DOCUMENT RESUME

ED 124 170 IR 003 560

AUTHOR Clarkson, Marcia Shonnard
TITLE Maximizing the Mini for Administratice Computing.
PUB DATE May 76
NOTE 17p.; Paper presented at the College and University Machine Records Conference (Cincinnati, Ohio, May 17-19, 1976)

EDRS PRICE MF-$0.83 HC-$1.67 Plus Postage.
DESCRIPTORS Ancillary Services; *Computer Oriented Programs; Costs; *Data Processing; Educational Facilities; Electronic Equipment; Higher Education; Information Systems; *Institutional Administration; On Line Systems; Time Sharing
IDENTIFIERS BASIC; Computer Hardware; CUMREC 76; *Minicomputers

ABSTRACT

A medium sized time-sharing minicomputer performs both administrative and academic computing for the University of the South. Some advantages of the system include the opportunity for student access to the terminals, and the creation of new student jobs involving computer activities. Also, the minicomputer system hardware is less expensive to run than a combined center. The main advantages of the system accrue to academic users because of the use of BASIC. Problems with the system center on administrative applications, difficulties using BASIC in commercial applications, the lack of a file management system, lack of an efficient sorting system, and contention for system time. (CH)

21st Annual College and University Machine Records Conference

"Maximizing the Mini for Administrative Computing"

Marcia Shonnard Clarkson
Director of Data Processing
The University of the South

CUMREC '76

University of Cincinnati / Miami University May 17,18,19 1976

Southwestern Ohio Regional Computer Center

Maximizing the Mini for Administrative Computing

Marcia Shonnard Clarkson
Director of Data Processing
The University of the South

A stand alone minicomputer is being used for both academic and administrative computing at The University of the South. I would like to explain to you some of our reasons for purchasing this system, discuss some of the difficulties we faced in developing administrative applications, describe some of the techniques we used to overcome the difficulties, and finally suggest that we might all learn from some of our academic colleagues who have reduced computer costs by developing software for less sophisticated systems like the mini and then sharing their work with each other.

So that you will have some idea of the extent of data processing possibilities involved, I will describe the situation to you. The University of the South is an educational institution occupying ten thousand acres on top of the Cumberland Plateau in south central Tennessee. The school which has three major divisions--the College of Arts and Sciences with one thousand students, the School of Theology with seventy-five students, and the Sewanee Academy, a secondary school with two-hundred students--is owned and supported by the Episcopal Church. Although the University is similar to other liberal arts colleges in the administration of academic affairs, our situation is complicated because the University's domain includes Sewanee, Tennessee, a town of two-thousand. The University operates a hospital, an inn, a drug and supply store, a grocery store, a police department, a fire department, and a substantial crew to maintain the land. Approximately five-hundred regular employees are required to provide these services.

In the fall of 1972 the University hired a consultant to investigate replacing unit record equipment which was installed in the Treasurer's Office with a computer which could serve all administrative users. At this time students in the College of Arts and Sciences were using an overloaded single-terminal mini for an expanding academic computing program. The College also needed to increase its computing power and felt that only a multi-terminal interactive system would satisfy its needs. The result was the purchase of a medium sized time-sharing minicomputer to do <u>both</u> administrative and academic computing.

Our present equipment as shown in figure one includes a 32K main processor, an 8K communications processor, a 22 million byte (IBM 2314 type) disk drive, a teletype 35 console, a 45 ips tape drive, a high speed paper tape reader, a 200 line per minute printer, four administrative CRT's, one administrative teletype, one administrative 30 cps upper and lower case printing terminal, a teletype used by students at the Sewanee Academy, and four teletypes, a CRT, and a 30 cps printing terminal for academic computing at the College.

When the computer was purchased, the University hired a Director of Data Processing and a half time Director of Academic Computing. As originally designed the Data Processing Department would provide systems design and programming assistance to all administrative users and would be responsible for co-ordinating the use of the time-shared system by both academic and administrative users. Each administrative department would provide its own data entry and program running personnel. The Data Processing staff was therefore planned to comprise only the director and a half time clerk-secretary-operator. This organization has worked fairly well providing service to the large departments of the University; but, as I will explain later, we have had to make some adjustments to our original plan.

