SOME REMARKS ON A LARGE NEW TIME-SHARING SYSTEM

Butler W. Lampson
Berkeley Computer Corporation
2398 Fourth Street
Berkeley, California 94710

September 15, 1970

During the last 18 months the Berkeley Computer Corporation (BCC) has designed and constructed a large new multi-access system, the BCC Model 500. Both hardware and software for this system have been built from scratch, and the system is now operational. This presentation describes the characteristics of the Model 500 which we now see as important to its success. These may be classed under four general headings:

- efficiency, obtained through specialized hardware
- reliability, which depends on redundancy, error-checking and good protection mechanisms
- modularity in both hardware and software
- high-level language programming

I Hardware

The Model 500 is an integrated hardware and software system designed to support a large number (up to 500) of time-sharing users; hence the name. This system consists of two central processors, several small processors, a large central (core and integrated circuit)memory, and rotating magnetic memory. The latter contains more than $500X10^6$ bytes, including approximately $12X10^6$ bytes of drum having a transfer rate of more than $5X10^6$ bytes per second. These transfer rates allow a large number of processes to be swapped in and out of core every second. For example, a small user might have 10^4 bytes of private storage to be read from drum and written back. It is possible to swap 250 such users in and out in one second.

An unusual central memory design is needed to handle these drum transfer rates without excessive interference with processors. The Model 500 central memory consists of a fast memory device and eight interleaved 1μs core modules. The fast memory interfaces the eight core modules with the rest of the system through four ports. Two of these ports are dedicated to the two CPUs; one port handles all auxiliary memory traffic; and the remaining port serves the rest of the system. The fast memory device contains fast, associative "sticky registers" which interface the four ports to the core modules. The basic function of the fast memory is to allow simultaneous access by the four ports to the contents of the core and to provide the access at better than core speed. Simultaneous access is provided by a combination of the register action and core address interleaving. This allows the CPUs to execute programs while the auxiliary memory system is swapping other programs in and out of core without interference. The fast memory will accept a request on each port every 100 nanoseconds. This is a port request bandwidth of 40 Mhz. The core modules, each with an access of two words per microsecond, provide a 16 Mhz transfer bandwidth. The difference in bandwidths is tolerable since a port request may be satisfied by a register without requiring a core access. Also, if a port request is rejected, it can be made again 100 nanoseconds later, thus enhancing the probability for a reasonably early acceptance of a request.

In addition to the two central processors, the Model 500 has three microprogrammed processors, each of which can execute more than 10^7 operations/second. These microprocessors contain read-only microprograms which implement the basic resource management functions of the system. As a result, the CPUs spend very little time executing operation system code, and can devote almost all of their energies to user programs. That this is a substantial consideration in increasing throughput can be seen from the following considerations. The auxiliary memory processor (AMC) must do all the work connected with the transfer of a page between drum and core in less than 1 ms, since the drum transfers a (6000 byte) page every 1.4 ms and some time must be left to service the disk. In this time it is necessary to

allocate core or drum space

queue the request for the drum

select an optimal request at the current drum position, bearing in mind the core requirements and state of all the processes which have requests queued at that position

clean up after the transfer is complete, check for errors and update the tables which keep track of the page's location

The AMC has enough computing power to do this work for two drums and two disks transferring simultaneously. It is the equivalent of several 6600s when performing its assigned tasks.

The character input/output processor (CHIO) has a throughput of about 15,000 characters/second. This sounds unimpressive, but these characters are coming from 30 multiplexed telephone lines, each of which is carrying data from 50-100 terminals. All the work required to buffer these characters and deliver them to the correct user programs in the system is done by the CHIO. Errors on the phone lines are detected and corrected by retransmission, echoing to full-duplex terminals is correctly assigned to the CHIO or the remote DCC, and the lines are multiplexed with an overhead of about 3% for typical system output; no CPU time is required for these functions. A third processor coordinates interrupts, schedules the CPUs and performs diagnostic and recovery operations. It is much less heavily loaded.

The central processor is designed to run compiled user programs in an interactive environment. To make this operation efficient, its instruction set has been chosen as a suitable target language for a compiler. Automatic byte and part-word addressing, bounds checking on all array references, complete argument type checking on function calls, monitoring of references to or stores into selected variables, and read-only protection for the program all minimize the amount of software checking required and make it possible to communicate with the user in source-language terms.

To sum up, the Model 500 is not an unusually fast machine when measured in instructions/second, nor does it use extremely fast logic. It is the specialization of comparatively modest amounts of hardware for particular purposes which make its overall efficiency high.

II <u>Reliability</u>

A number of techniques are used to make the Model 500 system highly reliable in spite of imperfect hardware and software. Two of these are quite conventional:

- The system can run with half its memory, one of two CPUs, one of many drums or disks

- all data and address transfers between modules carry parity bits, and all data stored in core, drum or disk memory also has parity information

Some other points may be less familiar:

- all parity errors, environmental warning conditions and other errors detected by hardware are gathered in system 'warning registers' accessible to the error recovery software

- power comes from a motor-generator, so that dependence on the utility company for constant voltage is eliminated

- operators and electromechanical equipment other than rotating memories are in a different room

- every page in the memory system carries a 48 bit code which identifies it uniquely. All access to memory requires possession of the correct code. This prevents unintentional access to or destruction of data even if the system's tables which keep track of the location of pages are damaged.

The software also contributes substantially to reliability:

- the supervisor is divided into modules protected from each other by hardware

- the integrity of all system tables and all data can be verified after an error, and all unaffected users can continue without any disturbance. This process takes about 15 seconds if it is not necessary to verify the disk, 10 minutes if it is.

- Since the 48-bit code on a page defines its position in the directory system, it is possible to retrieve all files from the disk even if the directory information is lost.

- extensive consistency checking in the supervisor and microprocessors brings hardware errors and program bugs to light before their consequences become disastrous or untraceable.

III Modularity

Of course being able to expand the system is important: more core, drum, disk, processing or communications capability may be required. Equally important in the Model 500 design is the ability to subdivide it, so that a larger number of very small users can use it with no more overhead than a few big users. For example, the system could support about 2,000 terminals doing simple text editing without any specialization of the basic operating system for this application. Furthermore, a single more demanding user could be substituted for a suitable number of these minimal users at any time.

