JAMES F. COREY
Systems Librarian
University of Illinois Library
Urbana-Champaign

Configurations and Software: A Tutorial

This paper, together with the preceding paper by Divilbiss, is intended to provide a background for the characteristics of minicomputers and the ways in which they can be configured to tackle library problems. As part of the discussion of minicomputer features, an attempt is made to compare and contrast them with larger, general purpose computers. An acquaintance with basic computer concepts is assumed. Concepts of a more technical nature are explained as they are introduced. The first section below discusses configurations; the second section covers minicomputer software. Hardware and architectural features are covered in the paper by Divilbiss.

MINICOMPUTER CONFIGURATIONS

Minicomputers and their associated peripheral devices can be combined in a great variety of ways, but this variety can be categorized into four basic configuration types: (1) stand-alone batch, (2) stand-alone on-line, (3) front end, and (4) remote concentrator.

The first type, stand-alone batch, is the most familiar (see figure 1). In this configuration, the mini looks like and performs like larger batch computers. In most cases the mini will have a card reader and punch, a line printer, tape drives and disks. (The minis in all four configuration types will have operator consoles, but to avoid cluttering the figures, the consoles have been omitted.) The term stand-alone is part of the name of this configuration in order to stress the fact that the mini is not connected to a larger computer. The mini stands alone. The main use of the stand-alone batch configuration is the generation of printed products, e.g., serials lists, purchase orders, circulation lists and overdue notices. This configuration could also be used for

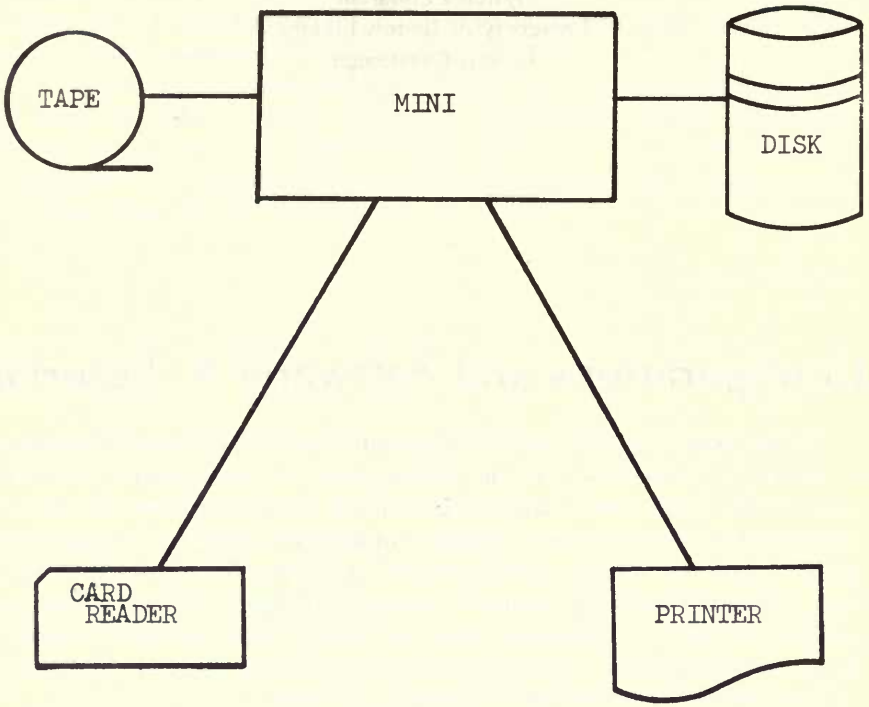

Fig. 1. Stand-Alone Batch

training librarians in programming techniques and machine operation and, in that capacity, might be of use to library schools as well as to libraries.

The second configuration, stand-alone on-line, is shown in figure 2. It has the peripheral devices used in the batch configuration but in addition has at least one kind of terminal device. Four common kinds of terminals are shown in the figure; cathode ray tubes and typewriters are two kinds shown attached to the mini. Both have keyboards and are used by library staff in "conversational mode" with the mini. Two kinds of circulation terminals are also shown. The one in the upper-left-hand corner of the figure represents the circulation terminal which reads Hollerith punched patron badges and book cards. The other terminal possesses the latest device, the "wand" reader, that can read coded strips about the size of a spine label. The strips may be magnetically coded or optically coded. The latter code is represented by a series of black and white stripes of varying thickness which can be interpreted by a light sensitive wand. An example of optically coded labels may be found