The administrative applications now running are these: accounts receivable (a more involved application than at most universities our size because of our auxiliary enterprises), accounts payable, payroll, ledgers (including general, operating, restricted, plant and endowment), securities, budget preparations, financial aid, personnel, student master, and admissions. We hope to expand the student records system to keep course data, and hope to add a hospital accounts receivable system and a system for the development office which will include both gift records and our data base of potential givers.

The main reason for purchasing this system was cost. The average monthly rental on commercial systems that were proposed to meet our administrative needs was over $3,000 a month, and the cost of upgrading the academic system then installed was estimated at $55,000. We purchased the combined system for approximately $100,000 and pay $2,144 a month on a five year loan. Our maintenance was $608 a month the first year and is now $789 a month. Our monthly hardware costs for <u>both</u> academic and administrative computing are approximately the monthly rental on the commercial systems that were proposed, and we will own the system in three more years.

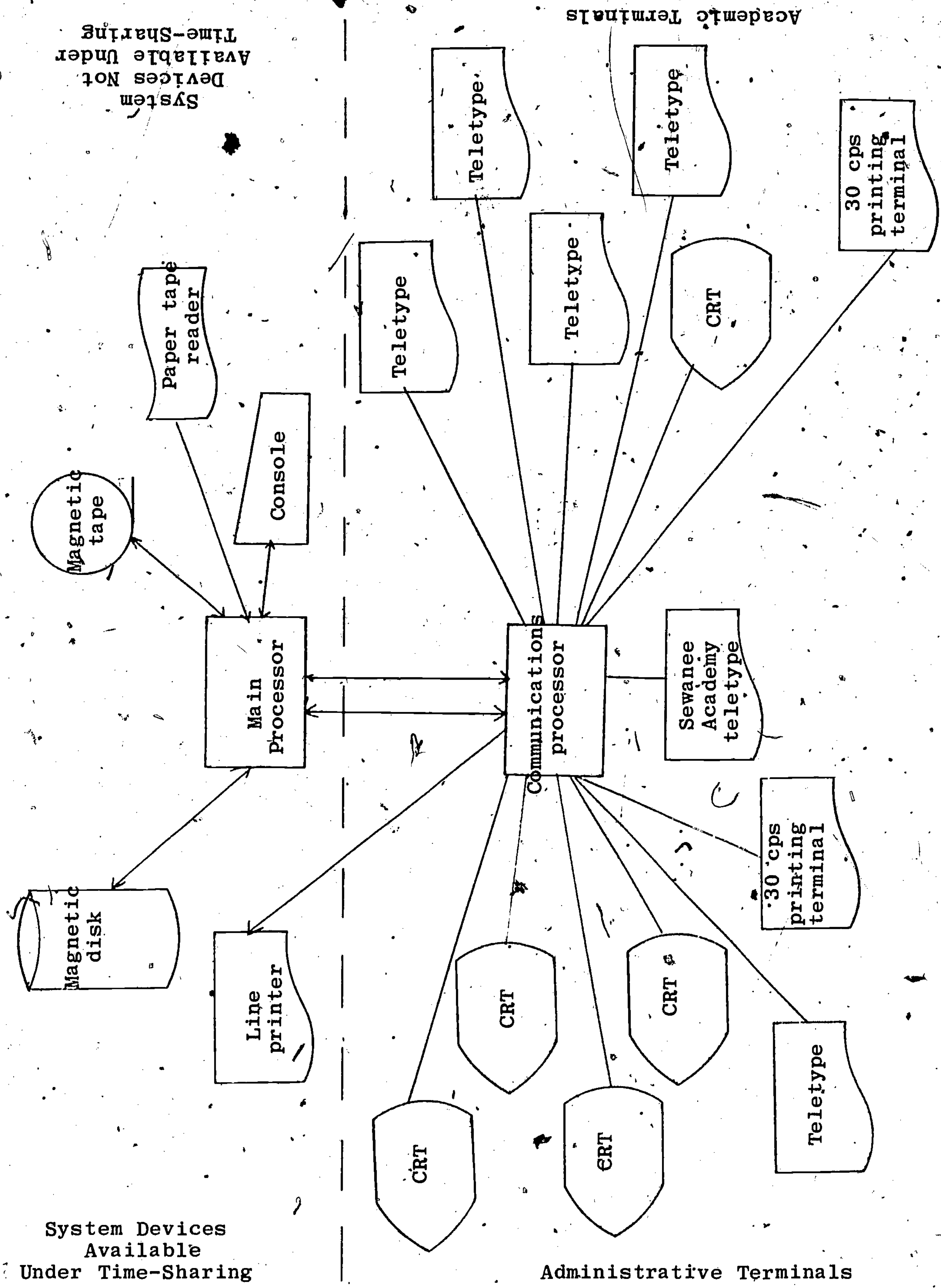
System Devices Not Available Under Time-Sharing
Academic Terminals
Teletype
Teletype
Teletype
Teletype
30 cps printing terminal
CRT
Paper tape reader
Console
Magnetic tape
Main Processor
Communications processor
Sewanee Academy teletype
30 cps printing terminal
Magnetic disk
Line printer
CRT
CRT
CRT
CRT
Teletype
System Devices Available Under Time-Sharing
Administrative Terminals

