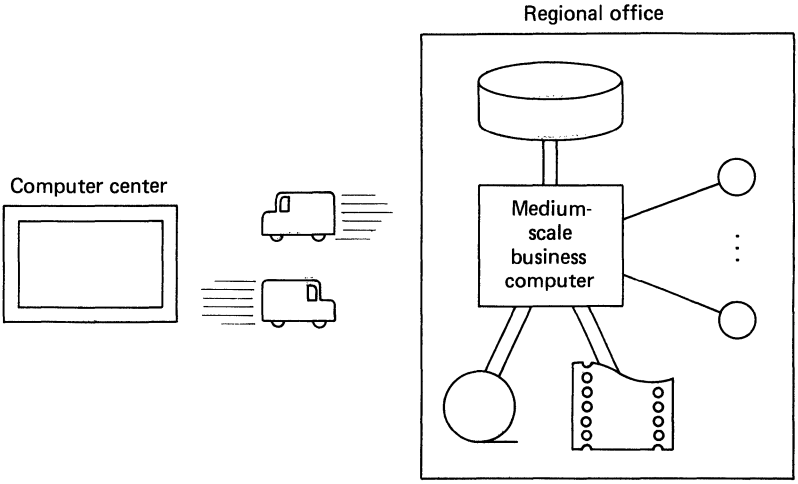


Figure 10.7
System configuration options, case study 3. *(a)* option 1 *(b)* option 2 *(c)* option 3 *(d)* option 4


Regional office
Computer center
Programmable controller
Programmable controller

(c)


Regional office
Computer center
Medium-scale business computer

(d)

