SPERRY RAND

UNIVAC 494 REAL-TIME SYSTEM

SYSTEM DESCRIPTION

CONTENTS

1. INTRODUCTION

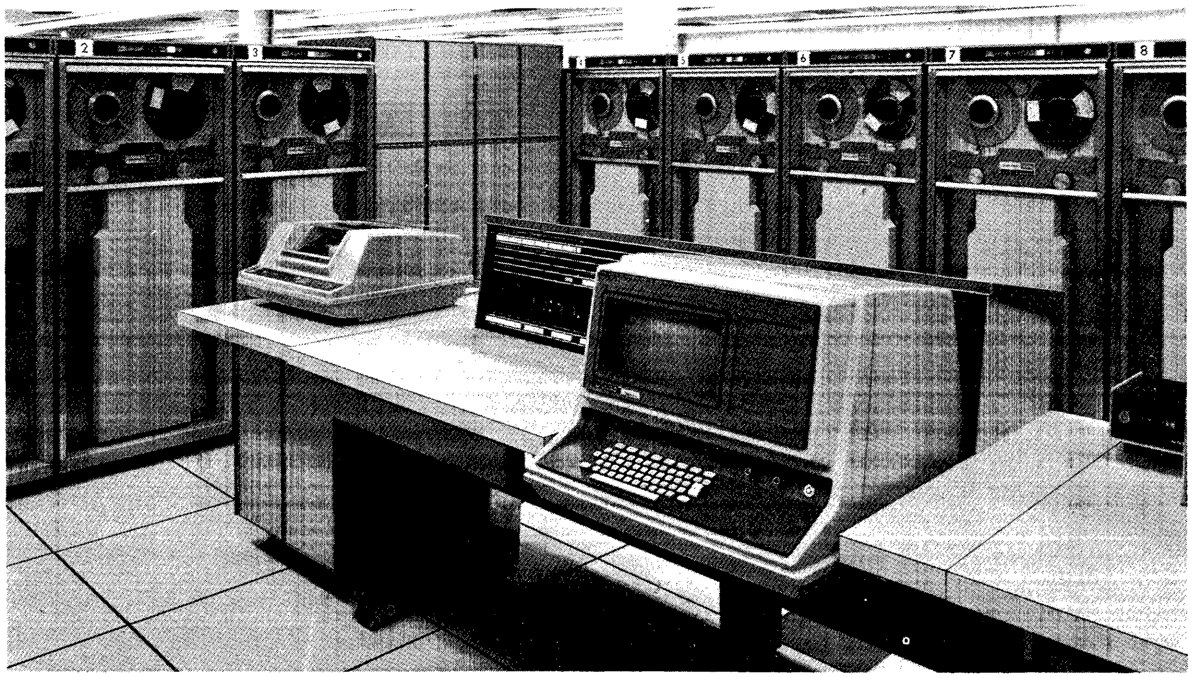


Figure 1–1. UNIVAC 494 Real-Time System

1.1. UNIVAC 494 REAL-TIME SYSTEM

The UNIVAC 494 Real-Time System is a versatile, high capacity computer system with outstanding capability in efficient response to the demands of real time processing for online application programs, while maintaining background activity in accomplishing multiple batch oriented jobs. The system is based upon the extremely high operating speed of the UNIVAC 494 computer and its ability to manipulate and communicate large volumes of data in a multiprogram environment. Information from diverse on-site and remote operations can be rapidly and effectively accessed and output through a variety of user options. The system is applicable to scientific, commercial, and communications oriented information handling operations.

1.2. SYSTEM CHARACTERISTICS

As the logical successor to the UNIVAC 490/491/492 Real-Time Systems, the UNIVAC 494 System embodies advanced concepts in computer design, system organization, and programming technology. The UNIVAC 494 computer possesses higher speeds, larger capacities, increased versatility, and other enhancements, providing greater power and flexibility in program development and execution. Selective focusing of system hardware and software resources to the specific information and data processing needs of the individual system user is a basic system concept. The modular structure of both hardware and software provides a precise blend of system components for fulfilling processing requirements in applications ranging from basic job shop operations to comprehensive, large scale information systems. Under control of the UNIVAC 494 Operating System, the system software permits maximum utilization of computer facilities and optimal specification of information requirements with minimum user effort. Primary and auxiliary storage, and peripheral subsystems, are designed for fast storage and retrieval of information as needed, and are available in configurations which may be expanded from medium to large scale as required. A variety of programming languages are accommodated for program development. Information may be input and output through various subsystems connected directly or indirectly to the computer. These subsystems include mass storage and unit record devices, communications equipment, and satellite computer systems. Inherent system compatibility provided in the hardware logic (490 operating mode) and system software components permit the UNIVAC 494 System to accommodate programs developed for, or oriented to, the UNIVAC 490/491/492 Systems.

Principal features of the UNIVAC 494 Real-Time System are summarized in the following list:

- Random access primary storage, consisting of a ferrite core memory, expandable from 65,536 to 131,072 30-bit words with a complete cycle time of 750 nanoseconds. Parity is maintained for each half-word during data transfers.

- Modular organization of primary storage, permitting memory overlap (the next sequential instruction is read simultaneously with the last operand cycle of the present instruction).

- Twelve full-word input/output channels, field expandable to 24, in groups of four.

- Buffered input/output, providing capability for program execution to proceed concurrently with I/O transfers.

- Program protection to prevent reference to primary storage addresses outside of the defined limits of a worker program.

- Direct arithmetic operations on fixed-point binary single and double precision operands, fixed-point binary coded decimal (BCD) double precision operands, and floating-point binary double precision operands.

- A wide variety of peripheral subsystems including random access magnetic drums, magnetic tape units, unit record devices, communication and display equipment, and satellite computers.

- A highly efficient and flexible Operating System, including a comprehensive library of programming languages, utility routines, and application packages.

- Multiprogram and multiactivity capability providing for a mix of real time and batch oriented programs.
- A comprehensive set of programming languages including two assembler languages, and FORTRAN IV and COBOL.
- Priority control network within the computer I/O logic which determines the order for honoring data transfer requests (function priority) on the various I/O channels, and for responding to system interrupts (interrupt priority), which control system contingencies.

The following information and description will serve as introduction to the UNIVAC 494 Real-Time System design, organization, hardware components, and Operating System. All software discussed in this document is operational rather than proposed.

An extensive library of hardware and programming documentation is published for the system. As applicable, these publications will be referenced in the text.

2. SYSTEM DESIGN CHARACTERISTICS

2.1. GENERAL

The UNIVAC 494 Real-Time System is user-oriented and is modular. The system comprises many and varied hardware and software components designed to increase the system capability and to meet the information processing needs of individual system users. An overview of the salient features of the system, hardware and software characteristics, is presented in this section. The hardware is described in greater detail in Section 3; the software is discussed in Section 4.

The UNIVAC 494 System provides a powerful and flexible information handling facility, with capabilities extending the full range of computer applications. The system components are fully complementary, and are designed for employment in configurations which will provide optimum utility and efficiency for a particular installation. The components function under control of the UNIVAC 494 Operating System, which imposes the coordination necessary for concurrent activities on multiple programs. The system design accommodates interchange and build-up of components in related classes or with related functions so that the system may be operated initially at specified minimum capacity, and may be expanded and upgraded to maximum capacity in accord with increases in informational needs of a user.

The planning of a specific UNIVAC 494 System configuration, while viewing the system itself as an entity, should, at the same time, consider the interrelationships, characteristics, and controls provided by specific system components in selecting the most practical installation.

2.2. HARDWARE COMPONENTS

The UNIVAC 494 System hardware provides highly sophisticated components for efficient execution of a wide latitude of information processing requests through a minimum of program specifications and operator intervention. The components are designed for full exploitation by Univac- and user-developed software, and permits the user to avail himself of the capabilities and speed of the main components, while permitting him great selectivity in choice of other system equipment and functions. In the main components, the power, speed and manipulative capabilities of the central processor unit (CPU) are combined with equally fast primary storage facilities, which are easily accessible and expandable, for efficient handling of both time and volume dependent activities. The diverse functions of available input/output (I/O) and communication peripheral devices and subsystems accommodate a variety of user input record formats, and furnish equally varied output record forms and displays. Random access peripheral storage devices provide high volume auxiliary storage with fast access times which, in combination with primary storage, facilitate program and data relocation and movement necessary in a real time, multiprogram environment. In addition, the random access storage facilities serve as I/O buffer areas between the main hardware components and the relatively slower peripheral units, allowing maximum utilization of the inherent speed of the system in both real time and batch processing.

The major functional divisions of the UNIVAC 494 System hardware are discussed briefly in this section under the following headings:

Central Processor
Primary Storage
On-site Peripheral Subsystems
Remote Peripheral Subsystems

2.2.1. Central Processor

The central processor (CPU) performs or controls all system activities, and is equipped with all functions for the execution of instructions, including arithmetic, logical, and input/output control. In addition to arithmetic, communication, and indexing registers, the CPU contains a unique system of program control and modification registers. Also provided are up to 24 I/O channels, with capabilities for multiplexing and interleaving of system elements on communication channels; a Day Clock; and a Real Time Clock for time-orientation of computer activities. The CPU operations and interrupt features, instruction repertoire, register formats, primary storage interface, and interface with its control consoles are discussed in Section 3 of this manual, and in *UNIVAC 494 Real-Time System Central Processor General Reference Manual, UP-4049* (current version).

2.2.2. Primary Storage

Primary storage consists of continuous core memory, randomly addressable in multiple, single, or partial word reference by the CPU. Primary storage contains instruction code in the execution process and buffers data transfers to and from input/output devices. With the exception of locations used in the CPU/primary storage interface, primary storage is a separate component from the CPU, providing storage capacities of 65,536 or 131,072 30-bit words. Read/restore cycle time is 750 nanoseconds. Detailed features of primary storage, including basic storage configurations and storage modules, are discussed in Section 3 of this manual.

2.2.3. On-site Peripheral Subsystems

On-site peripheral subsystems are connected directly to an input/output channel of the CPU. The peripheral subsystems perform various complementary functions in the system including input/output, auxiliary storage, and data preparation and communication. The following types of on-site peripheral subsystems are available for use with the UNIVAC 494 Real-Time System:

- MAGNETIC DRUM SUBSYSTEMS

  The FH-432, FH-432/FH-1782, and FH-880 Magnetic Drum Subsystems comprise random access, auxiliary storage units with average access times ranging from 4.33 to 17 milliseconds, and transfer rates of 60,000 to 240,000 words per second, which are expandable from one to eight units (three to nine for the FH-432 Drum Unit) per I/O channel, providing storage capacity of 786,432 to 16,777,216 30-bit words, dependent upon the unit configuration.

- MASS STORAGE SUBSYSTEMS

  The FASTRAND II and FASTRAND III Mass Storage Subsystems comprise random access, auxiliary storage magnetic drum units with average access times ranging from 35 to 92 milliseconds and with transfer rates of 50,688 to 76,032 words per second, which are expandable from one to eight units per I/O channel, providing storage capacity of 25,952,256 to 311,427,072 30-bit words.

- MAGNETIC TAPE SUBSYSTEMS

  The UNISERVO VIII C and UNISERVO VI C Magnetic Tape subsystems provide recording densities of 200–800 frames per inch, with transfer rates of 8,500 to 96,000 frames per second, dependent upon the unit and the recording density. Features include industry-wide compatibility, optional 9- or 7-track formats, forward/backward read, and simultaneous and nonsimultaneous operation capabilities. A maximum of 16 magnetic tape units per I/O channel is permitted.

- HIGH SPEED PRINTER SUBSYSTEMS

  The High Speed Printer Subsystems, Type 0755 and Type 0758, are capable of printing 700–1600 132-character lines per minute.

- PUNCHED CARD SUBSYSTEM

  The Punched Card Subsystem, comprises the Type 0706 Card Reader which senses 80-column data at the rate of 900 cards per minute, and the Type 0600 Card Punch which punches and stacks cards at the rate of 300 cards per minute.

- UNIVAC 1004 SYSTEM

  The UNIVAC 1004 System is a self-contained computer system, having a CPU and input/output peripheral units, which may serve as an online or remote unit record peripheral subsystem to the UNIVAC 494 System, and which may function as an independent computer when not serving as part of the UNIVAC 494 System. The UNIVAC 1004 System operates with plugboard programs, and can transfer and receive data at high speeds. The UNIVAC 1004 System may also perform some editing and manipulation operations on the data.

- UNIVAC 9300 SYSTEM

  The UNIVAC 9300 System is a self-contained computer system, having a CPU and input/output peripheral subsystems, which may serve as an online or remote unit record peripheral subsystem to the UNIVAC 494 System, and which may function as an independent computer when not serving as part of the UNIVAC 494 System. The UNIVAC 9300 System is an internally programmed system which transmits, processes, and receives data at higher speeds than the UNIVAC 1004 System. Greater storage capacities, processing capabilities, and peripheral facilities also are provided.

- COMMUNICATION SUBSYSTEM

  The Communication Terminal Modular Control (CTMC) Subsystem furnishes communication capability to the UNIVAC 494 Real-Time System, and permits time-shared data transfers between the UNIVAC 494 CPU and up to 64 diverse remote terminals. The CTMC Subsystem serves as the interface between the central processor and any device meeting the accepted Electronic Industries Association (EIA) standards for serial data transmission.

- TRANSFER SWITCH

  The transfer switch provides a means for switching peripheral units between 494 CPU's. The switch unit may be used to switch entire units to offline condition for maintenance and to bring replacement units online to the system.

2.2.4. Remote Peripheral Subsystems

Remote peripheral subsystems perform complementary functions similar to the on-site subsystems, and are connected to the UNIVAC 494 System through the Communication Terminal Modular Control (CTMC) Subsystem. The remote peripheral units include the following data exchange and display devices, and satellite computer systems:

- DATA COMMUNICATION TERMINAL (DCT) 2000

  The Data Communication Terminal (DCT) 2000 is an input/output subsystem that furnishes capability for sending and receiving large quantities of data in conjunction with other subsystems or computers over common-carrier land lines in remote-site operations. The DCT 2000 consists basically of a sending and receiving control unit with facilities for printing, card reading, and punching.

- UNISCOPE 300 VISUAL COMMUNICATION TERMINAL

  The UNISCOPE 300 Visual Communication Terminal transmits, receives, and displays information for applications requiring direct interaction between the operator and the central computer. The UNISCOPE 300 comprises a cathode-ray tube (CRT) display, keyboard, and memory, and is available in single station and multistation arrays of up to 48 units.

- UNIVAC 9200/9300 SYSTEMS

  The UNIVAC 9200 System, and the UNIVAC 9300 System as mentioned for online operation, are complete computer systems which may be linked to the 494 CPU as unit record peripheral subsystems through the CTMC subsystem of the UNIVAC 494 System and the Data Communication System (DCS) of the UNIVAC 9200/9300 Systems (see 3.6.3), and which may serve as independent computers when not connected with the UNIVAC 494 System. The UNIVAC 9200 System is an internally programmed card-processing system; the UNIVAC 9300 System is an internally programmed system supplying both card and magnetic tape facilities. Both systems transmit and receive data at high speeds, and perform editing and manipulation operations.

- UNIVAC 1004 SYSTEM

  The UNIVAC 1004 System, as mentioned, may be used as either on- site or remote unit record peripheral equipment to the UNIVAC 494 System, or may be used as an independent computer system when not connected to the UNIVAC 494 System.

2.3. SOFTWARE COMPONENTS

The UNIVAC 494 System software is designed to control and to make full use of the capabilities of the system hardware in the development and execution of user programs. The software library, the 494 Operating System, is a set of integrated programs and routines which furnish the control necessary for coordination and concurrent execution of real time and batch processing operations. The executive routine or control program provides a powerful and flexible control language for defining system operations and for invoking system components in a multiprogram, real time and batch processing environment. A variety of programming languages permit program development according to the desires of different users. A system of program libraries, with ancillary file directory and maintenance services, provides for effective storage, update, and retrieval of Univac- and user-supplied program elements and data. Random access storage, which is used as the system operating base and as a buffer for primary input/ output devices in cooperative or controlled interfaces with monitor routines, significantly reduces the time delays normally inherent in multiple activity operations involving job equipment switching and discontinuous patterns of data inputs and outputs.The overall effect of the executive routine on I/O operations, and on job scheduling and execution, is to furnish the user with an efficient automatic information processing complex, requiring minimum time and user effort.

The UNIVAC 494 System software is discussed briefly under the following topic headings (for detailed information, see Section 4 of this manual and *UNIVAC 494 Real-Time System Operating System Programmers Reference, UP-7504,* (current version):

Executive Operation

Utility Package

Application Package

Language Processors

2.3.1. Executive Operation

The user conveys information to the Operating System in the form of job decks comprising ordered sequences of tasks. The job information includes scheduling routines, limited data sets, program parameters, system processor calls, library specifications, and other pertinent communications. The executive routine also maintains a library system for storage and retrieval of programs or program components, called elements, according to application. The elements are of three types: Source, comprising input to language processors as the basis for programs; Relative Binary (RB), comprising language processor output as intermediate code for synthesis into executable elements; and Load, comprising object or executable programs. The element libraries are of three kinds: System, comprising elements which are an integral part of the software system; Job, comprising elements collected for a particular job; and Group, comprising elements with application to more than one job.

The information from unit record and other external devices, together with information from the libraries as required, is accepted in multiple streams through the Input Cooperatives (see 4.2.2) of the Data Management section and placed on queues for processing. In cooperative action between the Loader, which synthesizes the program components into complete, executable programs, and Job, Activity/Task, Remote Device, and Service Control Mechanisms, jobs are scheduled for processing according to priority; processor and peripheral facilities are assigned; jobs and tasks are selected and activated; the program is executed; and the desired output is made through the Output Cooperative, either to library storage media or to unit record devices. The control language is openended and expandable, permitting user direction of the system through control statements and service requests, some of which may be submitted through the control stream and others which may be submitted dynamically during job processing.

2.3.2. Utility Packages

The software utility packages are incorporated in the UNIVAC 494 Operating System for enlarging and enhancing the data processing procedures and the output information offered to the user. The following list, which is not exhaustive and final, shows the variety of utility programs and routines available in the UNIVAC 494 System. Detailed information concerning utility packages is discussed in Section 4 of this manual.

File Control

Report Writer

Utility Generator

Logging and Accounting

REXecutor

CONVAID

Random Storage File Handler

2.3.3. Application Packages

The application packages are programs and routines included in the software library to implement specific functions and uses of the UNIVAC 494 System. Currently available application packages include the following:

Sort/Merge

Network Simulator

Transaction Control System

Critical Path Method

Linear Programming

2.3.4. Language Processors

Language processors convert symbolic language to machine amenable language; that is, input programs written in source language are converted into intermediate output code, producing RB elements. Currently available language processors comprise the following two UNIVAC 494 System assemblers and two high-level compilers:

UNIVAC 494 Assembler (ASM)

UNIVAC 494 SPURT Assembler

COBOL Compiler

FORTRAN Compiler

2.4. SYSTEM CONFIGURATIONS

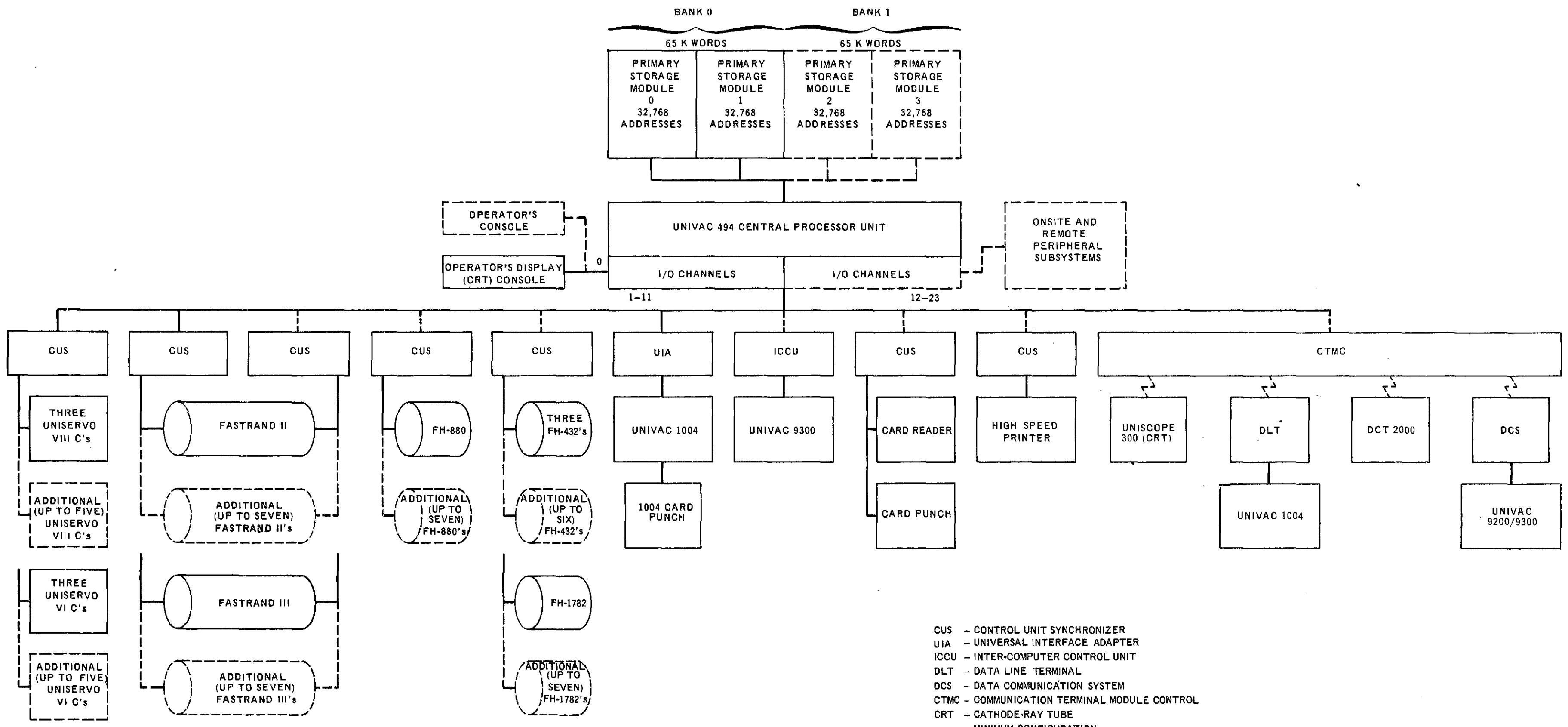
The variety of hardware characteristics and software controls afforded by the UNIVAC 494 System components provide an almost limitless number of possible system configurations. A wide range of choice and a high degree of selectivity is permitted the user in setting up a particular installation to meet his information requirements on the basis of both present and future needs. Basic considerations in configuration planning should include:

- the characteristics and volume of the input information, such as file size, record formats, and record content, as for example, scientific information as opposed to commercial information;

- the number, frequency, complexity, and time dependent relationships of queries and other transactions, as in real time/communication applications, which may require a large volume of rapid transactions utilizing much I/O and communication equipment without great need for extensive storage facilities, in contrast to scientific applications, requiring generally complex operations with more need for storage facilities than for I/O and communication facilities.

