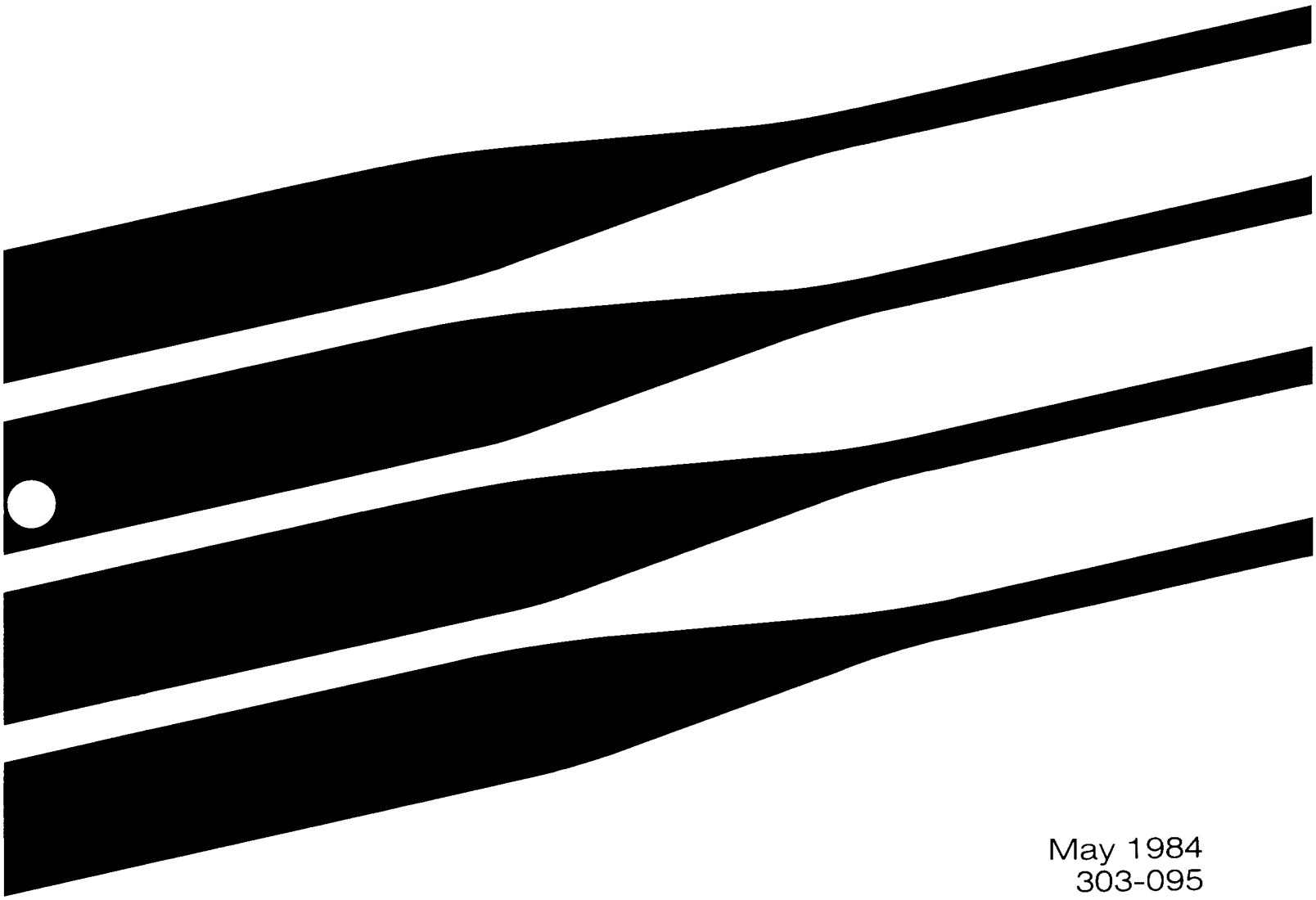
UNIX™ RTR Operating System Architecture Overview AT&T 3B20D Computer



May 1984
303-095

AT&T

CONTENTS

UNIX is a trademark of AT&T Bell Laboratories

This document is a detailed overview of the AT&T 3B20 Duplex Computer (3B20D) and its UNIX RTR operating system for system programmers and for those interested in obtaining information about the computer.

ORGANIZATION

This document is divided into six sections:

1. **Introduction** — Briefly describes the 3B20D computer and its UNIX RTR operating system and discusses some existing applications.

2. **3B20D Hardware Architecture** — Shows the 3B20D architecture and gives a detailed description of each frame or cabinet.

3. **UNIX RTR Software Architecture** — Gives information on the basic structure of UNIX RTR, describes the system management software, and delineates the input/output and maintenance features.

4. **Language Architecture** — Describes microinstruction language, assembly language, and C language used to write programs for the 3B20D computer.

5. **Development Support** — Describes System Alpha and other testing and debugging systems that augment, support, or run under the UNIX RTR operating system.

6. **Acronyms** — Lists and defines the acronyms used in this document.

1. INTRODUCTION

The 3B20 Duplex Computer is a high-speed, high-availability computer that monitors and controls operations in various switching and processing applications. It operates in a duplex configuration, with all major functional units duplicated to ensure uninterrupted and reliable service.

The computer uses LSI and VLSI technology and includes a family of peripheral devices.

The computer's central control unit is microcoded to accommodate multiple-resident instruction sets. The central control can emulate other instruction sets as well as the native instruction set, which is a superset of the IS25 assembly language.

The hardware contains self-checking and error correction circuitry. The software detects faulty processes and equipment, reconfigures or reinitializes the system, and diagnoses and identifies faulty equipment.

Application software is supported by the UNIX RTR operating system and by a comprehensive set of software development tools.

UNIX RTR and the 3B20D computer are designed for high-reliability, real-time responsive applications, for example, in telecommunications, military, and transaction processing systems. UNIX RTR provides a high-reliability, real-time operating system as well as a standard UNIX environment, and comprises a kernel and a set of modular, cooperating processes. UNIX RTR and application processes communicate via messages, events, interprocess traps, shared memory, UNIX pipes, and the file system. UNIX RTR provides four executing environments (or virtual machines) including the kernel. Applications can introduce code at three of these levels:

1. Kernel Process — Provides limited, real-time processing with interrupt-driven dispatching.

2. Supervisor Process — Provides more services with time-shared scheduling.

3. UNIX Process — Provides a UNIX environment with time-shared scheduling via a UNIX public library.

UNIX RTR emphasizes high availability through reliable hardware and software and a set of maintenance procedures. Although UNIX RTR performs fault recognition and recovery and hardware reconfiguration, some processes must perform process- or application-specific recovery after a fault or initialization.

For field maintenance, UNIX RTR provides an operator interface package with both Program Documentation Standard (PDS) and Man-Machine Language (MML) syntax, an equipment configuration data base, program updating and growth systems, and diagnostics.

1. INTRODUCTION

High availability is achieved via a duplicated computer and input/output (I/O) system. Additionally, the computer's concurrent, self-checking design simplifies maintenance and allows early detection of faults occurring during normal operation.

The 3B20D computer together with the UNIX RTR operating system:

- Achieves highest performance consistent with current technology.
- Provides high reliability and fault tolerance and meets the reliability down-time objective of two minutes per year maximum.
- Reduces software complexity by providing modern operating system facilities via UNIX RTR.
- Reduces programming effort by using the C language and a comprehensive set of software development tools.
- Provides facilities for system integrity and security (such as memory management and privileged instructions).
- Provides hardware and software support for large, real-time data bases and extensive data link capabilities.
- Allows execution of programs written for earlier ESS* processors and other computers, thus providing an upgrade path for earlier systems.

AT&T 3B20 DUPLEX COMPUTER/UNIX RTR APPLICATIONS

The 3B20D computer and the UNIX RTR operating system provide a base for a wide variety of high-availability, real-time, and time-sharing applications (Figure 1). The wide applicability of the 3B20D computer/UNIX RTR combination is due to the flexible hardware configuration of the computer coupled with the multilevel structure of the UNIX RTR operating system. The versatile, intelligent Input/Output Processor and the high-speed, dedicated I/O channels provide for varying peripheral configurations. The microcode support (for multiple-resident instruction sets) and the operating system support (for application software at several different operating system functionality levels) provide the diverse operating environment needs for both teleprocessing system and data processing applications.