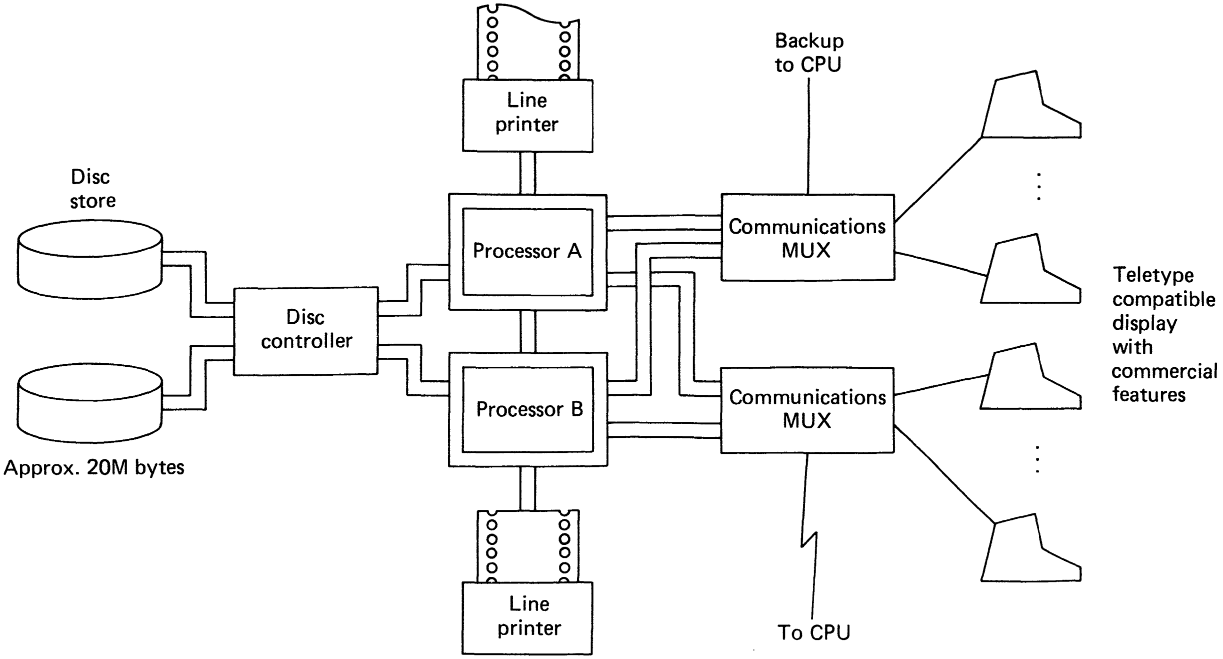


Figure 10.8
Regional office configuration A, case study 3

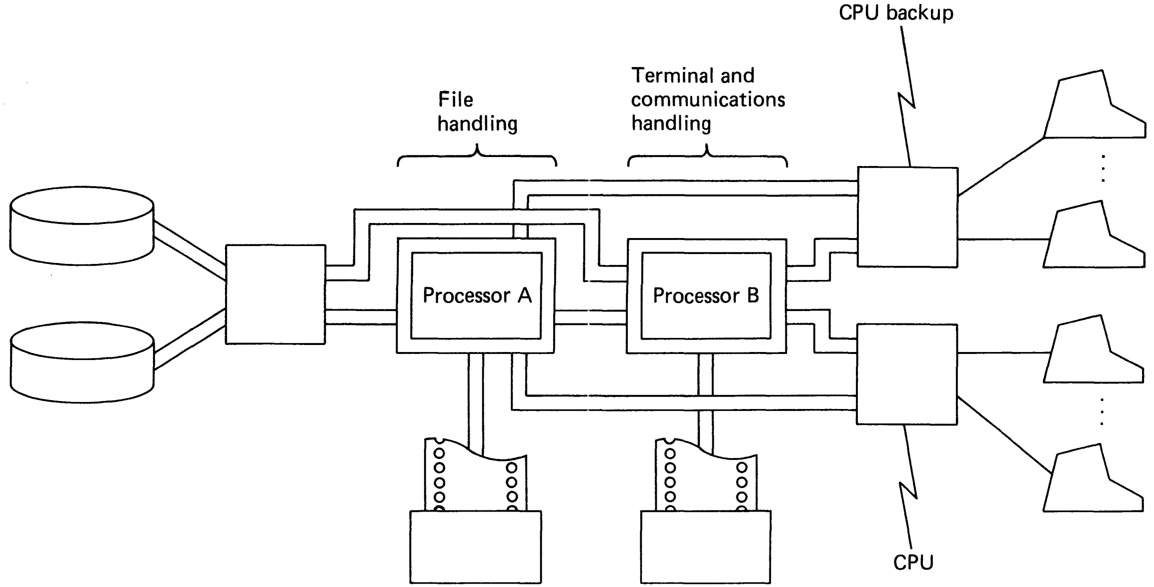


Figure 10.9
Regional office configuration B, case study 3

support it. The few suppliers that have developed HDLC/SDLC support have tended to envelop it in a "network product" such as SNA. It seemed, at the time, that the client would need to develop special software to support FDX working either at the remote site or the central site (or, heaven forbid, both sites).

It is still difficult to obtain good commercial VDUs at less than $3000 (the rapid descent of terminal prices in recent years seems to have been blocked by inflation or something). However, Teletype-compatible displays are becoming increasingly sophisticated. It is possible to obtain 2000 character-screen units with forms mode capability and maximum 9600 bps transmission rate for one-off prices of between $2000 and $2500. Quantity discounts should get this significantly lower, by up to 30 percent or more. The terminals can be connected to minis using V24 or 20-mA current loop circuits. It is generally not possible to connect them to the special-purpose devices that come with their own terminals.

Summary

Figure 10.10 illustrates the operational configuration of this system. The major suppliers chosen were IBM for the central mainframe; DEC for the remote minicomputers; Newbury Laboratories for the visual displays; and Racal-Milgo for the 9600-bps synchronous modems. The following comments will illustrate some of the difficulties encountered in translating the system from the drawing board to the operational set-up shown.

Interprocessor Communications

It was originally intended to link the PDP-11s to the 370 using DEC's 2780 emulation package and a 3705 communications controller at the center. Two major factors turned the development team against this approach. First, the 2780/BSC protocol is half duplex only, and second, when running, the 2780 emulation package absorbed some 30 percent of the PDP-11's processor capacity, a loss that could not be afforded.

The problem was solved by moving the interfacing problem somewhere else. An additional front-end processor was installed at the computer center so that intersite transmission could take place using DEC's standard (and much more efficient) DDCMP link control. This created the requirement to connect the FEP directly to the IBM 370 block multiplexor channel. Fortunately, DEC already had the hardware needed for this (the DX-11), and the company bought a package from a software house that made the FEP look (more or less) like two magnetic tape units, one for the incoming data stream and another for the output. A custom monitor (written by one of the authors) provided a multitasking environment and support for networking via the FEP.