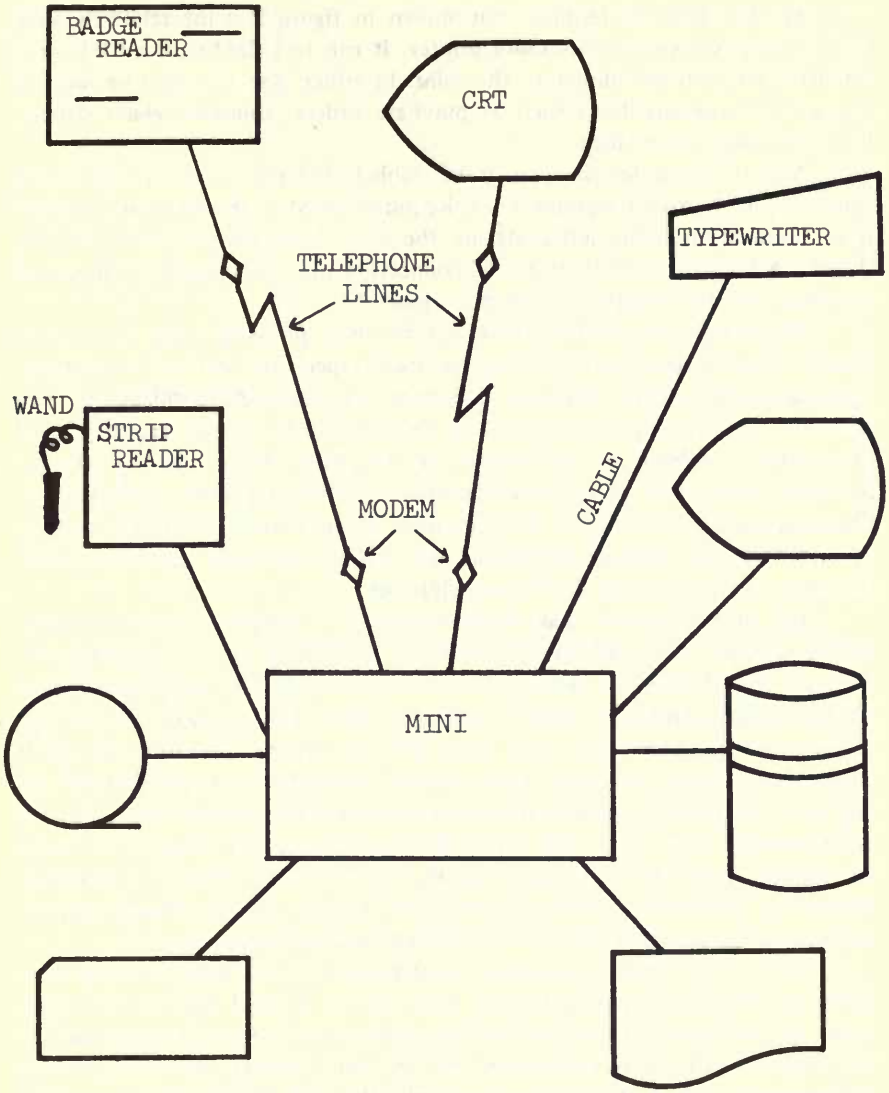

Fig. 2. Stand-Alone On-Line

on the side of an ordinary package of JELL-O. Whether optically or magnetically coded, the function of the labels is the same: they replace Hollerith punched book cards and badges. Their main use to date has been in automated circulation systems.

Another kind of terminal not shown in figure 2 is the relatively slow (30-120 characters/second) remote printer. It can be paired with a CRT, or it can have its own line direct to the mini. In either case it would be used to produce on demand items such as purchase orders, vouchers, claim notices, book cards and spine labels.

Any of these terminals may be cabled directly to the computer or connected to it over telephone lines. Required speed of transmission, distance from terminal to mini, and costs are the three factors which determine the choice. A mixture of both types of connection may be present on the same machine, and this situation is shown in figure 2.

The stand-alone on-line mini can be used to assist with almost any library recordkeeping job. It can be used especially well in acquisitions, circulation, and serials check-in. However, in cataloging applications, two problems are encountered: character set and disk storage capacity. The character-set problem is not unique to the mini. It is really a terminal problem, and as such affects computer-assisted cataloging on any size machine. Currently, it is difficult to find terminal manufacturers who will provide character-sets that include diacriticals and special alphabetic characters found in foreign languages using the Roman alphabet.

The other problem, disk storage capacity, is unique to minicomputers. As the number of MARC records produced at the Library of Congress grows, so does the problem of storing them, plus the library's own original cataloging, on-line. Thus far, larger computers have kept up with the storage problem, while minis have not been able to do so. Storage capacity is expanding, however, as larger disks become available for minis. But for the moment, the ability of the mini to store large bibliographic files on-line is more promise than fact, and minicomputers installed in libraries are more successfully used for acquisitions, serials and circulation than for cataloging. Papers by Beaumont, Grosch, Brudvig and Lourey in this volume describe some actual uses of the mini in a stand-alone on-line configuration.

Both of the above stand-alone configuration types apply to large computers as well as to minis. But the point to be stressed here is that minis today are not harshly restrictive in the number and types of peripherals they can support. The only restriction lies in disk capacity and even that is disappearing. This is quite a contrast with the first minis which supported only an operator's console and a paper tape reader/punch!