Essential to the kind of divisibility is rather precise machinery for controlling the resources which can be expended by each user. This is done with objects in the system called resource allocations, each of which authorizes the use of a minimum and maximum amount of certain resources (CPU, drum, disk) during a specified time. Resource allocations can be combined and split according to obvious rules, and the restraints imposed by them are enforced by the scheduler. Thus a program which has CPU resources of 10% min., 15% max., can be sure that every 5 seconds he will get at least 500 ms of CPU time, and not more than 750 ms. The maximum is important to prevent the service from fluctuating wildly as the system load changes.

Modularity in software depends on adequate protection and linkage. In this system a program can be divided into sections whose interrelations can be specified with complete freedom. Several sections can share the data required for the performance of a common task, and at the same time preserve the privacy of other data. At one extreme, it is possible to write a debugging system which maintains complete control over its subject program; at the other, programs written with no thought of coexistence can be combined in the same process, and the system will ensure that there is no interference. An interesting and commercially important intermediate case is a proprietary program which is to be used as a subroutine by some other program. The privacy of both must be protected, parameters must be passed back and forth, some data files may have to be shared, and the use of the proprietary program must be properly accounted for.

IV High-level Programming

It is generally recognized that programs written in high-level languages are coded and debugged more quickly and are easier to read and maintain than assembly-language programs. On the other hand, system programs need access to all the facilities of the machine, and their efficiency is a matter of great concern. It is therefore necessary to have a language which is 'powerful' in the right sense, with constructs which generate most useful combinations of machine instructions, but without facilities which cannot be implemented efficiently. The Model 500 System Programming Language (SPL) was designed with these aims, and has proved to be quite successful in achieving them, as is evidenced by the following two statistics.

- hand examination of 2000 generated instructions revealed that no more than 10% of the instructions could be removed by hand coding a typical routine

- The BCC operating system, a highly machine-oriented program of more than 10,000 instructions, contains fewer than 100 explicit machine instructions

Important secondary design goals relating to interactive use were that SPL be simple to use from a terminal, helpful in error situations, and efficient in its use of machine resources when responding to simple requests. We also felt it was important to provide facilities for analyzing programs statically (for example, obtaining information about the size of subroutines or the places where certain variables are used) and monitoring them dynamically (for example, by being able to run a program until an arbitrary statement about the state of the world becomes true). In short, SPL is meant to provide substantial assistance in programming, debugging, and even low-level program design, for users of our terminal-oriented system.

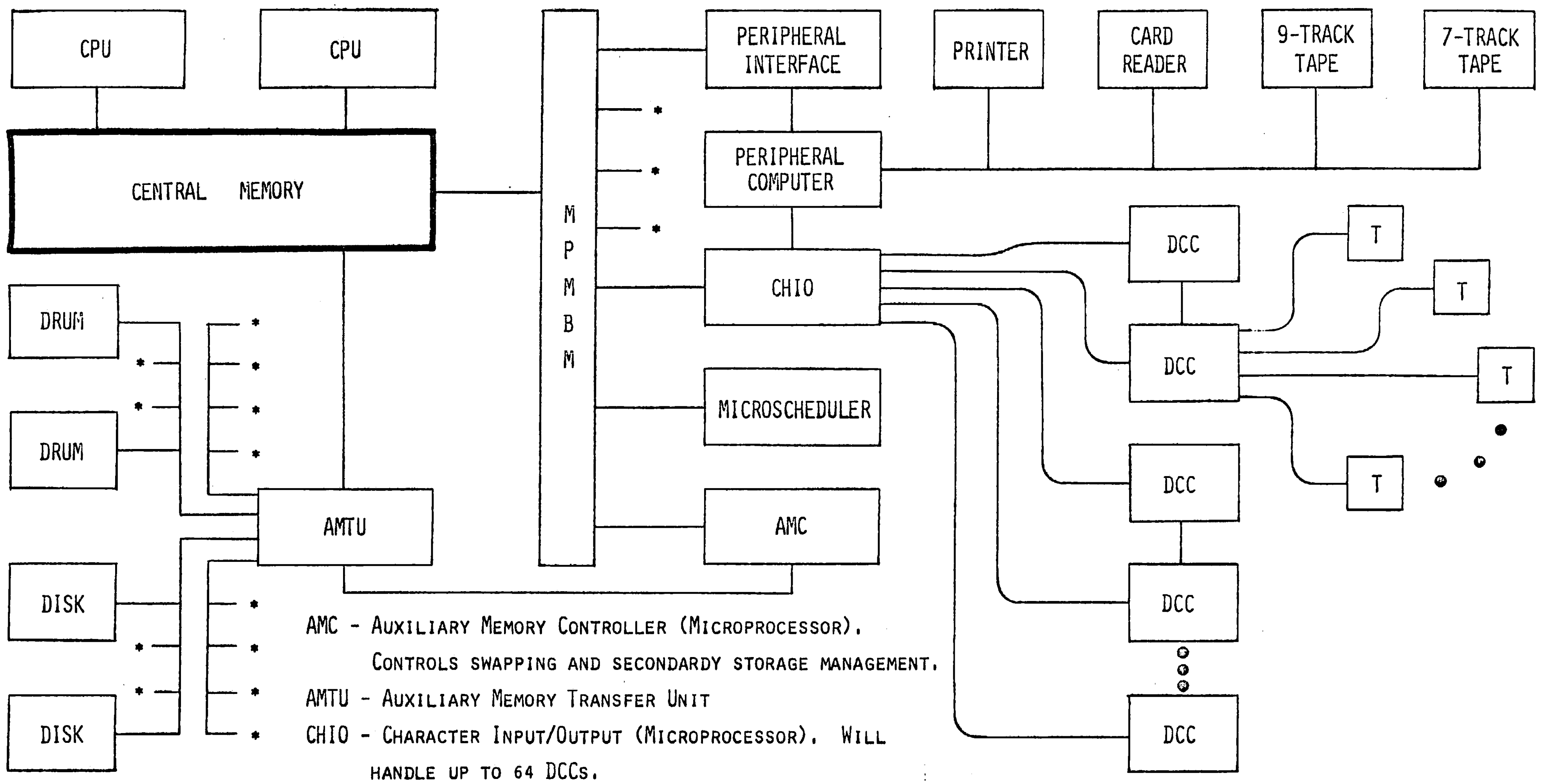
The result of these considerations is an integrated system for composition, analysis, execution, and manipulation of programs, with facilities covering the range from semi-novice to expert in sophistication. SPL does not try to please everyone; there is no decimal, scaled or multiple-precision arithmetic for example. It does, however, satisfy the requirements of most scientific and system programmers. The major novelty in SPL is the fact that it allows source-language editing and debugging and incremental compilation together with nearly optimal use of the machine.