DDCMP is also employed on the 1 M-bps link between the remote PDP-11s. A specialist software house was commissioned to write the software needed to

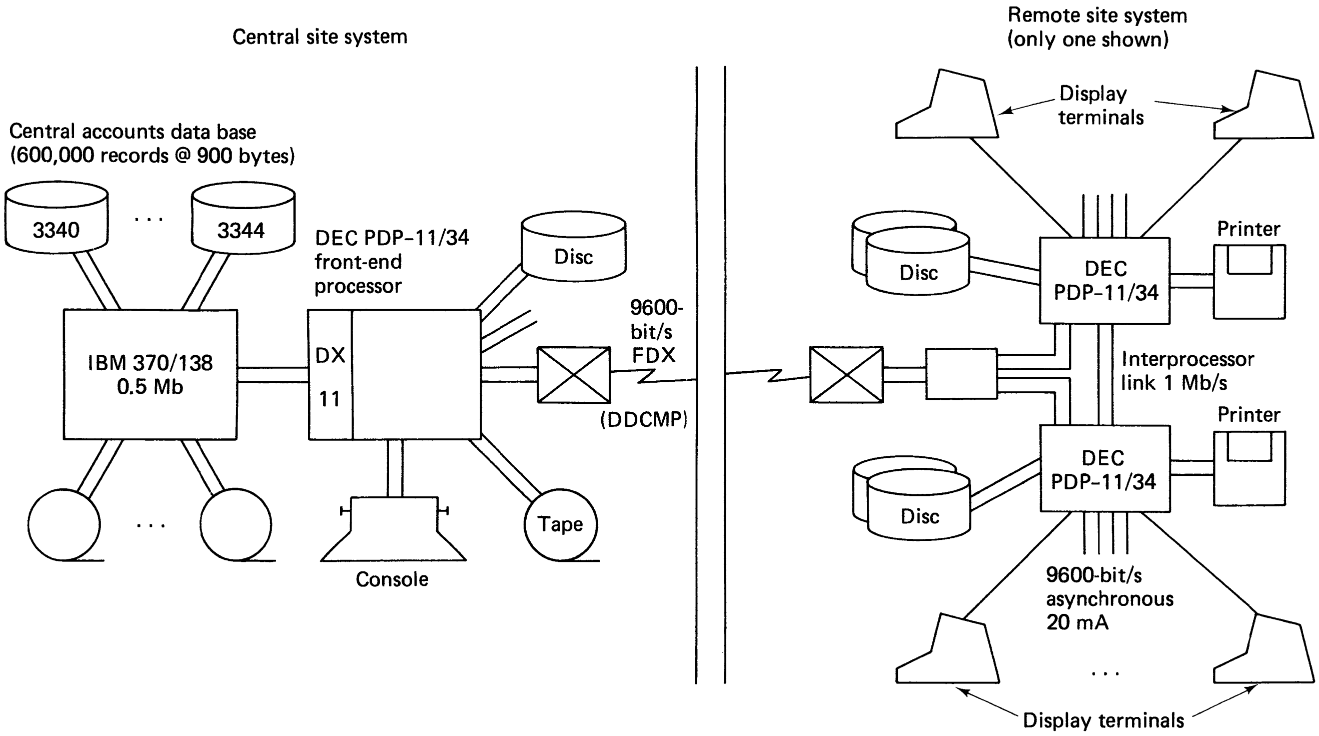


Figure 10.10
Operational configuration, case study 3

control the flow of data among the various systems. (DECNET was not used because it provided much more than was needed and, at the time, lacked maturity.) It has been in this area of networking, however, that most software problems have occurred.

Transaction Processing Software

A package was bought to provide a full transaction-processing environment on the remote PDP-11s. This was based upon the RSX-11M multiuser operating system and provided screen formatting facilities, file handling, and restart/recovery support. The package included a high-level structured programming language.