The third configuration type is called the front end configuration and is shown in figure 3. The peripheral devices attached to the mini are the same as in the stand-alone on-line configuration. The difference is the addition, in the front end configuration, of another, usually larger, computer with which the

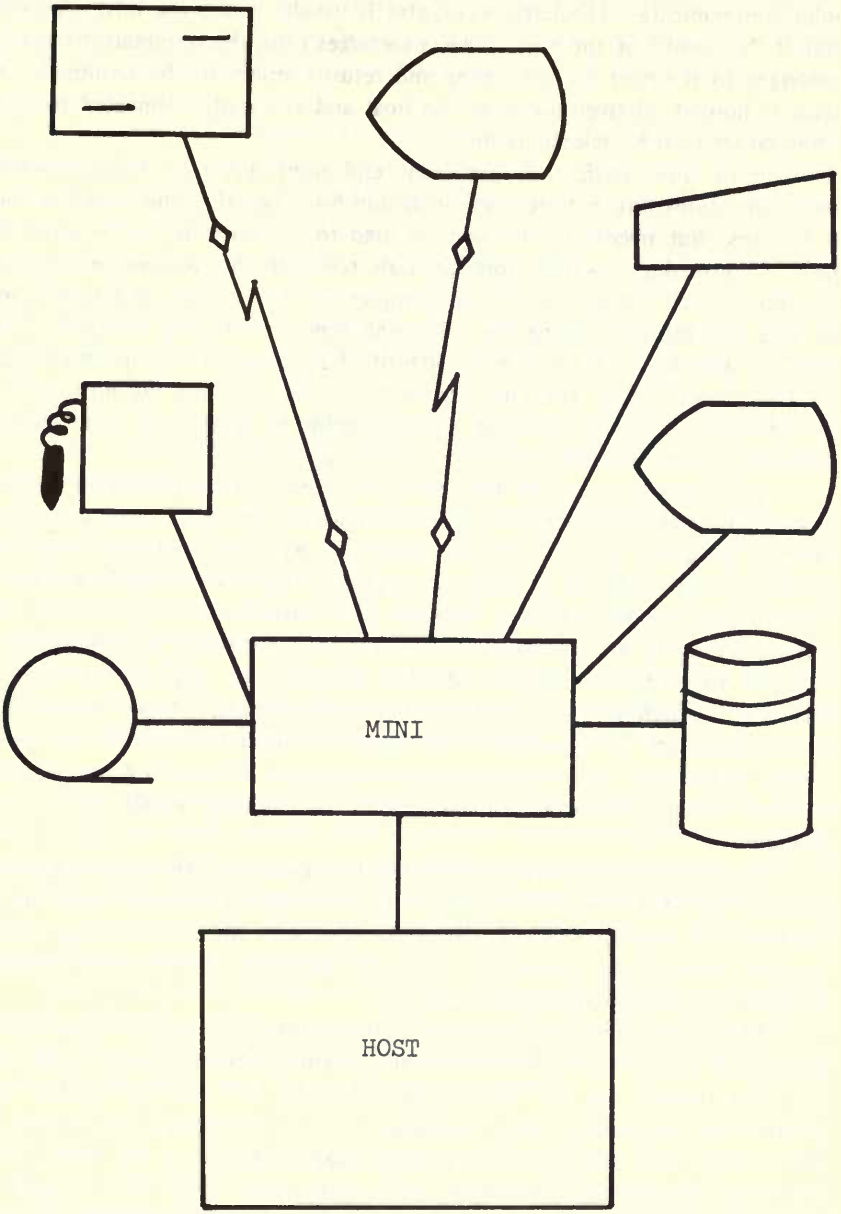

Fig. 3. Front End Configuration

mini communicates. The larger computer is usually called the host. The mini stands "in front" of the host, collects messages from the terminals, routes the messages to the host for processing and returns replies to the terminals. The mini is housed relatively close to the host and is usually connected to it by cable rather than by telephone line.

In its most basic use, the front end mini serves as a simpleminded communications link between terminals and host, i.e., the mini forwards what it receives. But invariably the mini is used to do more. It can be given the task of converting messages from disparate terminals into a common code and common format to make processing simpler for the host. It can handle errors in data transmission, asking for retransmissions and sending messages to the terminal operators when errors are identified as permanent equipment failures needing service repair. The mini can serve as back-up when the host is down by informing terminal operators and continuing to accept and store message requests for later processing when the host is again available.

Such tasks as the above are known as systems programs because they are designed to make the *computer system* as a whole perform as well as possible. Systems programs are not directly concerned with the particular work, expressed in the form of a particular message, that the terminal operator is trying to accomplish. Systems programs are contrasted with application programs. The latter are programs written to do a given job as seen from the point of view of the terminal user. For example, one terminal user may be preparing a purchase order; another may be discharging a book. For each of these transactions there would be an application program to fulfill the needs of that transaction. But a communications program written to get messages in and out without regard to the content of the message would be a system program.