References

1. Lampson, B.W., "Dynamic Protection Structures," Proc. AFIPS Conf. 35 (Fall 1969), pp 27-37

2. Lampson, B.W., "A Scheduling Philosophy for Multiprocessing Systems," Comm. ACM 11, 5(May 1968), pp 347-360.

3. Lee, Francis F., "Study of 'Look Aside' Memory," IEEE Trans. Computers, C-18, 11 (november 1969), pp 1062-1064.



AMC - Auxiliary Memory Controller (Microprocessor).
Controls swapping and secondardy storage management.

AMTU - Auxiliary Memory Transfer Unit

CHIO - Character Input/Output (Microprocessor). Will handle up to 64 DCCs.

DCC - Data Communications Computer (Microprocessor). Will handle up to 224 full-duplex, low speed terminals.

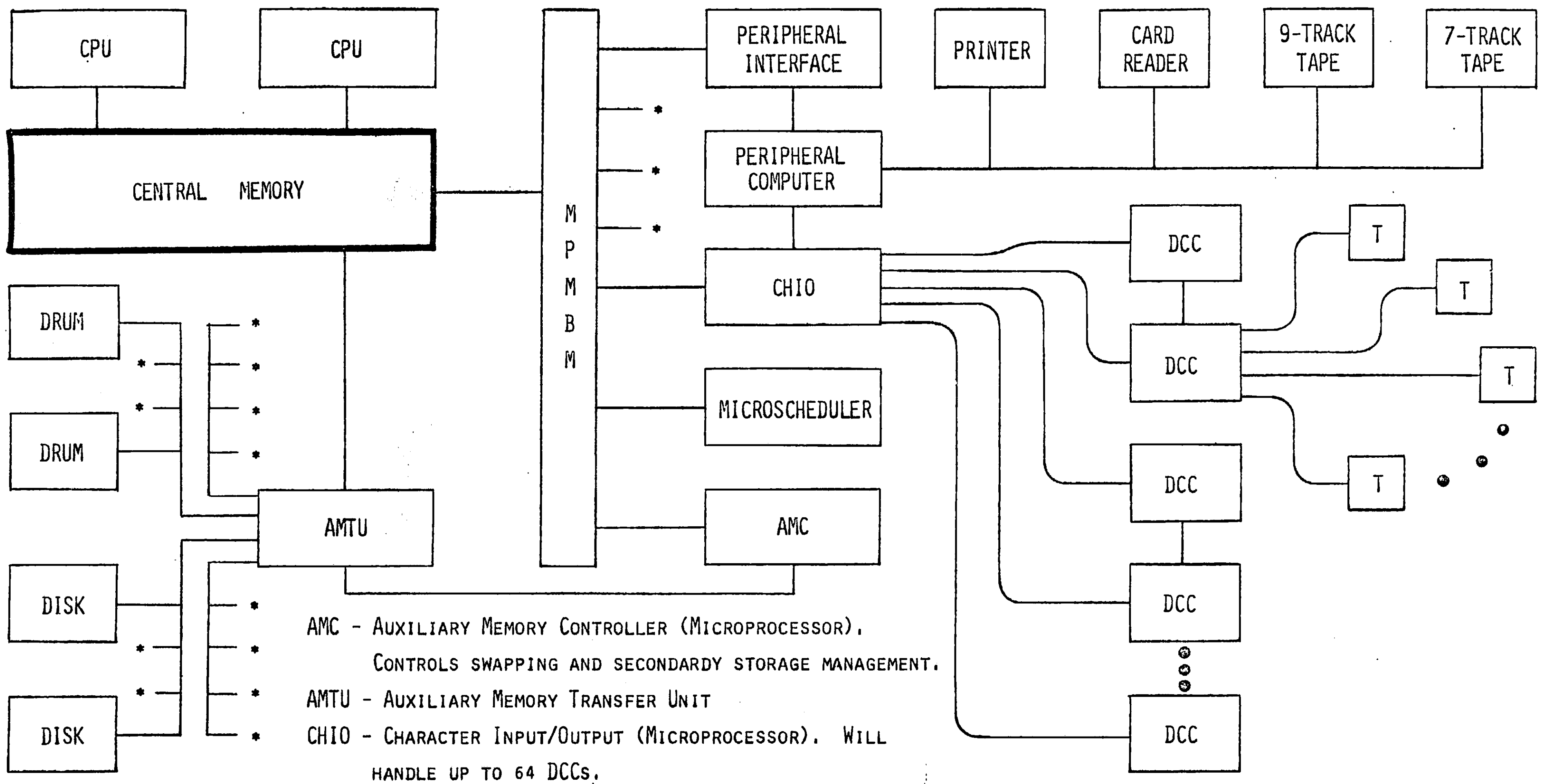
MPMBM - Microprocessor Memory Bus Multiplexer.

MICRO-SCHEDULER - (Microprocessor). CPU Scheduling.

T - Terminal: Teletypes 33, 35, 37; IBM 2741 (both models); 300 characters per second line printer; keyboard display.

Figure 1

BCC 500



AMC - Auxiliary Memory Controller (Microprocessor). Controls swapping and secondardy storage management.

AMTU - Auxiliary Memory Transfer Unit

CHIO - Character Input/Output (Microprocessor). Will handle up to 64 DCCs.

DCC - Data Communications Computer (Microprocessor). Will handle up to 224 full-duplex, low speed terminals.

MPMBM - Microprocessor Memory Bus Multiplexer.

MICRO-SCHEDULER - (Microprocessor). CPU Scheduling.

T - Terminal: Teletypes 33, 35, 37; IBM 2741 (both models); 300 characters per second line printer; keyboard display.