Distributed Files

A particularly interesting feature of the system is the successful application of some innovative file-handling techniques. For example, in order to take advantage of the fast response times possible on the remote-site systems, copies of the master records for which there are transactions that day are retrieved from the central site and held on the PDP-11 disc. This facilitates a very high level of verification of input data, which is then added to a transmission queue file for sending to the mainframe. The local files are for reference only and are scratched at the end of each day. In the meantime, the central master file is updated in batch mode overnight.

The performance of the system is really put to the test when inquiries are made across the network to the central file. This can take 5 to 20 seconds, depending on the time of day. Since the inquiry responses can fill many screens, they are stored locally on disc and flipped through quickly by the terminal user.

Comments

This system combines the best capabilities of mainframe and minicomputer architectures. Because of the pioneering nature of many of the techniques used, its development has been far from trouble-free. However, a steady program of tuning and enhancement sees the system moving toward impressive levels of performance and reliability.

CASE STUDY 4

Use of Microprocessors in Special-Purpose Data Communications Devices

Overview

The problem with communicating between two independent computers is nearly always compounded by the fact that each was developed seperately and seldom are the I/O interfaces truly compatible. Similar problems can occur with connecting nonstandard terminals to an existing system.

The advent of the microprocessor has provided a flexible means of adapting interfaces and line protocols in low-volume situations, avoiding the problem of redesign in the computer. The line adapter is essentially a simple front-end processor. A box is built with two I/O ports, one obeying the protocols of device A and the other of device B. The actual data from A is buffered in the adapter and retransmitted to B using the appropriate protocol. The adapter will normally contain its own power supplies and a printed circuit board (PCB) for the two I/O systems, the microprocessor and its support circuitry, some RAM for the data buffers, and sockets for EPROMs. The amount of RAM depends upon the size of the buffers and the amount of EPROM on the complexity of the control program and the size of any translation tables. The adapter is in fact a programmable device, but only in special cases is it possible to down-line load the program into RAM. The program is normally developed in a laboratory using a microprocessor development system (MDS), which provides software development aids supported by VDUs, discettes, and printers. The MDS also supports a "prom blaster" for transferring a program developed in the MDS RAM to an EPROM which can be physically transferred to the adapter hardware.

The adapter hardware can take a variety of forms. There are boxes on the market with I/O systems that can be partially programmed by switch selection, specially made for the job and with type approval from the PTT. Others use programmable USARTs and clocks set to a specific baud rate, number of bits, and so on, by executing an initialization routine from the EPROM on power-up or reset. Alternatively use can be made of the standard cards supplied by microprocessor maunufacturers; typically a single-board computer has sufficient RAM and EPROM, which is then coupled to an I/O card. Often the I/O card may need to be specially designed and will not have PTT approval. Another approach is to use one of the manufacturer's single-board-computer kits, which have an area on the board reserved for wire-wrapped circuits that is used for the I/O.

Debugging of special communications devices is a major problem. Seldom is the programmer's interpretation of the two-line protocol specifications exactly correct, and detailed modifications are usually needed. The real communications system can never be taken into a laboratory; the installation must take place on-site with minimal engineering facilities. The adapter is only reprogrammable via EPROMs and a laboratory MDS, there being no surplus memory or other facilities to enable program development on the installed hardware. The programs can be laboratory tested to a degree by the use of simulators, but a device known as an in-circuit emulator (ICE) provides a real-time debugging tool. The microprocessor is unplugged from its socket on the communications device and an umbilical cord that links into a full MDS is substituted. Now, temporarily, the full facilities of the MDS, including VDU, spare RAM, trace facilities, symbolic debugging, program development software, and so on, are available. ICE, however, only debugs the microprocessor itself; it does not debug errors on the data communications lines or inside the new complex interface chips (for example the Zilog Z80/SI0, which can be initialized to support asynchronous, synchronous, or HDLC data link controls). It consequently is not the complete answer, and additional tools such as the Hewlett-Packard logic and serial data analyzers can be used.

Once debugged the program can be committed to EPROM and plugged with the microprocessor into the final system. One further feature required is automatic start on power-up for the adapter, since it has no console.

The following sections briefly describe some typical applications, all of which were developed and commissioned by Hoshi Kalami, a research fellow at University College, Cardiff.