There is no hard line between the two concepts. The communications program may have a small routine to check for valid transaction codes at the beginning of messages, rejecting invalid codes with an error message to the terminal rather than forwarding them to the host. The "system" program becomes just slightly application-oriented.

Front end minis are frequently programmed to execute a number of application programs in addition to basic systems programs. Small volatile files are often placed on the mini's disk, allowing the mini to handle some transactions completely. Brief circulation records containing only patron ID, call number, and due date, for instance, can be placed on the mini's disks, permitting charging and discharging to be done on the mini. Yet a request for an overdue notice would go to the host to access the bibliographic and patron data associated with the overdue. When front ends are programmed in this

way, the configuration is also termed a distributed logic system because the logic of the application is distributed across more than one computer. Papers by Davison and Payne in this volume describe cases of minis used in a front end configuration.

A variation of the front end configuration is the network configuration shown in figure 4. Instead of communicating with terminals, the mini routes messages to other computers. The key function performed by a front end mini in a network is the translation of the machine language code of its host into a common communications code. By using a single communications code and front end minis, host computers from different manufacturers can be easily interconnected on the network. The much-heralded U.S. bibliographic network where each host serves as a state or regional node in the network has, so far, not materialized. When the network does materialize, it may be configured in the manner shown in figure 4. At present, the ARPA network in the United States[1] and the SITA airline reservations network in Europe,[2] two of the world's biggest, are configured in this manner.

The fourth and last configuration type is called the remote concentrator configuration (see figure 5). It is like the front end in practically every respect. The one essential difference is the location of the mini viz-à-viz the host. In the front end configuration, the mini is fairly close to the host and is connected to it by cable. In the remote concentrator configuration, the mini is farther away from the host and is connected to it by telephone line. The configuration gets its name because it is "remote" from the host and takes messages from several terminals and "concentrates" them over a single line to the host.

Other than the factor of distance, the same comments made above about front ends apply equally well to remote concentrators. Remote concentrators contain systems programs and, usually, application programs. They can provide back-up to the host, and they may process some transactions completely. The paper by Waite in this volume describes a minicomputer configured as a remote concentrator.

From this brief overview of minicomputer configurations, it should be apparent that there are many potential ways in which minis can be put to use to assist libraries with their data processing needs. But no mini will produce results for the library until it is properly programmed.

MINICOMPUTER SOFTWARE

In the area of minicomputer software, it is hazardous to state generalizations because the situation is changing so rapidly. Manufacturers of