Figure 1

BCC500

SOFTWARE AND FIRMWARE INVENTORY

NOVEMBER 7, 1970

BERKELEY COMPUTER CORPORATION

TABLE OF CONTENTS

I - INTRODUCTION

This document contains an inventory of all the BCC software and firmware. An abstract is presented for each program which outlines briefly its function, current state, and extent of documentation.

Programs operating on the BCC 500 fall into three general operating environments depending on whether they are coded in the Model 500 instruction set, the 940 emulated instruction set, or the micro-processor firmware instruction set. All firmware has been written in a commented assembly language and all of the 940 and Model 500 programs have been written in high level system programming compiler languages. Thus, in addition to normal documentation, the program listings provide a straight forward means for understanding.

In addition to an extensive set of design and implementation documentation, a set of user manuals exist for most of the user interface software. These include finished manuals for the GE 400 series compatible system, initial manuals for the XDS 940 emulated system, and system programmer level manuals for the Model 500 system and all its constituent parts. There are also broad brush overviews covering the BCC 500 capabilities, the central processing units, the communications system, and the BCC system programming language(SPL).

A considerable amount of operating software and firmware has been developed in the year and a half that has been devoted to this effort. This is due to the following points:

1. All software and firmware is written in efficient high level languages.

2. Initial implementations were done on an XDS 940 system which had a Model 500 simulator on it.

3. Two years of design in a university environment preceded actual implementation.

4. High caliber, experienced system programmers have consistently devoted much more time than is the case in the usual programming environment.

5. Critical algorithms and portions of the system were first simulated before they were implemented.

6. A substantial set of diagnostics, test routines, and debugging tools were developed.

All of these points, coupled with the hardware development, has made it possible for the BCC 500 system to be currently operational in a limited capacity. The system will now support time sharing services sufficient to finish the remaining tasks.

II.- PROGRAMS WHICH INTERFACE THE USER

940 SUBSYSTEMS

These subsystems originated on the SDS 940 and are completely operational on the Ml. Most 940 subsystems can be converted to the emulated 940 on Ml at minimal expense.

QED

This is a powerful editor capable of modifying any symbolic file. It is the mainstay for maintaining all software.

NARP

This is a fast single-pass macro assembler. It is a very convenient tool for creating assemblers for BCC processors.

QSPL

This is the 940 Quick System Programming Language. It is intended for the programmer who desires complete control over the programming environment. It's syntax is similiar to SPL such that programs written in QSPL can be converted to SPL at moderate expense.

DDT/QRUN

DDT is the 940 system debugger. It provides many features including breakpoints, searches, symbolic typeout of cells, and ability to write programs which facilitate debugging programs. QRUN is DDT with the QSPL runtime imbedded in it.

SNOBOL

This is an interactive SNOBOL system including an editor, compiler-interpreter, and debugger. It includes features to control a subprocess which were used heavily on the SDS-940. However, as the control I/O stream logic has not been implemented yet, the subprocess control features have not yet been redone.

CAL

An interactive interpreted language for doing short numerical problems.

MICRO/FNARP/DIMS

These three subsystems represent the two pass compiler for microprocessor code and debugger.

MICRO: (Bo's comments)

Bob Van Tuyl

8. SYSTEM-TEST

Provides various facilities for monitoring the system, testing the system, and changing the system. Includes a debugger. Completely operational. See memo by Lewendal.

9. SPL

The interactive SPL system consists of a compiler, debugger, editor, Model 500 simulator, and executive. This is a highly integrated and efficient system and has been used to produce most of the software running on the Model 500. The SPL system runs in 940 emulation mode on the Model 500 and has run in the past on an actual 940. The Model 500 simulator allowed implementation and debugging of the Model 500 system to occur on a 940 before the BCC hardware was ready. SPL does not yet run with the Model 500 instruction set. See documents CSED/M-12, SPLDS/W-17, SPLAL/W-16, SPLEP/W-21, SPLAPF/W-25, SPLEX/W-32, SPL/M-1.2, PREP/M-10, UPREP/M-11.

10. CROSS-REFERENCE

This program takes an arbitrary number of files of symbolic information and produces a file containing an alphabetic list of all references to each symbol. Its main features include a fast sort and a yet-to-be-overflowed symbol table. The program is operational. See memo by Van Tuyl.

11. READ PAPER TAPE

Read paper tape is a program which is used to read and edit a paper tape made off-line by a secretary.

It incorporates some very simple rules for erasing characters and lines of text. The editing conventions were designed to be quite natural and easy to learn.

12. USER LIBRARY

This is a standard set of declarations and programs designed to give the casual user of SPL a convenient starting point for writing programs. It is in final stages of debugging.

UTILITY

1. The Command Processor

This program recognizes the user's command and processes it. The commands may be built in commands such as RELEASE, SAVE-CURRENT, STATUS, RESET, etc. or they may be link names or file names. The QUIT command is not yet implemented. See document CMP/W-6.

2. Process Creation and Manipulation

This program contains the routines necessary for process creation. It will provide automatic resource allocation and now provides user extensions such as command and file search control by the Process-Profile. See document PMTSPT/W-2.

3. Sub-process Creation and Manipulation

This program handles the Utilities needed for sub-process manipulation as well as providing the user with a simple way of creating and attaching sub-processes. This program also maintains tables which extend the user's controls as well as protect him somewhat from the intricacies of the Monitor. See documents SPCAP/W-33, PMTSPT/W-2.

4. Control Input/Output Streams

This program handles the users control I/O, i.e., that I/O which is usually thought of as controlling a process or sub-process, usually a terminal. This program allows a sub-process to be terminal driven, file driven, or sub-process driven. See document CIOSUC/W-31.

5. File Handler

This program contains the usual mechanism for manipulating files except the actual reading and writing of the files. The primary feature of this section is the abbreviated name searching routines. These routines are also used by other parts of the system. This program also implements the Utility side of the file locking mechanism. It also provides the user with simplified techniques for using the system access and security mechanism. See documents MlFS/W-1Ø, SCBFS/W-9, FNS/W-4.

6. User Profile Manipulation

This program provides the calls necessary for manipulation of the User Profile. The User Profile contains information about user validity and capabilities. It also provides for user creation and Account and Group considerations. See document AUD/W-7.1, PM/W-48.