TELEX to V24 Line Adapter

An international company needed to link its message-switching system in Hong Kong into its European system. The European system is built on a PDP-8 based Business Master sited in London, the Hong Kong system on a Cable and Wireless unit. The Business Master uses 7-bit ASCII codes (plus parity), at 1200 bps with a CCITT V-series interface. The Cable and Wireless system uses TELEX at 50 baud, quarter-rate (to use lower tariffs), 5-bit Baudot code with no error checking. Both systems have a defined header and trailer message, including one source and multiple destination I/D codes and a sequence number. Both systems use different headers and trailers. The TELEX is full duplex by virtue of a four-wire system; the Business Master could be constrained to half duplex mode by hold and free messages forming a simple handshake protocol (see figure 10.11).

The line adapter employed was a general-purpose data communications device known as a DATAMAX. This 19-inch rack-mounted unit fits into the Business Master cabinet along with the Post Office's TELEX data communications equipment. The DATAMAX has a Post Office-approved interface. The DATAMAX uses a Motorola 6800 microprocessor with two 1-KB EPROMs and three programmable USARTs. The processor runs a subprogram on power-up that initializes the USARTs, one to 5 bits, the other to 7 bits plus parity. The 5-bit USART is run at 50 baud, the 8-bit one at 1200 baud, and the third one as a dummy at a quarter of 50 baud for control of the one-in-four-character TELEX

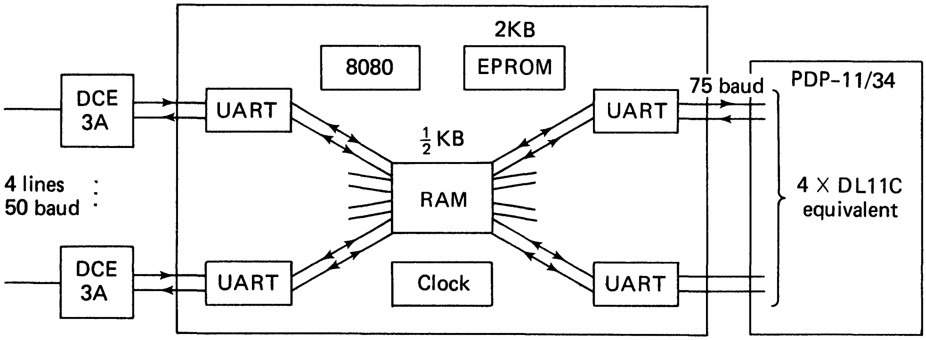


Figure 10.11
On-line TELEX system, case study 4

transmission. On TELEX transmission both USARTs are loaded, but only the 50-baud one is transmitted; the status for transmission complete, however, is read from the quarter-speed USART. Incoming characters are checked for header codes and the data messages are translated by software using table look-up for code translation, remembering the letter-shift/figure-shift characters in the Baudot code. Further tables define the header identifying codes used to reform the ongoing headers.

Since the TELEX line has no error-checking capability, a corrupted character in a header could cause extreme confusion. To improve this the header was checked by a block rather than individual characters and recognized if most—but not essentially all—characters were matching. The DATAMAX program was developed on a SWTP 6800 discette-based D-I-Y computer using the resident DOS operating system and Assembler with a simulator to aid debugging.

On-line TELEX System

A wholesaler in London uses a systems-house-developed minicomputer similar to the one described in case study 1. The firm has fourteen sales outlets spread around the United Kingdom which need to make occasional on-line references to the stock file. Since the traffic rate is unlikely to exceed 5 to 20 inquiries per locality per day, the cost of fourteen VDUs and modems was considered too high. Each remote site already uses the normal Post-Office-switched telegraph system and each has its own TELEX terminal. With the low-volume rate it therefore makes sense to put up with the poor quality of the TELEX keyboard and its low speed and to use it for both normal messages and as an on-line terminal. At the central computer site a rack was installed mounted with four Post-Office-supplied TELEX data terminal equipment units, type DCE3A. This type accepts incoming calls but cannot originate calls. The signals from the DCE3As are V24-compatible voltage levels, but in Baudot code format. The Baudot code has fewer printable characters than the ASCII set, but the applications programs are written to avoid using non-Baudot characters. It would have been possible to write either a BASIC-PLUS program to translate ASCII to Baudot or to modify the line handlers in the RSTS/E operating system. Both are undesirable because of maintenance, compatability with further upgrades, and possible bugs affecting the smooth day-to-day running of the other systems. The same inquiry programs may also be run from local VDUs as well as the remote terminal.