- the characteristics and volume of the desired output information, as in generation of exception reports, tape files, displays, and other outputs requiring more, or less, use of various peripheral subsystems as the application warrants.

For the foregoing reasons, a maximum system configuration is hardly specifiable, as the entire system, including the number of CPU's and consequently the number of primary storage units, may be expanded. A minimum configuration may be specified on the basis of the hardware necessary for I/O transfers through the CPU and a 65K primary storage unit, and for containing the basic system software. As shown in Figure 2–1, a minimum configuration may use a UNIVAC 1004 System for I/O transfers, providing also an additional computer system at low cost. The UNIVAC 1004 System may be replaced by the UNIVAC 9200 or UNIVAC 9300 Systems, providing enhanced computer facilities as well as I/O capability; or the punched card, high speed printer, and punched paper tape subsystems may be used. In the same way, for higher processing speeds, the FASTRAND Subsystem may be replaced by magnetic drum units furnishing the same storage capacity.


BANK 0
BANK 1
65 K WORDS
65 K WORDS
PRIMARY STORAGE MODULE 0 32,768 ADDRESSES
PRIMARY STORAGE MODULE 1 32,768 ADDRESSES
PRIMARY STORAGE MODULE 2 32,768 ADDRESSES
PRIMARY STORAGE MODULE 3 32,768 ADDRESSES
OPERATOR'S CONSOLE
OPERATOR'S DISPLAY (CRT) CONSOLE
0
UNIVAC 494 CENTRAL PROCESSOR UNIT
I/O CHANNELS
I/O CHANNELS
1–11
12–23
ONSITE AND REMOTE PERIPHERAL SUBSYSTEMS
CUS
CUS
CUS
CUS
CUS
UIA
ICCU
CUS
CUS
CTMC
THREE UNISERVO VIII C's
ADDITIONAL (UP TO FIVE) UNISERVO VIII C's
THREE UNISERVO VI C's
ADDITIONAL (UP TO FIVE) UNISERVO VI C's
FASTRAND II
ADDITIONAL (UP TO SEVEN) FASTRAND II's
FASTRAND III
ADDITIONAL (UP TO SEVEN) FASTRAND III's
FH-880
ADDITIONAL (UP TO SEVEN) FH-880's
THREE FH-432's
ADDITIONAL (UP TO SIX) FH-432's
FH-1782
ADDITIONAL (UP TO SEVEN) FH-1782's
UNIVAC 1004
1004 CARD PUNCH
UNIVAC 9300
CARD READER
CARD PUNCH
HIGH SPEED PRINTER
UNISCOPE 300 (CRT)
DLT
UNIVAC 1004
DCT 2000
DCS
UNIVAC 9200/9300

CUS – CONTROL UNIT SYNCHRONIZER
UIA – UNIVERSAL INTERFACE ADAPTER
ICCU – INTER-COMPUTER CONTROL UNIT
DLT – DATA LINE TERMINAL
DCS – DATA COMMUNICATION SYSTEM
CTMC – COMMUNICATION TERMINAL MODULE CONTROL
CRT – CATHODE-RAY TUBE
——— MINIMUM CONFIGURATION
– – – OPTIONAL, EXPANDABLE, OR INTERCHANGEABLE

NOTE: 7 OR 9 TRACK TAPE MAY BE USED
FASTRAND UNITS MAY BE REPLACED BY EQUIVALENT FH-432, FH-880, OR FH-1782 UNITS
FH-432 UNITS MAY BE REPLACED BY EQUIVALENT FH-1782 OR FH-880 UNITS
ONLINE UNIVAC 1004 MAY BE REPLACED BY UNIVAC 9300 OR BY HIGH SPEED PRINTER AND CARD SUBSYSTEMS

Figure 2–1. System Configuration

3. SYSTEM HARDWARE COMPONENTS

3.1. GENERAL

The UNIVAC 494 System hardware components comprise four types of equipment based upon functions within the system: the central processor unit (CPU), primary storage, on-site peripheral subsystems, and remote peripheral subsystems. Each equipment type embodies advanced design features for providing maximum utility and efficiency in its particular function. In keeping with the modular, expandable system concept, each component is compatible with similar units or with units having similar functions in the system. In this way, a large selection of peripheral subsystems and different storage capacities can be used in conjunction with the CPU to make the system responsive to a wide range of business, scientific, and industrial applications.

The CPU is a word-addressable, multipurpose digital computer, which operates on fixed-word length (double-, whole-, or half-word) data and instructions, and possesses full capability for operation in a multiprogram environment. Primary storage, which is separate from the CPU, is available in capacities of 65,536 30-bit words or 131,072 30-bit words, and is random access, linear select ferrite core memory. On-site magnetic drum, magnetic tape, and unit record subsystems provide a variety of capacities, speeds, and functions permitting selectivity in system configuration design. Remote communication subsystems and satellite computers provide a range of information exchange and data handling capabilities, one of these being the ability of the satellite computers to function as independent data processing systems when not operating with the UNIVAC 494 System.

Detailed information concerning the characteristics and capabilities of the system hardware is presented in this section. Additional information about the peripheral subsystems may be found in manuals written specifically for these subsystems. The 494 System hardware components are discussed under the following headings:

Central Processor

Primary Storage

On-site Peripheral Subsystems

Remote Peripheral Subsystems

3.2. CENTRAL PROCESSOR

The UNIVAC 494 Central Processor Unit (CPU) is the principal component of the UNIVAC 494 System, and is generally the component by which the entire 494 System is identified. The CPU has the responsibility for accepting data and tasks from external equipment, queuing tasks to meet the demands of the system, processing the tasks, and returning the results to external equipment.

The CPU is characterized by a control section, arithmetic section, input/output section, storage interface, and control console. The processor sections are completely independent of memory with the exception of input/output control. Input/output requires the use of fixed locations in primary storage (Buffer Control Registers) for control of input/output and for interrupt locations. A block diagram of the various sections of the CPU is given in Figure 3–1.

3.2.1. Control and Arithmetic Section

The control and arithmetic section provides the basic phasing and logic for instruction decoding and execution, and contains the following principal parts:

- A 17-bit Program Location Counter used for sequentially accessing instructions residing in memory.
- A series of Instruction Registers used to contain the instruction during the decoding and execution cycle.
- A Relative Index Register (RIR) which provides the base address bias or effective address, for programs (relative addressing). The RIR permits a program to be entered or moved to any location in memory without modification of the program code. A second level of relative addressing (Dual Index Mode) is provided by the Lower Lock address of the Program Lock-in Register in combination with the RIR. The Dual Index Mode is used by programs which have instructions and data residing in distinctly separate areas of storage, such as common subroutines, and provides for instructions to be biased by the RIR and data references to be biased by the Lower Lock.
- An Operand Address Register and a Program Address Register (P register) used in conjunction with the Program Location Counter, RIR, and index (B) registers in establishing and accessing instruction locations in primary storage.
- A Memory Select Register (MSR) used for manual selection of addressing mode (see 3.3.3) by determining which primary storage module and memory bank will receive instructions.
- A 30-bit Internal Function Register (IFR) used principally to facilitate executive routines. The IFR is used to specify: index mode, index register length, index register set, and privileged or nonprivileged program execution mode (Guard Mode). Guard Mode is activated to prevent operating programs from executing restricted instructions, such as input/output and executive instructions, and to activate memory limit checks.

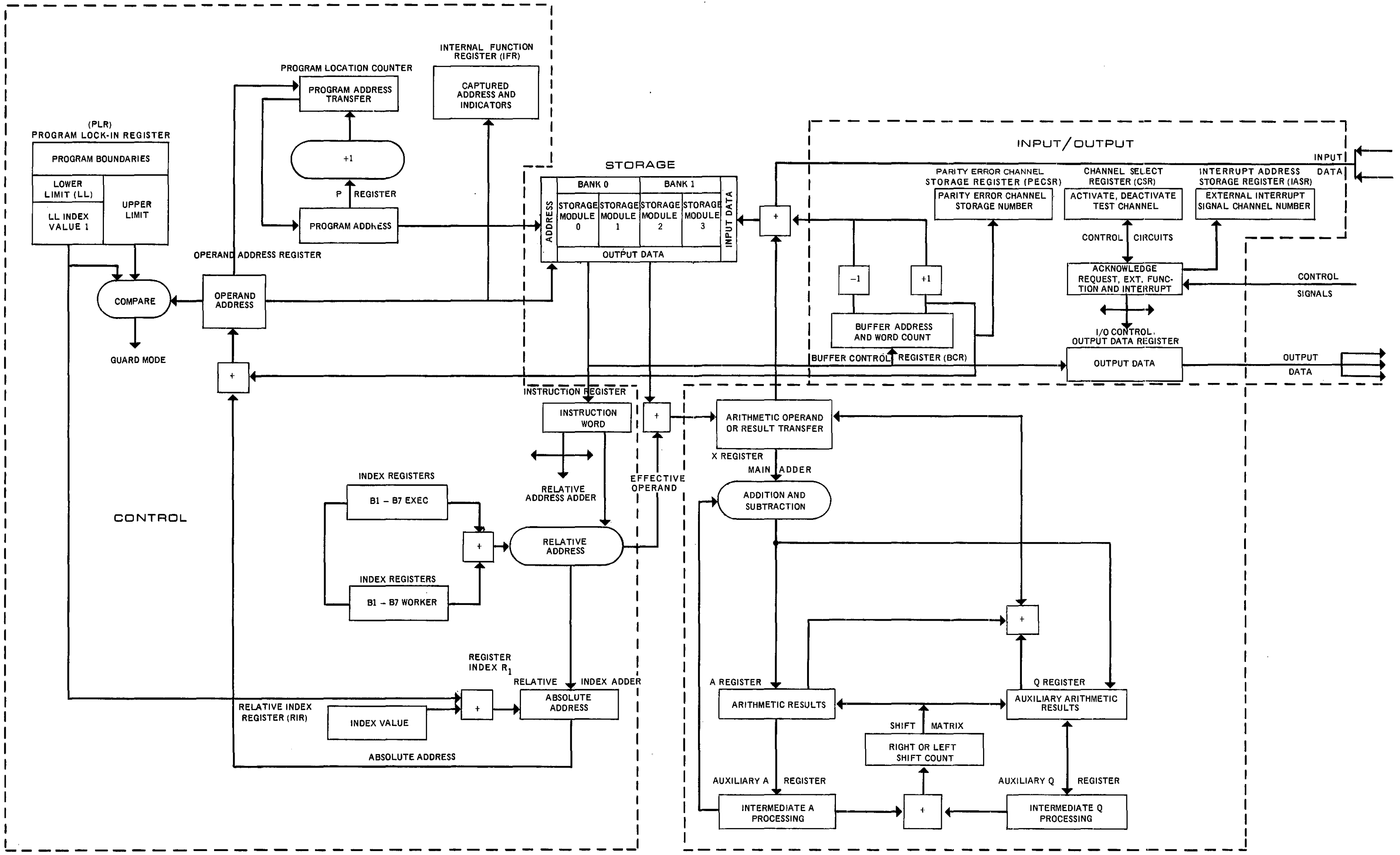


Figure 3–1. Central Processor Operation, Block Diagram

- A Program Lock-in Register (PLR) used to define the upper (upper lock limit) and lower (lower lock limit) storage address assigned to the program. The PLR prevents concurrently operating programs from inadvertently referencing outside their assigned memory locations.
- Fourteen addressable index (B) registers which provide for operand address modification, index codes, counters and modifier incrementation. The B registers comprise two groups of seven registers: the executive set, which is reserved for the executive routine when operating in a privileged mode, and the worker set, which is used by programs operating in a nonprivileged mode. Two options are provided in the use of B registers: the length option which specifies that all registers operate as 15-bit registers, or that three registers (B1, B2, and B3) operate as 15-bit registers and four registers (B4, B5, B6, and B7) operate as 17-bit registers; and the bias option which calls for bias of seven B registers by the RIR when used in an instruction performing a data access, or bias of B1, B2, and B3 by the RIR and bias of B4, B5, B6, and B7 by the PLR when used in an instruction performing a data access.
- An A (accumulator) and a Q (quotient) register, which are programmable registers of 30 bits each, used by operating programs to perform arithmetic and logical operations. The registers may be used as two distinct arithmetic registers or combined to form a 60-bit (AQ) register. Auxiliary A and Q registers are also used to store intermediate results in arithmetic operations.
- A number of communication registers, such as the X (arithmetic) and R (index) registers, used for communicating between other registers.
- Other components include various adders for performing arithmetic operations and establishing instruction addresses, and various counters and registers which perform matrix shifting (K register) and other functions.

3.2.1.1. Arithmetic Operations

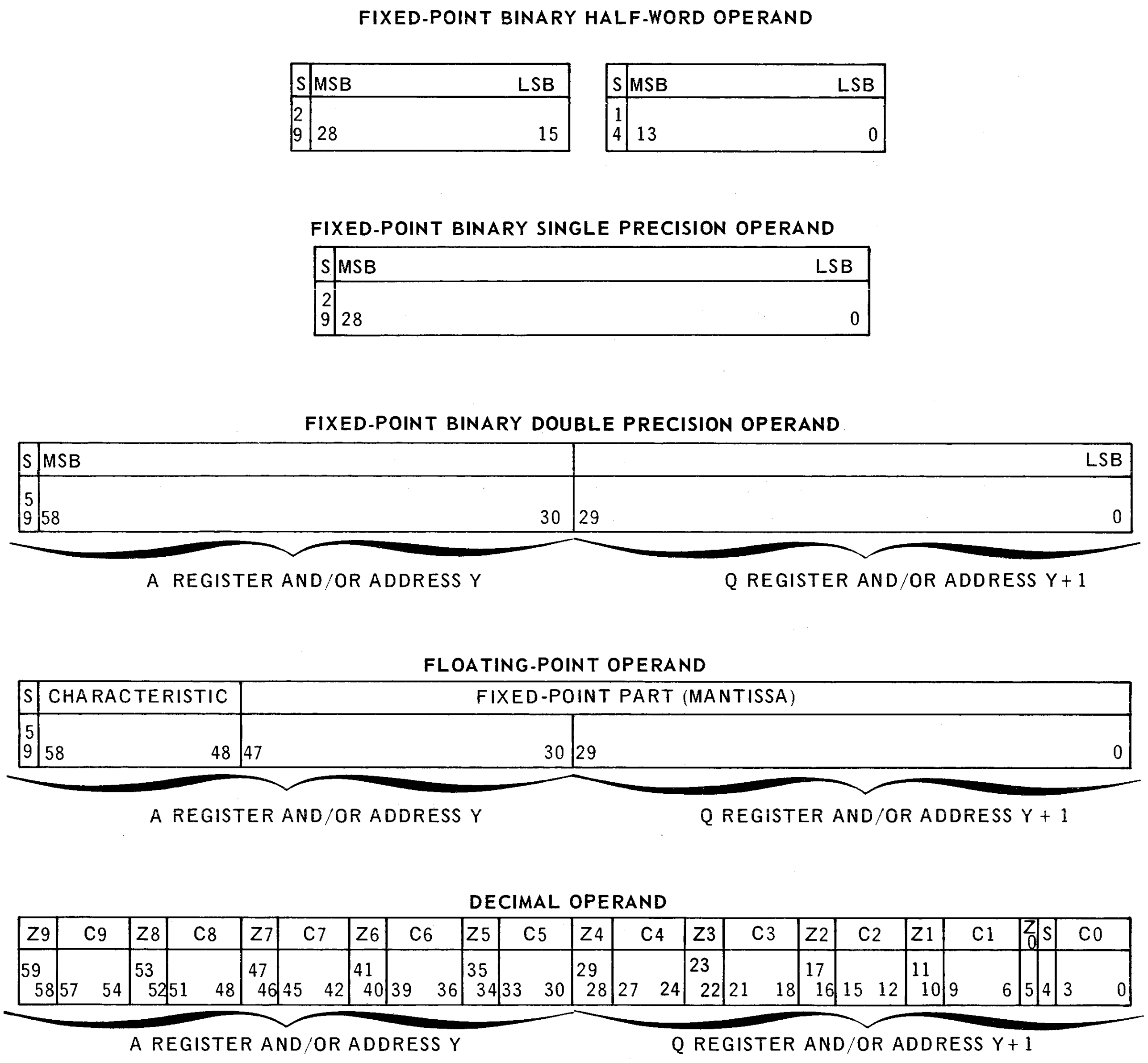
Arithmetic operations may be performed in the fixed-point binary coded decimal (BCD) mode, floating-point mode, or in the fixed-point binary mode. Operations in fixed-point binary may be either single or double precision.

- Fixed-Point Binary Formats

  The format for a fixed-point binary single precision operand, shown in Figure 3–2, assigns three bit positions per octal digit, except for the most significant digit which has two bit positions. The order of binary digits is from 00 to 29, right to left, with position 00 containing the least significant digit. The sign bit, contained in position 29, is 0 for a positive number, or 1 for a negative number.

  Fixed-point binary operands may also be half-words as indicated in Figure 3–2.

  The format for a fixed-point binary double precision operand, shown in Figure 3–2, is 20 octal coded digits, requiring two consecutive memory locations for storage and the use of both the A and Q registers for operations. The most significant portion of the operand (including the sign) appears in the accumulator and/or the first of the two consecutive memory locations.


FIXED-POINT BINARY HALF-WORD OPERAND
S MSB LSB
29 28 15
S MSB LSB
14 13 0
FIXED-POINT BINARY SINGLE PRECISION OPERAND
S MSB LSB
29 28 0
FIXED-POINT BINARY DOUBLE PRECISION OPERAND
S MSB LSB
59 58 30 29 0
A REGISTER AND/OR ADDRESS Y
Q REGISTER AND/OR ADDRESS Y+1
FLOATING-POINT OPERAND
S CHARACTERISTIC FIXED-POINT PART (MANTISSA)
59 58 48 47 30 29 0
A REGISTER AND/OR ADDRESS Y
Q REGISTER AND/OR ADDRESS Y + 1
DECIMAL OPERAND
Z9 C9 Z8 C8 Z7 C7 Z6 C6 Z5 C5 Z4 C4 Z3 C3 Z2 C2 Z1 C1 Z0 S C0
59 58 57 54 53 52 51 48 47 46 45 42 41 40 39 36 35 34 33 30 29 28 27 24 23 22 21 18 17 16 15 12 11 10 9 6 5 4 3 0
A REGISTER AND/OR ADDRESS Y
Q REGISTER AND/OR ADDRESS Y+1

The S bit is the sign bit. MSB and LSB are most and least significant bits, respectively. Z bits are zone bits associated with the character bits (C bits).

Figure 3–2. Computer Word Formats

- Fixed-Point Binary Coded Decimal (BCD) Format

  Binary Coded Decimal (BCD) operations may be performed upon operands stored in 30-bit combinations of any one of a variety of six-bit zoned BCD forms, such as Fieldata code. Because arithmetic operations are performed directly upon the BCD characters, no programmed coding conversion is necessary to convert the characters to straight binary characters. Ten decimal digits, including the sign, may be stored and operated upon, requiring two consecutive memory locations and/or the A and Q registers in combinations as shown in Figure 3–2.

  The most significant characters (C) and zone bits (Z) are stored in the accumulator and/or the first of the two consecutive addresses. Bit 04 denotes the sign(s) of the number; 1 for positive and 0 for negative. In accord with standard Fieldata code, the zone bits are both 1's, and are ignored in all decimal operations, the same as the original zone bits in the AQ register. After each BCD operation (addition or subtraction) an indicator is set (or reset) to indicate a carry or borrow. A decimal test instruction provides capability for testing the overflow indicator and also for testing for positive/negative and zero/nonzero results.

- Floating-Point Format

  The floating-point operand format uses 60 bits to express a positive or negative number as a fixed-point part (the mantissa) multiplied by a power of 2 (the characteristic). Two consecutive memory locations or the AQ register store the floating-point number, as shown in Figure 3–2.

  Bit 59 is the sign bit for the floating-point number; and is 0 for positive, or 1 for negative. A negative number is represented by the 60-bit ones complement of the positive representation of the number (e.g., $+2_{10}$ is expressed as 2002.40---0; -2_{10} is expressed as 5775.37---7).

  The characteristic is biased to represent both positive and negative powers of 2. The bias value is 2000_8. An eleven bit biased characteristic can represent any power of 2 from -2000_8 to $+1777_8$ (i.e., from -1024_{10} to $+1023_{10}$).

  A floating-point number is normalized when the most significant bit of the mantissa (bit 48) is different from the sign bit of the number. A positive number may be normalized by left shifting the mantissa until a 1 bit appears in the leftmost bit position of the mantissa, and by decrementing the characteristic by the number of bit positions shifted (e.g., 2003.20---0 and 2004.10---0 are nonnormalized representations of the number $+2_{10}$). A normalized floating-point number in this format can express any number, N, in the range $2^{-1025} \leq N < 2^{+1023}$, or between the approximate limits of 10^{-309} and 10^{+308}.

3.2.1.2. Arithmetic Processing

Arithmetic operations are performed in the parallel mode with all bits of an operand transferred to the arithmetic unit simultaneously for highest speed. The arithmetic required for effective and relative addressing is performed in the control section, which is separate, to prevent interference with arithmetic operations in the arithmetic section.

The UNIVAC 494 control and arithmetic section utilizes the operands shown in Figure 3–2.