7. Trap and Error Logic

This program provides for all system and user caused traps and errors. It attempts to reflect user caused traps or errors to the user if necessary. However, it will handle all problems including those the user program is not prepared to handle. See document IWS/W-11.1.

8. The Utility also contains miscellaneous routines for support purposes such as time and date routines and error message generators.

9. Listener

The Listener is a system extension routine. It has two distinct parts which will probably be split into two separate processes at a later date.

The first part is the initializer. This is needed when the system is brought up fresh, i.e. no files exist. The Listener reads an Initial Drum (Disk) Load tape which contains the initial system software configuration. It creates the necessary MIB objects and copies the files from tape to drum. It then goes into its second phase which is to listen to the unattached device lines.

When a line becomes active, the Listener carries on a conversation with the CHIO and DCC to connect the line and initialize certain system tables. The Listener then creates a process to carry on necessary conversation to verify the user. This process is called the Enter process. The Listener passes the line to this process. See document UTENT/W-26.

1Ø. Enter

This routine carries on the dialog necessary to establish the validity of the user and his resources. It runs in the user ring. When the user is validated it creates a process or restarts a process to perform the users task. By definition all processes created by ENTER have a Utility attached as the first sub-process. See document UTENT/W-26.

11. Logout

The Logout routine cleans up the users world and if necessary causes the process to be destroyed or causes it to be saved away for a later restart and hangs up and returns the line.

12. Profile Maintenance

User ring program which through simple user dialog manipulates the users profile or the process profile. An owner (group, account) may manipulate other profiles if they have access and control. Users are created and deleted with this routine. See document PM/W-48.

13. File Maintenance

File-Maintenance routine allows manipulation of MIB objects given that the user has sufficient access.

14. Operators Control Routine

Allows the account, group, or systems owner to make certain observations about and have control over his customers (users). Terminals may be shut off or turned on, resources allocated, etc. See document TM/W-50.

MONITOR

1. The File System

This code keeps track of all objects - like files, processes, etc., - belonging to users in the system. It implements our access and protection schemes for these objects. There is a currently working version of this system. There are plans to re-work it to make it more efficient. See document SCBFS/W-9.

2. The Interrupt and Wakeup System

The IWS processes wake-ups directed toward a process by the system and implements our means of inter-process communication. The IWS is a working part of the system. A more efficient version has been written but not yet incorporated into the system. See document IWS/W-11.1.

3. The Trap-Handling System

This system fields traps caused by malfunctioning user programs and either takes corrective action or reflects the traps back into a user program prepared to deal with them. There is a working version. There are plans to make it more sophisticated. See document IWS/W-11.1.

4. The Process Memory System

This system handles page-faults, maintains the working set of a process, protects the memory of one sub-process from that of another, and performs other memory management functions with the process. A version of the PMS is currently working. A new and more efficient version has been written but not yet put into the system. See document PMS/M-19.

5. The Sub-Process System

This system divides the process into up to 8 sub-processes, which are more or less independent programs which can communicate with each other, share memory, and protect themselves from malfunctions of other sub-processes. See document MISPS/M-7.

6. Resource and Scheduling System

This supervises the distribution of system resources among the users and processes in the system. Only a primitive version of the RSS is in the current system. A more complete version is being written. See document BSRA/W-27.

EMULATOR

1. BCC 940 EMULATOR

The Emulator is a large program which provides a Monitor and Executive for programs which run in 940 mode on the Model 500.

Every system call, made by a 940 mode program being executed under the control of the Emulator, is either directly performed by the Emulator, or is transformed into system calls to the Model 500 system. This has been operational for 4 months.

III - PARTS OF THE SYSTEM

1. Auxiliary Memory Controller (AMC)

The AMC is implemented on a BCC microprocessor. Some of the microcode implements a processor (APU) which is similar to the SDS 940. Another major part of the microcode implements a watch dog which responds to stimuli from the CPU, drum and disk. The rest of the microcode implements subroutines which are fixed and must be fast.

The Phase 1 has been redesigned twice since it first became operational. The first rewrite allowed more efficient allocation of the drum. It is operational. See AMC Phase 1 Notes.

2. Communications System Phase 1

This consists of microcode for the CHIO to run 16 teletypes. It is part of the current system and has been operational for six months. See IHTWD/W-38, ATINFIC/W-37, FOO/S-2.1, The BCC Communication System.

3. CPUs

a. Micro-code for Phase 1 CPU, having an instruction rate $2\text{x}1\emptyset^5$ per second. The code is 27 micro-code boards long, each board containing 64 words of 90 bits each.

The micro-code includes routines for address calculation, fixed point multiply/divide, 6 kinds of shifts, string, field and array operations, floating point arithmetic, subroutine jumps, mapping, processor-state switching and Xds-940 emulation. See CPUPG/M-14.1, MICPU/M-4.4, TSTX/W-45. This CPU has been operational for several months.

b. Micro-code for Phase 1.5 CPU, instruction rate is $5\text{x}1\emptyset^5$ per second. Similar to the Phase 1 code with general improvements and service routines for the instruction prefetch unit added. This CPU is now running an experimental version of the Monitor/Utility.

4. Microscheduler Microprocessor

a. Microscheduler

This microprogram does in-core scheduling and handles real-time interrupts for the system. Operational for several months. See PRUN/W-8, USI/W-14.

b. Integral Test Processor

This microprogram emulates an approximate 940 subset. It resides in the same microprocessor as the microscheduler with which it shares the capability of that processor. The ITP is used for system checkout and maintenance. Operational for several months. See ITPEX/W-22, ITPRM/W-23.

c. SYSDDT

SYSDDT is a debugging, monitoring, and hardware support program which runs in the UTP (microscheduler-test processor) microprocessor. It provides facilities for examining memory, starting and stopping the various processors, reconfiguring the system for certain hardware conditions, taking statistics on system operation, and writing short programs to run in any processor. SYSDDT is fully operational. See SDDTCOM/W-46.