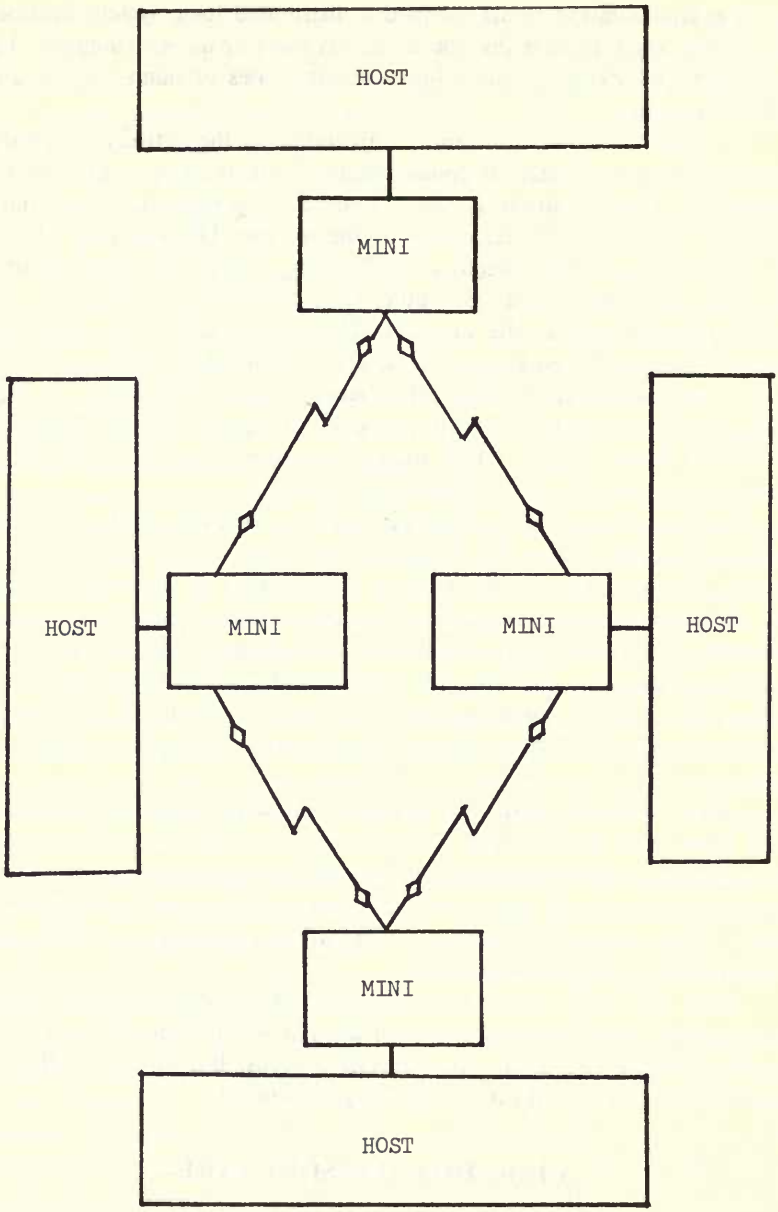

Fig. 4. Network Configuration

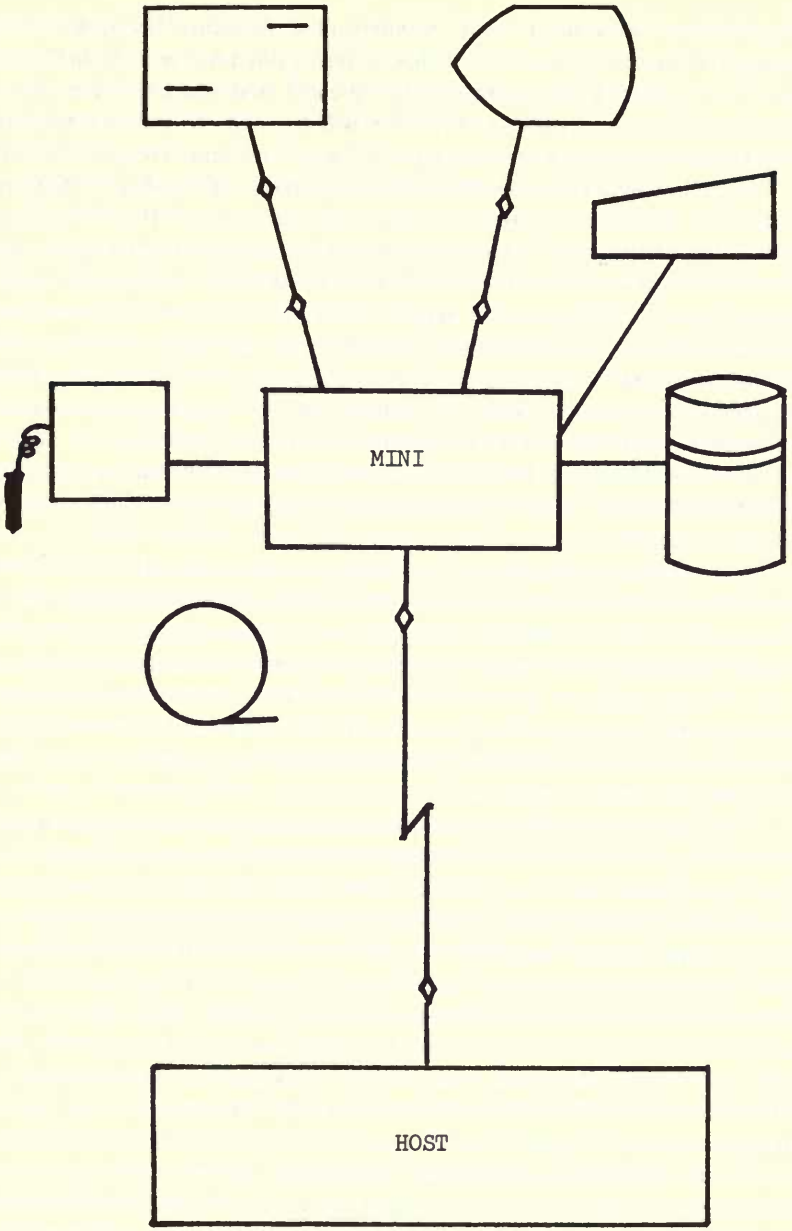

Fig. 5. Remote Concentrator Configuration

minicomputer hardware, as well as independent software firms, are steadily bringing out new programs or enhancements to old programs. Nonetheless, it might be worthwhile to summarize the present and identify trends for the future. As with configurations, an effort will be made to indicate similarities and differences between minicomputer software and large computer software.

In the computer business there is a commonly used saying: "Hardware is potentiality; software is actuality." The expression means that the hardware provides the potential to solve a problem, but well-written, thoroughly tested software is required to make the solution actual. This is understandable in light of the way computers are marketed. The hardware manufacturer builds the computer and provides some basic systems software consisting of two main components—an operating system and some number of programming languages. The customer, using the support of the operating system and the programming languages, writes programs (application programs) to solve his own set of problems. As purchased, the machine has only the potential to do useful work for the buyer. This situation is equally true for minicomputers and large computers.