* ESS is a trademark of AT&T Technologies

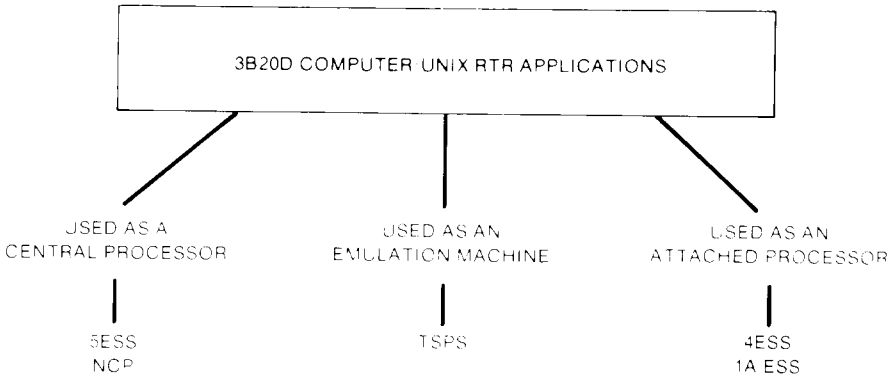


Figure 1. Existing 3B20 Duplex Computer/UNIX RTR Applications

Some existing applications of 3B20D computer/UNIX RTR use include:

- 5ESS Switching System

  The 5ESS switch is an AT&T-developed, local digital switching system designed as a world-class product for both domestic and international markets. In the 5ESS switch architecture, the 3B20D computer is used as the central processor complex and is responsible for various functions including maintenance, control, administration, human interface, system integrity, and centralized call processing functions such as routing and control of global data. Some of these functions are performed by the 5ESS application software, some are performed by UNIX RTR, and some are joint responsibilities requiring close coordination between the 5ESS application and UNIX RTR.

  The multilevel hierarchy in UNIX RTR allows the different functions to be placed at the most appropriate level. Each function involved was evaluated and the most appropriate capability level for 5ESS processes was selected. For example, many functions such as administration, recent change, and basic diagnostics are not real-time sensitive. Therefore, the performance of the kernel process level was not required and the easier development environment of the UNIX process level was the deciding criterion. In contrast, the call processing function required the best possible real-time performance so the kernel process level was chosen.

The 5ESS application added an application-specific environment having features such as task management, message handling, and timing specifically oriented to 5ESS call processing needs. The same environment is provided on other processing components within the distributed 5ESS architecture. This similarity of environment results in a uniform interface for 5ESS developers with subsequent advantages in training, portability, and ease of development.

The 5ESS application demonstrates the flexibility of the UNIX RTR multilevel architecture. Depending on the required flexibility, performance, and ease of development, a specific process can be placed where it fits most appropriately. Because of the kernel process interface, a unique environment also can be implemented for specific needs.

- Traffic Service Position System

Since its introduction in 1969, the Traffic Service Position System No. 1 (TSPS No. 1) has been deployed rapidly throughout the AT&T nationwide telecommunications network. More than 150 systems are now in operation in the continental United States. To meet growing needs, a major evolution from TSPS No. 1 to a new system, called TSPS No. 1B, was required. The goals of this evolution were to provide substantially increased call capacity, memory, and computer peripheral capability.

To preserve as much of the original investment as possible, the existing software and peripheral hardware were retained in the transition to TSPS No. 1B. Concurrently, new high-level language software and additional peripheral hardware were added to the system for new features.

The 3B20D computer and the UNIX RTR operating system meet the goals of TSPS No. 1B. The 3B20D computer replaces the existing TSPS No. 1 processor. The existing TSPS No. 1 periphery is retained and, through emulation, the existing TSPS software is preserved. The existing TSPS No. 1 periphery is kept by using a peripheral system interface (PSI) circuit that interfaces the TSPS buses with the 3B20D computer. The software is preserved by defining (through microcode) one of the multiple 3B20D instruction sets to be that of the existing processor, thus emulating that processor and allowing existing TSPS software to be transported to the 3B20D computer almost intact. A single instruction within a single process switches between the emulated instruction set and the 3B20D native instruction set. Therefore, new software can be added into either environment, as appropriate. Both emulated and native mode software run under the UNIX RTR operating system, allowing operating system services to be available to both forms of software. The emulated, existing TSPS No. 1 assembly language code is structured as a single kernel process executing under UNIX RTR to permit efficient control of real-time resources where

required. Other administrative and diagnostic functions that are not real-time sensitive are implemented as UNIX-level processes. The UNIX RTR operating system provides computer maintenance and administration. Thus, both the multilevel operating systems and multiple-resident instruction set capabilities of 3B20D computer/UNIX RTR are essential elements of the TSPS No. 1B design.

- Network Control Point

The Network Control Point (NCP) is a new AT&T development that adds an on-line, real-time data base to the Common Channel Interoffice Signaling (CCIS) network. Because the NCP was designed to use the 3B20D computer and the UNIX RTR operating system, the software is written entirely in the C programming language. Those portions of the NCP software that are real-time critical are implemented at the kernel process level; those portions with less stringent real-time requirements are implemented at the UNIX level. Thus, all features of the UNIX RTR operating system are used to create an efficient, homogeneous system.

Hardware for the NCP is composed of standard 3B20D units except for one special peripheral controller board used to link the NCP to the Signal Transfer Point of the CCIS network. The UNIX RTR input/output driver module is modified to handle this board and a diagnostic module is integrated with the standard UNIX RTR diagnostics. The remainder of the NCP hardware consists of duplex processor systems equipped with 10 to 16 megabytes of memory each, four Input/Output Processors, five Moving Head Disk units, six to ten X.25 data links and a Magnetic Tape unit. A typical NCP is one of the larger 3B20D systems in operation.

- Attached Processor System for the 4ESS and 1A ESS Switches

The 1A processor is the central processing unit of both the 4ESS and the 1A ESS switch. The 4ESS switch handles toll and tandem switching functions and the 1A ESS switch handles local, tandem, and toll switching functions. The growth of telephone traffic and customer demand for new services on these systems required increased processing capacity, new service features, and expanded memory spectrum. A new system architecture involving the attachment of the 3B20D computer to the 1A processor configuration was designed to meet these needs. The 3B20D computer with the UNIX RTR operating system was chosen because of its low cost, high reliability, high processor and memory resource capacity, and a high-level language in the software development environment.

The 1A file store was the earliest resource exhausted because of increased traffic load and new feature development. The Attached Processor System (APS) with the 3B20D computer and UNIX RTR operating system was designed as a replacement for the 1A file store. In addition, the APS can also relieve the 1A of operational and administrative tasks that potentially limit its real-time performance at high traffic loads. The 1A and 3B20D computer support different programming languages, and new development software can be written in the more appropriate language.

The APS system includes hardware and software to connect the 3B20D computer to the 1A processor. The hardware includes an Attached Processor Interface (API) unit to interconnect the direct memory access (DMA) channels of the 3B20D computer and the 1A processor. The software includes an API driver consisting of 3B20D program modules and corresponding 1A program modules. Together, these modules administer and maintain a 10 megabit/ second, fully-duplicated processor/computer communication link. The 3B20D modules are designed at the kernel process level to meet the stringent, real-time requirements for processor/computer intercommunication. To maintain integrity with the existing 4ESS and 1A ESS system software environment, the 1A disk administration mechanisms are provided in the APS. The 1A also has full access to the 3B20D computer/UNIX RTR File Manager and the entire 3B20D file system.

The 3B20D architecture is shown in Figure 2.