3.2.1.3. Timing Clocks

The central processor unit contains two clocks that may be used for program timing: the Day Clock and the Real Time Clock.

- The Day Clock is a twenty-four hour clock that records the time of day in hours, minutes and hundredths of minutes. The updated Day Clock time value is automatically stored every six hundred milliseconds in a fixed address of primary storage and is also displayed at the operator's console. Approximately every six seconds a Day Clock interrupt is generated to a fixed address where a routine, conditioned by the time of day, may be started. The format of the Day Clock time, as stored in primary storage, is shown in Figure 3–3.
- The Real Time Clock is an 18-bit counter contained in primary storage. The counter is incremented every two hundred microseconds, and an interrupt is generated when the counter overflows (i.e., recycles from all one bits to zero, over a maximum of fifty-two seconds). The clock may be set by the program and is used by the Operating System as an interval timer. The format of the Real Time Clock is shown in Figure 3–3.

HOURS		MINUTES		HUNDREDTHS (OF MINUTES)		ALL ZEROS
TEN	UNIT	TEN	UNIT	TEN	UNIT	
29 28	27 24	23 20	19 16	15 12	11 8	7 0

DAY CLOCK FORMAT

NOT USED	COUNTER
29 18	17 00

REAL TIME CLOCK FORMAT

Figure 3–3. Clock Formats

3.2.2. Instruction Words

An instruction repertoire is provided in the UNIVAC 494 System which utilizes and promotes the full power of the CPU in computer activities. The repertoire furnishes a complete set of standard instructions for data transfers, shifting, sequence modification, address and operand modification, logical operations, tests, fixed decimal and floating-point arithmetic, input/output control, and partial word selection. The repertoire also includes instructions which permit fast and simplified control by the executive routine operating in a multiprogram mode.

3.2.2.1. Instruction Word Formats

Three instruction formats are used: the normal instruction word, the extended instruction word, and the input/output (I/O) instruction word. Formats for the instructions are shown in Figure 3–4. Each type of instruction word provides for given sets of designators for defining the operation, and specifies the operand which will be affected.

NORMAL INSTRUCTION WORD

f	j	k	b	y
29 24	23 21	20 18	17 15	14 0

EXTENDED INSTRUCTION WORD

77_8	g	b	y
29 24	23 18	17 15	14 0

490 MODE I/O INSTRUCTION WORD

f	$\hat{j}$	$\hat{k}$	b	y
29 24	23 20	19 18	17 15	14 0

Figure 3–4. Instruction Word Formats

- **Normal Instruction Words**

Normal instructions are subdivided into classes which are: read, store, and replace. A read class instruction transfers data from primary storage to an appropriate register. A store class instruction transfers data between registers or from a register to primary storage. A replace class instruction, which is a combination of read and store operations, reads the data from primary storage or a register, performs an operation on the data, and places the result in primary storage or a designated register.

The f designator is a six-bit function code that specifies the operation to be performed. The j designator is a three-bit code that may be interpreted as a skip designator, register designator, repeat modification, or jump designator dependent upon the type of instruction. (See Table A–2 for interpretation of j designators.) The k designator is a three-bit code which, together with the class of the instruction, defines the portion (whole-word, upper half, or lower half) of the operand which will be processed and specifies its source and/or destination (see Table A–3), permitting the use of whole word operands of 30 bits or half-word operands of 15 bits. The y designator specifies the operand, Y, or the operand address which will be processed. The b designator indicates the index register (1–7), whose contents (B_b) are added to the 15-bit y designator to form a 15-bit or a 17-bit effective operand ($\bar{y}$ designator) or to form a primary storage address which, when added to the RIR or the lower portion of the PLR, is the location of the operand or the location of the next instruction to be performed.

- Extended Instruction Words

  The extended instruction word repertoire is applicable to worker and executive program functions which have been added as standard features of the UNIVAC 494 CPU beyond the standard functions of other computers in the 490 series. The functions include a number of operations which were performed by earlier computers through subroutines. In the extended repertoire, the f designator is 77_8. Following the 77 is a six-bit g designator which, together with the f designator, defines the function to be performed. The b and y designators are similar to their counterparts in the normal instruction word. No j or k designators can be used in the 77 instruction set; therefore, interpretation of the operand is implicit in the instruction itself and not by class.

- Input/Output Instruction Words

  Input/output instructions alert the input/output (I/O) section of the central processor to begin operations on a specified channel. The processor is then free to execute instructions while data is being transferred between primary storage and peripheral equipment by the I/O section. The format of the 490 mode I/O instruction word varies slightly from the normal instruction word format, as shown in Figure 3–4. The j-field is expanded to four bits (bits 20–23) and the k-field is decreased to two bits (bits 18–19), and are designated by the symbols $\hat{j}$ and $\hat{k}$, respectively. The $\hat{j}$-field is used as a channel designator and the $\hat{k}$-field defines the interpretation of the operand.

3.2.2.2. Instruction Repertoire

The complete 494 instruction repertoire is shown in Table A–1. The instructions are listed by instruction type as follows:

Transfer instructions
Shift instructions
Compare instructions
Jump instructions
Sequence modifying instructions
Arithmetic instructions
Logical instructions
Input/Output instructions

The description of each instruction contains: the mnemonic code used in the UNIVAC 494 assemblers, SPURT, and ASM, the function code (f) in octal notation, the description of the operation, the class of instruction, and the execution time. The interpretations of j designators are listed in Table A–2; k designator interpretations are listed in Table A–3. Additional information concerning the instructions may be found in *UNIVAC 494 Real-Time System SPURT Programmers Reference, UP-4090* (current version) and *UNIVAC 494 Real-Time System Assembler Programmers Reference, UP-4133* (current version).

- Transfer Instructions

  Transfer instructions move data between primary storage and the CPU. All transfers are nondestructive (the original source remains unchanged). Transfers may consist of 60, 30, or 15 bits or, in the character packing and unpacking instructions, 6 bits, as determined by the k designator of the instruction or the instruction itself. The character packing instruction composes a 30-bit word of five successive six-bit characters, each in a different sequential address. Similarly, the character unpacking instruction breaks up a 30-bit word into five successive six-bit characters and stores them in successive addresses of primary storage.

- Shift Instructions

  Shift instructions move the contents of a selected register either to the left or right by a specified number of positions. If the instruction is a right shift, all bits shifted out are lost, and all vacated positions may be zero filled or be filled with the highest-order bit (sign-fill). A left shift is a circular shift; as a bit is shifted out at the left, it is returned to the vacated bit position at the right. With the exception of the Scale Factor Shift (7730) instruction, the number of shifts is the six-bit binary number formed by the lowest-order six bits of the operand, Y. The number of shifts by the 7730 instruction is determined through the normalizing associated with arithmetic instructions.

- Compare Instructions

  Compare instructions test sets of register values against certain criteria with skip or no-skip operations being performed by the program as the result of the comparison. The comparisons may be performed in either the alphanumeric or arithmetic mode, the difference being that the highest-order bit is not treated as a sign in the alphanumeric mode, but as a binary 1 or 0. This is especially useful in the sorting of Fieldata characters. The compare instructions may use masking so that comparisons can be made only on specified bit positions of the words involved.

- Jump Instructions

  Jump instructions transfer control of the program from the next sequential address or a specified address to an instruction at a specified primary storage address, providing that the selected conditions (conditional jump) for the jump are satisfied, or that certain program operations occur (unconditional jump).

- Sequence-Modifying Instructions

  Sequence-modifying instructions cause repeated execution of an instruction a specified number of times, or cause skips or jumps, while capturing the relative P-value (address) for future reference in the case of jump operations.

- Arithmetic Instructions

  Arithmetic instructions in the fixed-point, binary, single precision (integer) mode handle 30-bit binary numbers with the highest-order digit (bit position 29) reserved for sign (binary 0 = +, binary 1 = –) with absolute value up to $3,777,777,777_8$ or $536,870,911_{10}$. A negative number is presented as the ones complement of the same positive number (complement each binary digit). Arithmetic instructions in the fixed-point double precision mode can handle operands with absolute value up to $37,777,777,777,777,777,777_8$ or $576,460,752,303,423,487_{10}$. Arithmetic instructions in the fixed-point zoned BCD (decimal) mode can operate directly upon numbers made up of ten decimal digits received as signed, zoned BCD characters such as in the Fieldata code. The zone bits remain unchanged by the arithmetic operation. Arithmetic instructions in the floating-point (exponential) mode may be used to operate upon positive or negative numbers greater than or equal to 2^{-1025} and less than 2^{+1023} (approximately 10^{-309} to 10^{+308}).

- Logical Operations

  Logical operations enable masking operations upon a word or upon selected bits of a word. These operations are: logical product, which selects specified bits of a word (leaving binary 0's in unselected bit positions); selective set, which forces binary 1's into selected bit positions of the accumulator; selective clear, which forces binary 0's into selected bit positions of the accumulator; selective complement, which selectively changes the bits (binary 1's to 0's, binary 0's to 1's); and selective substitute, which substitutes the bits in selected bit positions into the corresponding bit positions of the accumulator.

3.2.3. Interrupts

The CPU is conditioned to respond to interrupt signals which may occur for various reasons, including programming errors, hardware faults, notification of incoming data or of the availability of peripheral units after completion of a previously scheduled task, full or empty buffers, and other contingencies.

An interrupt sends the program to a fixed address for further action. The fixed address contains an instruction which captures the relative address of the next instruction in the interrupted program and then jumps to the applicable interrupt routine, temporarily suspending further processing of the interrupted program. After the interrupt routine is completed, the interrupted program is usually resumed. Two types of interrupts may occur: an unconditional interrupt, which cannot be locked out, or a conditional interrupt, which can be locked out by either another interrupt or by a Return Jump (octal function code 64) instruction with j designator 0 or 1. A Return Jump instruction, issued while in the guard mode, may not lock out interrupts for more than 100 microseconds. Conditional interrupts are enabled by executing a Jump instruction (octal function code 60) with a j designator of 0 or 1.

3.2.3.1. Unconditional Interrupts

Unconditional interrupts indicate program contingencies which require immediate intervention, and include the following:

- Memory Parity Interrupt, caused by a parity error involving data which is not related to buffer operation. A memory parity error sends the program to a fixed address associated with the particular memory bank involved. When a memory parity error is detected, the data is written back into memory (at the same address) with the same incorrect parity, so that the interrupt routine can locate and test operation at the same core memory address.

- Memory Protection or Timeout Interrupt, generated by an attempt to violate the read and/or write protection mode in effect, or if interrupts are locked out for more than 100 microseconds by a Return Jump instruction when operating in guard mode.

- Executive Return Interrupt, caused by the Executive Return (7754) instruction which sends the program to an interrupt routine enabling capture of the P value of the program which is interrupting. This interrupt is the normal mode of program entry to the Operating System.

- Test and Set Interrupt, caused by a Test and Set (7752) instruction which tests for a 1 bit in bit position 14 at a selected core memory location. If this bit is already set to 1, the interrupt is generated; if the bit is not set, bits 0–14 are then set and no interrupt is generated.

- Floating-Point Underflow Interrupt, generated if the result of a floating-point operation has a nonzero mantissa, and an exponent less than -1024_{10}.

- Floating-Point Overflow Interrupt, generated if the result of a floating-point operation has an exponent greater than $+1023_{10}$ or if floating-point division by zero is attempted.

- Illegal Instruction Interrupt, generated if an attempt is made to execute a priviledged instruction when in the guard mode or to execute an instruction with a function code of 00 or 7700.

3.2.3.2. Conditional Interrupts

Conditional interrupts indicate program contingencies which may not require immediate intervention, and include the following:

- BCR Parity Interrupt, caused by detection of a parity error when reading the word contained in a Buffer Control Register (BCR). Each time that a data transfer is made between the central processor buffer and external equipment, a Buffer Control Word (BCW) is read by the I/O section, updated, and written back into a BCR (see 3.2.4.1). Data Transfers may be made through I/O channels using either the Internally Specified Index (ISI) or Externally Specified Index (ESI) modes (see 3.2.4.2). If there is a parity error during the read cycle, the write cycle will write all 0's back into the BCR. An ISI channel will be terminated; however, an ESI channel will not be terminated (to prevent loss of data from other sources on the multiplexed channel).

- I/O Data Parity Interrupt, caused by detection of a parity error during I/O operations when reading a data word addressed by a BCW for output or ESI input transfer. Data transfers will be completed and the channel will not be deactivated.

- Power Loss Interrupt, caused by detecting that the input line voltage has dropped below a predetermined value. The interrupt routine has a fixed time of about five milliseconds during which the CPU and primary storage power supplies will furnish sufficient output to permit normal operations to store the state of the program so that shutdown is orderly; and, when power is restored, the program is supplied with an entry point for recovery in an orderly manner.

- External Interrupt, caused by a signal sent from a peripheral device to the CPU, together with a status word on the data input lines. The interrupt routine contains a Store Channel (17) instruction for storing the status word in primary storage. The channel number is automatically entered into the Interrupt Address Storage Register (IASR). The status word is then analyzed to condition operation of the central processor.

- Monitor Interrupt, caused when a buffer activated with monitor is either filled (input) or emptied (output).

- Day Clock Interrupt, generated approximately every 6 seconds, which permits processing operations to be conditioned by the time of day.

- Real Time Clock Interrupt, generated each time that the real time clock recycles from all 1 bits to all 0 bits. Since the recycling time can be program-controlled, that is, the clock can be set to specify the length of time until the next recycle, the interrupt can be used to measure the elapsed time of a program, as a timer to prevent program looping, and for other functions.

3.2.4. Input/Output Control Section

The input/output control section of the CPU controls and multiplexes data flow between primary storage and peripheral subsystems through the use of input/output (I/O) channels. The processor has a basic set of 12 I/O channels, which may be expanded in units of four to a maximum of 24 channels (channel 0 is reserved for the operator's console and Day Clock). Each normal channel may operate at a 30-bit word transfer rate of 555 KC. Compatible channels may operate at 250 KC transfer rate.

3.2.4.1. Input/Output Registers

The input/output section contains the data paths and control registers used in the transfer of data and in the processing of interrupts. The registers are as follows:

- Buffer Control Registers (BCR), each of which contains a Buffer Control Word (BCW) as an index for control of transfer of data between a buffer area of core storage and a peripheral device.
- Output Data Registers, which hold data being sent on the output data lines long enough to meet the requirements of slow speed peripheral devices.
- Channel Select Register (CSR), which is referenced by I/O instructions during normal operations to determine which channel to activate, deactivate, or test.
- Interrupt Address Storage Register (IASR), which contains the identifying number of a channel receiving an external interrupt or monitor interrupt signal. During the interrupt subroutine, the IASR is used in place of the CSR to specify the channel. When the Store Channel Number instruction (7772) is executed, the number of the IASR is stored in a memory location. The reference to the IASR does not alter the contents of the CSR; after the subroutine is completed, the I/O instructions again refer to the CSR. The IASR is enabled from the time the interrupt is honored (enter subroutine) until the interrupt lockout is released.

  NOTE: If interrupts are locked out under program control, the CSR rather than the IASR, is referenced by I/O instructions.
- Parity Error Channel Storage Register (PECSR), which is used in error routines initiated by a BCR parity error interrupt or I/O data parity error interrupt. The number of the I/O channel being serviced at the time of the interrupt is automatically stored in the PECSR. When the Store Channel Number instruction (7772) is executed during the parity error routine, the contents of the PECSR rather than the IASR or CSR are stored in a memory location. Within the BCR or I/O data parity error routine, the CSR specifies which channel to activate, deactivate, or test.

3.2.4.2. Index Modes

The Buffer Control Word (BCW) contained in a Buffer Control Register (BCR) specifies the address of the next data word to be read or written and is the index to an input/output operation. The address of the BCW (or index) may be specified by either of two possible modes of operation when a peripheral subsystem requests an input or output data transfer: Internally Specified Index (ISI) mode, used with standard peripheral devices connected directly to I/O channels of the CPU; and Externally Specified Index (ESI) mode, used with peripheral devices connected to I/O channels of the CPU through a communications multiplexer.

■ Internally Specified Index (ISI) Mode

The Internally Specified Index (ISI) mode provides a unique BCR for each channel which serves a standard peripheral subsystem. Only one I/O subsystem path is assigned to a channel, and the data to and from a peripheral is stored in continuous locations within a defined buffer (see Figure 3–5). Buffer control of I/O operations is maintained by the BCW, which has been loaded into the BCR at a fixed location internally specified by the CPU, through the I/O initializing instructions. The BCW has the following format:

BUFFER CONTROL WORD (BCW) FORMAT

WORD COUNT (minus decrement)		FIRST ADDRESS OF BUFFER (plus increment)
29 — 18	17	16 — 0

Bits 00–16	Indicate the first address of the allotted buffer. As each data transfer takes place, this address (in the BCR) is incremented by 1. Since 17 bit positions are available, all locations in primary storage are accessible.
Bit 17	Not used.
Bits 18–29	Indicate the allotted word count. With each data transfer, this count (in the BCR) is decremented by 1. The 12 bits made available for the count permit a maximum buffer length of 4096_{10} words. For data transfers greater than 4096_{10} words, requests may be segmented into several BCW's. When the count is equal to 0, following a transfer, the buffer is terminated.

■ Externally Specified Index (ESI) Mode

The Externally Specified Index (ESI) mode provides different buffer areas, each defined by a separate BCR, or pair of BCR's, for each of a number of I/O devices that are connected to the I/O channel by a multiplexer. Buffer control is maintained through the BCR, with the exception that more than one input BCR and output BCR are used for an ESI channel. The index is externally specified by the external subsystem (the CTMC) which gives a value to the CPU through I/O channels, with each value given corresponding to a particular I/O device and uniquely defining whether input or output is requested. The location of the appropriate BCR is placed on the input data lines by the requesting peripheral unit. Each input data word address is placed in bits 15–29 of the word in the defined buffer. Each output data word address is taken from bits 00–14 of the buffer. For accessing the full range of primary storage, two additional bits are taken from the MSR. Input word storage for the ESI and ISI modes is compared in Figure 3–5.

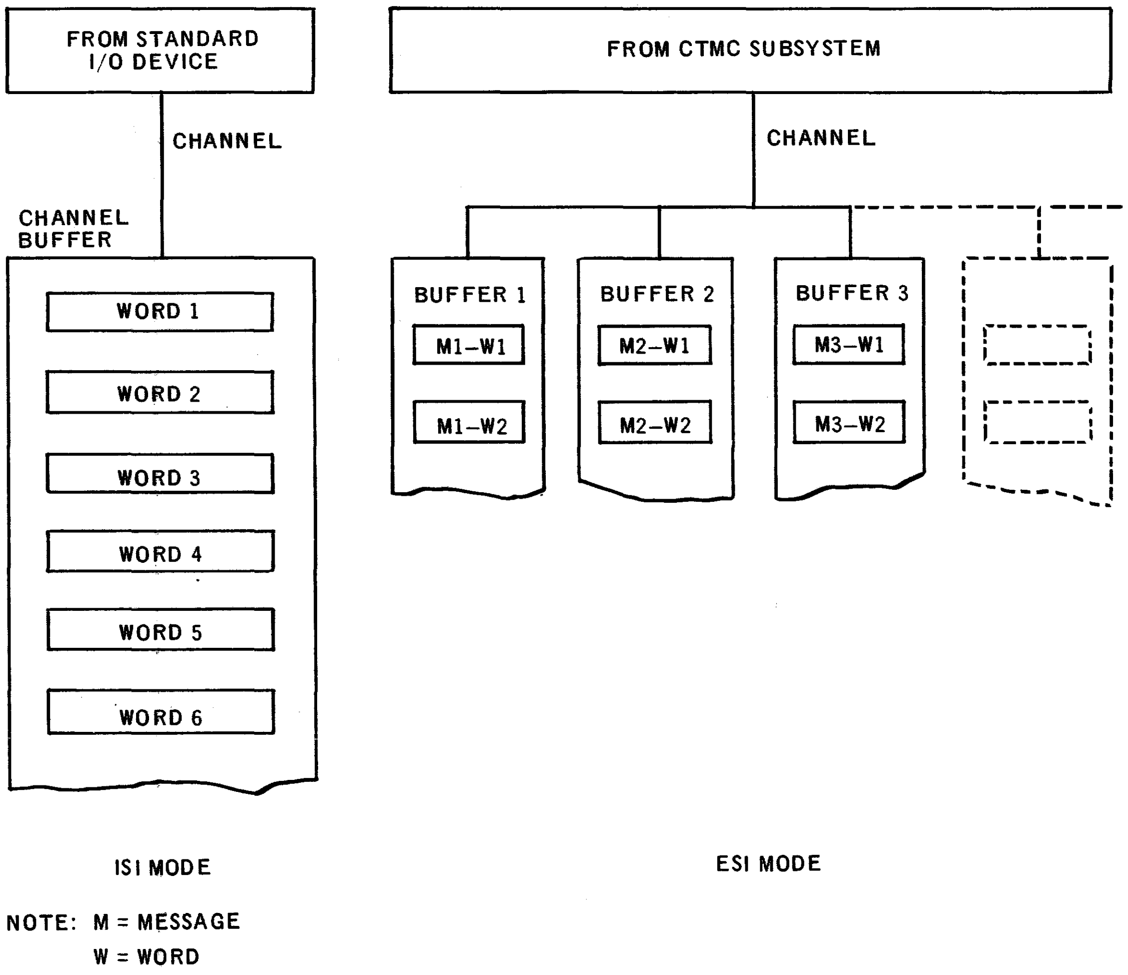


Figure 3–5. Input Word Storage, Internally Specified Index (ISI) Mode Versus Externally Specified Index (ESI) Mode

3.2.4.3. External Equipment Control

External peripheral equipment control by the CPU is attained by function (control) codes and status words conveyed over the data lines of I/O channels connected to the peripheral equipment. A function code is sent by the CPU to a peripheral unit to initiate a particular operation at the unit. The function code is distinguished from a data word by activation of the External Function control line. The use of function codes rather than a set of instructions makes the CPU independent of the characteristics of the I/O unit, enabling connection of a variety of peripheral equipment to the CPU with no modification to computer logic.