The question arises: "How hard is it to achieve actuality?" The answer, expressed in terms of cost, is: "Very hard." Over the past ten years the cost of hardware has been steadily dropping while programming costs have been steadily increasing. The computer facility manager finds a greater percentage of his budget going to personnel each year. With respect to minicomputer users, a study has shown that the ratio of software costs to hardware costs is running two to one.[3] Software costs are twice the hardware costs! "Actuality" does not come cheaply.

The objective of the cost-conscious minicomputer user, then, should be to strive to get as much software support from the hardware manufacturer or other commercial source as possible. First, take the minicomputer hardware manufacturer. What software does he furnish? It can be categorically stated that no minicomputer manufacturer provides what could even remotely be called an automated system for libraries. There is no package from any manufacturer that will order library materials, receive them, pay for them, catalog them or circulate them. Library application programs must be written by the customer. Admittedly, some library packages are available from independent firms, but these firms were first of all customers of the hardware manufacturer, and, as such, had to take the software provided by the manufacturer and develop the application using what was useful of the manufacturer's systems software. Let us consider then the manufacturers' software to see what will help the developer of library applications. In explaining manufacturers' software, the progression will be from the simplest versions to the

most complex. This approach is not only easier, it also recapitulates the history of minicomputer software. The software was initially very basic, but it has been improving steadily. The information on manufacturers' software was derived from brochures and technical manuals supplied by several minicomputer manufacturers and from a survey of a number of books and articles on minicomputers, the most notable of which is one by Auerbach.[4]

The first minis had quite rudimentary input and output devices—one operator console and one paper tape reader/punch. This configuration is still available and is called the paper tape mini. The software is commensurate with the rudimentary hardware. It consists of four programs. The first program is a bootstrap loader which has to be laboriously dialed into memory by setting switches and pushing buttons on the front panel of the mini. The bootstrap loader does one thing: it loads the second program—the real loader. The loader in turn loads the third program—the assembler. The assembler and loader are on paper tape, so merely getting the programs into memory is time consuming. At ten characters/second, the standard speed of some paper tape readers, a 4,000 character assembler takes over six minutes just to load. The assembler reads and translates an application program and punches it on paper tape. The loader then loads the application program and, if the latter fails, the fourth vendor-supplied program—the debugger—is loaded. The debugger can write to the operator console, printing values from the collapsed program to assist the programmer in diagnosing the problem.

The assembler furnished in the vendor package is itself quite basic. It cannot accept macros or external names and it does not produce relocatable code. Space does not allow an accurate explanation of macros, external names and relocatable code, but a rough explanation is possible.

A macro is a single written instruction that translates into several machine instructions. Without macros the programmer has to write down one instruction for every instruction the machine is to execute. With macros, any group of instructions that is frequently executed together can be symbolized by a shorthand notation—a macro. Whenever the programmer wants that group of instructions, he can write the macro and the assembler will expand it into its corresponding group of instructions.

External names are names not defined in the program. They allow programs to be broken into separately written modules which have the ability to communicate with each other. One module can refer by name to an item of data in another module as long as the assembler knows the data item is external. Without support for external names, program modules could not talk to one another, thus defeating most of the purpose of modular programming.

Relocatable code is code that can be placed anywhere in the machine and still run. It is contrasted with absolute code which will run at only one place in computer memory. This one place must be known in advance to the assembler. A control card is furnished to the assembler, stating, for example, that the program is to begin at main memory location 2370. Absolute code, besides forcing programs to fit in certain places, makes modular programming difficult. It is seldom possible to know in advance how big modules will be. If modules A, B, and C are eventually to come together to make one program, module B must fit after module A. Since the final size of A cannot be known in advance, the starting location of module B cannot be specified.

Modular programming permits division of labor because several programmers can work on separate modules of a total program. Modular programming also permits frequently used subroutines to be saved and combined into new programs. External names and relocatable code, by supporting modular programming, and macros, by supporting code efficiency, all help to increase programmer productivity. The basic assembler, lacking these features, causes longer development time and higher development costs.

A mini with only a console and paper tape for input/output is of no value to libraries because the input and output is too limited. A minimal increase producing a useful machine is the addition of communications lines and terminals. The mini can then be attached to a host to become a front end or remote concentrator. However, unless the software is improved, program preparation will still be difficult, hence, costly.

The next step up in vendor-supplied software is the improvement of program preparation software, a phrase that refers to the subset of programs supplied by the vendor which are used most often in the preparation of new programs. Besides the assembler, program preparation software includes other language translators, a program called a linkage editor and a set of programs known as utilities. The first improvements are usually made to the assembler. The vendor makes the assembler capable of accepting macros, external names and generating relocatable code. Relocatable code, as mentioned, allows programs to be written in modules; but the existence of modules introduces a new problem. To function as a total program the modules must be "linked" together by the linkage editor, the program which unites modules. It is run after the assembler is finished. It uses external names to link referenced items with referencing instructions, and it relocates relocatable code to make all modules contiguous.