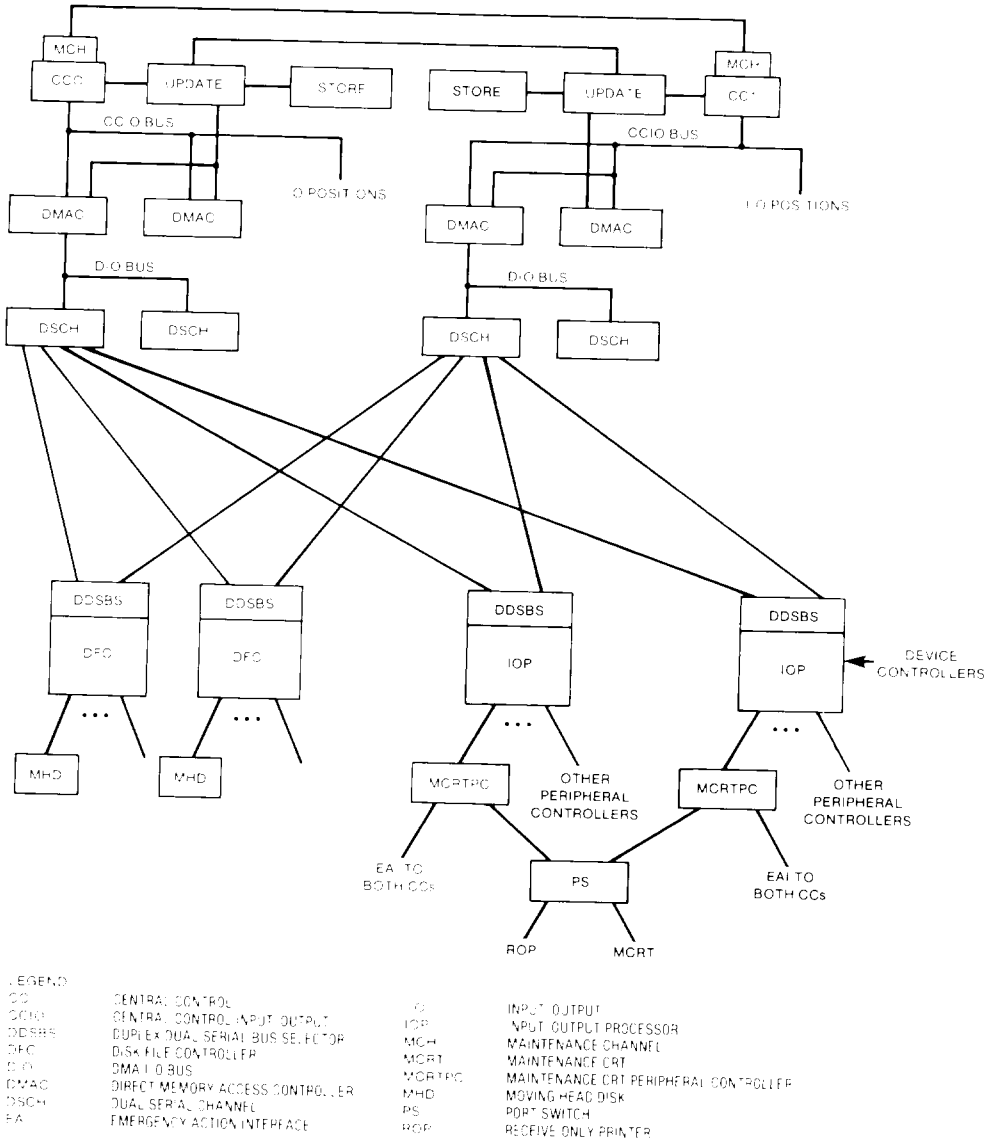


Figure 2. 3B20 Duplex Computer Architecture

2. 3B20D HARDWARE ARCHITECTURE

The central control (CC), store, and most of the input/output (I/O) community are duplicated for reliability. The CCs operate in an active/stand-by or active/out-of-service mode. The duplicated stores are normally kept up-to-date by the update link. Each CC has diagnostic access to the other via the maintenance channel (MCH).

All communication between the CC and the I/O devices occurs over the central control input/output (CCIO) bus, which connects the CC to one or two DMA controllers (DMACs) and up to two other I/O positions. The DMACs provide direct DMA access to the store, bypassing the CC. The I/O positions do not use DMA and communicate with the store through the CC, using programmed I/O. The I/O positions can be application channel interfaces (ACHIs) or dual serial channels (DSCHs).

Each DMAC can control two DSCHs via the direct memory access input/output (DIO) bus. Each DSCH can control up to 16 devices. The devices (such as the Input/Output Processor, IOP, and the disk file controller, DFC) are dual ported to two dual serial channels via the duplex dual serial bus selector (DDSBS) to allow each processor access to all devices.

The DFC controls moving head disks (MHDs); each DFC can support up to eight MHDs.

The IOP controls peripheral controllers, which provide access to terminals, tape, and the maintenance CRT peripheral controller (MCRTPC). The MCRTPC supports the Maintenance CRT (MCRT) and the Receive-Only Printer (ROP). It also has emergency action interface (EAI) to both CCs to control configurations and request boots. A second MCRTPC provides backup support. The MCRTPCs are connected to a common MCRT and ROP via a port switch (PS).

Figure 3 shows the physical hardware composing the 3B20D computer. The hardware includes the following:

- Processor frame or cabinet
- 300- or 160-Megabyte Moving Head Disk frame or 340-Megabyte Tape/Disk cabinet
- Tape Unit frame
- Power Distribution cabinet (not shown).

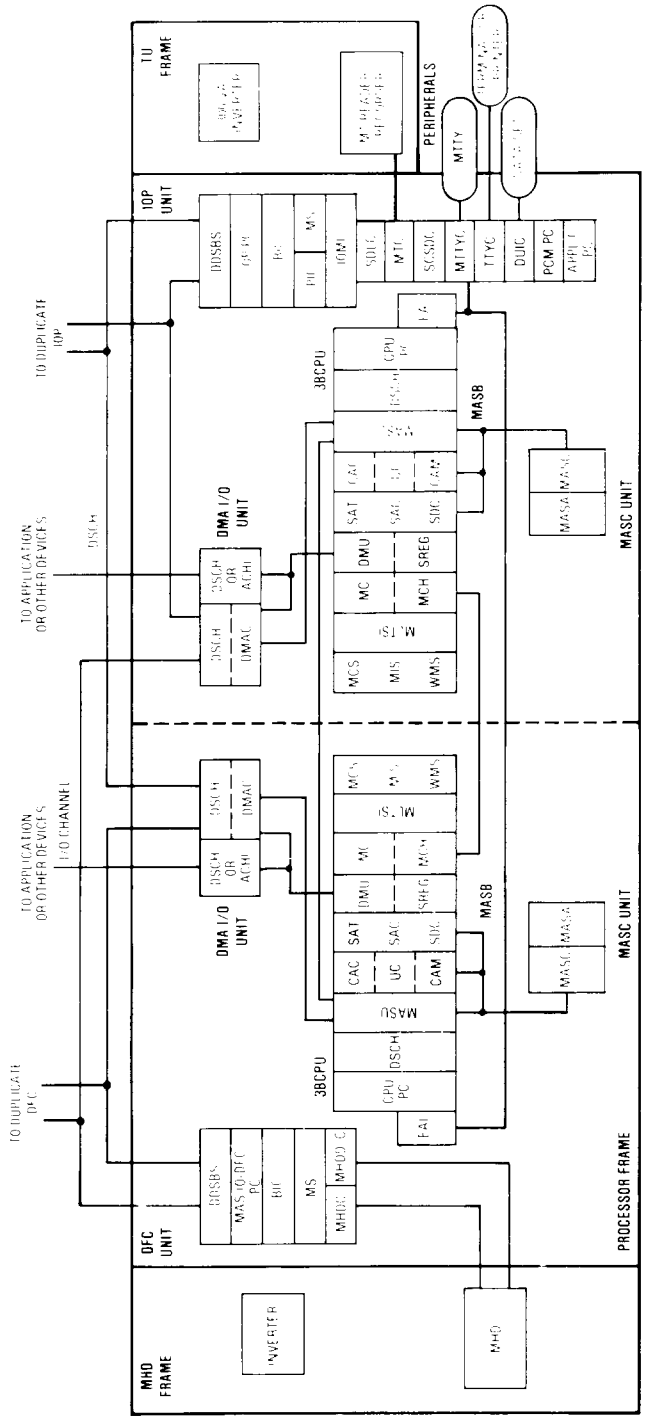


Figure 3. 3B20D Physical Hardware

All units have a distributed power arrangement that allows excellent spatial efficiency and reduced hardware.

Power in the 3B20D computer is distributed directly to the backplane from WE* converters; fuses are not used. Instead, a converter feature called current programming provides backplane protection. Each circuit pack has a resistor on it with a value related to the amount of current drawn by that version of the pack under normal conditions. The current-programming resistors of all the packs supplied by a given converter are wired together to form a parallel network whose value determines how much current the converter should normally supply. If a high-current fault occurs, causing a heavier-than-normal load on the converter, the converter shuts down and generates an alarm signal. The fault may occur either on the backplane or on a circuit pack; the protection is the same. This arrangement eliminates the cost, bulk, and impedance problems associated with fuses.

PROCESSOR FRAME OR CABINET

The Processor frame incorporates a processor, disk file controller (DFC), and Input/Output Processor into a single frame or cabinet. The single frame or cabinet has two bays, each of which contains a processor.

Central Processor Unit

The Central Processor Unit (CPU) or Central Control (CC) is the core of the 3B20D Processor frame or cabinet. It can be optionally equipped with user microcode, cache memory, microlevel test set interface and utility circuit.

The physical organization of the CPU is as follows:

- Microinstruction Store (MIS) — Stores the microinstructions necessary to decode main store instructions.
- Microstore Controller (MC) — Controls the MIS and provides the needed control signals to start microinstruction decoding.
- Data Manipulation Unit (DMU) — Performs all arithmetic and logic functions required.
- Special Registers (SREG) — Provides identify, signal, fetch, record, interface, and control functions within the CPU.

* WE is a trademark of AT&T Technologies

- Store Address Control (SAC) — Controls the store interface circuits and contains the store address.
- Cache Memory (CAM) — Provides memory with high-speed access.
- Cache Memory Controller (CAC) — Controls cache memory addressing and data transfer.
- Store Address Translator (SAT) — Translates a virtual address to a corresponding physical address.
- Store Data Control (SDC) — Interfaces the CPU to the main store (MAS).
- Main Store Update (MASU) — Allows an on-line CPU to keep both the active MAS and standby MAS updated with the latest information. The MASU allows the MAS update bus (MASUB) to be shared by both CPUs. In addition, the MASU selects CC and DMA access to the store bus.
- Maintenance Channel (MCH) — Provides a diagnostic access at the microcode level over a serial bus. MCH forms and transmits messages requiring stop-and-switch action under certain error conditions.
- Microlevel Test Set Interface (MLTSI) — Provides access to the internal CPU buses through the Microlevel Test Set (MLTS).
- Emergency Action Interface (EAI) — Provides status and communications with the processor during processor recovery phases.
- Utility Circuit (UC) — Provides utilities for software debugging and troubleshooting.
- Dual Utility Circuit (DUC) — Used in place of the UC for interface with the Field Test Set (FTS).
- CPU Power Control (CPU PC) — Provides power to the CPU.