Following the receipt of a function word by a peripheral subsystem, and either completion of the requested operation or detection of an error or abnormal condition which prevents normal completion, a status word is sent to the CPU by peripheral equipment to inform the computer program of the status of the equipment and the readiness of the peripheral subsystem to initiate or terminate data transfer. A status word is distinguished from a data word by activation of the External Interrupt (EI) line. Activation of an EI line forces program control to a specific primary storage address where the Operating System initiates a routine to analyze the status word and to take the appropriate action.

3.2.4.4. Operator's Console (Alternate)

The Operator's Console, Type 0400, is an input/output system hardware component which, when provided, is an integral part of the UNIVAC 494 System. Although separate from the CPU, the console functions as part of the I/O section of the CPU. The console provides the means for operator communication with the system and for monitoring of system operations. The console informs the operator when manual intervention is needed and/or accomplished, particularly in readying peripheral subsystems, performing special functions, and correcting abnormal operating conditions.

The operator's console consists of a keyboard and printer, control and display panel, and the Day Clock display (see 3.2.1.3), all of which share channel 0 of the CPU. The console may be replaced by the operator's display console (see 3.2.4.5) for enhanced programming flexibility and the added feature of a cathode-ray tube (CRT) display of messages.

- Keyboard and Printer

  The keyboard and printer unit comprises a four-bank typewriter keyboard for sending information to the CPU, and a printer for furnishing hard copy records of the input and output of the console. The keyboard contains all of the necessary keys and special symbols for encoding the 64-character Fieldata code, and has an interrupt key for generating an EI signal for gaining access to the CPU. Program control characters, limited sets of data, and messages indicating operator initiation of certain procedures or operator response to CPU messages may be input to the CPU through the console keyboard. The printer, in turn, contains the necessary characters and symbols for printing Fieldata code messages. The printer shows CPU replies to the keyboard input, prints messages indicating system status or requests for actions to be taken, and outputs other information as called for by the program or the operator.

The keyboard and printer unit also contains controls and indicators for monitoring and directing its functions.

- Control and Display Panel

  The control and display panel contains controls and indicators for monitoring system operations and for making certain adjustments as desired. The indicators show fault or abnormal conditions in both the hardware and the program being executed, such as illegal instructions, improper switch settings, abnormal power input, and disabled condition of equipment, which may affect the system in program execution. They also show the orientation, or address and status, of the program. Controls are provided for clearing and resetting the hardware components, loading of program instructions, and stopping, starting, and sequencing of program execution, such as performance of jump instructions.

- Day Clock Display

  The Day Clock display shows the time of day as carried by the CPU for use in time orientation of the program. The time is given in hours, minutes, and hundredths of minutes. Controls for clearing, stopping, starting, and setting the Day Clock are contained on the console control and display panel.

3.2.4.5. Operator's Display (CRT) Console

The Operator's Display (CRT) Console, Type 4009-97, (see Figure 3–6) is an input/output, visual communication, system hardware component which is an integral part of the UNIVAC 494 System and which, although separate, functions as part of the I/O section of the CPU. The display console provides a means for operator communication with the system and for monitoring of system operations.

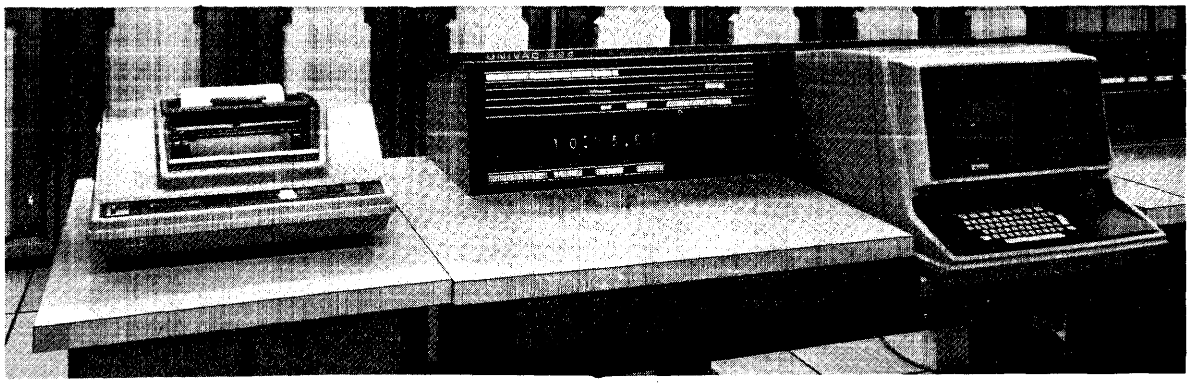


Figure 3–6. Operator Display (CRT) Console

In addition to the functions provided by the operator's alternate console (see 3.2.4.4), which are enhanced, the display console furnishes a cathode-ray tube (CRT) display of the CPU and operator generated messages. Through both the CRT display and printed output, the console informs the operator when manual intervention is needed and/or accomplished, as in readying peripheral subsystems, performing special functions, and correcting abnormal operating conditions. Selection of CRT display or printed output may be made through system software or by manual intervention at the console.

The display console consists of a keyboard and CRT display unit, PAGEWRITER, Day Clock display, and control and display panel, all of which share channel 0 of the CPU. The display console is the standard console for the UNIVAC 494 System. A detailed description of the display console is given in *UNIVAC 494 Real-Time System Display Console Component Description, UP-7610* (current version).

- Keyboard and CRT Display Unit

  The keyboard and CRT display unit comprises a four-bank typewriter keyboard for sending information to the CPU and a viewing screen component for display of information from the CPU. The keyboard contains all of the necessary keys and special symbols for encoding the 64 characters of the Fieldata code, and eight interrupt keys for generating external interrupt signals (with a unique status code for each key) for gaining access to the CPU, thus providing the operator with more flexibility in program control than does the standard console. Program control characters, limited sets of data, and messages indicating operator initiation of certain procedures or operator response to CPU messages may be input to the CPU through the console keyboard. The CRT component has a 1024-character buffer memory and a disassembly register for receiving and storing CPU messages after disassembly into six-bit character codes. The CRT unit then reassembles the codes into 16-line, 64-character message display formats, character addressable under program control. No direct communication exists between the console keyboard and the CRT; all messages are handled through the CPU. However, when a key is pressed, the corresponding character is displayed on the CRT through cooperative action between the CPU and the display console control program so that the input message may be edited before its entry into the operative system.

  The keyboard and CRT display unit also contains other controls and indicators for monitoring and directing its functions.

- PAGEWRITER

  The PAGEWRITER is an online printer unit, separate from the keyboard and CRT display unit, and not in direct communication with either component. The PAGEWRITER logs or records all transactions between the CPU or control program and the operator, and is equipped to handle all of the characters and symbols of the Fieldata code. The PAGEWRITER prints CPU replies to the keyboard input, prints messages indicating system status or requests for actions to be taken, and outputs other information as called for by the program or the operator. Output messages are printed in the format 80 characters per line (with horizontal spacing of 10 characters per inch) and six lines per inch. The printing rate is 25 characters per second.

The PAGEWRITER does not operate automatically or simultaneously with the CRT display. Messages may be directed to the PAGEWRITER or to the CRT display through the system software or by manual intervention at the console.

- Control and Display Panel

  The control and display panel contains controls and indicators for monitoring system operations and for making certain adjustments as desired, in the same manner as the alternate console. Indicators and switches include Program Address Counter, Disables, Modes, Faults, Select Jumps and Stops, Release Jumps and Stops, System Controls, and Day Clock controls.

- The Day Clock display shows the time of day carried by the CPU in hours, minutes, and hundredths of minutes, with controls on the control and indicator panel.

3.3. PRIMARY STORAGE

Primary storage, or main memory, is a major component of the UNIVAC 494 System, separate from the CPU, and allocatable in the same manner as peripheral equipment. Through efficient memory organization and design, in the interface with the CPU, primary storage provides a high performance, immediate access facility for storage and retrieval of instructions and data, and for accommodation of input/output buffer areas.

Significant characteristics of primary storage include:

- 750 nanosecond read/restore time
- 65,536 or 131,072 30-bit word capacity
- Multiple word, single word, or partial word reference
- Parity checking/generating on all storage references

3.3.1. Storage Design

Primary storage is composed of 34-plane, coincident current, ferrite core arrays, providing for 30-bit words with one parity bit for each half-word. Memory transactions may transfer 60-bit double precision words, 30-bit whole words, 15-bit half-words, or program selected bits of a word.

The primary storage unit is made up of independently accessible storage modules of 32,768-word capacity. Two modules are contained in one cabinet to form memory banks providing 65,536-word ranges of addresses. Each module presents a continuous addressing structure to the CPU for increased processing efficiency. In normal system operation, for decreasing processing time, the memory banks use odd-even addressing, with all odd addresses in the 65,536-word range being referred to one module of a bank and all even addresses being referred to the other (see Figure 3–7). In certain applications, as for bypassing memory fault areas if necessary, straight addressing can be selected manually, with all addresses, odd and even consecutively in a 32,768-word range, being referred to one or the other module in the same bank.

3.3.2. Storage Configuration

Available storage configurations permit the use of one memory bank (cabinet containing two modules), providing the minimum system capacity of 65,536 words, or two memory banks (four modules), providing maximum capacity of 131,072 words. The growth increment from the minimum capacity is one memory bank.

3.3.3. Storage Modules

Storage modules are essentially passive elements which react with great speed and efficiency to the demands of the CPU in furnishing random access core storage. Each module contains the necessary controls and circuitry for performing the following functions:

- Grant storage access to the CPU.
- Accept an address from the CPU.
- Store or retrieve a word at the address.
- Check or generate parity on each access, and generate an interrupt signal to the CPU if a parity error occurs.
- Issue an acknowledgement signal indicating that a storage reference has been completed.

Each module provides storage for 32,768 addressable words of 30 data bits, with each word carrying a parity bit on a nonaddressable level for each half of the word. With the exception of certain fixed locations reserved for system usage, all storage areas are available to program assignments. Each storage module has the following physical components:

- a 15-bit address register
- a 30-bit read/write register
- parity checking/generating circuits
- request/acknowledge circuits
- maintenance switches allowing the module to be removed for servicing

The 15-bit address register of each storage module provides a continuous addressing structure of 32,768 words. The CPU generates a 17-bit address for each storage reference, specifying which of 131,072 possible memory locations is involved. For odd-even addressing, two of the address bits, 2^0 (representing the module) and 2^{16} (representing the memory bank), are used by the Memory Select Register (MSR) for identification and selection of one or another of the four possible modules in the memory configuration; for straight addressing, bits 2^{16} and 2^{15}, are used for the memory bank and module, respectively. The remaining 15 bits are sent to the address register of the selected module providing a unique set of addresses for each module. This organization permits continuation of addressing from one module to the next to attain the desired locations over the entire memory range (see Figure 3–7).

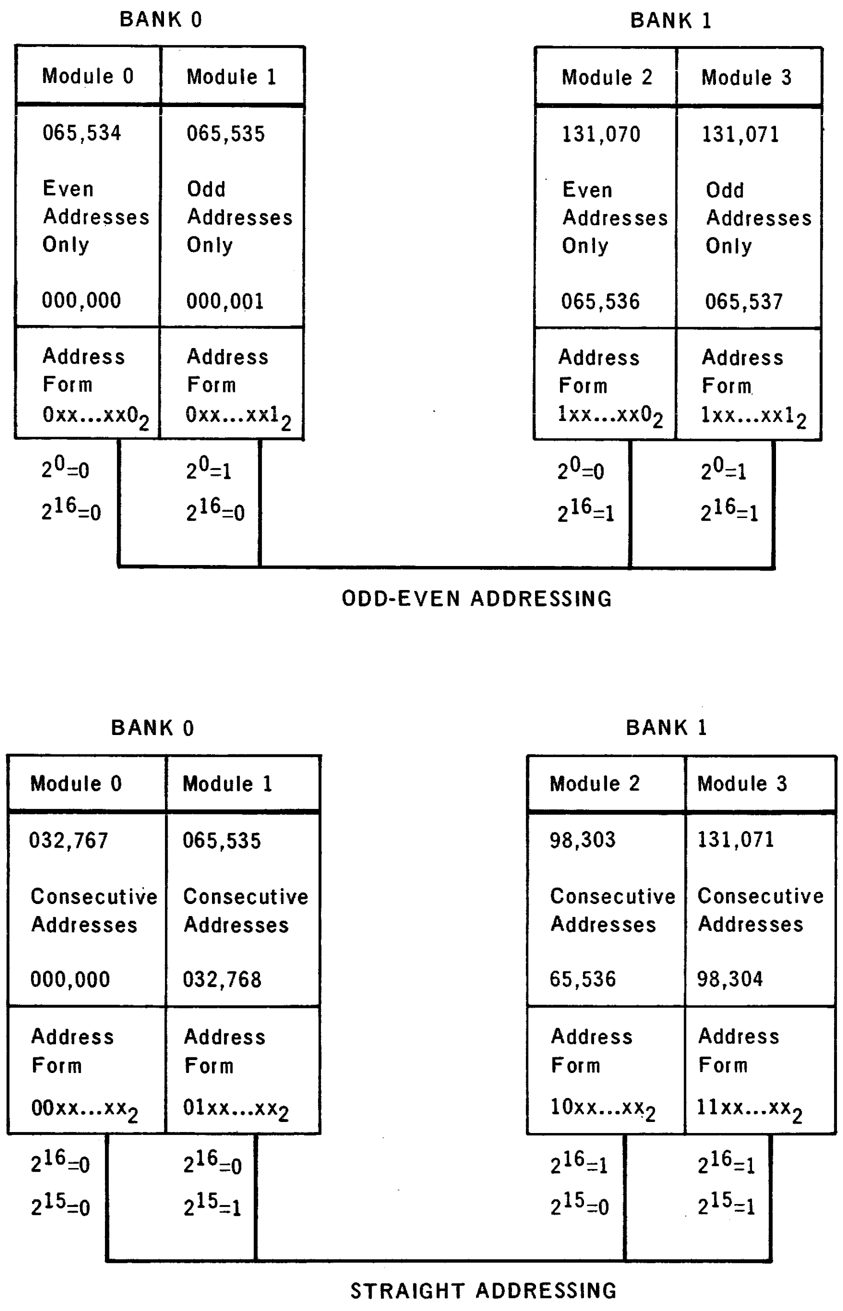
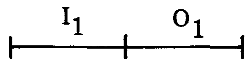


Figure 3–7. Memory Organization

Parity is checked (reading) or calculated (writing) on each 15- or 30-bit storage access. If a parity error is detected, the module issues a parity interrupt signal to the CPU. The word in its incorrect form will be rewritten to memory, thereby assuring that the word may again be located.

3.3.4. Primary Storage Interface

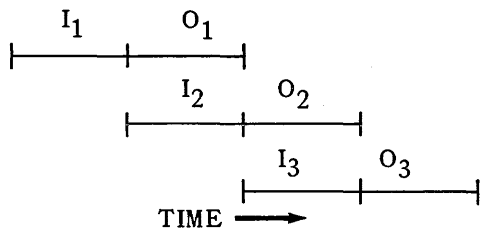
The interface between the CPU and primary storage is responsible for retrieving and restoring data and instructions. The execution of each instruction by the CPU may require two or more primary storage cycles, namely, an instruction cycle and one or more fetch or result cycles as follows:



The speed and efficiency of memory transactions are enhanced by the ability of the CPU to use independent paths to the storage modules, that is, two modules may be cycled simultaneously. The capability of the storage modules to be cycled independently is used to increase the frequency of use of facilities in memory overlap and to promote processing performance, through interleaving, as in odd-even addressing.

3.3.4.1. Overlap

Overlap occurs as the CPU requests the next sequential instruction word from memory during the last operand fetch or result storage cycle of the current instruction. If the next instruction lies in a memory module different from the module of the current operand/result, the two words are retrieved from primary storage in parallel. Overlap may be illustrated as follows:



The overlap technique may achieve up to a fifty percent performance increase over other memory organization subject to the following conditions depending upon the instruction involved (see Appendix A).

- If the current instruction is a read or store class instruction and a memory reference is not required to obtain the operand (that is, a k-designator of 0, 4, or 7), the next instruction is always retrieved during the operand cycle of the current instruction.
- If the current instruction is a read or store class instruction and the operand cycle requires a memory reference (that is, a k-designator of 1, 2, 3, 5, or 6), overlap occurs only if the current operand is in a memory module different from that of the next sequential instruction.
- If the current instruction is a replace class instruction, overlap occurs when the store address of the replace instruction is in a memory module different from that of the next sequential instruction.

- If the execution of the current instruction results in a jump or skip condition, no gain in performance is achieved by the overlap feature because the next sequential instruction (hardware, not program) is not the next instruction to be executed; the next sequential instruction is discarded. The reading of the instruction at the jump-to/skip-to address, is not initiated until completion of the cycle to obtain the instruction which is discarded.

3.3.4.2. Interleave

Interleaving increases the probability of overlap which occurs when the current operand and the next sequential instruction lie in different memory modules. In the interleave technique (see odd-even addressing), two memory modules of 32,768 words each are merged so that all even addresses within 65,536 sequential words are contained in one memory module, and all odd addresses are contained in the other memory module. Thus, with the interleave technique, overlap is achieved with significant frequency, so that data and instruction areas need not be separate in order to take advantage of the overlap feature.

NOTE: Conclusions on the type of instruction to be used within a program should not be necessarily drawn from the percentage gain described for overlap and interleave techniques. The UNIVAC 494 instruction repertoire contains many instructions which are equivalent to two separate but less powerful instructions. For example, a replace instruction is equivalent to an enter and store instruction. A load, store, or arithmetic operation plus a test and conditional skip are combined in many individual instructions.

3.3.5. Preassigned Storage Addresses

A set of fixed memory locations is reserved for system usage in such functions as information displays, interrupt subroutine entrance, ISI buffer control registers (BCR), and clock data. The fixed memory locations are defined in Table 3–1.

ADDRESS	FUNCTION
x00000	Illegal Instruction
x00001	Memory Protection or Timeout Interrupt
x00002	Power Loss Interrupt
x00003	Memory Parity Bank 0 Interrupt
x00004	Memory Parity Bank 1 Interrupt
x00005	BCW Parity Interrupt
x00006	I/O Data Parity Interrupt
x00007	Executive Return Interrupt
x00010	Floating Point Underflow Interrupt
x00011	Floating Point Overflow Interrupt
x00012	External Synchronization #1 Interrupt
x00013	External Synchronization #2 Interrupt
x00014	Real Time Clock Interrupt
x00015	Day Clock Interrupt
x00016	Day Clock Time
x00017	Real Time Clock
x00020	ESI External Interrupt
x00021	ESI Input Monitor Interrupt
x00022	ESI Output Monitor Interrupt
x00023	Memory Parity Bank 2 Interrupt
x00024	ISI External Interrupt
x00025	ISI Input Monitor Interrupt
x00026	ISI Output Monitor Interrupt
x00027	Memory Parity Bank 3 Interrupt
x00030	Test and Set Interrupt
x00031 to x00037	Not Used
x00040	Output BCW – Channel 0
x00041	Output BCW – Channel 1
x00042	Output BCW – Channel 2
x00043	Output BCW – Channel 3
x00044	Output BCW – Channel 4
x00045	Output BCW – Channel 5
x00046	Output BCW – Channel 6
x00047	Output BCW – Channel 7

Table 3–1. Fixed Memory Locations
(Part 1 of 2)

ADDRESS	FUNCTION
x00050	Output BCW – Channel 8
x00051	Output BCW – Channel 9
x00052	Output BCW – Channel 10
x00053	Output BCW – Channel 11
x00054	Output BCW – Channel 12
x00055	Output BCW – Channel 13
x00056	Output BCW – Channel 14
x00057	Output BCW – Channel 15
x00060	Output BCW – Channel 16
x00061	Output BCW – Channel 17
x00062	Output BCW – Channel 18
x00063	Output BCW – Channel 19
x00064	Output BCW – Channel 20
x00065	Output BCW – Channel 21
x00066	Output BCW – Channel 22
x00067	Output BCW – Channel 23
x00070 to x00077	Not Used
x00100	Input BCW – Channel 0
x00101	Input BCW – Channel 1
x00102	Input BCW – Channel 2
x00103	Input BCW – Channel 3
x00104	Input BCW – Channel 4
x00105	Input BCW – Channel 5
x00106	Input BCW – Channel 6
x00107	Input BCW – Channel 7
x00110	Input BCW – Channel 8
x00111	Input BCW – Channel 9
x00112	Input BCW – Channel 10
x00113	Input BCW – Channel 11
x00114	Input BCW – Channel 12
x00115	Input BCW – Channel 13
x00116	Input BCW – Channel 14
x00117	Input BCW – Channel 15
x00120	Input BCW – Channel 16
x00121	Input BCW – Channel 17
x00122	Input BCW – Channel 18
x00123	Input BCW – Channel 19
x00124	Input BCW – Channel 20
x00125	Input BCW – Channel 21
x00126	Input BCW – Channel 22
x00127	Input BCW – Channel 23
x = MSR	

Table 3–1. Fixed Memory Locations
(Part 2 of 2)

3.4. TRANSFER SWITCH

The transfer switch unit in an optional device which provides a means for switching peripheral subsystems between CPU's. The switching is static rather than dynamic; the subsystem is physically removed from the processor. Switching is accomplished manually.

The switch unit may be used to take peripheral subsystems offline for maintenance while switching standby subsystems to online condition.

The transfer switch may be used in an X, Y, or cascade configuration. Figure 3–8 illustrates these configurations.