Figure 1

There are other financial advantages. Academic computing's printing terminal has been the back-up for the line printer for producing payroll checks and other critical reports during an emergency or during periods such as the end of a fiscal year when administrative computing has an extremely heavy printing load. Likewise, college students have access to three administrative terminals after five o'clock and on weekends. Because these are among the heaviest periods of student use, we have been able to serve more student and faculty users with fewer strictly academic terminals. Probably the most significant advantage to me in developing administrative applications has been the availability of work study students at a cost to the University of 40¢ an hour (the federal government contributes 80% of the $2.00 per hour salary of students who qualify for the college work study program). We can select our best computer science students who are on this type of financial aid; and, because of their previous experience with the computer and the administrative programming language, quickly turn them into productive programmers. Students have written the financial aid system, much of the student master system, a sick leave vacation system, and a new file management system for the accounting files. If we estimate the normal salary for a beginning programmer at $3.50 an hour, we saved over $4,000 in programming costs during our first two years of operation by using work study students. Having two professionals (that is the director of academic computing and myself) who are familiar with the equipment and daily use the same operating system and programming language has also been a great help. The Director of Academic Computing backs me up when I am on vacation, and I teach a section of computer science for him. Many aspects other than the cost of hardware make it less expensive to run a combined center.

The main advantage of our system in developing administrative applications is that it is an on-line system. For example, the Treasurer's Office has four terminals on which clerical personnel who are familiar with the data enter it into the system. Preliminary editing is performed as the data is entered. This editing has considerably reduced keying errors. Also, much information that used to be keyed can now be pulled from the on-line master files. Because there is less information to be keyed, and because errors are caught as they are made, the data entry function in the Treasurer's Office has been reduced from a full time job on the keypunch to a half time job on the new equipment. Furthermore, since we have six administrative terminals online, we have considerable flexibility in file inquiry and scheduling jobs.

Although there are advantages to a time-sharing system in administrative application development, the main advantages of this type of system are to the academic users. The BASIC language is easy for students and faculty to learn. An interactive system that gives syntax errors as a program is entered is perfect for teaching programming. And, academic users are much more comfortable with interpreting than administrative users. Academic users tend to use the same programs fewer times and are much more likely to make program changes at execution time. Furthermore, many CAI (computer assisted instruction) packages are available in BASIC, and our system has a very nice Coursewriter-type package that can be used to develop our own courses. Probably the greatest advantage, though, to academic users is that there are many programs available for the system which are circulated by users' groups and professional organizations.