The solution used was to build a microprocessor-controlled box (see figure 10.12) that accepted the I/O from the four DCE3As and buffered, code-converted, and retransmitted the data so that it looked like four Teletypes coupled to a standard low-speed asynchronous I/O multiplexor in the PDP-11. The microprocessor was programmed to trap and ignore echo-back of characters fed to the PDP-11, since the TELEX system is half duplex and will double-print the character if it is echoed. The TELEX runs at 7 cps (50 baud); the minimum speed on the PDP-11 was 7.5 cps (75 baud). This could cause overrun, particularly if the PDP-11 transmitted a continuous message of alpha, numeric, alpha, and so on, since the Baudot code would have to send double the number of characters due to figure and letter shifts. Thus the PDP-11 was set to send 5 nulls at the end of each line which were trapped out by the micro.

The system was built using an Intel 8080 SKD development package and a special board for the four input and four output UARTs. A zener diode barrier unit was included to meet the Post Office requirements, and the box was granted one-off rather than full-type approval. The 8080 proved fast enough to handle the four

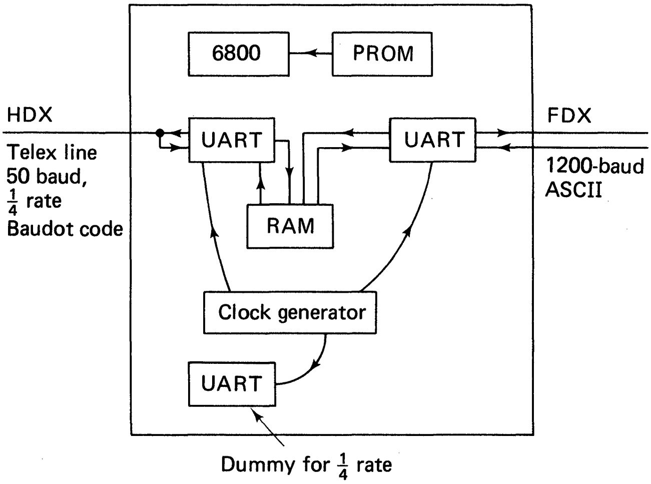


Figure 10.12
TELEX to V24 adapter, case study 4 (courtesy of ICSC, London)

DCE3As, using input character polling, with ease. It was programmed on an INTELEC 8 MOD 80 MDS in Assembly language.

Synchronous Line Protocol Controller

It was necessary to connect a Texas Instruments 990/10 minicomputer running under the DX10 operating system to an ICL 1902 for batch mode data communication. The 1902 scanner supported only ICL 7020 protocols; the Texas Instruments system supports only an IBM 2780/3780 emulator package. The user did not like the 3780 emulator because of its lack of an operator console, and the ICL scanner was therefore upgraded to a Digico FEP, emulating the ICL 7902. While this machine still supports 7020 protocol the supplier, unlike ICL, was prepared to modify line protocols to suit. This could possibly have been 3780-compatible to allow use of the Texas Instruments software package, but this was not considered progressive enough. Further, software emulation on minicomputers consumes a large percentage of processing time.

The solution was to provide micro-based dedicated front-ends to the Texas Instruments and Digico minis, connected through 9600-baud asynchronous interfaces as local terminals. The Digico, serving as an FEP to the 1902, was specially programmed. The Texas Instruments 990 used a modified Teletype driver (largely eliminating the trapping of special characters) with a simple COBOL program to dedicate a VDU as a console and to spool incoming data to disc and transfer outgoing data from disc. No communications software was needed, only an application program. A simple mini-to-micro handshake was implemented to stop overflow of buffers, particularly if the micro is involved in transmission problems. An end-to-end protocol was defined to allow the Texas Instruments and Digico programs to request actions and send and receive error condition reports in addition to direct data transfer. The line protocol was completely controlled in the microprocessor systems (see figure 10.13).