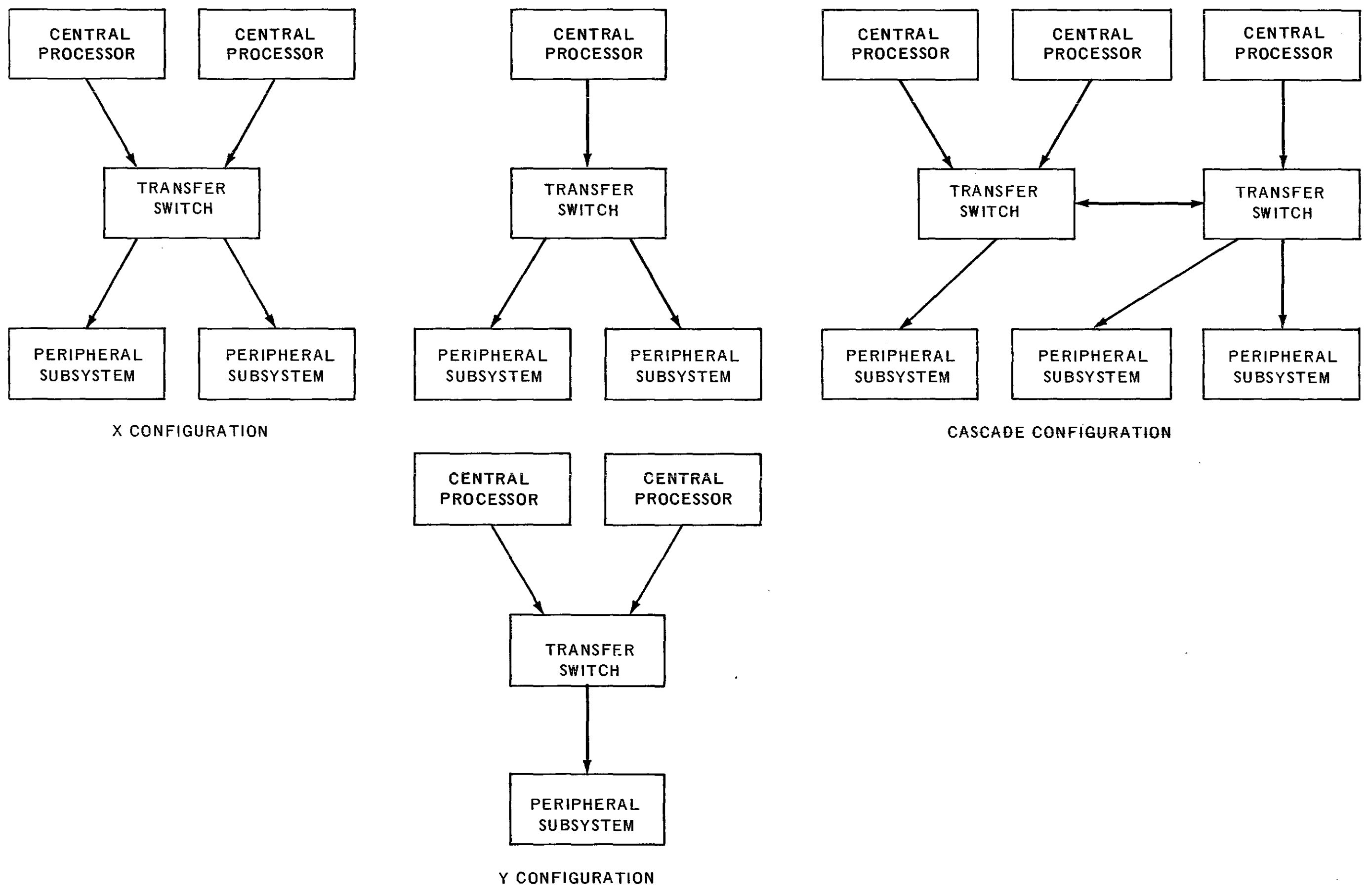


Figure 3–8. Transfer Switch Configurations

3.5. ON-SITE PERIPHERAL SUBSYSTEMS

On-site peripheral subsystems comprise one or more peripheral units of the same kind, with the appropriate control unit(s), connected directly to the desired CPU input/output channel(s). The peripheral subsystems perform diverse functions complementary to the CPU according to the design characteristics of the peripheral equipment and the needs of the system. The functions may include input/output, auxiliary storage, data preparation and communications, file handling, and such other activities as required.

The following types of peripheral subsystems are available for use on-site with the UNIVAC 494 System:

- Random Access Storage Subsystems
- Magnetic Tape Subsystems
- Unit Record Subsystems
- Communication Subsystem

When an operating program requires access to a subsystem, the CPU issues control signals which select the needed subsystem and initiate the desired action. Program execution by the CPU continues automatically until the subsystem has completed the requested activity. The subsystem signals the CPU when the activity is completed, and the CPU deals with the result of the action.

Each subsystem is controlled by a control unit which performs the following functions:

– Interprets the control signals and instructions issued by the CPU.

– Effects the transfer of data to or from the selected unit and the CPU.

– Indicates the status or availability of the peripheral units to the CPU.

– Informs the CPU when errors or faults occur which affect the operation of the subsystem.

3.5.1. Random Access Storage Subsystems

Random access storage subsystems provide high speed auxiliary storage facilities for program elements and libraries, subroutines, and data which, for reasons of economy and efficiency, may not be kept in primary storage, but which must be called into the operating programs rapidly and/or often, as in program development and control, and language translation operations. Available random access subsystems afford a variety of access times, transfer rates, and storage capacities to meet the requirements of the individual user.

Each storage system comprises magnetic drum devices with read/write heads which "fly" or float on the boundary layer of air created by the rotation of the drum.

The following subsystems are included:

- FH-432 Magnetic Drum Subsystem
- FH-432/FH-1782 Magnetic Drum Subsystem
- FH-880 Magnetic Drum Subsystem
- FASTRAND II Mass Storage Subsystem
- FASTRAND III Mass Storage Subsystem

■ Control Units

Each subsystem has a control unit(s) which interfaces the subsystem with the CPU, and which has the following principal functions:

- Receives function words (control information) from the CPU, and translates this information to control signals for the drum units.
- Controls the orderly accessing of drum locations.
- Assembles and disassembles data and control words for acceptance by the CPU and drum units.
- Requests and acknowledges data transfers, and synchronizes the flow of data between the CPU and drum units.
- Interprets signals from the drum units for both normal and abnormal conditions, and notifies the CPU of the drum conditions.

3.5.1.1. FH-432 Magnetic Drum Subsystem

The FH-432 Magnetic Drum Subsystem (see Figure 3–9) is a high speed, large capacity, random access storage medium consisting of one Type 6013 FH-432 Drum Control Unit (containing one FH-432 Drum Unit) and from two to eight Type F0696 FH-432 Drum Units, or one Type 5012-02 FH-432/FH-1782 Control Unit and up to eight Type 6016 FH-432 Drum Units (see 3.5.1.2). Each drum unit can store 262,144 individually addressable 30-bit words with parity or the equivalent of 1,310,720 alphanumeric characters. The average access time for any word in the subsystem is 4.33 milliseconds.

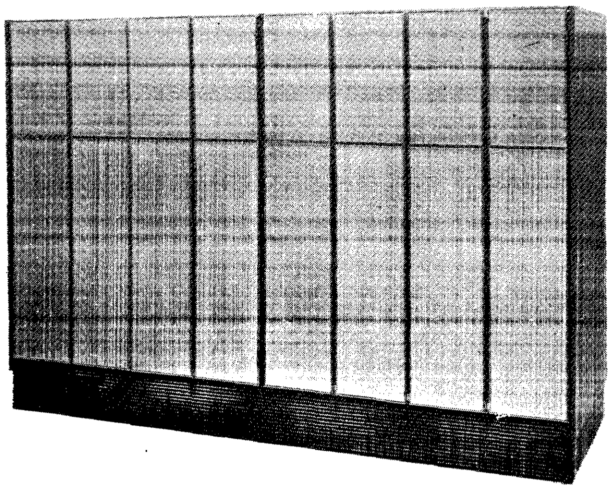


Figure 3–9. FH-432 Magnetic Drum Subsystem

The FH-432 drum is a magnetic coated cylinder containing 432 recording tracks, each equipped with a read/write head, of which 384 tracks are used for storing data; the remaining tracks are used for spares and for timing purposes. Data and parity bits are recorded on 128 three-track bands, each with a capacity of 2,048 words. Reading and writing functions occur in a three-bit parallel mode on the three tracks of a band simultaneously at a maximum transfer rate of 240,000 words or 1,200,000 characters per second.

The accuracy of data recording is verified by odd parity checking. When data is recorded on the drum, three parity bits per word are generated by the control unit and stored with the word. Parity is checked automatically when data is read from the drum. If a parity error occurs, an external interrupt signal is generated and the CPU is notified of the address of the word in which the error was detected.

The control unit for the subsystem interfaces the FH-432 Drum Units to an I/O channel of the CPU with principal functions as given in 3.5.1.

- Functions

  The function repertoire of the FH-432 Drum Subsystem consists of the following:

  - Write
  - Terminate Without Interrupt
  - Terminate With Interrupt
  - Bootstrap Without Interrupt
  - Continuous Read
  - Search
  - Search Read
  - Bootstrap With Interrupt
  - Block Read
  - Block Search
  - Block Search Read

- Characteristics

  Characteristics of the FH-432 Magnetic Drum Subsystem are given in the following table. Detailed information may be found in *UNIVAC 494 Real-Time System FH-432 Magnetic Drum Subsystem Programmer/Operator Reference Manual, UP-4102* (current version).

PARAMETERS	SPECIFICATIONS		
	PER DRUM	MINIMUM CONFIGURATION	MAXIMUM CONFIGURATION
Storage Capacity (30-bit words)	262,144	786,432	2,359,296
Alphanumeric Characters	1,310,720	3,932,160	11,796,480
Access Time			
Minimum	120 microseconds		
Average	4.33 milliseconds		
Maximum	8.55 milliseconds		
Drum Speed	7120 RPM		
Number of Read/Write Heads	432 (one per track)		
Word Transfer Rate	240,000 words/second (maximum)		
Character Transfer Rate	1,200,000 characters/second (maximum)		
Number of Drums (per subsystem)	3 to 9		
I/O Channels Required	1 channel		

3.5.1.2. FH-432/FH-1782 Magnetic Drum Subsystem

The FH-432/FH-1782 Magnetic Drum Subsystem (see Figure 3–10) is a high speed, large capacity, random access storage medium consisting of one or two Type 5012 Control Units, and from one to eight Type 6015 FH-1782 Drum Units or Type 6016 FH-432 Drum Units, or a combination of both types not exceeding eight drum units.

The hybrid subsystem provides both the fast access time of the FH-432 Drum Subsystem and the large storage capabilities of the FH-1782 Drum Subsystem. The combination affords nonvolatile random access storage with a maximum transfer rate of 240,000 words per second, average access time as low as 4.33 milliseconds, and storage capacities as great as 2,097,152 data words per drum unit.

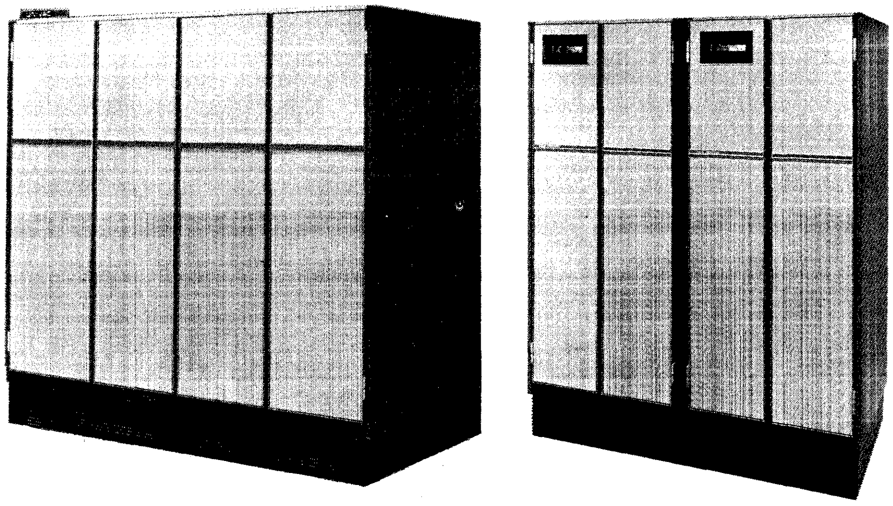


Figure 3–10. FH-432/FH-1782 Magnetic Drum Subsystem

- Control Unit

The control unit interfaces the FH-432 and/or FH-1782 Drum Units to an I/O channel of the CPU, and has the principal functions as given in 3.5.1.

With the use of two Type 5012 Control Units, fully simultaneous operation can be achieved with this subsystem in a dual access configuration. One unit can control the execution of any function on any drum while the other unit can control any function on any other drum in the same subsystem. The two control units, operating on separate CPU I/O channels, are completely independent, providing backup capabilities.

The physical characteristics of the Type 6016 FH-432 Drum Unit are identical to those of the Type F0696 FH-432 Drum Unit described in 3.5.1.1, with the exception that the Type 6016 Drum Unit has 486 recording tracks, with a read/write head for each.

The Type 6015 FH-1782 Drum Unit is a magnetic coated cylinder containing 1,890 recording tracks, each equipped with a read/write head, of which 1,556 are used for storing data; the remaining tracks are used for spares and for timing purposes. Data and parity bits are recorded on 256 six-track bands having a capacity of 8,192 words each, providing a total storage capacity of 2,097,152 words per drum. Reading and writing functions occur in a six-bit parallel mode, simultaneously, at a maximum transfer rate of 240,000 words or 1,200,000 characters per second. Each word is individually addressable with an average access time of 17 milliseconds.

The accuracy of data recording is verified by odd parity checking. When data is recorded on the drum, three parity bits per word are generated by the control unit and stored with the word. The parity of each word is checked automatically as it is read from the drum. If a parity error occurs, an external interrupt signal is generated and sent to the CPU with a status code.

■ Functions

The function repertoire of the FH-432/FH-1782 Magnetic Drum Subsystem consists of the following:

- Automatic Bootstrap
- Continuous Write With Interrupt
- Continuous Write Without Interrupt
- Continuous Read With Interrupt
- Continuous Read Without Interrupt
- Block Read
- Block Search
- Block Search Read
- Read Early (parity error recovery)
- Read Late (parity error recovery)
- Search
- Search Read
- Send Angular Address
- Terminate With Interrupt
- Terminate Without Interrupt

■ Characteristics

Characteristics of the FH-432 drum and FH-1782 drum are given in the following table. Detailed information may be found in *UNIVAC 494 Real-Time System FH-432/FH-1782 Magnetic Drum Subsystem Programmer/Operator Reference, UP-7630* (current version), and in UP-4102 (for the FH-432 drum) as previously mentioned.

PARAMETERS	SPECIFICATIONS	
	FH-432 DRUM	FH-1782 DRUM
Storage Capacity (words per drum)	262,144	2,097,152
Drum Speed	7120 RPM	1770 RPM
Number of Read/Write Heads	486	1890
Recording Density (bits per inch)	889	750
Word Transfer Rate	240,000 words/second (maximum)	
Character Transfer Rate	1,200,000 characters/second (maximum)	
Access Time		
Minimum	120 microseconds	200 microseconds
Average	4.33 milliseconds	17 milliseconds
Maximum	8.55 milliseconds	34 milliseconds
Number of Drums (per subsystem)	8 maximum – any combination	
I/O Channels Required	1 or 2 channels (one for each control unit)	

3.5.1.3. FH-880 Magnetic Drum Subsystem

The FH-880 Magnetic Drum Subsystem (see Figure 3–11) is a large capacity, word-addressable, random access storage medium consisting of one Type 8103 Control Unit and from one to eight Type 7304-01 FH-880 Magnetic Drum Cabinets. Each drum has the capacity to store 786,432 30-bit words plus parity, or the equivalent to 3,932,160 alphanumeric characters. The average access time for any word in the subsystem is 17 milliseconds.

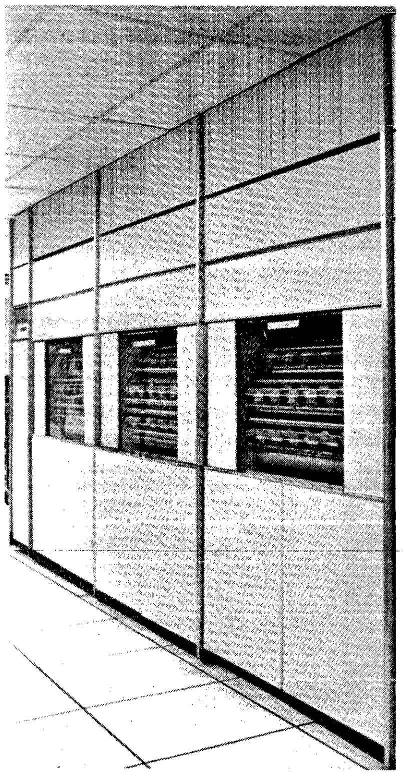


Figure 3–11. FH-880 Magnetic Drum Subsystem

The FH-880 drum is a magnetic coated cylinder containing 880 recording tracks, each equipped with a read/write head, of which 768 are active data tracks, 32 are parity tracks, and the remainder are timing tracks and spares. The 768 tracks are organized into 128 six-track bands, which are divided into 2,048 Angular Addresses that are subdivided into three Angular Sections, each capable of storing one 30-bit word.

When a word is written on the drum, the word is divided into five groups of six data bits each, with each group being recorded in parallel mode on the six tracks of a band, starting with bit positions 29 through 24 of the data word followed by bit positions 23 through 18, and so on. A parity bit is generated so that the total number of 1 bits recorded (data bits plus the parity bit) is an odd number. The parity bit is recorded in a parity track location which is internally associated with the data word location.

When a word is read from the drum, groups of six data bits each are read in parallel mode from the band and are assembled to form a 30-bit word, the first group in bit positions 29 through 24, the second in bit positions 23 through 18, and so forth. An odd parity is calculated and checked against the previously recorded parity bit. If the parity comparison is correct, the assembled data word is made available to the CPU; if a parity error occurs, an external interrupt signal is sent together with a status word to the CPU.

- Control Unit

  The control unit is connected to both an I/O channel and the individual drum units, and has the principal functions, as given in 3.5.1.

- Functions

  The function repertoire of the FH-880 Drum Subsystem consists of the following:

  - Write
  - Terminate Without Interrupt
  - Terminate With Interrupt
  - Bootstrap Without Interrupt
  - Continuous Read
  - Search
  - Search Read
  - Bootstrap With Interrupt
  - Block Read
  - Block Search
  - Block Search Read

- Characteristics

  Characteristics of the FH-880 Magnetic Drum Subsystem are given in the following table. Detailed information may be found in *UNIVAC 491/492/494 Real-Time System FH-880 Magnetic Drum Subsystem Programmer/Operator Reference Manual, UP-7533* (current version).

PARAMETERS	SPECIFICATIONS
Storage Capacity (per drum)	
30-bit Words	786,432
Alphanumeric Characters	3,932,160
Access Time	
Minimum	160 microseconds
Average	17 milliseconds
Maximum	34 milliseconds
Drum Speed	1770 RPM
Number of Read/Write Heads	880 (one per track)
Word Transfer Rate	60,000 words/second (maximum)
Character Transfer Rate	300,000 characters/second (maximum)
Number of Drums (per subsystem)	1 to 8
I/O Channels Required	1 channel

3.5.1.4. FASTRAND II Subsystem

The FASTRAND II Subsystem is a large capacity, sector-addressable, random access storage medium consisting of one Type 5009 FASTRAND Control Unit and from one to eight Type 6010-00 FASTRAND II Mass Storage Units (see Figure 3–12). Each mass storage unit can store up to 25,952,256 words of 30 data bits plus parity characters, or 129,761,280 alphanumeric characters. The average access time for any word in the subsystem is 92 milliseconds.

Each FASTRAND II Mass Storage Unit contains two magnetic drums, mounted one above the other, and having a total of 13,068 tracks around the drum surfaces. There are 32 groups of 192 tracks on each drum, totalling 6,534 tracks, of which 6,144 are used for storing data; the remaining 390 tracks are used for spares and for special hardware timing purposes. Data tracks are divided into 64 sectors, with a capacity of 33 words each. The read/write functions occur in bit-serial mode at a maximum transfer rate of 37,040 words or 185,200 characters per second.

The accuracy of data recording is verified by odd longitudinal parity checking. A parity character is produced by the control unit on the basis of the data recorded within a particular sector and is then recorded at the end of the sector. When data is read, parity is checked automatically. If a parity error occurs, an appropriate status word and external interrupt signal are sent to the CPU.

The basic configuration of the FASTRAND II Mass Storage Subsystem can use a single I/O channel. The storage capacity of the subsystem can be expanded by the optional addition of from one to seven mass storage units on the channel, providing a maximum capacity of 207,618,048 30-bit words.

- Control Unit

  The FASTRAND control unit interfaces the FASTRAND II unit with an I/O channel of the CPU and has the principal functions, given in 3.5.1.

  A special dual channel FASTRAND control unit is available, permitting communication with the CPU over two I/O channels and access to two storage units simultaneously, thereby providing reading or writing on two units, or reading on one unit and writing on another at the same time. Simultaneous reading or writing from the same storage unit is impossible. If both control units attempt to gain access to the same unit simultaneously, one control unit will have priority over the other. If one control unit gains access before the other, the second control unit waits until the first is finished.

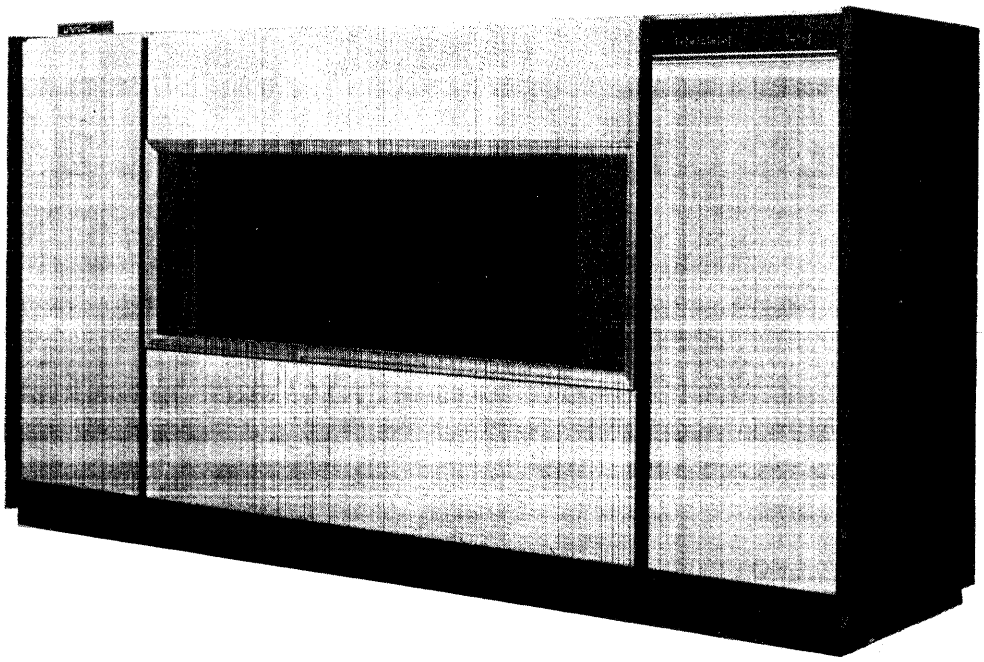


Figure 3–12. Type 6010-00 FASTRAND II Mass Storage Unit

There are 40 read/write heads used with each drum (or 80 heads per mass storage unit). The spare and timing heads are mounted in fixed positions; each head has access to only one track. The remaining 32 data heads and two special purpose heads are movable on a boom that has 192 positions, giving each movable head access to 192 tracks.