There are of course difficulties in implementing administrative applications on a mini--difficulties that have prevented their wide spread use. I would like to spend most of my time discussing these problems, relating them to applications that we have installed, and telling how we or the minicomputer vendors are solving them.

One of the main hardware limitations to the sort of application I am describing is that many minis have six digit accuracy and no decimal arithmetic. Can you imagine telling the Treasurer that the new computer rounds figures greater than $9,999.99, and cannot store 10¢ accurately (because .1 has no binary equivalent)? In a payroll check reconciliation program that we were recently testing, we kept coming up with the message "AMOUNTS DON'T MATCH" when comparing the calculated amount that was stored on the check reconciliation file with the amount that was entered from the cancelled check. The error occurred very infrequently and always involved amounts ending in .31 or .61. It was a rounding problem. The calculated number from the check reconciliation file could have been something like $75.31001 and the number from the check could have been $74.3099. The error was eliminated by multiplying each number by one-hundred and by then comparing the integral values. In general, there are two main ways of dealing with these arithmetic limitations. When a number is over six digits it can be stored as multiple whole numbers and then hardware arithmetic can be performed on each number separately. Although this method avoids the problem that some decimal numbers do not have a binary equivalent by expressing all numbers as whole numbers, the user must keep track of where the decimal point should be; and, although it allows accurate hardware arithmetic to be performed on multiple split numbers, the user must somehow combine the split numbers into an alphanumeric field for printing. The method that we use (it was the simplest to get started with, but is probably not the most efficient), is to store numbers that require over six digits of accuracy as strings (alphanumeric characters), and then to use subroutines which were available in our vendor's contributed library to perform calculations on these strings. The subroutines are extremely slow, but they do simulate decimal arithmetic that is accurate to seventy-two places. Let me give you an example of throughput degradation. Our accounts receivable aged trial balance branches to a one-hundred seventy-five statement BASIC adding routine as many as six times for each line that we print. The line printer prints at about twenty-five lines per minute instead of its rated speed of two-hundred lines per minute. These hardware limitations certainly do not prevent us from using the system for administrative applications, but they increase implementation time and degrade throughput. Many minicomputer vendors are announcing hardware string arithmetic which will greatly improve the use of minis in financial applications.

Traditionally minicomputer vendors have manufactured and marketed only central processors. As the market for minis has expanded, vendors have been forced to offer peripheral equipment for the processors. Our vendor sold us a disk unit and printer that he maintains but did not manufacture, and we have had much more down time on these units than on the equipment that was manufactured by the vendor. Furthermore, the disk unit is an IBM 2314 plug compatible drive without the control unit that would be required to attach it to a larger system. The hardware diagnostics are left out of the inexpensive interface that our system uses to replace the control unit. It is difficult to estimate the effect this unsophisticated interface has on service. I was shocked the first time the customer engineer came to work on the disk. He took down the whole system and started loading paper tape diagnostics. I think many times we would be operational more quickly after a disk failure if the hardware gave the service person more data on the cause of the error; but, a good engineer can make up for the limitations of the equipment. Not only are the peripherals that can be attached less sophisticated than when attached to a general purpose computer, but also, many peripherals do not have vendor supported interfaces or drivers available for the mini. When running a payroll application, a user cannot afford to risk using unsupported equipment.