MAS-I/O-DFC Unit

A MAS-I/O-DFC unit is located directly below the CPU unit. The Main Store portion has a main store controller and positions for 8 MAS circuit packs, allowing up to 8 megabytes of MAS. MAS can be expanded to 16 megabytes via a growth unit discussed later. Two positions are available for DMA controller circuit packs and two positions are available for I/O channel circuit packs.

Each DMA controller (DMAC) can connect to two dual serial channel (DSCH) circuit packs; and each DSCH can connect to 16 devices. The DMAC provides direct memory transfers between the main store and the peripheral devices and controls its channel(s) via the DMA I/O bus (DIOB). The two I/O channel circuit packs can be either DMA DSCHs, non-DMA DSCHs, or non-DMA application channel interfaces (ACHIs). The growth unit (described below) provides room for four more I/O channel circuit packs, subject to the limits of two DSCH per DMAC (four total) and two non-DMA circuit packs.

The DSCH and ACHI circuit packs are described below:

- DSCH — Interfaces with up to 16 peripheral devices over 16 sets of five-pair serial data cables. Two serial data streams are simultaneously transmitted (dual serial). The DSCH operates in either a word transfer or block transfer mode.

- ACHI — Provides peripheral application devices with access to the CCIOB. The ACHI provides the interface logic and connection to the CPU, allowing communication with the application device by programmed I/O.

The DFC has a microprocessor to transfer, receive, and buffer data between the CPU and up to eight MHDs. The major subcircuits are:

- Duplex Dual Serial Bus Selector (DDSBS), — Connects the DFC to the two processors.

- Microstore (MS) — Contains the program store (8K by 40 bit) for the peripheral interface controller (PIC).

- PIC and MHD control — Controls the interface with the MHD command and control circuits.

- MHD Data, Clock Parallel Serial Data Interface (MHDDC/PSDI) — Provides parallel/serial and serial/parallel data format conversions, error detections, and voltage level conversions on serial data/clock streams to and from the MHD; it also interfaces with up to eight MHDs.

Interconnection between the CPU and the MAS-I/O-DFC unit is via tape cable assemblies.

IOP Basic Unit

The IOP has its own microprocessor to transfer, receive, and buffer data between slow- and medium-speed peripheral devices and the computer. The subcircuits of the IOP and their functions are:

- Duplex Dual Serial Bus Selector (DDSBS) — Connects the IOP to the two computers.
- Bus Interface Controller (BIC) — Buffers data and commands from the DDSBS to the peripheral interface controller (PIC), and buffers data and status information from PIC to DDSBS.
- Peripheral Interface Controller (PIC), containing a controller and a microstore — Manages the peripheral controllers (PC) through the input/output microprocessor interface.
- Input/Output Microprocessor Interface (IOMI) — Buffers the information between the PIC and the PCs.
- IOP Power Controller — Provides power to the IOP unit.
- Peripheral Controllers (PCs), microprocessor-based controllers — Serve as intelligent interfaces between the input/output microprocessor interface and slow- to medium-speed peripheral devices. Communication between the IOMI and the PC is through a dual-access memory. PCs are grouped into four communities, two in the IOP basic unit and two in the growth unit (discussed below). Each community is powered separately and supports up to four peripheral controllers. Therefore, the IOP basic unit holds eight PCs, and the IOP supports a total of 16.
- Peripheral Controller Memory Power Control — Provides independent +12 and -5 volts for peripheral controller memories and j12 volts for communications interfaces.

Growth Unit

A growth unit can be located between the MAS-I/O-DFC unit and the IOP basic unit. This unit has positions for eight 1-megabyte MAS circuit packs (8 megabytes, giving a per processor total of 16 megabytes), four I/O channels, and eight peripheral controllers. Tape cables connect the MAS growth unit with the store.

Each peripheral community (2 and 3) in the growth unit also has independent ±12 volts.

300-MEGABYTE MOVING HEAD DISK FRAME

The 300-Megabyte MHD frame contains three major units:

- Disk Inverter
- Control Panel
- MHD unit.

The disk inverter converts -48 Vdc from the local power source to 208 Vac to power the disk drives.

The control panel manually powers up/down the MHD unit and inverter. In addition, it provides the interface to the system alarms.

The 300-Megabyte Moving Head Dick unit provides the disk memory storage medium for programs and data.

160-MEGABYTE MINIMODULE DISK FRAME

The 160-Megabyte Minimodule Disk (MMD) frame contains up to three 160-megabyte fixed media disk drives and up to three disk inverters.

The 160-megabyte disk drives provide the disk memory storage medium for programs and data. The 160-megabyte disk has nonremovable storage media sealed into a plastic disk module. The modules are "clean-room" assembled to prevent environmental contaminates from damaging the media and heads.

The disk inverters convert -48 Vdc from the local power source to 120 Vac to power the disk drives. The inverter also contains the power control for the MMD.

TAPE/DISK CABINET

The Tape Disk cabinet contains from one to eight 340-megabyte fixed media disk drives, or a tape unit and up to four 340-megabyte fixed media disk drives. Each disk has its own power supply.

The 340-megabyte disk drives provide the disk storage medium for programs and data. The 340-megabyte disk has nonremovable storage media sealed into a plastic disk module. The modules are "clean room" assembled to prevent environmental contaminates from damaging the media and heads.

The tape unit contains a nine-track Magnetic Tape Drive.

TAPE UNIT FRAME

The Tape Unit frame contains the nine-track Magnetic Tape Drive and 300VA Inverter Synthesizer. The Magnetic Tape Drive performs tape read/write functions and controls tape movement. Microprocessor circuitry within the drive controls tape formatting. The Inverter Synthesiser inverts -48 Vdc to provide 120 Vac for the Magnetic Tape Drive.

POWER DISTRIBUTION CABINET

The Power Distribution cabinet distributes -48 Vdc to all units within a 3B20D configuration. This cabinet contains filter fuse panels and a control panel.

As an option, this cabinet can also contain the filter fuse, control panel, and ac-to-dc rectifiers to convert the 208 Vac power to -48 Vdc required for the 3B20D computer.

An uninterruptible power supply is available for continuous power.

The primary 3B20D system software is the UNIX RTR operating system. UNIX RTR includes procedures that allow users to efficiently share system resources, such as computer time, storage space, and peripheral devices. Software interfaces for the 3B20D computer provide maximal portability, achieved through the UNIX environment, C language, and IS25 assembly language common instruction set.

BASIC SYSTEM STRUCTURE

UNIX RTR provides a high-reliability, real-time operating system and comprises a kernel and a set of modular, cooperating processes (Figure 4).

UNIX RTR and application processes communicate via messages, events, interprocess traps, shared memory, UNIX pipes, and the file system. UNIX RTR provides four executing environments or virtual machines. These are:

- Kernel
- Kernel Process
- Supervisor Process
- UNIX Process.

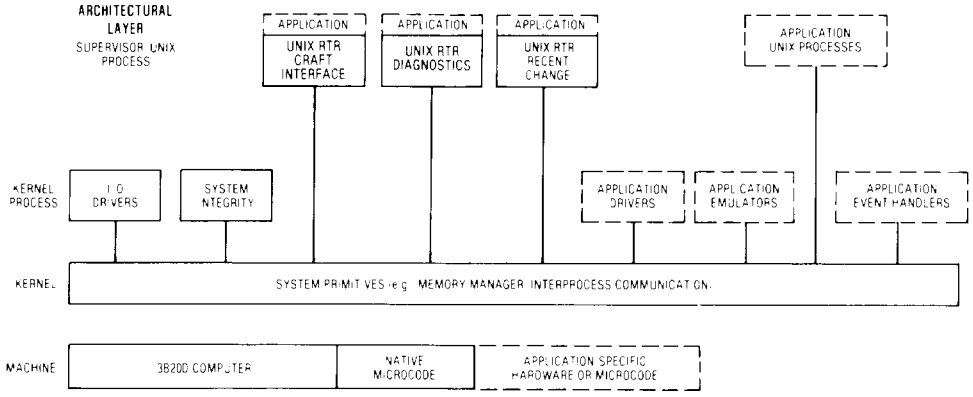


Figure 4. UNIX RTR Functional Software

Successive levels restrict access rights, and further remove users from the details of the physical machine and more primitive levels.

Applications can introduce code at the kernel process, supervisor process, or UNIX process levels.

Kernel Level

The UNIX RTR kernel is what some operating systems might call a "nucleus" or an "executive". The kernel's primary functions are:

- Bootstrap and initialization
- Timing
- System and maintenance primitives
- Scheduling
- Memory management
- Job queues
- 3B20D control.

The kernel is closest to the hardware, controls computer hardware directly, and does not depend on other operating system services. Significant portions of the kernel are written in C language; however, some of the kernel is written in assembler.

Kernel Process Level

Kernel processes are driven by hardware and software interrupts, are locked into main memory, and can execute certain privileged instructions. Kernel processes are used for functions with stringent real-time requirements, such as peripheral I/O and file system management.