Each data track is divided into 64 sectors. Each sector contains 33 data words of 30 bits each. When data words are written on the drum, bits are serially recorded in the track. After the last word, a sentinel and parity character are written in the appropriate place within the sector. The parity character is the result obtained from the exclusive OR of all 165 data characters and the sentinel character. When a sector of information is read from the drum, data bits are read serially and transferred to the control unit where parity is checked. If parity errors occur, an external interrupt signal and status code are sent to the CPU.

- Functions

The function repertoire of the FASTRAND II Subsystem consists of the following:

- Write With Interrupt
- Position Without Interrupt
- Terminate Without Interrupt
- Position With Interrupt
- Terminate With Interrupt
- Read
- Data Recovery Read
- Search First Word (long)
- Search First Word (short)
- Search All Words (long)
- Search All Words (short)

- Optional Capabilities

Optional features which may be obtained with the FASTRAND II Subsystem are Fastbands and write lockout.

- Fastbands

Fastbands are 24 additional data tracks at the end of the drums which are accessed by fixed position heads and which are used for storing data to which rapid access is required. Since no boom movement is required to position the heads, the access time for the Fastbands is usually reduced to the drum latency time. However, this reduced timing is effective only if a head boom positioning operation is not in process on the specified mass storage unit, in which case an additional delay of up to 86 milliseconds is added to the access time. Fastband tracks are exactly the same as regular data tracks except for reduced access time. The Fastband storage capacity of a FASTRAND drum unit is 50,688 words. Total Fastband capacity of a subsystem is 405,504 words.

- Write Lockout

The write lockout option protects selected drum tracks from erasure by overwriting, thereby preventing the accidental loss of permanent data. A FASTRAND unit, so equipped, contains a key-controlled multiposition switch which, if not in the OFF position, prevents writing on 1, 2, 4, 8, 16, 32, or 192 contiguous tracks under each head, beginning with track 000. This feature, however, does not interfere with read operations.

■ Characteristics

Characteristics of the FASTRAND II Subsystem are summarized in the following table. Detailed information is given in *UNIVAC 491/492/494 FASTRAND Mass Storage Subsystem Programmer/Operator Reference Manual, UP-7528* (current version).

PARAMETERS	SPECIFICATIONS
Storage Capacity (per FASTRAND II Mass Storage Unit)	25,952,256 words (129,761,280 characters)
Recording Mode	Bit-serial
Latency Time Average Maximum	 35 milliseconds 70 milliseconds
Head Switching Time	20 microseconds
Head Boom Positioning Time Minimum (one position move) Average Maximum	 30 milliseconds 57 milliseconds 86 milliseconds
Access Time* Average Maximum	 92 milliseconds 156 milliseconds
Effective Transfer Rate for Multisector Operation	31,000 words (155,000 characters) per second
Maximum Transfer Rate (within a Sector)	37,000 words (185,000 characters) per second
I/O Channels Required	1 channel for nonsimultaneous operation 2 channels for simultaneous operation
Drum Speed	880 RPM
Number of Data Read/Write Heads (per FASTRAND II Mass Storage Unit)	64 heads
Number of FASTRAND II Mass Storage Units Per Subsystem	1 to 8
Number of FASTRAND Control Units per Subsystem	1 for nonsimultaneous operation 2 or 2-channel unit for simultaneous operation
Number of Fastband Heads per Mass Storage Unit	24
Number of Fastband Tracks per Mass Storage Unit	24
Fastband Storage Capacity per Mass Storage Unit	50,688 words (253,440 characters)
Mean Access Time	35 milliseconds
Maximum Access Time	70 milliseconds

* Access Time: Latency Time Plus Head Boom Positioning Time.

3.5.1.5. FASTRAND III Subsystem

The FASTRAND III Subsystem is a large capacity, sector-addressable, random access storage medium consisting of one Type 5009 FASTRAND Control Unit and one to eight Type 6010-10 FASTRAND III Mass Storage Units. Each mass storage unit can store up to 38,928,384 words of 30 data bits plus parity characters. The average access time for any word in the subsystem is 92 milliseconds.

The FASTRAND III Subsystem has been developed to provide greater online storage capacity than the FASTRAND II Subsystem (see 3.5.1.4). The FASTRAND III Mass Storage Unit has the same general and physical characteristics, and provides the same functions, as the FASTRAND II unit; moreover, these functions have been generally enhanced and the data storage capacity increased by 50 percent. The FASTRAND III unit may be used to replace the FASTRAND II unit in existing configurations for promoting system capabilities without programming change; FASTRAND III and FASTRAND II units may not, however, be intermixed in the same subsystem.

The higher capacity and corresponding increase in data transfer rate for the FASTRAND III is developed through incorporation of a greater number of sectors on the same number of tracks, 96 sectors as opposed to 64 sectors, and a higher recording density, 1600 bits per inch rather than 1000 bits per inch, than for the FASTRAND II. Drum speeds and average access times for both units remain the same. However, because of the increase in data available with each drum revolution, the size of the index tables is reduced accordingly.

The basic configuration of the FASTRAND III Subsystem can use a single I/O channel. The storage capacity of the subsystem can be expanded by the optional addition of one to seven mass storage units on the channel, providing a maximum storage capacity of 311,427,072 30-bit words.

■ Control Unit

The FASTRAND control unit for the FASTRAND III Subsystem has the same functions as for the FASTRAND II Subsystem, with the dual channel feature also available.

■ Functions

The function repertoire of the FASTRAND III Subsystem is the same as that of the FASTRAND II Subsystem.

■ Optional Capabilities

Optional features which may be obtained with the FASTRAND III Subsystem are Fastbands and write lockout.

– Fastbands

The increased sector density of the FASTRAND III results in a 50 percent increase in the amount of data stored in the 24 tracks of the Fastband area. However, the access time of 35 milliseconds remains the same as for FASTRAND II.

– Write Lockout

The number of tracks which can be locked out for FASTRAND III are the same as for FASTRAND II. However, as more data is contained on each track, one and one-half times as much data is under lockout protection on FASTRAND III.

■ Characteristics

Characteristics of the FASTRAND III Subsystem are summarized in the following table.

PARAMETERS	SPECIFICATIONS
Storage Capacity (per FASTRAND III Mass Storage Unit)	38,928,384 words (194,641,920 characters)
Recording Mode	Bit-serial
Latency Time Average Maximum	 35 milliseconds 70 milliseconds
Head Switching Time	20 microseconds
Head Boom Positioning Time Minimum (one position move) Average Maximum	 30 milliseconds 57 milliseconds 86 milliseconds
Access Time* Average Maximum	 92 milliseconds 156 milliseconds
Effective Transfer Rate for Multisector Operation	45,257 words (226,283 characters) per second
Maximum Transfer Rate (within a Sector)	55,560 words (277,800 characters) per second
I/O Channels Required	1 channel for nonsimultaneous operation 2 channels for simultaneous operation
Drum Speed	880 RPM
Number of Data Read/Write Heads (per FASTRAND III Mass Storage Unit)	64 heads
Number of FASTRAND III Mass Storage Units per Subsystem	1 to 8
Number of FASTRAND Control Units per Subsystem	1 for nonsimultaneous operation 2 or 2-channel unit for simultaneous operation
Number of Fastband Heads per Mass Storage Unit	24
Number of Fastband Tracks per Mass Storage Unit	24
Fastband Storage Capacity per Mass Storage Unit	76,032 words (380,160 characters)
Mean Access Time	35 milliseconds
Maximum Access Time	70 milliseconds

* Access Time: Latency Time Plus Head Boom Positioning Time.

3.5.2. Magnetic Tape Subsystems

Magnetic tape subsystems provide high speed auxiliary storage media for program elements and libraries, subroutines, and data which for various reasons may not be kept in random access storage and which may be necessarily brought as input/output, sort/merge, and file maintenance operations. Magnetic tape subsystems consist of one or two Type 5008 Control Units, standard, or standard and auxiliary, (see Figure 3–13) and 1 to 16 Type 0859 UNISERVO VIII C or Type 0858 UNISERVO VI C Magnetic Tape Units, connected to either a compatible or a normal I/O channel of the CPU. The UNISERVO VIII C and VI C units have the same functions, the difference being in appearance and timing, that is, tape speed, start/stop/time, and other functions, and may be used in various combinations according to system configuration design, and control unit options, in simultaneous or nonsimultaneous mode operation.

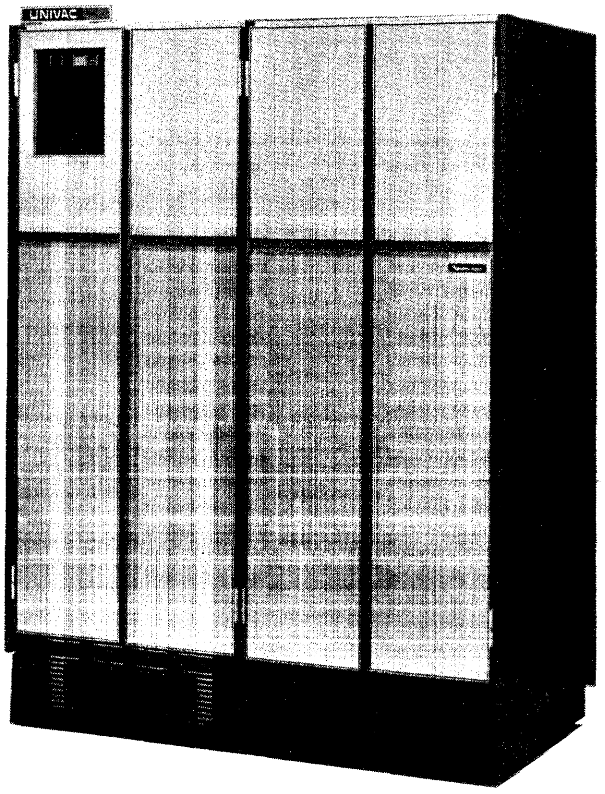


Figure 3–13. Standard and Auxiliary Control Units for Simultaneous Operation Magnetic Tape Subsystem

■ Control Unit

The control unit interfaces the magnetic tape units with the CPU and has the following principal functions:

– Receives function words (control information) from the CPU and translates this information into control signals for the magnetic tape units.

– Requests and acknowledges data transfers, initiates tape movement on the selected unit, and handles the data transfers and checking required.

- Assembles and disassembles data and control words for the CPU and magnetic tape units.
- Sends a status code to the CPU indicating the completion or result of subsystem activity and the condition of the subsystem.

In the nonsimultaneous mode of operation, the standard control unit is used, controlling all of the magnetic tape units in the subsystem through one I/O channel. In the simultaneous mode of operation, the standard control unit is connected to one I/O channel and the auxiliary control unit is connected to another or two standard control units may be used, each connected to its own I/O channel. In either case, both control units are connected to all magnetic tape units in the subsystem, permitting all subsystem units to be accessed from two I/O channels.

Simultaneous operation permits write functions on one magnetic tape unit in the system through an I/O channel at the same time as read functions on another magnetic tape unit in the same subsystem through another I/O channel.

During the execution of a write function, the control unit requests 30-bit data words from the CPU, disassembles the words received into 6-bit groups, adds the appropriate parity bit to each group, and sends the resultant 7-bit groups to be written on the selected magnetic tape unit. The control unit also checks each write operation through readback from the tape unit read head for proper lateral parity, and checks for provision of even parity for each track by the longitudinal check frame.

During the execution of a read function, the control unit checks each 7-bit frame read from tape for proper parity, strips off the parity bit, assembles the resultant 6-bit groups into 30-bit words, and makes each word available to the CPU. The control unit also checks for provision of even parity for each track by the longitudinal check frame.

■ Functions

The magnetic tape subsystems are capable of performing the following functions:

- Write
- Skip/Write
- Read Backward
- Read Forward
- Bootstrap
- Rewind To Load Point
- Rewind With Interlock

■ Optional Features

Optional features which may be included in the Magnetic Tape Subsystem are the translate, 9-track format, and the UNISERVO VI C feature.

– Translate

The optional translate feature provides the translation hardware which relieves the program of the time-consuming task of translating from a six-bit processor code to a tape code for equipment which utilizes different six-bit codes to represent the various characters. The translate feature also provides hardware translation from tape code to internal code. The option is available for both nonsimultaneous and simultaneous operations.

– 9-Track Format

The optional 9-track format feature provides ability for reading tapes prepared on IBM 2400 Series magnetic tape configuration at 800 bytes per inch and for writing tapes which can be read on such equipment. The 9-track feature affords increased data transfer rates and permits a configuration which includes both 9-track tape units and 7-track tape units. The 9-track format is available for both nonsimultaneous and simultaneous operations.

– UNISERVO VI C Feature

The optional UNISERVO VI C feature is a hardware addition to the Magnetic Tape Subsystem control unit which permits UNISERVO VI C Magnetic Tape Units to be included in the same subsystem as the UNISERVO VIII C.

3.5.2.1. UNISERVO VIII C Magnetic Tape Subsystem

The UNISERVO VIII C Magnetic Tape Unit (see Figure 3–14) operates at 120 inches per second for read and write operations and at 240 inches per second for rewind operations.

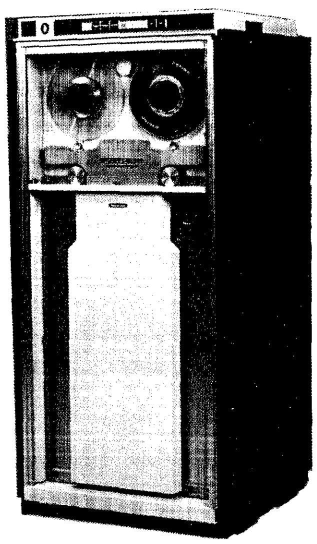


Figure 3–14. UNISERVO VIII C Magnetic Tape Units

Each UNISERVO VIII C Magnetic Tape Unit includes an operator's Control/ Indicator Panel, a tape supply reel hub, a permanently mounted tape take-up reel, a photoelectric reflective marker detection system, a 7- or 9-track read/ write head, an erase head, and a write enable/disable facility.

- Characteristics

Characteristics of the UNISERVO VIII C Magnetic Tape Unit are given in the following table. Detailed information may be found in *UNIVAC 491/492/494 Real-Time System UNISERVO VIII C Magnetic Tape Subsystem Programmer/ Operator Reference Manual, UP-7523* (current version).

PARAMETERS	SPECIFICATIONS
Number of Units per Subsystem	1 – 16
Tape Handling Speed	120 inches per second for data transfer operations
Recording Densities	200, 556, or 800 frames per inch
Transfer Rates	
200 frames per inch	24,000 frames per second
556 frames per inch	66,666 frames per second
800 frames per inch	96,000 frames per second
Rewind Speed	240 inches per second
Rewind Time (2400 foot reel of tape)	2.0 minutes
Tape Reversal Delay	12.5 milliseconds
Interblock Gap Size	.75 inches (7 channel) .60 inches (9 channel)
Tolerance	+.16 inch –.06 inch (7 channel) +.15 inch –.1 inch (9 channel)
Start Time	2.5 milliseconds
Start Distance	.10 – .19 inch
Stop Time	2.5 milliseconds
Stop Distance	.11 – .16 inch
Tape Width	0.5 inches
Tape Thickness	1.5 mils
Tape Length	2400 feet
Block Length	variable
Channels on Tape	7 channels: 6 data, 1 parity 9 channels: 8 data, 1 parity
Read/Write Operation	Read in forward or backward direction; Write in forward direction
Compatibility	Ability to read or write tapes in Binary Coded Decimal (BCD) format
Write Enable Ring	Manually inserted ring in the tape supply reel enables the write operation

3.5.2.2. UNISERVO VI C Magnetic Tape Subsystem

The UNISERVO VI C Magnetic Tape Unit (see Figure 3–15) operates at 42.7 inches per second for read and write operations and at 160 inches per second for rewind operations. The UNISERVO VI C has two basic forms: a master unit and a slave unit, which are identical in external appearance. A master unit contains a power supply unit and electronic circuitry which are also shared by a maximum of three slave units, with the group being referred to as a quad.

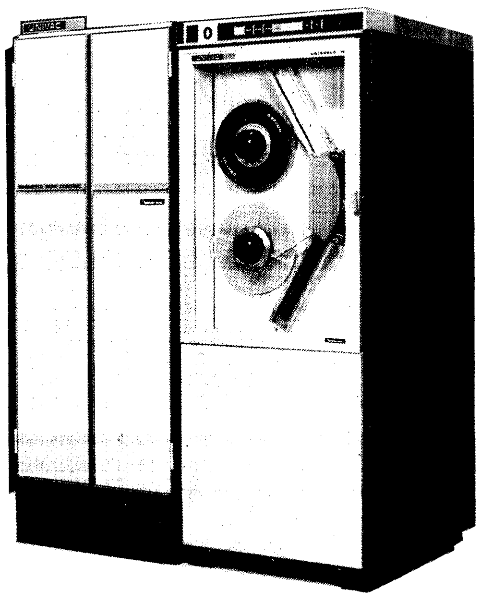


Figure 3–15. UNISERVO VI C Magnetic Tape Units

The master unit used in a simultaneous operation subsystem provides for concurrent communication between the two control units in the subsystem and any two tape units in the quad. The slave units used in a nonsimultaneous operation subsystem are identical to the slave units used in a simultaneous operation subsystem.

Each UNISERVO VI C Magnetic Tape Unit includes an operator's Control Panel, a tape supply reel hub, a permanently mounted tape take-up reel, a photoelectric reflective marker detection system, a 7- or 9-track read/write head, an erase head, and a write enable/disable facility.

■ Characteristics

Characteristics of the UNISERVO VI C Magnetic Tape Unit are given in the following table. Detailed information may be found in *UNIVAC 491/492/494 Real-Time System UNISERVO VI C Magnetic Tape Subsystems Programmer/ Operator Reference Manual, UP-4101* (current version).

PARAMETERS	SPECIFICATIONS
Number of Units per Subsystem	1 – 16
Tape Handling Speed	42.7 inches per second
Recording Densities	200, 556, or 800 frames per inch
Transfer Rates 200 frames per inch 556 frames per inch 800 frames per inch	 8,500 frames per second 23,720 frames per second 34,160 frames per second
Rewind Speed	160 inches per second
Rewind Time (2400 foot reel of tape)	3.0 minutes
Tape Reversal Delay	25.0 milliseconds
Interblock Gap Size Tolerance	0.75 inches (7 channel) 0.60 inches (9 channel) +.16 inches – .06 inches (7 channel) +.15 inches – .1 inch (9 channel)
Start Time Start Distance	7.0 milliseconds .10 – .15 inches
Stop Time Stop Distance	5.0 milliseconds .09 – .16 inches
Tape Width	0.5 inches
Tape Thickness	1.5 mils
Tape Length	2400 feet
Block Length	variable
Channels on Tape	7 channels: 6 data, 1 parity 9 channels: 6 data, 1 parity
Read/Write Operation	Read in forward or backward direction; Write in forward direction
Compatibility	Ability to read or write tapes in Binary Coded Decimal (BCD) format
Write Enable Ring	Manually inserted ring in the tape supply reel enables the write operation

3.5.3. Unit Record Subsystems

Unit record subsystems provide facilities for preparation, input, output, display, and storage of information according to the design functions of the subsystem. Each subsystem is governed by a control unit which is connected to an I/O channel of the CPU.

The unit record subsystems available include the following:

- The High Speed Printer Subsystem, which furnishes capability for hard copy printouts of program listings and program results.

- The Punched Card Subsystem, which prepares and submits input data and provides punched card output of program listings and program results.

- The UNIVAC 1004 System, which is a satellite computer system that includes peripheral devices for input and output of data to the UNIVAC 494 CPU. The UNIVAC 1004 System uses its capabilities as a computer for performing certain ancillary functions such as editing, arithmetic, and logical operations on data during transfer.

- The UNIVAC 9300 System is a satellite computer system that includes peripheral devices for input and output of data to the UNIVAC 494 CPU. The UNIVAC 9300 System uses its capabilities as a computer for performing ancillary functions such as editing, arithmetic, and logical operations on data during transfer. The system provides faster, more flexible, and more powerful processing facilities than the UNIVAC 1004 System.

3.5.3.1. High Speed Printer Subsystem

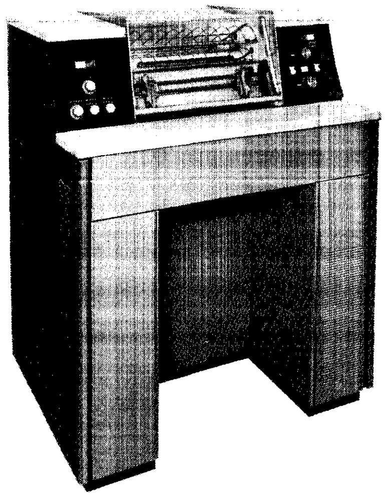


Figure 3–16. High Speed Printer

The High Speed Printer Subsystem is an output unit which is capable of printing single or multiple copies of data. Each line of output data may contain up to 132 printed characters.

The subsystem comprises a High Speed Printer (see Figure 3–16), Type 0755, or 0758, connected to a High Speed Printer Control Unit, Type 8120-02 or Type 5011 (for Type 0758 only). The Type 0755 High Speed Printer is capable of printing from 700 to 922 lines per minute; Type 0758 High Speed Printer is capable of printing from 1200 to 1600 lines per minute. The printing rate depends upon the character set to be printed. The full character set is printed at the lower speed. The maximum rate is attained by printing an alphanumeric subset of the complete character set.