5. Model 30

a. Model 3Ø Driver

This is a program which multiplexes requests and data to tape units, card readers, and printers attached to the M3Ø. It has been operational for several months. See M3Ø/S-7, M3ØT/M-18.1.

b. Initial Program Library

This is an ITP program which allows magnetic tapes to be read and written. It resides in the microscheduler's private memory and is used to bring up the system on a bare machine. It has been operational for several months. See memo by Thompson.

c. M3ØCOM

This program allows any number of processes to communicate with the peripheral equipment connected to the 36Ø-3Ø. To be used mainly by the Disk Backup System and Print program. The program is in the latter stages of debugging. See DKBK/S-26.

IV - DEVELOPMENT WORK

1. Subsystems

a. FORTRAN IV compiler is completely written and has been 90% debugged in a 940 environment. The translation to SPL is complete. It must now be compiled in the Model 500 environment before final checkout can proceed. Two man-months estimated for completion. User manual is completed.

b. The FORTRAN IV run time package is completely written and about 50% debugged. One man-month estimated to completion.

c. An Extended BASIC compiler is completely written and 90% debugged in a 940 environment. The SPL translation is complete. It must now be compiled in the Model 500 environment before final checkout. One man-month estimated for completion provided FORTRAN is completed first. User manual is completed.

d. The Extended BASIC run time package is completely written and 50% debugged. Two man-weeks estimated to completion provided the FORTRAN run time is completed first.

e. The GE Executive (command processor) is 80% written and debugged. Two man-weeks estimated for completion. User manual is completed.

f. The GE file system is 90% written and 80% debugged. Two man-weeks estimated to completion. User manual is completed.

g. The GE editor is 90% written and 80% debugged. Two man-weeks estimated to completion. User manual is completed.

h. Disk Backup System is a program for backing up disk files on magnetic tape. This program will be responsible for maintaining the permanency of disk files. It is in the latter stages of debugging. See DKBK/S-26.

i. The Print System provides a facility for queuing print requests. Initial version operational. Final version depends on M3Ø Communications Program.

2. Monitor

a. Interrupt/Wakeup System. The IWS has been completely rewritten, and is awaiting integration into the system along with the necessary debugging to accomplish that.

b. Software Lock. (software locking mechanism) Simple forms of Dijkstra's P and V synchronization have been written and are awaiting the integration of the new IWS mentioned above. Similar mechanisms have been designed, but not programmed for locks on the user's directory and on individual objects (files, etc.).

c. File System Rewrite. The file system rewrite has had some redesign done, but little actual programming or modification of the existing system has been done.

d. Resource Allocation Programs. (monitor level) The design for statically allocated resources has been done. Dynamically allocated resources, while partially designed, is dependent on the design of the Scheduler, which is not yet complete. See Resource Allocation.

e. Accounting Monitor Call. This call has been designed, and would take approximately one week to implement.

f. Crash Recovery. Reconstructs a good system from the wreckage of one which has failed. This program is in final debugging phase.

3. Microprocessors

a. AMC 1.03. The final redesign of the phase 1 AMC. This AMC includes many new features designed to speed monitor operations. It also includes logic to improve the flow of processes through core.

b. Phase II AMC

1) QSPL simulation of all microcode routines has been completed. This is being used as a pattern for the microcode.

2) The microcode has been written for about 90% of the routines and the declaration package is virtually complete. Of the 90% which has been written, 70% has been looked over very carefully and is ready for serious debugging.

c. Communications System Phase 2

This consists of microcode for two processors; the DCC and the CHIO, and a CHIOINIT function in the monitor. The communications system when working will connect the BCC 500 to several remote computers which will do bit scanning of teletypes. The remote computer can also connect to higher speed devices such as line printers.

The status of the communications system is that the individual processors have been extensively tested, but the system as a whole has not yet been run. There is very high confidence that no major software/firmware difficulties will occur in bringing up the Phase 2 communications system. The CHIOINIT function is written but undebugged. No other code is needed to run the Phase 2 system except the DCC test program. To take advantage of all of the communications system features, however, more DCC code must be written.

d. Data Communications Computer Microcode

The low-speed device bit scanning, multiplexing and error-correction procedures of the communications system are implemented by this code. In final debugging stages.

e. DCC Loader-Unloader

A self-contained routine to load binary paper tapes into the DCC and to dump portions of DCC core in loadable format. In final debugging stages.

f. Phase 2 CPU

High-level logic design and microprogramming for the Phase 2 CPU address calculation unit and instruction execution unit. The design work has been suspended.

4. Other

a. Utility Accounting Program.

This is called by the Utility at logon and logoff of each user and also at a pre-set interval. This program makes a monitor call to get current information which it massages and then puts (temporarily) on a one-page file called the Message Buffer. When the Message Buffer is full, it gets dumped onto the system DAEL. It is coded except for the intricacies of locking the Message Buffer and setting up a timed call of the program. It has not been debugged. The daily update program is 80% written with no debugging. The group multiplexing program is completely written and 40% debugged. The billing program is completely written with no debugging. See FSOC/S-36 and several memos.

b. Shell 940 Emulator

This emulator provides the Monitor and Executive for Shell Development Corporation's 940 system. It differs from the BCC 940 Emulator in that 1) it is emulating a different Monitor and Executive, and 2) it provides for all the Executive commands of the Shell system. In particular all commands are directly interpreted by the Emulator rather than by the Executive supplied by the Utility. This work has not been completed.

c. Parameter Collection

Parses a command line according to syntax supplied by a program, prompts for missing parameters and generates error messages. It is being coded.

d. Disk and Drum Diagnostics

A fairly sophisticated drum and disk diagnostic has been written and is virtually debugged. It uses a special purpose APU program which simulates the Phase II AMC's mode of direct I/O so that the program will run with only trivial changes on the Phase II.

V - DIAGNOSTIC PROGRAMS

1. Disk Diagnostic

Disk diagnostic (KTEST) intended to run under the time sharing system; this program's main purpose is the identification of faulty records on the disk file due to surface imperfections. The program operates in

a. sequential addressing mode and
b. random addressing mode.

This program is written in 940 assembly language and is fully operational.

2. Drum/Disk Exercizer

This ITP program reads and writes pages of pseudo-random data on a drum or disk using the APU direct I/O feature and reports errors detected by the AMTU during these operations. Error reporting is governed by the setting of switches in the Sense Switch Register. Also, as an option, the data read from the device can be compared with that written and discrepancies reported. Parameters set in core include number of reads, writes, pages, area on the device, and type of operation. This program is complete and in use daily.