Programs at the kernel process level are also strongly hardware related. These programs are structured as limited-service processes that access hardware using kernel calls, and that communicate with each other through interprocess messages. Kernel processes are written in C language. They are dispatched in a priority driven, preemptive manner, with the kernel process of highest priority (2 through 15) running first.

UNIX RTR uses this level for the file manager and for individual device (hardware unit) drivers. Code for applications programs may be written at this level.

Supervisor Process Level

Supervisor processes can use services provided by the kernel and kernel processes, and are scheduled in a time-shared manner. Within UNIX RTR, the process manager is a supervisor process.

UNIX Process Level

A shared library provides a UNIX environment for supervisor processes, thus hiding UNIX processes from UNIX RTR. New software can use the UNIX calls, and existing UNIX software can be ported to UNIX RTR under this environment. The library translates UNIX calls into sequences of UNIX RTR supervisor calls.

Conceptually, UNIX and supervisor processes are different, but they are the same from the operating system's viewpoint. UNIX processes can use many of the functions provided to supervisor processes, such as physical and asynchronous I/O, interprocess communication through messages and events, fault handling, and memory management.

Processes and Segments

Processes are the basic executable entities in UNIX RTR (Figure 5). They are composed of segments, which are portions of main memory with consecutive virtual addresses.

The basic segment types used in processes are text (executable code) segments, data segments, stack segments (used for C-function call interfaces), and process control block (PCB) segment. Segments consist of pages and are the basic swapped entity (that is, all pages in a segment are swapped simultaneously into and out of memory). Each segment can have a maximum of 64 pages; each page contains 2048 bytes.

When a process is created, its segments are created and loaded into main memory by the memory manager. The segments become known to the system through an entry made in a segment descriptor table. This entry remains until the segment is no longer needed for any process to execute. Segments can also be transferred to secondary memory if the main memory is needed for a higher priority process.

Segments are identified by the physical address of their segment descriptor table entries. Supervisor/UNIX process segments are also referenced by the segment numbers used to index the process' segment list table.

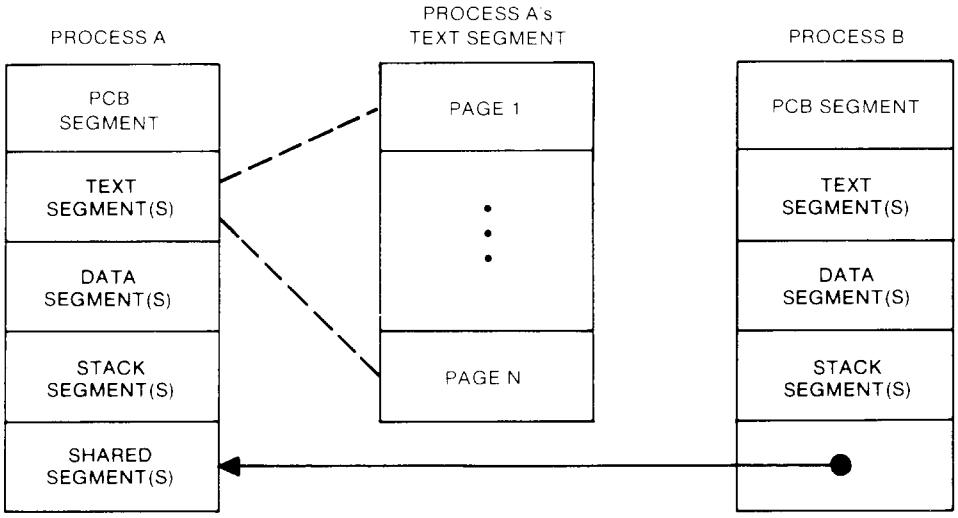


Figure 5. UNIX RTR Processes

Segments can be shared between processes to avoid duplicating commonly used procedures or to share data. This increases machine efficiency because a process already in main memory need only attach the segment to its virtual address space.

Segments can also be declared unswappable to allow them to be moved within main memory but not to be swapped to secondary storage.

Segments can contain between 1 and 128K bytes. Supervisor/UNIX process segments generally grow dynamically in 1-byte increments by using appropriate system primitives.

Segments can be executable, readable, or writable. Code segments are usually executable but not writable. The memory manager maintains protection keys to record the current protection status of a segment.

A process is a collection of segments in main memory that forms an executable entity. Each process can have 128 segments in its virtual address space. All processes contain a minimum of three segments: a code segment, a stack segment, and a PCB. A process may also have a data segment. The PCB defines the virtual address space of the process and saves the state of the process if it is preempted.

Processes are created in main memory by the operating system and disappear when they terminate. During system boot, the file manager, memory manager, system scheduler, and process manager are loaded into main memory along with the kernel.

Processes can be in a transient state (being created, being loaded into memory, being terminated); or in a stable state (ready to run, scheduled, running). Additionally, nonkernel processes can be in a nonbusy waiting state, called roadblock.

A supervisor-level process is roadblocked when it asks to receive a message that is unavailable. When the needed message is available, a message arrival event is sent and the roadblock is broken.

SYSTEM MANAGEMENT SOFTWARE

UNIX RTR manages system resources to allow users to efficiently share system capabilities, such as processor time, storage space, peripheral devices, and information. UNIX RTR regulates the system software via process management, file system management, performance measurement, utility management, and data base management.

Process Management

Because UNIX RTR has modular, independent processes, the concept of the process is to UNIX RTR, which is composed of a kernel and a collection of cooperating, concurrent processes.

Most of the functions that must be performed to start a process are performed by the process manager, a supervisor process. The scheduler, a special system process closely associated with the kernel, schedules all supervisor and UNIX processes for execution. The UNIX RTR memory manager, another special process, controls the contents of real memory by directing the movement of process segments between real memory and secondary storage, or swap space.

The UNIX RTR operating system also provides interprocess communication and synchronization mechanisms and has an interrupt structure that allows a currently executing program to be interrupted for maintenance or other purposes.

File System Management

The UNIX RTR file system is similar to the basic UNIX file system. Both are organized as a rooted tree, where each node is a directory and each leaf, a file. The file manager locates files and devices, opens all files and special device files, checks user access permissions, ensures the device handlers are enabled when needed, and translates I/O requests for logical files on block-structured devices. Unlike UNIX, UNIX RTR provides contiguous as well as normal UNIX files. Contiguous files are useful for data bases and object modules.

Performance Measurement Facilities

Many tools and techniques are available to the application developer and operating system programmer for measuring and evaluating the UNIX RTR software components and software activities. Operating system instrumentation can be accessed through software to measure and observe performance at a system level and for specific software activities.

Utility Management

The utility manager is a special kernel process that provides services for the UNIX RTR debugging tools. These services require access to data in the kernel's address space and, in some instances, write access to some other process' text segments. The services include:

- Identification of all currently running processes with a given utility ID
- Setting and removing of software breakpoints in a process' text segment
- Attachment and detachment of the utility process and the utility circuit interrupt
- Setting up of the process' address space when the utility circuit "fires."

Data Base Management and Recent Change

The Data Base Management System (DBMS) for the 3B20D operating system controls the Equipment Configuration Data (ECD) data base, the System Generation (SG) data base, and the Plant Measurement (PM) data base. A Recent Change system allows controlled changes to these and other application data bases.

INPUT/OUTPUT AND MAINTENANCE FEATURES

UNIX RTR is a flexible operating system that provides eight subsystems to efficiently share system resources while providing services unique to each subsystem.

Input/Output Subsystem Software

UNIX RTR supports communication with peripheral devices through a set of drivers and device handlers, which execute as kernel processes. These drivers isolate the user from the details of the peripheral system and provide efficient use of the peripheral devices by allowing access to them on an equitable basis.

The architecture of the I/O software closely resembles I/O hardware architecture. The I/O Processor (IOP) driver (IODRV) and the device handlers manage the IOPs, the peripheral controller (PCs), and the peripheral controller subdevices (PCSDs). The Disk Driver (DKDRV) manages the disk file controller (DFC) and the Moving Head Disk (MHD).

Nondeferrable Maintenance

All maintenance functions for the UNIX RTR operating system are classified as either deferrable or nondeferrable. Deferrable functions execute under the UNIX supervisor. The nondeferrable maintenance functions are closely associated with the UNIX RTR kernel and cannot be swapped. They include fault recovery, system initialization, and configuration management.

System Integrity Monitor

The UNIX RTR System Integrity Monitor (SIM) is primarily responsible for the software integrity of the UNIX RTR operating system. SIM is responsible for ensuring the integrity of UNIX RTR system initialization, for administering the 3B20D hardware sanity timer, and for monitoring UNIX RTR integrity processes and overload conditions. SIM also provides an interface between UNIX RTR and the Application Integrity Monitor (AIM).

Audit Control Subsystem

Several audits verify the validity of UNIX RTR data structures on a regular basis to determine the sanity of the system. These audits are controlled, scheduled, and dispatched by the SIM process.