The printer contains 63 printable characters – the 26 letters of the alphabet, the ten numerals, and 27 special characters. Different symbols may be factory supplied upon order.

■ High Speed Printer Control Unit

The high speed printer control unit may be connected to either a normal or compatible I/O channel of the CPU. Output information is routed by the CPU through the printer control unit to the printer. The printer control unit contains a 132-character core buffer for accumulating a full line of information before printing.

The control unit governs all the operations of the High Speed Printer Subsystem, and has the following principal functions:

- Receives function words from the CPU and translates them into control signals for the printer.
- Requests and acknowledges data transfers, and synchronizes the flow of data between the processor and the printer.
- Accumulates a line of print in its buffer memory from the data transmitted by the processor.
- Interprets signals, both normal and abnormal, from the printer and notifies the processor of printer and control unit conditions.

■ Characteristics

Characteristics of the High Speed Printer Subsystem are summarized in the following table. Additional information is contained in the *UNIVAC 491/492/494 Real-Time Systems High Speed Printer Subsystem Programmer/Operator Reference Manual, UP-7571* (current version).

PARAMETERS	SPECIFICATIONS
Printing Speed (with single-line spacing)	
Type 0755	700–920 lines per minute, depending upon character set printed.
Type 0758	1200–1600 lines per minute, depending upon character set printed.
Line Spacing Speed	
Type 0755	20 ms for spacing first line and for spacing each subsequent line as follows: 8 ms at 6 lines per inch; 6 ms at 8 lines per inch
Type 0758	11.5 ms for spacing first line and for spacing each subsequent line as follows: 5.1 ms at 6 lines per inch; 5.7 ms at 8 lines per inch
Characters Per Line	132 characters (including spaces) per line.
Spacing of Characters	0.1 inch along print line.
Ribbon Feed	Bidirectional, self-reversing, self-correcting.
Type of Ribbon	Fabric ribbon interchangeable with carbon Mylar* ribbon (optional) for "one-time" operation.
Vertical Line Spacing	Manually selected. Either 6 lines per inch or 8 lines per inch.
Number of Characters	Up to 63 different characters: standard font consists of alphabetic characters A–Z, numeric characters 0–9, 27 punctuation marks and symbols. Modified fonts available upon request.
Print Format	Full print width of 132 characters can be placed anywhere on 16.5 inch form. With 22 inch width form, only central 13.2 inch portion can be used. Format variation under full control of programming.
Paper Forms	Continuous forms with standard edge sprocket holes from 4 to 22 inches in width. Carbons may be attached or unattached with multicopy forms up to a maximum of six parts. Recommended pack thickness up to .0155 inch for optimum print quality.
Paper Container	Maximum dimensions accommodated entirely within base of machine: 16 inches long, and 22.5 inches deep.

**Trademark of DuPont Corporation*

3.5.3.2. Punched Card Subsystem

The Punched Card Subsystem is an input/output system consisting of a Type 0706 Card Reader, and/or a Type 0600 Card Punch, and a Type 5010-01 Card Control Unit. The control unit communicates with the CPU, and controls the operation of both the card reader and the card punch.

The card reader senses 80-column data at the rate of 900 cards per minute maximum, and transfers the data, column-by-column, to the card control unit.

The card punch receives data from the card control unit in the card image format, and punches row-by-row, and stacks cards at the maximum rate of 300 cards per minute.

The control unit affords the sensing and punching of data for 80-column cards in the following formats:

- Translate – Each successive card column represents a character, or six bits of a 30-bit word, providing for 16 5-character data words per card.
- Card Image by Column – Each successive two and one-half card columns represent a 30-bit data word, providing for 32 words per card.
- Card Image by Row – Each card row represents two successive 30-bit data words and a third word with data in the 20 most significant bit positions, providing for 36 words per card (the least significant 10 bits of 12 words will be zerofilled on subsequent reading and ignored on punching).

The components of the Punched Card Subsystem are described in the following paragraphs. More information may be obtained from *UNIVAC 491/492/494 Real-Time System Punched Card Programmer/Operator Reference Manual, UP-7522* (current version).

- Card Control Unit

  The card control unit decodes function words that are transmitted by the CPU as instructions for the card reader or the card punch. The control unit includes a buffer memory which collects the data characters read by the card reader, and translates and sends the information to the CPU. In the same way, data from the CPU is translated by the card control unit and sent to the card punch.

  The card control unit governs all the operations of the Punched Card Subsystem, and has the following principal functions:

  - Receives, from the CPU, function words that condition and prepare the subsystem for different modes of operation and for data handling.
  - Requests and acknowledges data transfers, and synchronizes the flow of data between the CPU and the card reader or card punch.
  - Accumulates the data to be punched, or the data being read from cards, in its buffer memory.
  - Interprets signals, both normal and abnormal, from the card reader or card punch and notifies the CPU of conditions within these units.

■ Card Reader

The card reader, shown in Figure 3–17, reads cards into the buffer memory of the Card Control Unit at a rate of up to 900 cards per minute, and stacks the cards in the same order as originally fed. The card reader is equipped with a control panel which is divided into two areas: the operator's control panel which initiates and monitors operation, and the diagnostic panel which indicates malfunctions.

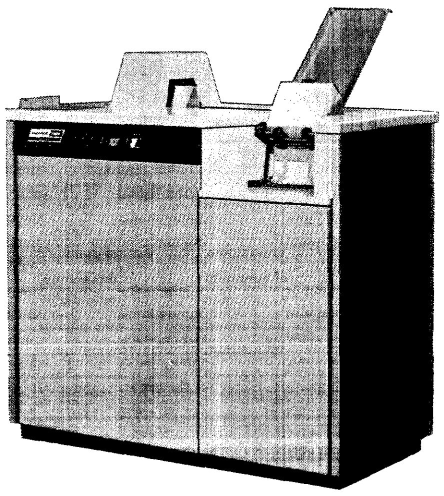


Figure 3–17. Type 0706 Card Reader

■ Card Punch

The card punch shown in Figure 3–18, feeds, punches, post-punch reads, and stacks 80-column cards at a maximum rate of 300 cards per minute on command from the CPU. The card punch is equipped with a control panel which is divided into two areas: the operator's control panel which initiates and monitors operation, and the diagnostic panel which indicates malfunctions.

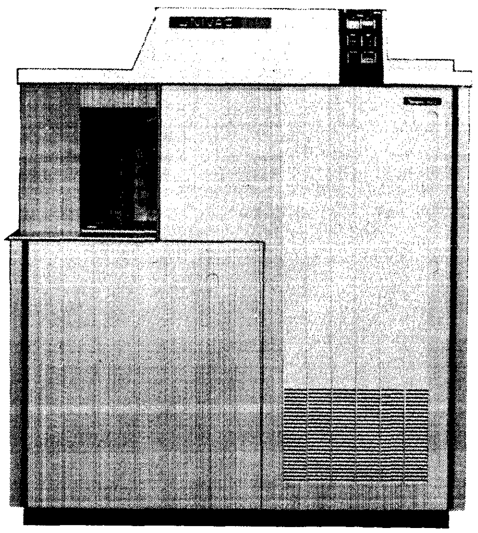


Figure 3–18. Type 0600 Card Punch

■ Characteristics

Punch Card Subsystem characteristics are summarized in the following table. Detailed information may be found in *UNIVAC 491/492/494 Real-Time System Punched Card Programmer/Operator Reference Manual, UP-7522* (current version).

PARAMETERS	SPECIFICATIONS
TYPE 0706 CARD READER	
Card Orientation	Fed face down, 9-edge leading
Card Rate	900 cards per minute (maximum)
Card Cycle	66.6 milliseconds per card cycle at 900 cpm
Input Hopper Capacity	300 cards
Read Station Sensing	Column-by-column
Output Stacker Capacity	
Stacker – Normal	2100 cards
Stacker – Error	100 cards
TYPE 0600 CARD PUNCH	
Card Orientation	Fed face down, 9-edge leading
Card Rate	300 cards per minute (maximum)
Card Cycle	200 milliseconds per card cycle at 300 cpm
Input Hopper Capacity	1000 cards
Punch Station Punching	Row-by-row
Read Check Station Sensing	Row-by-row
Output Stacker Capacity	
Output Stacker – Normal	850 cards
Output Stacker – Select	850 cards

3.5.3.3. UNIVAC 1004 System

The UNIVAC 1004 System is a complete data processing system, with its own peripheral units, which may be used online with the UNIVAC 494 System as a peripheral subsystem to provide input/output facilities and to perform supplementary processing of the transfer data.

The UNIVAC 1004 System shown in Figure 3–19, is a powerful processing unit in its own right, with arithmetic, logical, and editing capabilities, allied to a modular 961-character core storage. Standard peripheral units are a 615 cpm card reader, and a high speed printer operating at 600 lpm with a 63-character set and 132-character print line width.

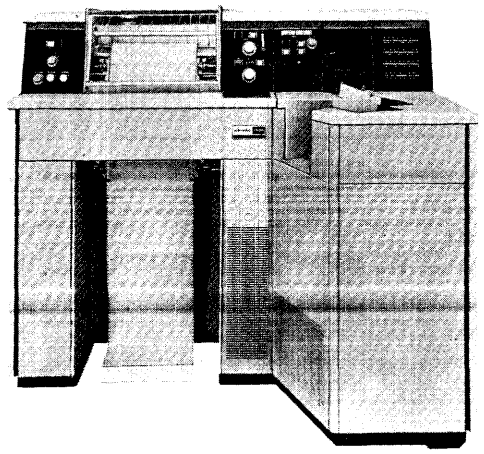


Figure 3–19. UNIVAC 1004 System

The subsystem may be used online to the UNIVAC 494 CPU by the inclusion of the Universal Interface Adapter. The adapter must include the 30-bit word transfer option and the External Interrupt feature, for utilizing the full capability of the peripheral subsystem.

Optional features supported by software are: a card punch (200 cards per minute), a paper tape reader (400 characters per second), and a paper tape punch (110 characters per second).

A special plugboard, called the OMEGA plugboard, is required when the subsystem is used online. The UNIVAC 1004 Subsystem retains its freestanding processing power when used in this configuration. At any time, the 1004 Subsystem can be switched to offline mode, and it then operates as an independent computer.

■ Characteristics

UNIVAC 1004 Subsystem characteristics are summarized in the following table. Detailed information may be obtained in *UNIVAC 1004 Card Processor, 80 Column, UT-2543* (current version) and *UNIVAC 494 Real-Time System Online 1004 Program Programmer/Operator Reference Manual, UP-7575* (current version).

PARAMETERS	SPECIFICATIONS
REQUIRED FEATURES	
External Interrupt	
Punch Stacker Select (with Card Punch)	
30-bit Word Transfer Interface	
CARD OPERATIONS	
Card Reading Speed	615 cpm
Card Punching Speed	200 cpm
Card Reading Modes	80-column Hollerith and column binary
Card Punching Modes	80-column Hollerith and column binary
PRINT OPERATIONS	
Printing Speed	600 lpm
Maximum Number of Characters per Line	132
Number of Printable Characters	63 (26 alphabetic, 10 numeric, and 27 special characters) plus space
Lines per Inch (Vertical)	6 or 8 (manually selected)
PAPER TAPE OPERATIONS	
Reading Rate	400 characters per second
Punching Rate	110 characters per second
Number of Channels	5, 6, 7, or 8
Characters per Inch	10
Tape Speed	40 inches per second reading 11 inches per second punching
Tape Width	11/16, 7/8, or 1 inch

3.5.3.4. UNIVAC 9300 System

The UNIVAC 9300 System is a complete, internally programmed, data processing system with its own peripheral subsystems, which may be used online with the UNIVAC 494 System as a peripheral subsystem to provide input/output facilities and to perform supplementary processing of the transfer data.

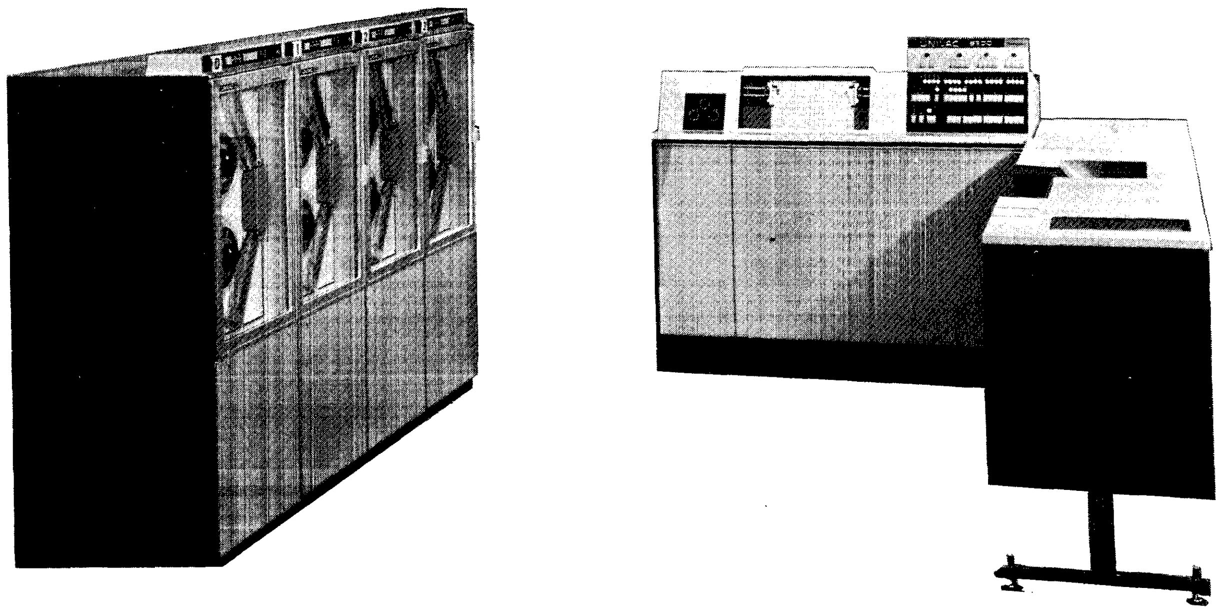


Figure 3–20. UNIVAC 9300 System

The UNIVAC 9300 System (see Figure 3–20) is a powerful and flexible high speed computer system which provides card processing, magnetic tape handling, and printing facilities as well as arithmetic, editing, I/O control, and data manipulation functions. The UNIVAC 9300 CPU contains the necessary controls, instructions, and synchronizers for directing its functions and the functions of up to seven peripheral units on its Multiplexer I/O Channel. Data transfers with the UNIVAC 494 CPU are also made on the Multiplexer I/O Channel through the Inter-Computer Control Unit (ICCU), which permits the UNIVAC 9000 Series computers to communicate on-site in 30-bit word format. Primary storage or main memory for the UNIVAC 9300 System is plated wire, with 600 nanosecond cycle time and storage capacity of 8,192 bytes of eight bits each, expandable to 32,768 bytes. Being a member of the UNIVAC 9000 Series, the UNIVAC 9300 System is compatible with both the UNIVAC 9200 and UNIVAC 9400 Systems.

The card reader processes 80-column cards, with the Multi-Strobe Read feature providing multiple sensing of card columns; and the printer outputs 132 character lines.

■ Characteristics

Characteristics of the UNIVAC 9300 System are given in the following list. Detailed information may be found in *UNIVAC 9300 System, System Description UP-4119* (current version).

PARAMETERS	SPECIFICATIONS
System Orientation	Card/Tape
Basic Memory Maximum Memory	8192 bytes 32,768 bytes
Memory Cycle Time	600 nanoseconds
Add (Decimal) Instruction Time (Two 5-digit Fields)	52 microseconds
Multiply, Divide, and Edit	Standard
Card Read-Basic Reader 1001 Card Controller	600 cpm 1000/2000 cpm
Card Punch-Column Row Selective Stacker	75–200 cpm (depends on last column punched) 200 cpm Standard on Row Punch Optional on Column Punch
Read/Punch (for either punch)	Optional
Print Speed (Alphanumeric)	600 lpm
Numeric Printing (Optional)	1200 lpm
Overlapped Input/Output Units	Standard
Magnetic Tape Rate	34,160 bytes per second
Simultaneous Tape Read, Write and Process	Optional
Multiplexer Channel (Standard for Tape Systems)	85,000 bytes per second transfer rate

3.5.4. Communication Terminal Modular Control Subsystem

The Communication Terminal Modular Control (CTMC) Subsystem provides communication capability to the CPU and provides for transfer of data between the CPU and remote site devices. The subsystem serves as the interface between the CPU and any device which meets the accepted standard for serial data transmission. Each subsystem permits time shared data transfers between the CPU and up to 32 diverse, remote terminals. The CTMC Subsystem is linked to the CPU by a single I/O channel, and is housed in two central site cabinets each of which contains: a Communication Terminal Module (CTM) Controller, which governs the interface between the CPU and the subsystem components; up to 16 Communication Terminal Modules (CTM's), which effect data transfers between the CPU and communication lines through the CTMC; and Interface Modules (IM's), which make the necessary conversion lines between the CTM's and communication lines and equipment.

The elements of the CTMC Subsystem are described in the following paragraphs. Detailed information may be found in *UNIVAC 418, 490/491/492, and 494 and 1108 Systems Communication Terminal Modular Control (CTMC) Subsystem Programmer/Operator Reference Manual, UP-7519* (current version).

3.5.4.1. Communication Terminal Module Controller

The Communication Terminal Module (CTM) Controller responds to service requests from the individual CTM's and, with the associated CPU, exchanges the signals required to effect data transfer. Internal cabling connects the CTM Controller to a maximum of 16 CTM's. External circuitry links the CTMC to the 494 CPU. One active CTM Controller and one unconnected spare controller may be installed in the same subsystem cabinet.

3.5.4.2. Communication Terminal Module (CTM)

The Communication Terminal Modules (CTM's) provide the logical interface between the communication lines and the 494 CPU through the CTM Controller. Four basic types of CTM's are available: low speed asynchronous, medium speed asynchronous, high speed synchronous, and dial.

The CTM's arrange data in the format required by the CPU or in the format demanded by the circuit with which the terminal is designed to operate. Incoming data is received from the associated IM's and arranged in bit-parallel character serial form for submission to the CPU. Output traffic is formatted according to the requirements of the associated communication circuit.

Each CTM has a service priority determined by its physical connection to the CTM Controller and presents a service request to the CTM Controller when the CTM has prepared a complete incoming character for transfer to the CPU, or when the final bit of a transmitted character has been submitted to the IM. Many CTM's could logically generate service requests simultaneously. The CTM Controller samples all terminals requesting service and grants priority to the highest numbered position, whether the CTM is an input or output terminal. After an acknowledge signal from the CPU frees the CTM Controller from one terminal, all remaining service requests will be evaluated and the CTM Controller will lock on the next terminal within 13 microseconds.

The CTM's are grouped into modules of two input terminals and two output terminals (with the exception of the dial CTM, which has no input terminals and has up to six output terminals per module). The terminals may be used in either simplex, half duplex, or full duplex mode.

- Characteristics

Characteristics of the CTM's are as follows:

– Low Speed CTM

Transmission Mode: Bit-serial

Transmission Method: Asynchronous

Input Rate: Clock in each input CTM adjustable from 20 – 300 bits per second

Output Rate: Output timing provided by clock in the CTM Controller

Character Compatibility: 5, 6, 7, or 8 data bits (field option)

IM: Either relay or data set

– Medium Speed CTM

Transmission Mode: Bit-serial

Transmission Method: Asynchronous

Input Rate: Clock in each input CTM adjustable from 300 – 1600 bits per second

Output Rate: Output timing provided by clock in the CTM Controller

Character Compatibility: 4, 5, 6, 7, or 8 data bits (field option)

IM: Data set

– High Speed CTM

Transmission Mode: Bit-serial

Transmission Method: Synchronous

Input and Output Rate: Established by clocks in user-supplied data sets (up to 50,000 bits per second)

Character Compatibility: 5, 6, 7, or 8 data bits (field option)

IM: Data set

– Dial CTM

Transmission Mode: Bit-parallel

Transmission Method: Asynchronous

Input Rate: Dial CTM is an output device only

Output Rate: Determined by Bell System Automatic Calling Unit (ACU) made available by user.

Character Compatibility: Four-bit (BCD) dial digits

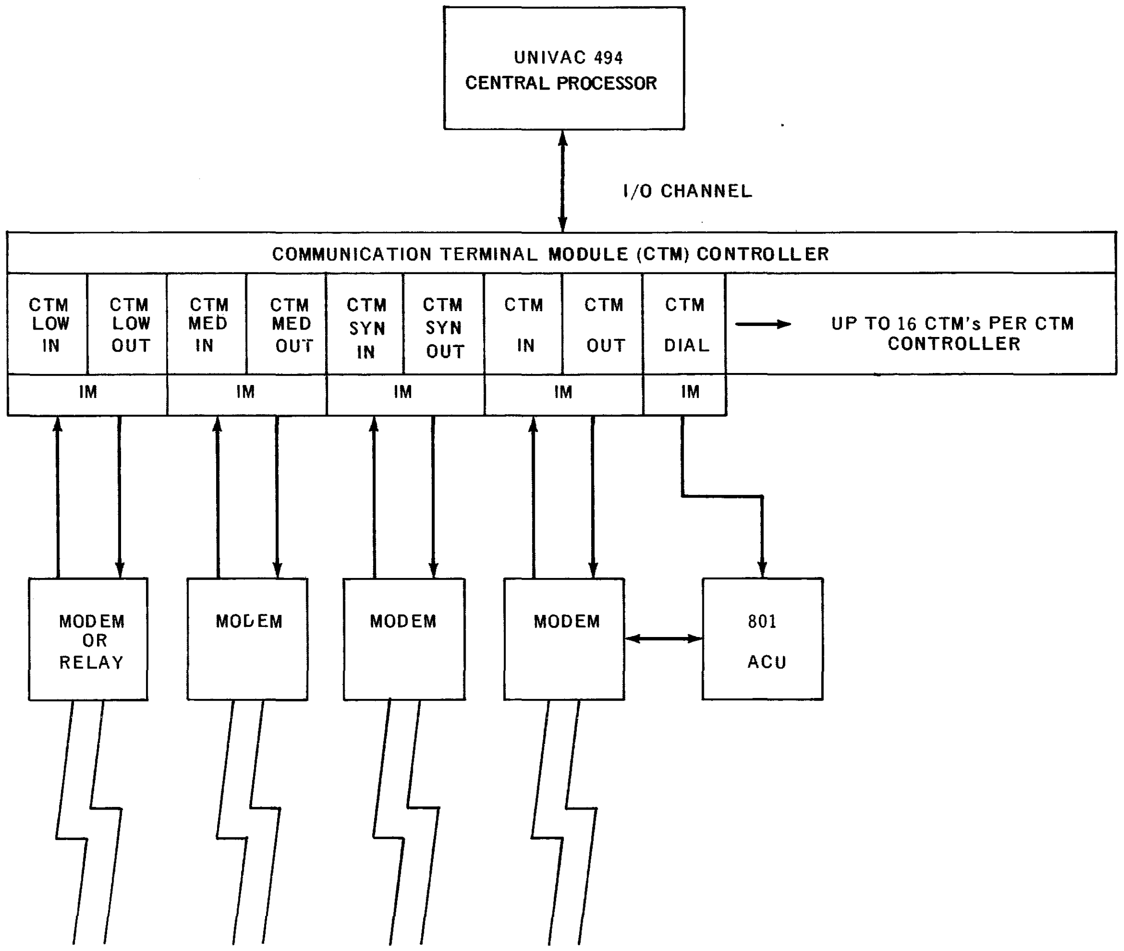
IM: Dial

3.5.4.3. Interface Modules (IM)

The Interface Modules (IM's) make the necessary conversion between the electrical operating levels of the CTM's and the levels of the particular external circuits with which the IM's are designed to operate. The terminations offered, electrical potentials supplied, signals presented, and responses anticipated by each IM, conform to the Electronic Industries Association (EIA) standard for that type of data communication. Different IM's are required for any one of the following applications: printing telegraphs employing DC circuits, transmission rates employing data sets, and specialized applications such as dialing. Data set IM's may be equipped with the Unattended Answering feature for responding to remotely originated dial connections.