3. Drum Basher

This APU routine runs the drum at full speed in order to test the memory system. It has been operational for several months. It has been rewritten in microcode but the boards have not been built yet.

4. DCC Test Program

This program makes a fairly elaborate test of the DCC to see if it and the microcode are working properly. This program, which has been working for three months, tests all of the important parts of the DCC microcode.

5. ITP Diagnostic

A self-contained diagnostic program for the ITP which verifies the proper functioning of microprocessor and microcode. Operational.

6. Diagnostic for APU

This program runs on the APU and checks a subset of the instructions. A companion program runs on the ITP and prints a message whenever an error is discovered. Operational.

7. CPU Test Programs (4 packages)

The set of four test programs exercises every micro-instruction in the CPU. In case of an error a message is given indicating the instruction which failed, the correct and the bad results. These test programs do not use the Monitor and require only a minimum of working hardware. (Memory and ITP) The four packages are as follows:

a. Test of the addressing logic
b. Test of instruction execution
c. Test of subroutine and Monitor calls
d. Test of traps.

8. CHIO Test Program

This test program has a large number (>150) of routines, each testing a micro-code subroutine in the character input/output (CHIO) processor. The program also contains a simulator for an error-free communication line (EFCL) to facilitate the debugging of the Data Communications Computer (DCC)-M1 linkage.

9. Core Memory Diagnostic

Generates worst-case and random test patterns for diagnosing core modules in the 500 system. This program is completed.

VI - SUPPORT SOFTWARE

1. Save Drum Program

This program saves the current state of the running system onto drum or disk, and restores it at a later time. Status: Completed.

2. MICRO/FNARP/DIMS

These three subsystems represent the two pass compiler for microprocessor code and debugger. See document MICRO/M-8.

MICRO: a compiler for translating programs written in a convenient higher level language into the 90-bit microcode words used by the BCC microprocessors. MICRO has been completely operational for well over a year.

FNARP: this is the second pass of the compiler. Micro outputs symbolic output to FNARP. FNARP is NARP which contains some extensive macros to output the correct format to DIMS.

DIMS: this is the debugger which includes the microprocessor simulator. All microcode generated in the past year has been tested to some degree with this debugger.

3. Microprocessor Simulator

This simulates BCC microprocessors including DCCs. This program which generates both a matrix of the bits needed for the ROMs and a paper tape that can be used to control the drilling machine that makes the printed circuit board has been operational for one year. See document PIG/M-13.

4. MNODT

This is a micro-coded debugger for the microscheduler. It is completed and has been used successfully for about four months. See document MNODT/W-34.

5. TPDDT

This program runs on the test processor and allows the engineers to write and debug programs. It has been operational better than one year. See document TPDDT/W-3.1.

6. Scan Simulator

This program simulates the bit scanning algorithms of a DCC as well as the characteristics of medium speed lines.

7. Simulator for TSU, Drum, and Disk

This program used in the debugger described below to simulate the interface between the AMC and the drum and disk.

8. Debugger for AMC

This program fits over the microprocessor simulator containing the microcode for the AMC. It allows the APU programs written for the AMC to be debugged. It provides some reasonably powerful debugging aids, such as breakpoints, searches, and listing features. See document DBAMC/W-43.

9. GE Dump Tape Translator

Program for reading files from a GE dump tape. Completely operational on the 940. This is used to bring files from 400 series GE systems into the BCC-GE compatible system.

10. WIRE-LIST

From a list of pin number and signal names produce a list of wires, organized by signal name. The specific purposes were accuracy, repeatability, legibility (for debuggers) and easy conversion to formats required by automatic wirewrapping machines.

This process has been running for over a year. It was first written in SNOBOL 4. Recently it has been rewritten for speed reasons in QSPL. All the processors and the CPUs of our system were wired automatically by wirelists produced by these programs. See document WL/M-22.

11. Operator Precedence Table Builder

This program generates operator precedence tables from BNF descriptions of the syntax of expressions. See document OPREC/R-4.

12. Lexical Analyzer Table Builder

This program generates the tables for a table driven lexical analyzer and keyword tables. See document GELAI/R-5.

13. DCC, APU, ITP, and TP2 Assemblers

These four assemblers produce machine code for the four mentioned microprocessors. See documents RPASS/M-21, PTPP/W-35.1.

14. Read, Punch, and Verify Paper Tapes

This program is used to check for errors on long punched tapes.

15. Directory for Incore Debugging

This program takes a symbol table generated by the assembler and produces an alphabetic list of the symbols and their locations.

16. Bootstrap Loaders for ITP, TP2

These loaders are imbedded in a program which prefixes the loader to a core image of a program which is to be loaded into the processor. See document PTPP/W-35.1.

17. Free Storage Allocator

An efficient subroutine for implementing a free storage allocator. Implemented in QSPL. See memos by Van Tuyl.

18. SNOWFLAKE

A package of routines written in SPL to make SNOBOL-like pattern matching facilities callable from SPL programs. Largely checked out.

APPENDIX A

INVENTORY OF SOFTWARE

Programs Which Interface User

1. Subsystems

   QED
   NARP
   QSPL
   DDT/QRUN
   SNOBOL
   CAL
   KDF-BACKUP
   SYSTEM-TEST
   SPL
   Cross-Reference
   Read-Paper-Tape-Made-by-Secretary
   SPL-User-Library

2. Utility

   Command Processor
   Process Creation and Manipulation
   Sub-Process Creation and Manipulation
   Control I/O Stream
   File Handler
   User Profile Manipulation
   Trap and Error Logic
   Listener
   Enter
   Logout
   Profile Maintenance
   File Maintenance
   Operator Control Routine

3. Monitor

   File System
   Interrupt and Wakeup System
   Trap Handling System
   Process Memory System
   Sub-Process System
   Resource and Scheduling System

4. 940 Emulator

Parts of the System

1. Auxiliary Memory Controller

   Microcode
   Auxiliary Processing Unit Code

2. Character Input/Output Processor

   Phase 1 Microcode

3. CPUs

   Microcode for CPU 1.Ø
   Microcode for CPU 1.5

4. Microscheduler Microprocessor

   Microscheduler microcode
   Integral Test Processor (ITP)
   System Monitoring and Debugging Program