Diagnostics

The 3B20D computer's high reliability is ensured through speedy repairs to duplicated hardware. Comprehensive diagnostic tests and multi-featured control software help to isolate faulty hardware anywhere in the computer. Diagnostics run automatically as a result of system abnormalities and optionally at preset times of the day. They may also be requested manually by actions at the 3B20D computer or by input messages from local or remote terminals. Some of the outstanding features of the diagnostics are:

- Noninterference with normal system operations
- Easy maintenance and update for new or growth hardware

- Special purpose test design language that facilitates test interpretation
- Common test database that covers all versions of the hardware
- Partitioning of tests into phases associated with specific hardware functions
- Control features allowing selective test execution and variable degrees of test result detail
- Trouble location process that outputs a list of suspected faulty circuit packs.

Additional features are provided in the diagnostic control software to extend diagnostic capabilities to *3B* 20D customer systems.

Program Update Facilities

UNIX RTR accepts software changes through three types of updating, none of which interferes with its normal operation.

- Field update — The method used by UNIX RTR to correct software errors and add small enhancements.
- Emergency update — The methods used to immediately correct severe, service-affecting problems.
- System update — The procedure used to introduce new versions of UNIX RTR and application software into the 3B20D system.

Operator Interface

The 3B20D's operator interface provides a complete maintenance package for the system. It gives the operator:

- Immediate feedback on system status
- Direction for correcting failures
- Absolute control over the system independent of 3B20D hardware and software sanity
- Protection from dangerous procedural errors.

The operator interface uses forms and menus on a graphics terminal. The application can add their own forms and menus to those provided by UNIX RTR.

Although composed of hardware, firmware, and software, the largest component of this operator interface package is the collection of software processes.

The main processes in this collection are the craft shell (CFTSHL), Control and Display Administration (DAP), and the output spooler.

Field Operations Tools

The following field operations tools help in performing routine maintenance and trouble-clearing activities:

- Physical Disk-to-Tape Writer (PDT) — Writes the image of selected disk partitions to tape. These tapes can then be used to rebuild the disk, if necessary.
- Clock Daemon (CRON) — Executes commands at specified dates and times.
- Generic Access Package (GRASP) — Provides a set of debugging utility functions used to dump, copy, and (with restrictions) overwrite data at any addressable location in the system.
- Plant Measurement System (PMS) — Reports, at specified intervals, information such as counts of the number of interupts, reinitializations, audits and audit failures, alarms, and the amount of time various units are simplexed due to maintenance activity.
- Field Test Set (FTS) — Provides a debugging tool that monitors processes executing on the 3B20D computer.

Application developers and 3B20D programmers can write code in three different programming language levels, depending on the program purpose. These languages are:

- Microinstruction language
- Assembly language
- C language.

Each language level is less dependent on hardware than the one before, and each has its own special use.

Microcode instructions directly manipulate the hardware elements of the CPU. Microcode interprets the object code and produces the desired machine responses. Each microcode instruction fits into a specific format that translates into a binary string. Voltage levels (the string's physical form) operate specific hardware gates, causing the computer to complete the action specified by the instruction.

Assembly language instructions compose the complete set of actions of the processor. These actions include basic arithmetic functions (add, subtract, etc.), logic functions (and, or, etc.), and manipulate instructions (jump, mask, etc.). The 3B20D computer uses a common core of assembly language instructions called IS25. Additionally, it has unique instructions that reflect its particular capabilities. All assembly language instructions, whether written directly by a programmer or produced by the C language compiler, are translated into object code by the 3B20D assembler.

C is a systems programming language. The programmer needs only basic familiarity with processor hardware to write a program because the C statements represent problem-solving actions. The C compiler translates each high-level instruction into a series of assembly language instructions that produce the desired results.

Microcode statements are organized into their full format by the micro-assembler (MICA). Assembly language statements assemble as short programs of object code instructions. C language statements compile as short programs of assembly instructions.

THE C LANGUAGE

C is a general-purpose programming language. It has been closely associated with the UNIX system because it was developed on that system, and because UNIX and its software are written in C. However, the language is not tied to

any operating system or machine. Although it has been called a "system programming language" because it is useful for writing operating systems, it has also been used to write major numerical, text-processing, and data base programs.

C is a low-level language in that C deals with the same type of objects that most computers do, namely characters, numbers, and addresses. These may be combined and moved around with the usual arithmetic and logical operators implemented by actual machines.

C provides no operations to deal directly with composite objects such as character strings, sets, lists, or arrays considered as a whole. For example, there is no analog of the PL/I operations that manipulate an entire array or string. The language does not define any storage allocation facility other than static definition and the stack discipline provided by the local variables of functions. Finally, C provides no input/output facilities: 'read' or 'write' statements, and wired-in file access methods are not available. All these higher-level mechanisms must be provided by explicitly called functions.

Similarly, C offers only straightforward, single-thread control flow constructions such as tests, loops, grouping, and subprograms. Multiprogramming, parallel operations, synchronization, or co-routines are not available.

Although the absence of some of these features may appear as a deficiency, C does have advantages. Because C is small, it can be described in a small space, and learned quickly. A compiler for C can be simple and compact. Because the data types and control structures provided by C are supported directly by most existing computers, the run-time library required to implement self-contained programs is small. Each implementation provides a comprehensive, compatible library of functions to do I/O, string handling, and storage allocation operations. Because all needed functions must be called explicitly, they can be avoided if not required; they can also be written portably in C.

Because the C language reflects the capabilities of current computers, C programs are normally efficient and writing in assembly language is not necessary. In addition, C is independent of any specific machine architecture. Therefore, portable programs (programs that can be run without change on different computers) can also be written.

In C, the fundamental data objects are characters, integers of several sizes, and floating point numbers. Additionally, a hierarchy of derived data types is created with pointers, arrays, structures, unions, and functions.

C provides the fundamental, flow-control constructions required for well-structured programs: statement grouping; decision making ('if'); looping with the end test at the top ('while', 'for'), or at the bottom ('do'); and selecting one of a set of possible cases ('switch').

C provides pointers and the capability of address arithmetic. The arguments to functions are passed by copying the value of the argument. In addition, the called function cannot change the argument in the caller. When 'call by reference' is desired, a pointer may be passed explicitly, and the function may change the object to which the pointer points. Array names are passed as the location of the array origin. Therefore, array arguments are effectively called by reference.

Any function may be called recursively, and its local variables are created anew with each invocation. Function definitions may not be nested, but variables may be declared in a block-structured fashion. The functions of a C program may be compiled separately. Variables may be internal to a function, external but known only within a single source file, or completely global. Internal variables may be automatic or static. Automatic variables may be placed in registers for increased efficiency.

3B20D ASSEMBLY LANGUAGE AND MICROCODE

Instruction Set 25 (IS25) provides an intermediate language for UNIX-level code written in C and is designed to produce portable assembly language programs. Although the use of assembly language programming is not recommended, some code might be written at least partially in assembly language. Because of this need, macros are available to interface the assembly language routines with C code.

All assembly language instructions, whether written directly by a programmer or produced by the C language compiler, are translated into object code by the 3B20D assembler. The object code loaded into the 3B20D computer is executed by the 3B20D resident microcode.

IS25

The IS25 instruction set consists of a complete assembly-level description of data and common instructions and the object-level description of data. It is designed to be a space- and time-efficient instruction set for compiled C programs, and is based on extensive measurements of production C programs.

An IS25 program consists of three sections: 'text', 'data', and 'bss'. The 'text' section contains all executable instructions and some initialized data, the 'data' section contains the initialized data that is not in the 'text' section, and the 'bss' section contains only uninitialized data. Each section may begin at any address that is a multiple of four and consists of a contiguous sequence of bytes.

Microcode

Microcode is written when no other language provides the facilities, when increased performance is required, or when software written for other computers is run on the 3B20D computer (emulation).

The microcode controls the functions of the 3B20D hardware. For example, accessing registers, transferring data between registers, and transferring data from microinstructions to the registers. I/O and store functions are done by transferring data to special function registers that affect I/O devices or main store. Internal and external registers are the two main groups of registers in the 3B20D computer. Most registers are external to the arithmetic logic unit (ALU). The only internal registers are the 16 general purpose registers and the Q register.

COMMON OBJECT FILE FORMAT

Object files are specified by section, physical address, and virtual address. A section is a portion of the object file that is the smallest unit of relocation; it is treated as a separate, distinct entity. In the default case, there are three sections: '.text' (instruction), '.data' (initialized data), and '.bss' (uninitialized data). Additional sections are used for accommodating multiple text segments, shared data segments, or user-specified sections. The physical address is the physical location in memory where a section is loaded. The virtual address is the location that all relocatable references in a section assume the section will occupy at execution time.

An object file consists of a file header, optional header information, a table of section headers, the data corresponding to the section headers, relocation information, line numbers, and a symbol table. The symbol table of the common object file format is more extensive than that for any previous C-based development system, allowing complete access to symbols declared in a C program. Such access simplifies C source-level symbolic debugging.

Several software systems are available to develop, augment, support, or run under the UNIX RTR operating system. These systems are described in this section.