Another assist for the programmer is a set of utilities. Utilities are off-line programs best explained by examples of their functions:

1. moving data from cards to tape or vice versa,
2. moving data from cards to disk or vice versa,
3. moving data from tape to disk or vice versa,
4. moving data from cards, tape or disk to the printer,
5. combining two files of data into one,
6. splitting one file into two,
7. inserting new data into the middle of an existing file, or
8. deleting selected data from a file.

If the data are programs or a string of program instructions, the utilities can be seen as instruments to create and modify programs. Utilities are also used to maintain a program library. A program library is a set of individual files (programs) with an index which tells the location and size of each program or program module. Program libraries are accessed by the linkage editor to retrieve modules for linking into programs. Program libraries are also used to store the macros accessed by the assembler to convert the single macro instruction into a set of real instructions.

The next improvement in program preparation software is usually a higher-level language. The most commonly offered is FORTRAN, followed by BASIC, both numerically oriented languages. COBOL is beginning to be available on a few machines, while PL/1 is nonexistent on minis. All higher level languages support external names and relocatable code. By their very nature, they do not need macros.

If the minicomputer is very small and its I/O capacity limited, e.g., a front end mini with no tape or disk, the program preparation programs are very slow and cumbersome to run on the mini. The higher level language translators may even require more main memory than the mini has available. To alleviate this problem, vendors often supply language translators, linkage editors and utilities that run on large, general purpose machines, but which, nevertheless, produce programs that run on the mini. This concept is often difficult to understand. The programs are written in the mini's language. They are translated on a different machine, but the output of the translation process is machine code for the mini that will run only on the mini. The machine code will not run on the translating machine. It has to be physically transported to the mini for execution. The machine code could be punched into cards, and read into the mini through its card reader. Another technique is used at the University of Illinois where an IBM System/7 is connected by cable to an IBM 370/158. The System/7's programs are assembled and link edited on the 158 and then sent by a utility over the cable to the System/7. Program preparation on a different machine is called cross-assembly.[5]

Cross-assembly is not limited to machines from the same vendor. Since mos mini vendors do not make large computers, their cross-assemblers necessarily run on computers of other vendors.

The next hardware expansion applied to a mini is the addition of tapes disk, card readers and line printer. These I/O devices greatly increase th mini's power. The program preparation software no longer needs to be run o another machine. The language translators, program libraries, linkage edito and utilities can be run on the mini. Additional software can also be provided

At this stage, an input/output control system appears. The two mos important sets of routines in the IOCS are the peripheral drivers and th interrupt handlers. Both sets of routines do I/O at the physical level, i.e., they are concerned with status bits that are set on and off by the hardware a electrical signals are sent back and forth between the CPU and its peripherals Bits can be set to indicate a variety of conditions: device ready, device busy temporary error, permanent error, request completed, etc. Peripheral driver are responsible for initiating I/O operations. When the operation successfully starts, they are finished. There is one routine for each type of device becaus the meaning of the bit settings can vary from device to device. The routine are "device dependent." The interrupt handlers are responsible for processin an I/O operation when it is complete. The term interrupt handlers is use because most computers are designed so that an electrical signal is sent to th CPU by the peripheral on completion of the I/O event. The signal interrupt the CPU to call attention to the completed event. Like peripheral drivers interrupt handlers are device dependent.

The value of an IOCS is in its assistance to programmers. Programmer can then do I/O on the logical level instead of the physical level. Instead o testing bits, the programmer writes a statement which says, in effect: "Here i a record. Write it to tape drive number three, and tell me when you ar done."

With IOCS, programmers need never know which bit settings mea what—except in one circumstance. Vendors are very good about providin peripheral drivers and interrupt handlers for their own peripherals as it make the peripherals easy to use. However, they usually make it a policy not t provide such routines for peripherals of competitive vendors. When a use wants to connect company X's terminal to company Y's mini, extra program ming for the peripheral driver and interrupt handler must be done, adding t costs and development time.

Another systems software feature is known as a console comman language. It is a set of commands that the operator can type in on the consol to cause programs already in the main memory of the mini, or in

predesignated program library, to be executed. This saves the programmer from having to put batches of cards in a card reader every time he wants to run a standard program. Translators and utilities are some of the programs callable from the console via the command language. Another program callable may be a text editor, a program which will retrieve lines of text from a file, allow the operator to correct the text by inserting or deleting characters or whole lines, and return the corrected lines to the file. A text editor is, in effect, an on-line version of one of the utilities. Since programs can be treated as lines of text, the existence of a text editor increases programmer productivity by allowing for faster program correction.

A better debugging program is usually furnished for minis that have line printers. Extensive information about the state of the machine at the time of failure can be printed very quickly. This feature is very important both for new programs in the testing stage and for operational ones which encounter an unforeseen combination of circumstances causing a malfunction.