Although the hardware limitations have troubled the users who have tried to install commercial applications on a mini, the lack of software is probably the main reason why minis are not used more widely in administrative applications. Our system has only one programming language available under time-sharing, BASIC. Although this language is easy for engineers and scientists to learn and use, it is more difficult than RPG or COBOL for business people to learn and use. The literature and programmed instruction courses in BASIC use examples familiar to mathematicians, like the quadratic formula and arithmetic progressions, instead of examples from commercial applications. As a result, clerical people that I have tried to interest in programming have been scared away before they even got to the details of the language. Furthermore, the language itself was designed to solve numerical algorithms, but not to easily store, massage, and print large amounts of data which is of course the first requirement of a commercial language. Extended BASIC, which we are using, has enhanced the original language so that it can format a report and manipulate alphanumeric data. But as I will demonstrate later, since we store numbers as strings, we have to write statements to perform tasks the programming language would usually take care of, like aligning the decimal point in a column of numeric data or placing the minus sign for a credit to the right of the number. This additional programming is not difficult once the routine is written, but adding it before each number is printed does increase programming time and execution time. BASIC also does not take advantage of some of the features of our hardware. We cannot access the line printer's carriage tape under time-sharing which slows down throughput when writing statements, checks or other reports that involve a lot of skipping. Another limitation is that our BASIC can store only twenty-six alphanumeric fields. Since numbers are stored alphanumerically to insure the proper accuracy, programs sometimes become very complicated. More effort is spent switching and saving string variables than developing program logic. Furthermore, since numeric variables can only be the letters A through Z, or a single letter followed by a single digit, and alphanumeric variables can only be the letters A-Z followed by a dollar sign, it is very difficult to read a BASIC program once it is written.

Let's look at an example of a BASIC program. The program in figure two searches the open invoice file for vendors with a net debit balance. It must total the invoices for each vendor to determine if the balance is negative. When it finds a debit balance it pulls the vendor's name off the vendor master, and prints a line with vendor number, vendor name and amount. At the end of the listing

PAY23

```
10   Q2=TIM(0)
30   Q3=TIM(1)
50   Q4=TIM(2)
70   ENTER #Q0
90   REM ** This program lists the vendors with net debit balances
110   FILES VEND,OPINV,ACTNG
130   DIM A$[5],B$[40],V$[25],D[6],C$[5],E$[10],F$[10],G$[10],H$[12],I$[12],J$[5]
150  G$="0"
170  H$="0"
190  P=1
200  D=2
210  READ #1,1;R1,R2
230  IF  END #2 THEN 1450
250  GOSUB 1310
270  READ #2;V$,C$,V$,V$,V$,V$,V$,E$,F$
290  J$=C$
310  IF F$#"0.00" THEN 370
330  Z$=E$
350  GOTO 450
370  Z9=2
390  Y$=E$
410  Z$=F$
430  GOSUB 9000
450  Y$=G$
470  Z9=1
490  GOSUB 9000
510  G$=Z$
530  READ #2;V$,C$,V$,V$,V$,V$,V$,E$,F$
550  IF C$=J$ THEN 310
570  IF G$[1,1]#"-" THEN 610
590  GOSUB 650
610  J$=C$
620  G$="0"
630  GOTO 310
650  Z1=R1
670  Z2=2
690  Z3=INT((Z1-Z2)*.5+Z2)
710  READ #1,Z3;A$,B$
730  IF J$=A$ THEN 1050
750  IF J$>A$ THEN 810
770  Z1=Z3
790  GOTO 830
810  Z2=Z3
830  IF Z1-Z2>1 AND Z3#2 THEN 690
850  IF Z3=2 THEN 890
870  Z3=Z3-1
890  READ #1,Z3
910  IF  END #1 THEN 1030
930  FOR K=1 TO 9
950  READ #1;A$,B$,V$,V$,V$,V$
970  MAT  READ #1;D
990  IF J$=A$ THEN 1050
1010  NEXT K
1030  B$="Missing master"
```