5. Model 30

   Model 30 Code
   Initial Program Library (IPL)
   M3Ø Communication Program

Development Work

1. Subsystems

   Fortran IV
   Basic
   GE Executive, Editor
   Disk Backup System
   Print

2. Monitor

   Interrupt/Wakeup System
   Software Locking Mechanism
   File System
   Resource Allocation
   Accounting Monitor Call
   Automatic Crash Recovery

3. Microprocessors

    Phase 1.Ø3 AMC
    Phase 2 AMC
    Phase 2 Communications System
    Data Communications Computer Microcode
    Phase 2 CPU

4. Other

    Accounting System
    Shell Emulator
    Parameter Collection
    Drum and Disk Diagnostic

Diagnostic Programs

- Disk Diagnostic (on time sharing system)
- Drum/Disk Exerciser
- Drum Basher (full speed)
- DCC Test Programs
- ITP Diagnostic
- APU Diagnostic
- CPU Test Programs
- CHIO Test Programs
- Core Memory Diagnostic

Support Software

- Save System on Drum
- Microprocessor Compiler/Assembler
- Microprocessor Interactive Simulator/Debugger
- Microcoded Octal DDT
- Bare Machine Debugging Program (for ITP, TP2)
- Bit Scanning Simulator
- TSU, Drum and Disk Simulator
- AMC Debugger
- Read-GE-Tape
- Wire-Listing
- Operator-Precedence-Generator
- Lexical-Analyzer-Generator
- Assemblers for:
    - DCC
    - APU
    - ITP
    - TP2
- Read and Punch Paper Tapes with Error Checking
- Directory for Incore Debugging
- Boot Strap Loader for ITP, TP2
- Free Storage Allocator
- SNOWFLAKE

APPENDIX B - SOFTWARE DOCUMENTATION

NUMBER	TITLE
FOO/S-2	CHIO/CPU Interface
M3Ø/S-7	Communications with the IBM Model 30
BCCPU1/S-10.1	Special and Branch Conditions in CPU1
PIF/S-21	Specification of Program Image Files
COMCON/S-22	Command Writing Conventions
RC/S-23	The Remote Concentrator Design
CS/S-24	The Communications System--Phase II
INITT/S-25	System Initialization Tape
DKBK/S-26	Disk Backup Subsystem
TEXTF/S-27	M1 Text File Standard
RPU/S-33	The Remote Processor Unit
MMS/S-35	Memory Management System
FSOD/S-36	Format of System Owner's DAEL
SPL/M-1.2	SPL Reference Manual
MISPS/M-7	MCALLs on the Model 1 Sub-Process System
MICRO/M-8	MICRO Reference and User Manual
PREP/M-10	Phase One Preprocess Interface
UPREP/M-11	Phase One Unpreprocessor Interface
CSED/M-12	M1CS Phase One Language Editor
PIG/M-13	Interactive Microprocessor Simulator
CPUPG/M-14.1	CPU Programmers Guide
AKOCS/M-15	Character Sets
PTES/M-16	Paper Tape Reading and Punching
PCNR/M-17	CHIO Phase 1 Test Routines
M3ØT/M-18.1	Model 30 Trivia
PMS/M-19	Process Memory System
DCODT/M-20	DCODT Reference Manual
RPASS/M-21	The Remote Processor Assembler
WL/M-22	Wirelisting
MM1/W-1	Memory Management
PMTSPT/W-2	MCALLs for Manipulating PMT and SPT
TPDDT/W-3.1	Test Processor DDT
FNS/W-4	System 1 File-Naming System
MICPU/W-4.4	Model 1 CPU Reference Manual
CWS/W-5	The Core Working Set
CMP/W-6	The Command Processor
AUD/W-7.1	The Account and User Directories
PRUN/W-8	Running a Process
SCBFS/W-9	System Calls to the Basic File System
MIFS/W-10	Model 1 File System
IWS/W-11.1	Interrupt and Wake-up System
CSED/W-12	Preliminary Description of Compiler System Editor
USI/W-14	Micro-scheduler Implementation
SYSP/W-15.1	System Parameters

NUMBER	TITLE
SPLAL/W-16	Facilities for Allocation in SPL
SPLDS/W-17	SPL Command Language and Debugging System
SINIT/W-19	System Initialization and Error Recovery
CPUINT/W-20	CPU Interruptability
SPLEP/W-21	SPL Conventions for System-Defined Entry Points
ITPEX/W-22	External Interfaces for the Integrated Test Processor
ITPRM/W-23	Integral Test Processor Reference Manual
SPLAPF/W-25	Allocation and Program Format in SPL
UTENT/W-26	The Utility Enter Dialog
BSRA/W-27	Basic System Resource Allocation
MCHIO/W-28	CHIO Functions in Basic System
SSIN/W-29	Software System Initialization
MPREC/W-30	Microprocessor Crash Procedures Initialization and Breakpoints
CIOS/W-31	Control Input/Output Streams
SPLEX/W-32	SPL Executive Commands
SPCAP/W-33	Sub-Process Capabilities
MNODT/W-34	MNODT
PTPP/W-35.1	Procedures for TP Programs
PROC/W-36	Processes
ATINFIC/W-37	An Efficient Multiplexing Algorithm
IHTWD/W-38	CHIO Implementation Phase 1
DITP/W-39	ITP Diagnostic Program
FAROUT/W-40	An Error-free Communications Line
OHWOW/W-41	Local Echoing in the Communications System
DBAMC/W-43	Microprocessor Debugger for the AMC
MPT/W-44	Microprocessor Test Program
TSTX/W-45	CPU Test Programs
SDDTCOM/W-46	SYSDDT Commands
APUD/W-47	APU Diagnostic Program
PM/W-48	Profile Maintenance
P2CSS/W-49	Software for the Phase 2 Communications System
TM/W-50	Terminal Management
PAMC/W-51	Phase 1 AMC Notes
TRGEN/R-2	M1CS Phase One Tree Generator
SPLGR/R-3	SPL Grammar and Precedence
OPREC/R-4	Support Software for Operator Precedence Parsing
GELAI/R-5	GE Lexical Analyzer Implementation
GEFEQU/R-8	GE FORTRAN Equivalence Algorithm
SSE/T-5	Standard for Syntax Equations