The software features that have been described to this point are summarized in table 1. Two minis that have most of these features are the IBM System/7 and the Datapoint. Minis with these capabilities can do a great deal of valuable work. They can perform in any of the four configurations described earlier. Their main limitation is that they cannot run more than one program simultaneously, i.e., they do not permit multiprogramming. For example, a mini with the software described thus far could not run a circulation system on-line and concurrently test a new book fund accounting program. The circulation program would have to be stopped first and removed from the machine. Testing becomes inconvenient for the programmer who wants to develop a second application. Testing would begin when the library closed.

To alleviate this limitation, many mini vendors offer some form of multiprogramming operating system. Most frequently offered is the ability to run two programs simultaneously. A few vendors support three simultaneous programs, but few go beyond three. In contrast, large machine operating systems may support fifty. When the mini vendor supports two programs, the terminology used is foreground-background. The foreground is given first priority by the operating system. The background takes over only when the foreground is waiting. Returning to the above example, the operational on-line circulation system could run in the foreground while the book fund accounting program under development could be run in the background.

Allowing multiprogramming requires a much more sophisticated operating system. The basic problem is this: both programs want the same resources —the CPU, all of main memory and all of the peripherals. The operating

Paper Tape Software
- Bootstrap loader
- Loader
- Assembler (no macros, no external names, absolute code)
- Debugger

Program Preparation Software (runs on host or mini with disk)
- Assembler (macros, external names, relocatable code)
- Linkage editor
- Program library
- FORTRAN
- BASIC
- COBOL

Disk-Oriented Software
- Input/Output Control System (IOCS)
- Console command language
- Text editor
- Better debugger

Table 1. Vendor Supplied Software

system has to arbitrate the application programs' demands for these resources. The solution in the case of the CPU has already been described. The program entered into the machine as the foreground program gets priority on the CPU. Only when the foreground releases the CPU does the background get it.

Main memory management must fulfill two functions: it must allocate storage and it must protect allocated storage from aggrandizement by over-zealous programs. Storage allocation algorithms vary from machine to machine. Some are quite dynamic, expanding and contracting program space as the program runs. Others are static, giving the program one fixed piece of memory for the duration of the run. Reasons for the diffferent approaches are beyond the scope of this paper, but one of the most sophisticated, called virtual memory, is explained by Divilbiss in the preceding paper. Storage protection is a combination of hardware and software which defends allocated storage by canceling the offending program and booting it from the machine. The reason for cancellation is printed in a diagnostic report; cancellation is usually because of errors in programming that cause a program to try to store data outside its allotted area. Storage protection routines must keep any application program from reaching beyond its bounds in order to prevent damage to other application programs or even to the operating system.

Probably the most complex arbitration work required of the multi-programming system is the allocation of devices. Most programs want to read cards, print messages and write on tapes and disks. Running the purchase

order program and the overdue notices program simultaneously without device allocation is likely to result in the patron's overdue notice going to a bookdealer and the book order going to the patron. Methods for dealing with device allocation vary, but usually each program is required to state prior to execution what devices it needs. If all devices are available, they are assigned to the program and it proceeds. If not all devices are available, the program is either held up or it proceeds using temporary substitute devices, with the data being moved to the correct device later by the operating system.

Many mini vendors offer operating systems with multiprogramming. Digital Equipment Corporation, Hewlett Packard, Varian, and Data General, to name a few, have advanced their software to this stage. And these operating systems are continually being improved. Mini operating systems, formerly primitive, are now approaching the level of complexity of operating systems available on medium-sized, general-purpose computers.

In summary, minis have made dramatic improvements in the past ten years, both in price reduction and in hardware and software capabilities. For library work minis are still weak in several areas. They are weak in the higher level languages best suited to variable length strings of alphabetic characters. PL/1 is not supported. Character-string features in COBOL are not supported. Minis are still somewhat limited in their support of large capacity disks, and they are weak in the support of indexes to large data files. But if the previous ten years is any guide, limitations will disappear. Front end and remote concentrator minis are already working in several libraries. Stand-alone minis are also doing specialized library jobs such as book fund accounting and circulation. In the future, stand-alone minis may be dedicated to libraries, assisting them in all facets of library bibliographic control.

REFERENCES

1. Sher, Michael S. "A Case Study in Networking," *Datamation*, 20:56-59, March 1974.

2. Hirsch, Phil. "SITA: Rating a Packet-Switched Network," *Datamation*, 20:60-63, March 1974.

3. Ross, D. T. "Software Development for Minicomputers." In *Minicomputers: International Computer State of the Art Report* (Infotech State of the Art Report, no. 13). Maidenhead, Berkshire, England, Infotech Information Ltd., 1973, pp. 203-26.

4. Auerbach, Inc. *Auerbach on Minicomputers*. New York, Petrocelli, 1974.

5. Lamb, Vincent S. "All About Cross-Assemblers," *Datamation*, 19:77-80, July 1973.