SYSTEM ALPHA

System Alpha is a reliable, integrated, time-proven system designed to support the development of applications software for the 3B20 Duplex Computer. Currently, System Alpha runs on the 3B20S/A computer off line from the 3B20D computer and its operating system. The capabilities of System Alpha include:

- Software generation for both high-level and assembly language programs
- Source file change management
- Multimachine modification tracking and reporting
- Individual product construction environments
- Source-language level debugging facilities
- Load building tools.

System Alpha (Figure 6) comprises six distinct, interactive components, as follows:

- Change Management System (CMS) — Controls and records the source change activities of developers, administrators, and test teams associated with the software development project.
- Modification Request Tracking System (MRTS) — Serves as a central master data base from which administrators control and track all software changes and generate tracking and status reports.
- Build — Allows the construction of independent software packages based on the concept of sharing common files. The sharing of common files is permitted by a viewpath, which specifies the selection of a particular version of software.
- Load Building Tool Package (LBTP) — Combines system source files into an integrated product.
- Software Generation System (SGS) — Provides through a set of tools a cross-development environment for the generation of 3B20D computer executable modules. The SGS includes a C-language compiler, assembler, link editor, process loader, and other utilities.
- DART — Provides C-language program debugging capability. DART permits developers to inspect and change the execution of a program at the C-language level.

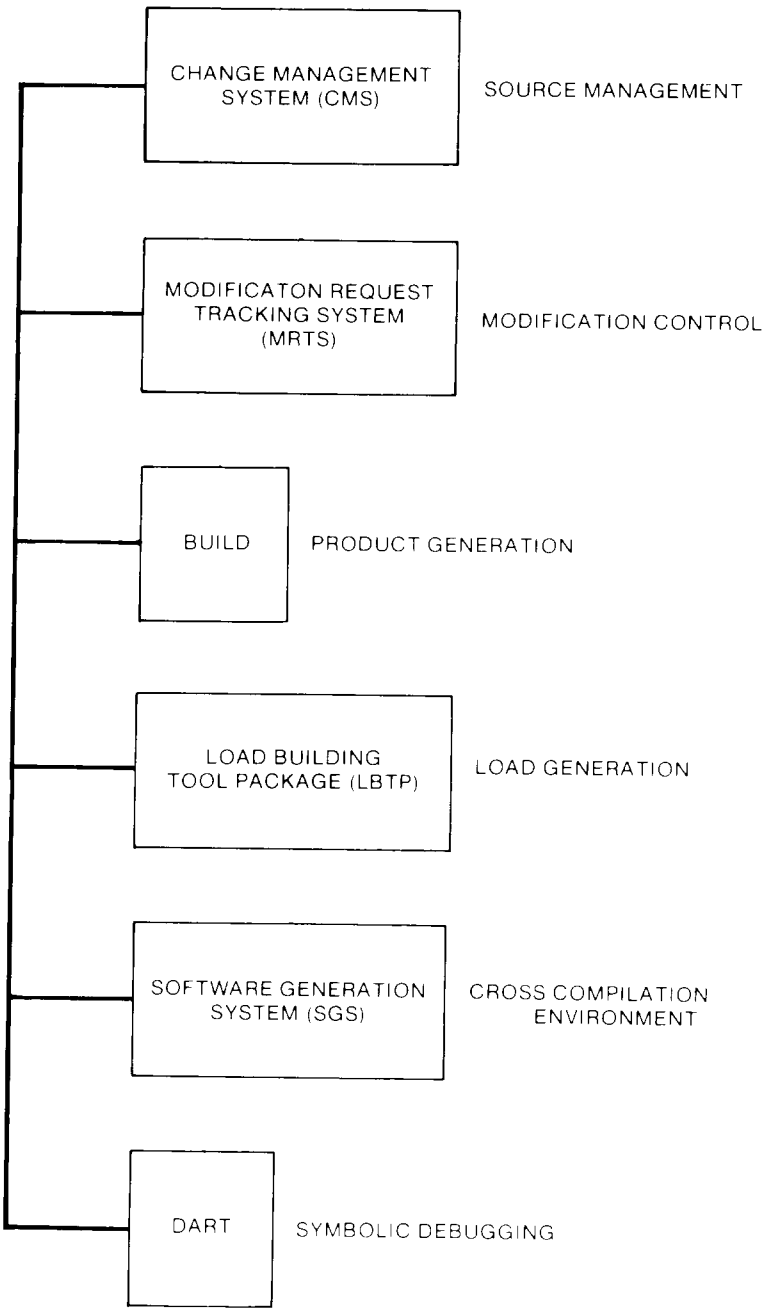


Figure 6. System Alpha

Change Management System (CMS)

CMS allows administrative control of project software by monitoring software changes. It also provides system security by controlling access to source files and project information. System integrity is guaranteed by this access control and by system audit routines. The CMS uses the Source Code Control System (SCCS) proven in the UNIX system and its own relational data base to track and relate each change made in a source file to a modification request (MR). The two major components of CMS are MR management and the source file managment.

CMS allows project managers to track the status of MRs from creation through system integration by recording MR information in the relational data base. Multiple releases, supported in CMS, allow projects to have more than one active version. Managers can track field fixes after a release as well as active development before a release. This information is accessible to both developers and administrators to permit coordination of changes to the system and communication of the status of these changes.

The source file management component of CMS records changes to source files and retrieves those changes on an MR basis. This component is easy to use and provides a simple retrieval mechanism while relieving developers of intricate, error-prone retrieval of MR-specific changes. In addition, source files can be observed during development and after field fixes have been incorporated. A desirable feature is the option to include selected portions of source files at retrieval time.

Modification Request Tracking System (MRTS)

MRTS tracks software changes on one or more machines via a master modification request (MR) data base. Each MR record in this data base contains information, such as the MR status, the releases (or system versions) affected by the MR, the MR due date, the supervisor responsible for the MR, and the portions of the software system that are affected by the MR.

MRTS has several advantages, as follows:

- A set of MR status-changing commands is provided to track the state of the MR in the development cycle. MR states such as under study, being fixed, assigned, submitted, under test, and released, are available.

- Each MR assigned to a developer is identified with a CMS instance. An instance is defined as a set of SCCS controlled source files and the relational data base that tracks the changes based on MRs. Within

MRTS, one associated CMS instance appears for each subsystem. The MR status-changing commands communicate applicable MR information to the CMS instance to ensure consistency between the MRTS and CMS MR data bases.

- MRTS provides an interactive report generation capability. Based on values found in the record, a user specifies which MR records are to be included in a report. The information included within the record, the order of the records in the report, and the format of the report are also specified by the user.
- Audits guarantee consistency between MRTS and CMS data bases and the consistency of the MRTS data base itself. These audits can be run at regular intervals. MRTS is a user-friendly system because most commands are entered in interactive sessions, with a user prompted for specific information.

Build

The *build* component benefits everyone involved in a software development project. *Build* automates product generation and regeneration ensuring that developers and administrators can produce correct and up-to-date products. *Build* is an extension of the UNIX system product generation *make* command. Like *make*, *build* reads an instruction file, called a *makefile*, that contains a description of the software to be generated. The *makefile* lists all files needed and commands executed to produce a product. Unlike *make*, *build* also allows each user independently to make changes to selected files and produce a unique product that incorporates those changes. Only the files selected for change need be duplicated; all other necessary files can be shared among all users.

Sharing of common files is permitted by a viewpath. A user's viewpath specifies a list of UNIX system directories, called nodes, to be searched when a file is required in the generation process. The viewpath instructs *build* to look for files in a private developer-controlled node and then in a project-controlled node that contains the files common to all developers.

To develop a new product or change an existing one, a developer places the changed or new files, including *makefiles*, in a private development node. The viewpath node list is set to look first in the developer's node, then in the common node. *Build* is run and, as a result, a product including the changed files is generated in the development node. Another developer, test team, or administrator can be simultaneously constructing the same product but with different changes. If the product already existed, the source files needed to construct it and the product itself are still in the common node. When the new changed product is tested, it is moved into the common node along with the necessary changed files and is available to the entire project.

Load Building Tool Package (LBTP)

Load building is the task of combining system source files into an integrated product. Load building is aided by a package of load-building tools. These tools assume the use of CMS, *build*, and the underlying concept of a modification request (see CMS and *Build* sections). The inclusion or exclusion of a set of source changes from a load is determined by including or excluding the controlling MRs. Therefore, load content is modification driven.

The tools extract source files from multiple CMS instances, possibly on different machines. The tools also ship these files to a central, designated load-building machine. Tools on the load-building machine arrange the source files and use the build command to create the system products. Other tools update the development machines with the new products and source files.

The package of tools and the underlying procedures eliminate manual handling of individual source files. A record of the MRs and associated source files for a given load is archived.

Software Generation System (SGS)

SGS includes the tools to compile C-language programs and to assemble assembly language into UNIX-level object programs.

A single command can be used to control the C-language program compilation process. This command involves a source code preprocessor, the C-language compiler and optimizer, assembler, and link editor. These tools, in turn, convert a collection of C-language programs into a single-object program with addresses that are either relocatable or absolute.

The preprocessor provides a macro expansion facility and directives for sharing common source or header files. Header files typically contain data and macro declarations shared by many programs.