Figure 2

PAY23

```
1050  I$="          "
1070  I$[11-LEN(G$),10]=G$
1090  PRINT  USING 1110;A$,B$,I$
1110  IMAGE 5A,5X,40A,X,10A
1130  X=X+1
1150  IF X#58 THEN 1190
1170  GOSUB 1290
1190  Z$=H$
1210  Y$=G$
1230  GOSUB 9000
1250  H$=Z$
1270  RETURN
1290  PRINT LIN(62-X)
1310  PRINT  USING 1330;P
1330  IMAGE 6x,"Accounts Payable Net Debit Balances",8x,"Page ",3d/
1350  PRINT  USING 1370
1370  IMAGE "Vend#",5x,"Vendor Name",35x,"Amount"/
1390  P=P+1
1410  X=1
1430  RETURN
1450  IF G$[1,1]#"-" THEN 1490
1470  GOSUB 650
1490  I$="            "
1510  I$[13-LEN(H$),12]=H$
1530  PRINT  USING 1550;I$
1550  IMAGE 49x,12a
1570  IF  END #3 THEN 1630
1590  READ #3;B$,B$,R1,R2,R3,R4,R5
1610  GOTO 1590
1630  PRINT #3;"pay23","opinv",Q0,Q4,TIM(1)-Q3,TIM(0)-Q2,0, END
1650  STOP
```

Figure 2 continued

there is a grand total of debit balances. If you look at the program listing, you will see that the program would be much easier to read if vendor number, vendor name, gross amount, and net amount had variable names that suggested their use a little more than C$, B$, E$ and F$. We try to keep standard variable usage within an application, but having only twenty-six available alphanumeric variables makes this almost impossible to do for a large program. As you can see gross amount and discount (E$ and F$) are expressed as strings. This is because we write checks for amounts over $9,999.99. Although I did not include the computation subroutine for string arithmetic--it is two hundred twenty-five lines long--we can get an idea of what is required in the main program to perform this type of arithmetic by looking at lines 310 through 510 of the listing in figure two. We only want to perform the lengthy subroutine if it is necessary, so line 310 checks to see if there is a discount. If there is not, line 330 sets net equal to gross. If there is a discount lines 370 through 430 assign gross and net to the proper variables for the string arithmetic routine and then branch to that subroutine. Lines 450 through 510 add gross to a vendor total (G$). Again, this is not difficult programming, but the variable manipulation slows down program production and the subroutine for addition and subtraction slows down execution. Furthermore, in this program we want the decimal points for net aligned in a column, but since all these nets are credits, we do not worry about moving the credit sign from the left of the field to the right. Lines 1050 and 1070 right align the decimal point in the vendor total (G$) by producing a new string (I$) that is exactly ten positions along with blanks at the beginning of the field and the decimal point the third position from the right. These general observations might be made from looking at the listing: 1) BASIC is certainly not self documenting 2) much of the coding is extraneous to the program logic 3) it would be difficult to go back and make changes to this program, and finally 4) BASIC would be difficult to learn for a programmer that is not mathematically oriented.

Another software limitation is that BASIC is an interpreter. We have no facility for compiling programs. An interpreter is certainly very useful in the development stage of a program, but it is much slower than a compiler at executing the program. We have a data entry program that is run about four hours each day. Each time the statement "PRINT 'ACT#'" is executed this statement must be translated to machine language. In the case of data entry the execution speed is limited by the keying speed, but in many applications this compilation phase of program execution each time a statement within a program is executed can seriously degrade throughput. Another problem with interpreting is that our external auditors have questioned the lack of audit control involved in using an interpreter for financial applications. Each time a program is run it could be modified and no record would be made of the modification. Because source programs are relatively easy to change, it is true that interpreted systems are comparatively vulnerable to computer fraud. But compiled systems are obviously not inviolable, for anyone familiar with machine language can modify object programs.

Eliminating these limitations of the programming language is another area in which the minicomputer vendors are working. There are systems similar to ours that allow BASIC programs to be compiled, and many systems offer FORTRAN, COBOL and RPG. Of course, part of the reason that our system was as inexpensive as it was is that the vendor did not have as much invested in software development.