The microprocessor systems use a Zilog Z80 processor, 3-KB RAM (double-buffered, 512-byte, full duplex), 2-KB EPROM for the control program, a USART for the asynchronous link to the minis, and a Zilog SIO chip for the line I/O. Full modem control, switch-selectable speeds, indicator LEDS, timers, and some special test aids are all included in a self-powered box, together with barrier diodes for modem protection. The SIO chip is a programmable super-USART which provides the synchronous HDLC full duplex line protocol employed. It checks CRC codes, controls bit stripping and stuffing, checks parity and the like, reporting errors through status bits to enable the Z80 program to control its buffers, initiate retransmission, encode error messages for the minis, and control the handshake and transmission between mini and micro. By using this system the minicomputer is able to use remote communication while running standard applications programs suited to local terminals. While the system has been implemented as a local Teletype emulator, a paper-tape reader would have been

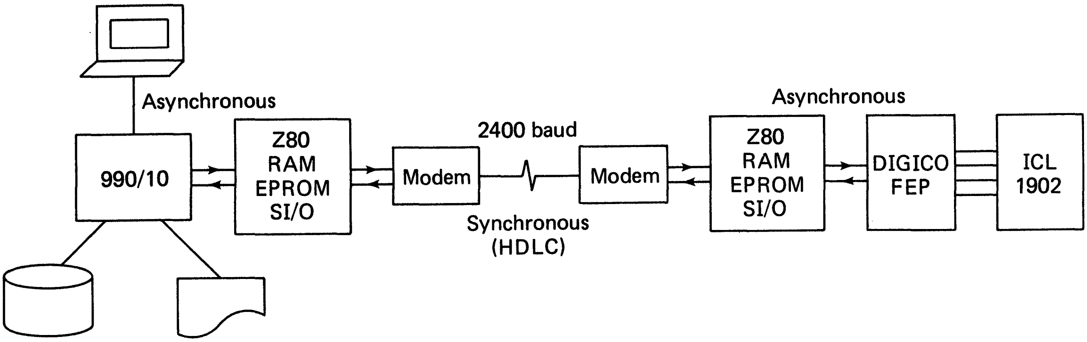


Figure 10.13
HDLC line controller, case study 4

better if available. This is because the standard device driver would have a handshake inbuilt to control the mechanics of the paper tape handlers.

The Z80 has been programmed using a Zilog MDS in Assembler language. An ICE has been used to aid debugging. Serious problems were encountered with the SIO chips which fell far short of its specification. Only by extensive trial and error have tricks been devised to overcome the chip errors; an upgraded version of the device is now available.

Paper-Tape Punch for IBM 3790

The IBM 3790 does not normally support a paper-tape punch, but one user wished to send confirmation messages to dockside by TELEX. The solution provided was to build a microprocessor-controlled box that emulated an IBM 3284/86 matrix printer at line level. This captured "printed" data in a RAM buffer, code-converted from EBCDIC to Baudot, and punched it on 5-track paper tape. When the buffer was emptied, the ready status was returned, as with the printer. The 3791 controller to 3277 VDU and 3284 printer uses the same 1 Mbit-per-second, pulse-width modulated, 13-bit character, polled half duplex protocol as in the 3270 system.

The microprocessor was built on an Intel SKD kit using the wirewrap area as a mounting point for the rather complex interface circuitry to the 3790. It was extremely difficult to get the system to work properly due to the differences between the 3790 and the IBM's supporting literature. In the end the literature was abandoned and the protocols determined by logging the data on the coax link connecting a working 3284 printer. The programs were developed on the Intel MDS in Assembler language, with specific hardware aids to emulate the 3790. An ICE would have been of little help in this case, since the problems encountered all lay in the 3790 protocol, not inside the microprocessor.

Conclusion

The foregoing examples show how microprocessors can be employed to link together systems with noncompatible I/O. It is also interesting to see how a relatively cheap microprocessor system can be employed to reduce the heavy loading of communications software on a standard business computer system. Since there is freedom at each end of the line, the most effective protocol available can be used. The technique is being adapted to enable the mini to communicate via the micro using standard emulations such as the 3780, so that they can be coupled into existing systems. A parallel, preferably DMA, interface between the micro and mini would be more efficient than a V24 interface, since there is direct control of status bits. However, this would then be minicomputer-specific and would lose the advantage of a standard interface.

Index