A C-language compiler and object code optimizer also are provided with SGS. These tools produce the input to the process loader, which creates the process file for execution on the 3B20D computer.

Full, high-level language support utilities, such as a disassembler, an intermediate loader, and a process file examiner compliment the tools mentioned and are provided by SGS.

DART

DART is a symbolic debugger for C-language programs that run on the 3B20D

computer. DART permits the user to interactively examine and modify the execution of C-language 3B20D computer application programs. This debugger provides execution controls that allow the user to set breakpoints and to specify action lists for execution when those breakpoints are reached.

Possible breakpoints include the function entry point, the function exit point, a particular C-language source line number in the function, or a particular labeled line in the function. Breakpoints can be set at various points in a function, removed, enabled, disabled, or listed with their associated statistics. The action lists are written in DART's command language.

The language is block structured and contains many C-like constructs such as while loops, if-else conditional statements, and function calls. Action lists can also include formatted or unformatted dumps of variable values and gotos to C-source line numbers or labeled lines. DART allows program examination of variable values, system registers, and virtual memory, and can exercise the program to determine its validity.

OTHER DEBUGGING AND TESTING AIDS

Other debugging and testing aids available are:

- Field Test Set (FTS)
- Generic Access Package (GRASP)
- IBROWSE
- File System Debugger (FSDB)
- CX
- Microlevel Test Set Interface Program (MIP).

Field Test Set (FTS)

A system problem could occur that leaves the system inoperative.On-line utility systems, which assume basic functionality and nonoverload conditions, are not appropriate in this case, but the FTS was designed specifically to meet this need in the field. It is strictly a monitoring device and does not affect computer performance or rely on system operability. This noninterfering characteristic is also important when trying to isolate problems at a field site carrying a heavy system load.

The FTS is a small, portable unit that is easily transported and connected to the 3B20D computer through the dual access utility circuit (DUC). The external

FTS unit connects to the DUC through an eight-foot cable. The FTS intelligence is contained in this external unit that includes a microprocessor with memory management, one megabyte of RAM, and a cassette transport. User access is provided through a local or remote terminal with phone access provided by the FTS.

The FTS/DUC system supports several trace modes. A transfer trace records program addresses of all transfers executed by a program. A function trace records program function call/return sequences. At a higher level, a record may be kept each time a different process begins execution. Multiple trace modes can be active simultaneously. Information is recorded into the trace memory under control of a variety of sophisticated matcher circuits. Matchers are included for address, address range, data, access type (read, write, or read/write) and process matching. They can be used to restrict tracing to particular processes, address ranges, etc.

The trace memory is operated in either a pre-trace or a post-trace mode. In the pre-trace mode, the trace memory records information until it receives a stop signal. The trace data then represents program flow leading up to that event. In the post-trace mode, the trace memory starts recording on receiving a start signal and stops when the trace memory is full. This provides a history of program flow after a specific event.

Generic Access Package (GRASP)

Software problems may occur when the system is functioning and providing services in a non-overload environment. Such problems can be solved in the 3B20D computer by using GRASP. GRASP is an on-site 3B20D resident tool for software debugging. Because it supports an interface to the DUC, GRASP provides a set of trace and data access trap functions similar to those provided by the FTS. In addition, it can place multiple breakpoints in code, print the contents of memory and many machine registers, and (with some restrictions) write memory and registers regardless of whether the DUC is available. GRASP has a regulating mechanism that prevents it from taking too much real time and thereby interfering with processing or driving the system into overload. GRASP is also useful when multiple breakpoints are needed, when breakpoints must be planted in several processes simultaneously, where register information is needed, or when investigation must be done remotely from a central maintenance facility.

IBROWSE

IBROWSE is an interactive tool used to symbolically examine stable data of any UNIX RTR process in main memory; it allows users to view the operating system

tables and message buffers in the kernel address space. IBROWSE also can be used on an off-line support computer to analyze tape dumps of main memory taken at field sites.

BROWSE

BROWSE allows interactive perusal of low-level access (LLA) data bases. It can be used to verify data and data base structures independent of data base application programs. It can also be used to find corrupted data base structures and to repair damage.

File System Debugger (FSDB)

FSDB allows interactive examination of the file system. It can be used to repair a damaged file system rather than restoring it from an old, saved copy.

CX

CX is a noninteractive, nonsymbolic program that dumps the contents of core files. It runs at low priority on the 3B20D computer or on a support computer making it practical to use in any environment.

Microlevel Test Set Interface Program (MIP)

MIP is a low-level test system aimed primarily at hardware register and microcode access. It consists of an interface circuit that plugs into the 3B20D computer and an external control circuit. Because the Microlevel Test Set (MLTS) is equipped with an RS-232 interface and a 212A data set, it may be configured with a terminal on-site or may be operated from a remote location.

MIP provides interfering read/write access to all internal hardware and firmware registers and is the only tool that provides access to the computer's microcode. MIP provides microcode breakpoints, can read and write microstore and main store locations, and can read and write machine registers that are not accessible to other troubleshooting tools. Although MIP is used primarily in a laboratory environment, occasionally such capabilities are required to solve problems during field tests.

ACHI	Application Channel Interface
AIM	Application Integrity Monitor
ALU	Arithmetic Logic Unit
API	Attached Processor Interface
APS	Attached Processor System
ASCII	American Standard Code for Information Interchange
BIC	Bus Interface Controller
CAC	Cache Controller
CAM	Cache Memory
CC	Central Control
CCIO	Central Control Input/Output
CCIOB	Central Control Input/Output Bus
CCIS	Common Channel Interoffice Signaling
CFTSHL	Craft Shell
CMS	Change Management System
CONT	Controller
CPU	Central Processor Unit
CPU PC	Central Processor Unit Power Control
CRON	Clock Daemon
CRT	Cathode Ray Tube
CU	Control Unit
DAP	Display Administration Process
DAM	Dual Access Memory
DART	Debugging a Remote Target
DBMS	Data Base Management System
DDSBS	Duplex Dual Serial Bus Selector
DFC	Disk File Controller
DIAGC	Diagnostic Control
DIO	Direct Memory Access Input/Output
DIOB	Direct Memory Access Input/Output Bus
DKDRV	Disk Driver
DMA	Direct Memory Access
DMAC	Direct Memory Access Controller
DMA I/O	Direct Memory Access Input/Output

6. ACRONYMS

DMU	Data Manipulation Unit
DSCH	Dual Serial Channel
DUC	Dual Access Utility Circuit
EAI	Emergency Action Interface
ECD	Equipment Configuration Data
FSDB	File System Debugger
FTS	Field Test Set
GRASP	Generic Access Package
I/O	Input/Output
IODRV	Input/Output Processor Driver
IOMI	Input/Output Microprocessor Interface
IOP	Input/Output Processor
IOP PC	Input/Output Processor Power Control
IS25	Instruction Set 25
LBTP	Load Building Tool Package
LLA	Low-Level Access
LSI	Large-Scale Integration
MAS	Main Store
MASA	Main Store Array
MASB	Main Store Bus
MASC	Main Store Controller
MASU	Main Store Update
MASUB	Main Store Update Bus
MC	Microcontrol
MCH	Maintenance Channel
MCRT	Maintenance CRT
MCRTPC	Maintenance CRT Peripheral Controller
MCS	Microcontrol Store
MHD	Moving Head Disk
MHDC	Moving Head Disk Controller
MHDDC	Moving Head Disk Data Clock
MICA	Microassembler
MIP	Microlevel Test Set Interface Program

MIS	Microstore
MLTS	Microlevel Test Set
MLTSI	Microlevel Test Set Interface
MMD	Minimodule Disk
MML	Man-Machine Language
MR	Modification Request
MRTS	Modification Request Tracking System
MT	Magnetic Tape
MTC	Magnetic Tape Controller
MTTY	Maintenance Teletypewriter
MTTYC	Maintenance Teletypewriter Controller
NCP	Network Control Point
PC	Peripheral Controller
PCB	Process Control Block
PCM	Peripheral Controller Memory
PCM PC	Peripheral Controller Memory Power Controller
PCSD	Peripheral Controller Subdevice
PD	Power Distribution
PDS	Program Documentation Standard
PDT	Physical Disk to Tape
PIC	Peripheral Interface Controller
PM	Plant Measurement
PMS	Plant Measurement System
PS	Port Switch
PSDI	Parallel Serial Data Interface
PSI	Peripheral System Interface
RAM	Random Access Memory
ROP	Receive-Only Printer
SAC	Store Address Control
SCCS	Source Code Control System
SCSDC	Scanner and Signal Distributor Controller
SAT	Store Address Translator
SDC	Store Data Control
SG	System Generation

6. ACRONYMS

SGS	Software Generation System
SIM	System Integrity Monitor
SMIP	Symbolic Microlevel Test Set Interface Program
SREG	Special Register
TSPS	Traffic Service Position System
TTYC	Teletypewriter Controller
TU	Tape Unit
UC	Utility Circuit
VLSI	Very Large-Scale Integration
WMS	Writable Microstore