Not only is the BASIC language difficult to use in commercial applications, but the operating systems available on minis were designed for a different type of user. A secondary school using a time-sharing system for CAI and for teaching programming needs an operating system that can be installed by the vendor and then run with very little intervention required from an operator. Although this type of system is ideal for academic use, it assumes the user does not need to control the location of information on auxiliary storage devices, that he or she very seldom needs to load information on and off the system, and that he or she can live with a fixed system of priorities for task execution. Our operating system includes the BASIC interpreter, a communications processor to control a maximum of thirty-two terminals, routines to manipulate a system library and two levels of user library, routines to backup and load all or selective parts of the system from magnetic tape, and a few other operator commands. Since we have a single disk system that is constantly merging and updating files, we could improve our throughput if we could cut down on disk seek time by controlling the location of files on the disk. Also, the tape unit is only accessible when the system is being brought up after backup. Since many monthly files are too large to keep permanently disk resident, we have to schedule very carefully which things can fit on the system together and when they should be loaded and removed. Similarly, the priorities of the system are fixed and are designed to give the best possible response to a user entering information. There are many times we would like to modify these priorities to have more control over the throughput of a particular job. Again, these limitations were designed into the system to make it easy to operate, but some are being removed by the vendors and some can be removed by understanding how the system is working. The magnetic tape unit is available under time-sharing on a later version of our system which considerably increases the possibilities of system design. And, by studying how the operating system allocates disk space, one can have some control over where information is stored. For example, within a given user area, files and programs are stored alphabetically unless they are new to the system. Therefore, by naming sort work files w1, w2, and w3, they can be forced to be adjacent, reducing disk seek time for the sort.

Probably the greatest limitation of our operating system for administrative use is the lack of a file management system. There are operating system commands to allocate space for disk files and to remove disk files, but the files are composed of 256 word fixed length sectors. There is no IOCS to handle blocking or buffering. The user contributed library has a package to create and maintain a file of unblocked records containing only alphanumeric data. The files created using this package waste space if logical records are not 256 words long, and slow processing because of the time required to transfer the unused part of the sector and because all numbers have to be stored as strings. But the files have a series of pointers for fast sequential retrieval, a chain of available sectors so that additions can be made without disturbing the sequence, and index records for fast random retrieval. Using this package for the master file and an unblocked sequential transaction file, we installed the accounts receivable system. The accounts receivable master and one month's transactions used 18,000 of our 44,000 available disk sectors and the system was unbelievably slow. Since that time we have added blocking routines to all programs that use the transaction file and developed our own file management system for the master file. The transactions are now blocked four logical records to a sector and the master file is blocked three. The new master file allows numbers that do not require more than six digits of accuracy to be stored numerically which saves both space and processing time. The file is in account number sequence and so can easily be accessed sequentially. For random retrieval we use a binary search routine. We still do not have a satisfactory method of updating the file. Since BASIC automatically writes an end of record mark after every write, we developed a system of double buffers so that a blocked file can be updated in place; but to make additions to a file, we have to add them to the end of the file and sort the added records into place. We hope to develop a more flexible file management system: possibly in conjunction with automating our development office.

The net debit balance program in figure two includes an example of random retrieval of the vendor master using BASIC. Lines 650 through 1030 binary search the vendor master to pick up vendor name for those vendors with a net debit balance. The method itself--binary searching--involves more disk seeks to locate a record than using an index, so it is not very efficient. And, in most operating systems, the programmer would only have to say "READ ON KEY" instead of writing twenty program steps. Not only is there more coding involved to access the file, but the program is certainly not file independent. If we change the structure of the file the routine would have to be rewritten. Furthermore, though I have tried to keep file structures similar so that programming from application to application will be similar, disk space is one of the most crucial constraints on our system. As a result, we have sacrificed ease of programming for efficient disk space utilization when necessary.

Another limitation of the operating system is the lack of utilities. When we first installed the system there was only a very primitive user contributed sort that took four hours to sort four hundred records with unpredictable results. But, the vendor finally realized that the system could not be used for administrative applications without an efficient sort and that we could not install on schedule if he did not provide one. The sort the vendor wrote is unsupported, but accurate and very efficient. And, some good, generalized utilities like a file list have been added to the user library.

One final technical problem is contention. Although the system services very efficiently thirty-two students entering programs or running a CAI course, two file updates cause such contention that each might take four times longer to run than if it were running alone. This contention has been one of the main reasons that we have had to modify our original decentralized organization. We thought that each department could use the system independently without any help or control by the data processing department. The contention that resulted from the registrar sorting the student master, financial aid searching the award file, and the treasurer updating the payroll, made it impossible for any office to schedule its work. Now the academic users and administrative data entry users who are limited by the speed of their I/O devices run as they please. Other jobs like sorts, merges and file updates are scheduled through my department to cut down the contention. We are using a time-shared system and forcing it to simulate a multi-programmed batch system.

Our original idea for a completely decentralized data processing organization had another flaw. We found that we could not use people or terminals efficiently in a small department with this approach. For example, the clerks in the registrar's office constantly serve students and faculty and could not come to the computer center to use a terminal, so we located a $4000 terminal in their office. We found that during some phases of the academic year the registrar's terminal was being used only four hours a week and that other offices were much more reluctant to go to the registrar's office to use a terminal than they were to come to mine. Furthermore, because they used it infrequently, the clerical personnel in the registrar's office never did understand the system well enough to be efficient at running programs. Clerks in the registrar's office still do the bulk of their data entry, concentrated at the beginning of each semester, but my office now runs programs other than data entry for most small departments.

Although I have mentioned numerous technical problems, I feel that The University of the South made the correct decision in choosing a minicomputer for both academic and administrative processing. Our student participation in computing is phenomenal. Every student uses the computer in a required freshman mathematics program, and over twenty-five percent of the students in our most definitely liberal arts college take computer science. And, at the same time we have implemented every administrative application that we have demand for. The applications not implemented are still manual because the department heads involved are not convined that computer processing is desirable. If we had chosen a strictly commercial machine we would probably have installed our business applications more quickly, but we would not have been able to make as much use of student programmers, and we probably would not have been able to support as wide an academic computing program, certainly not without greater expense.

Now that some administrative applications have been developed for the minicomputer and now that vendors are expanding their hardware and software so that it is easier to install commercial applications, I think the mini is an excellent computer for a small college or University. There are three things I would suggest keeping in mind to a school that is considering using a mini: first, be careful in selecting the vendor; second, insist on a benchmark; and third, visit an educational institution similar to yours that is using the proposed equipment.

It is very important to choose the vendor carefully because many minicomputer vendors are not used to supporting commercial applications. When running a payroll you cannot have the machine go down for forty-eight hours the day checks are to be run. Some mini vendors do not have the hardware and software service personnel to support commercial applications. Furthermore, many mini vendors are not capable of estimating your hardware requirements. For example, the proposal for our computer did not estimate any disk space for accounts receivable transactions. The vendor did not know enough about the application to know that detail transactions are required on the statement and no one then at the University knew enough about computers to know that this figure is critical in estimating disk requirements. Either the vendor must be very familiar with your applications or you must have someone available to your organization very familiar with administrative systems design and hardware evaluation.

I think a benchmark is essential. We did not know what a six digit accuracy limitation and what contention would do to our throughput because we did not see the proposed equipment running our applications. Be sure that the benchmark includes a sort of a fairly large file, various storage retrieval methods, and arithmetic to your required accuracy. Then, if you intend to do your own programming, look at the programs the vendor wrote to run your application so that you can see what kind of a programming effort you are getting involved in.

Finally, the visit to an educational institution similar to yours can alleviate all your fears or let you know right then that this is not the machine for you. If the school you visit is not running some applications that you hope to install find out why and try to get an idea of how difficult they think it would be to implement the application. These people may also be better able than the vendor to give you estimates of staffing requirements, help you develop an implementation schedule and suggest vendors of peripheral equipment like terminals.

We in higher education are not in competition with each other in the same way the manufacturing organizations are. We can learn from each other. And, we can use a less sophisticated machine like the mini because we have the capability ourselves to provide much of the software that is required. We can take away from the vendor some of the cost of software development by developing our own applications and by then sharing them with each other. Academic computing has been doing this for years, and academic administrations can do so in the future.