3.5.4.4. Communication Subsystem Configuration

The arrangement of the various elements within the communication subsystem is shown in Figure 3–21.



CTM – COMMUNICATION TERMINAL MODULE
IM – INTERFACE MODULE
ACU – BELL SYSTEM AUTOMATIC CALL UNIT

Figure 3–21. Communication Subsystem Configuration

3.6. REMOTE PERIPHERAL SUBSYSTEMS

Remote peripheral subsystems are made up of devices which provide similar functions as on-site peripheral units, but which usually transfer data some distance from the CPU or transfer data through interfaces different from the CPU, and which are connected to the CPU through the communication subsystem (see 3.5.4). Although some of these devices may be physically present at the computer site, they are not considered on-site subsystems as defined in 3.5.

The remote peripheral devices are connected to the Communication Terminal Modular Control (CTMC) Subsystem through common carrier facilities. The type of facility is dependent upon the particular remote device. In addition to individual control units, each remote device is interfaced to a communication subsystem which connects to the carrier facility connecting the CTMC Subsystems.

The remote devices included in this section are standard Univac products and adhere to the accepted interface standard for serial data transmission, EIA specification RS-232. The devices include the UNIVAC Data Communication Terminal (DCT) 2000, UNIVAC UNISCOPE 300, UNIVAC 9200/9300 Systems and UNIVAC 1004 System.

3.6.1. DCT 2000 Data Communication Terminal

The UNIVAC Data Communication Terminal (DCT) 2000, shown in Figure 3–22, is a combination printer and card reader/punch data transmission device designed to transfer large quantities of data efficiently through voice-grade private line or switched telephone network facilities. When tied into a network with computers or other DCT 2000 Systems, the DCT 2000 can handle up to 250 blocks per minute.

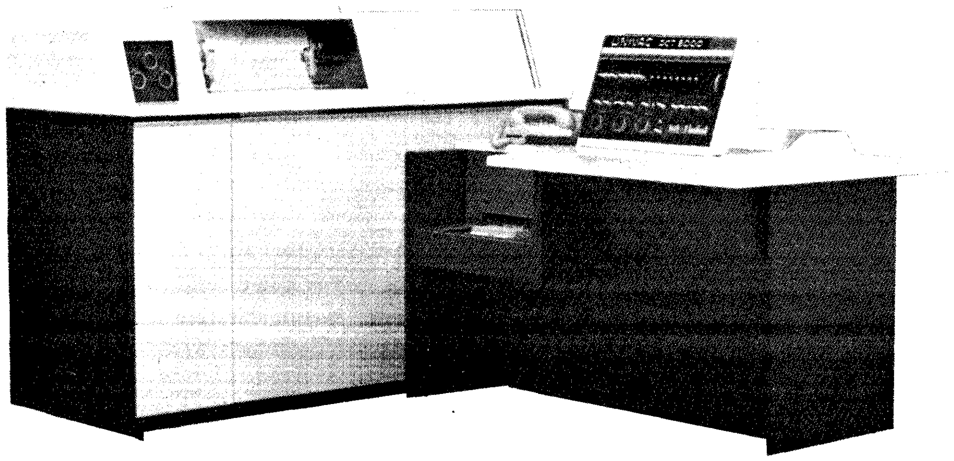


Figure 3–22. DCT 2000 Terminal

The basic components of the DCT 2000, including the control unit, the card reader/punch, the bar printer, and the communication interface are discussed in the following paragraphs. Detailed information may be found in *UNIVAC DCT 2000 General Description Reference Manual, UP-7511* (current version).

3.6.1.1. Control Unit

The basic control unit is designed to handle the United States of America Standard Code for Information Interchange (USASCII). An optional control unit provides compatibility with a UNIVAC 1004 System equipped with a Data Line Terminal (DLT 1 or DLT 3).

The control unit coordinates the transmit/receive operations within the UNIVAC DCT 2000. In general, the control unit comprises buffer storage, control logic, and data paths. The internal hardware is designed to perform a variety of functions such as buffer-memory addressing, control-character decoding, peripheral/control unit translation (USASCII/Hollerith), and error control.

Since the control unit is fully buffered, the terminal lends itself readily to retransmission techniques. These techniques are employed with the Error Detection and Retransmission feature which also provides capability for generating character and block parity, detecting errors in messages, preventing the loss of a data block, and providing protection against receiving duplicate blocks of data.

A variety of optional control unit features available with the DCT 2000 Subsystem are considered in detail in the following paragraphs. Separate consideration should be given, however, to the two versions of the control unit, USASCII compatible and Data Line Terminal (DLT) compatible.

- USASCII Control Unit

  The USASCII Control Unit coordinates and manipulates data in USASCII code, which is an eight-bit code that uses seven bits for alphabetics, numerics, special symbols, and transmission control characters. The eighth bit is used for odd parity. Within the control unit, the basic unit of seven information bits plus a parity bit is called a character. Groups of characters are called a block or a message. The DCT 2000 is designed to provide the operator with the option of specifying message lengths of 80 and 128 characters by setting a lever switch on the operator's console.

- Data Line Terminal (DLT) Compatible Control Unit

  The DLT Compatible Control Unit is required for communication between the DCT 2000 and UNIVAC 1004. Functionally, this control unit is the same as the USASCII version, with the following major difference: all transmission control, message control, and data characters appear in XS-3 code (the six-bit, internal machine code of the UNIVAC 1004). Again, for compatibility with the 1004, the DCT 2000 transmits fixed block lengths only, but can receive fixed- or variable-length messages. Also, Error Detection and Retransmission, as well as Short Block Capability, are included as standard equipment at no extra cost.

■ Control Unit Characteristics

Control unit technical characteristics and special features are summarized in the following list.

PARAMETERS	SPECIFICATIONS
Transmission Method	Block by block
Transmission Mode	Half duplex; 2 or 4 wire (non-simultaneous; two-way transmission)
Transmission Facilities	Voice Grade Telephone Toll Exchange or Private Line
Transmission Rate	2.4 KC (Private Line); 2 KC (Switched Telephone Network)
Transmission Code	USASCII XS-3 (DLT Compatible)
Buffer Storage	256 Character Capacity Two 128-Character Core Memory Buffers
Translation Capabilities	Card Code/Transmission Code Hollerith/USASCII Hollerith/XS-3 (DLT Compatible)

SPECIAL FEATURES

Error Detection and Retransmission*

Telephone Alert

Select Character Capability

Short Block Capability*

Peripheral Input/Output Channel

Unattended Operation

**With the DLT Compatible Control Unit, Error Detection and Retransmission, and Short Block Capability are included as standard equipment.*

3.6.1.2. Reader/Punch

The DCT 2000 Reader/Punch Unit feeds, reads or punches, and stacks 80-column cards. Read and punch operations cannot be performed on the same card during the same cycle. The punching rate of the unit is 75 cards per minute, and the reading rate is 200 cards per minute.

Cards are fed through the reader/punch from an input hopper to a wait station. When the preceding card has been completely read or punched, the next card is moved from the wait station to the read station. If the card is to be read, the card is sensed photoelectrically at the read station and passed unaltered to the output stacker. If card punching is specified, the card passes through the read station (read circuits inactive) to the punch station where it is punched two columns at at a time and ejected to the output stacker.

- Characteristics

  Characteristics and special features of the DCT 2000 Reader/Punch Unit are summarized below. The reading and punching speeds given reflect the capabilities of the unit; speeds attained in actual system use may be somewhat lower due to the transmission facilities available.

PARAMETERS	SPECIFICATIONS
Cards	Standard 80-column cards
Reading Speed	Maximum rate of 200 cards per minute.
Reading Method	Photoelectric Read Station
Punching Speed	Maximum Rate of 75 cards per minute for 80-column punching.
Punching Method	Two columns at a time.
Input Hopper Capacity	1200 cards
Output Stacker Capacity	700 cards
SPECIAL FEATURES	
Punch Check Error and Reject Stacker	

3.6.1.3. Printer

The DCT 2000 Bar Printer provides quality performance and economy while producing highly legible hard copy at the maximum rate of 250 lines per minute. Through engineering innovation, particularly in the printing method employed, a variety of cost avoidance features, such as the simplicity of operation and ease of maintenance, can be offered to the UNIVAC DCT 2000 user.

- Characteristics

Characteristics and special features of the DCT 2000 Bar Printer are summarized below. The printing speed given reflects the capability of the unit; speed attained in actual system use may be somewhat lower due to the transmission facilities available.

PARAMETERS	SPECIFICATIONS
Printing Speed	Maximum rate of 250 lines of alphanumeric characters per minute.
Printing Method	Removable type bar
Printing Positions	80
Printable Characters	63 different characters on an alphanumeric type bar: 26 alphabetic capitals 10 numeric characters 27 punctuation marks and symbols
Paper Speed	25 lines per second (Form Skip Speed)
Paper Spacing	6 lines per inch
Printing Flexibility	Maximum continuous forms of 22 inches in width and 14 inches in length provide an original and up to 5 copies.
SPECIAL FEATURES	
Optional 128 print positions Transmit/Receive Monitor Offline listing Form Control Optional print bars affording varied character sets	

3.6.2. UNISCOPE 300 Visual Communication Terminal

The UNISCOPE 300, shown in Figure 3–23, is a visual communication terminal designed for applications which require direct operator interaction with a centralized computer. Information generated by the operator is displayed on the UNISCOPE prior to transmission to the computer so that the operator can make any required changes or edit where necessary. Data transmitted from the computer is displayed on the UNISCOPE for operator information and interpretation.

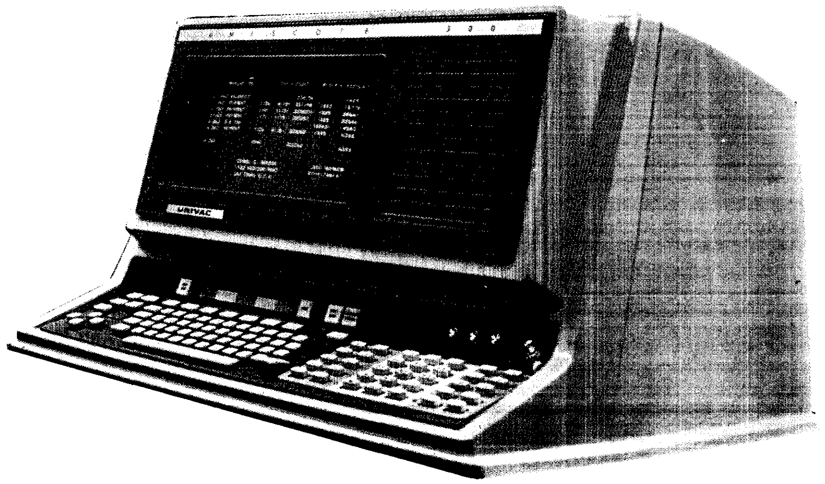


Figure 3–23. UNISCOPE 300 Terminal

3.6.2.1. Basic Concepts

The UNISCOPE subsystem consists of a cathode-ray tube (CRT) display, keyboard, and memory designed to minimize communication and maintenance costs. The UNISCOPE may be used in either a single or multi-station configuration. In the single station arrangement, only one UNISCOPE containing its own display memory and control section may be connected to a data set (modem). In the multi-station arrangement, a Multi-Station Control Unit (MSCU), containing the display memory for all its display units, is connected to the modem and interfaces with up to 48 multi-station UNISCOPE display units. The components of the single and multi-station configurations are discussed in the following paragraphs. Detailed information may be found in *UNIVAC UNISCOPE 300 Visual Communications Terminal General Description Manual, UP-7619* (current version).

The UNISCOPE Subsystem consists of a CRT display, keyboard, memory, and control logic. The subsystem is designed for synchronous transmission at speeds of 2,000 bits per second and higher, and interfaces a data set (modem) such as the Bell System 201 Series or equivalent through the EIA standard interface RS-232. In addition to operating individually on a private line, the subsystem may be connected to a multipoint party line, and will respond to a poll code from the central computer. Character and message parity are checked on each incoming message and generated for each outgoing message. Erroneous blocks may be retransmitted automatically upon request from the computer.

- Display Screen

  The display screen is a cathode-ray tube (CRT) with a viewing surface 10 inches wide and 5 inches high. A format of 64 characters per line on 16 lines per display is provided. The display uses a digital scanning technique, as opposed to TV scanning, to provide excellent linearity. Spacing between characters is consistent from end-to-end of the screen, and the size and shape of each character does not change with its relative position on the screen. The character style maximizes legibility and readability. Each character is .150 x .113 inches and is readable up to a distance of seven feet. The character brightness may be varied by the operator from 70 percent brightness to full brightness.

  The presentation is such that no flicker or jitter is perceptible to the operator. Each of the 1,024 characters is repainted on the display surface 60 times each second. The regeneration is synchronized with the power source to eliminate interference with the 60-cycle power supply.

- Display Memory

  The display memory is a computer core memory providing the same reliability, speed, and random access capability found in other Univac computer systems. The use of a core memory also simplifies control hardware, since timing restrictions are not required on the reading or writing of data. Because the display is regenerated from memory, the traffic between the UNISCOPE and the computer is for the purpose of inputting or sending out new data only, and the lines are not burdened by refreshing directly from the processor memory.

- Control Section

  The control section of the UNISCOPE Subsystem directs all of the UNISCOPE operations. The control section interprets all instructions and generates commands to the other sections of the unit, including the sequence of I/O operations, positioning of the cursor (position marker), addressing memory for painting characters, and handling block moves of data.

- Keyboard

  The keyboard is the interface with the UNISCOPE and the computer, by which the operator can control the UNISCOPE, input data to the memory, and request data from the computer. Each time that a key is depressed, a unique code is sent to the keyboard control section which examines this code to see if it is data or a function. Data is entered in the position indicated by the cursor. A variety of cursor control and function keys are also provided on the keyboard.

■ Characteristics

UNISCOPE 300 characteristics are summarized in the following table:

PARAMETERS	SPECIFICATIONS
Display	
Capacity	1,024 or 512 Characters
Viewing Area	10" Wide x 5" High
Format	64 Characters per Line x 8 or 16 Lines
Character Size	.150" High x .113" Wide
Refresh Rate	60 Cycles per Second
Scan Method	Digital
Character Set	56, 61, or 96 Symbols
Character Generation	Closed Stroke, Maximum 8 per Character
Keyboard	Basic Alphanumeric Typewriter 9 Cursor Control Keys 5 (7 Functions) Editing Keys 0, 5, or 40 Function Keys 122 Possible Function Key Overlays (Over 4,000 Functions are Possible)
Memory	
Single Station	1,024 Character Magnetic Core 7.2 Microseconds Cycle Time
Multi-Station	8,192, 16,384, or 24,576 Character Magnetic Core 1.8 Microseconds Cycle Time
Code	7-Bit Modified USASCII
Subsystem Size	
Single Station	1: 1,024 Characters
Multi-Station	2–24: 1,024 Characters each 2–48: 512 Characters each
Interface	Communication (Telephone) Line, AT&T 201 Data Set or Equivalent, 2,000 Bits per Second or Higher
Power	120 Volts + 10% – 15% Single Phase 60 Cycles per Second ± .5 Cycle
Size	
MSCU	36" x 24" x 64"
Display	25" Wide 17" High 24" Deep

3.6.3. UNIVAC 9200/9300 Systems

The UNIVAC 9200 System, or the UNIVAC 9300 System as discussed in 3.5.3.4, may be used as remote subsystems to the UNIVAC 494 computer. The UNIVAC 9200/9300 Systems are linked to transmission facilities through the Data Communication Subsystem (DCS), with the transmission facilities being connected to the UNIVAC 494 computer through the CTMC Subsystem.

The remote UNIVAC 9200/9300 Subsystems, shown in Figure 3–24, provide inexpensive and efficient means for transmitting data or job stream information to the UNIVAC 494 computer. Editing and data manipulation features of the 9200/9300 Systems provide powerful tools in the preparation of data, thereby saving valuable transmission time.

A valuable feature of the configuration is that the UNIVAC 9200/9300 Systems may be used as independent computers when not engaged in remote transmissions. Another outstanding feature of the systems is that they are compatible, being members of the UNIVAC 9000 Series, which includes the UNIVAC 9400 System, providing for expansion and growth from smaller to larger systems and configurations.

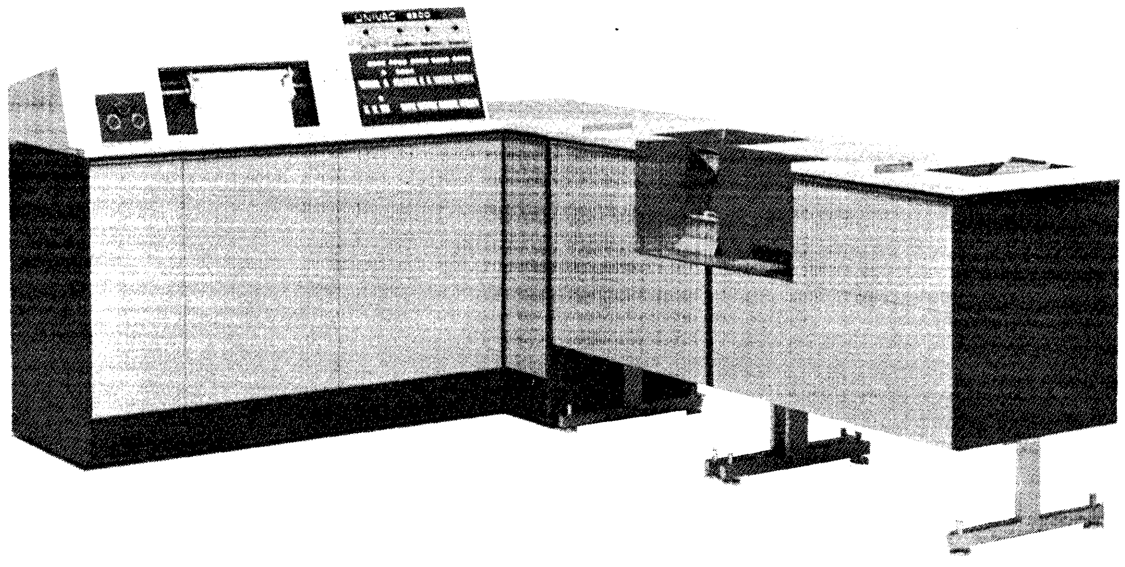


Figure 3–24. UNIVAC 9200/9300 Systems

3.6.3.1. UNIVAC 9200 System

The UNIVAC 9200 System is a compact, low priced, internally programmed computer system, which is card oriented. The computer is equipped with all functions for the execution of instructions, including arithmetic and input/output control. An integral card reader, card punch, and line printer are standard peripheral units. A multiplexer channel is available for use as the communication channel.

Memory is organized into bytes consisting of eight data bits plus one parity bit. Minimum memory size is 8,192 bytes and maximum memory size is 16,384 bytes. Memory cycle time is 1.2 microseconds.

- Characteristics

  Characteristics of the UNIVAC 9200 System are shown in the following table. Detailed information may be found in *UNIVAC 9200 System System Description Manual, UP-4086* (current version).

PARAMETERS	SPECIFICATIONS
System Orientation	Card
Basic Memory Maximum Memory	8192 bytes 16,384 bytes
Memory Cycle Time	1.2 microseconds
Add (Decimal) Instruction Time (Two 5-digit Fields)	104 microseconds
Multiply, Divide, Edit	Optional
Card Read – Basic Reader 1001 Card Controller	400 cpm 1000/2000 cpm
Card Punch	75–200 cpm (depends on last column punched)
Read/Punch Selective Stacker	Optional Optional
Print Speed (Alphanumeric)	250 lpm 300 lpm optional
Variable-Speed Printing (Optional)	250/500 lpm or 300/600 lpm
Overlapped I/O Units	Standard
Multiplexer Channel (Optional)	85,000 bytes/sec.transfer rate

3.6.3.2. UNIVAC 9300 System

The UNIVAC 9300 System is an internally programmed computing system which offers both a powerful 80-column card processing capability and a high speed magnetic tape system. The computer is equipped with all functions for execution of instructions including arithmetic and input/output control. The integral card reader, card punch, and line printer offer higher speeds than those available on the smaller UNIVAC 9200 System. The multiplexer I/O channel of the UNIVAC 9300 can accommodate up to eight peripheral subsystems. Maximum memory size is 32,768 bytes of plated wire memory with a cycle time of 600 nanoseconds.

- Characteristics

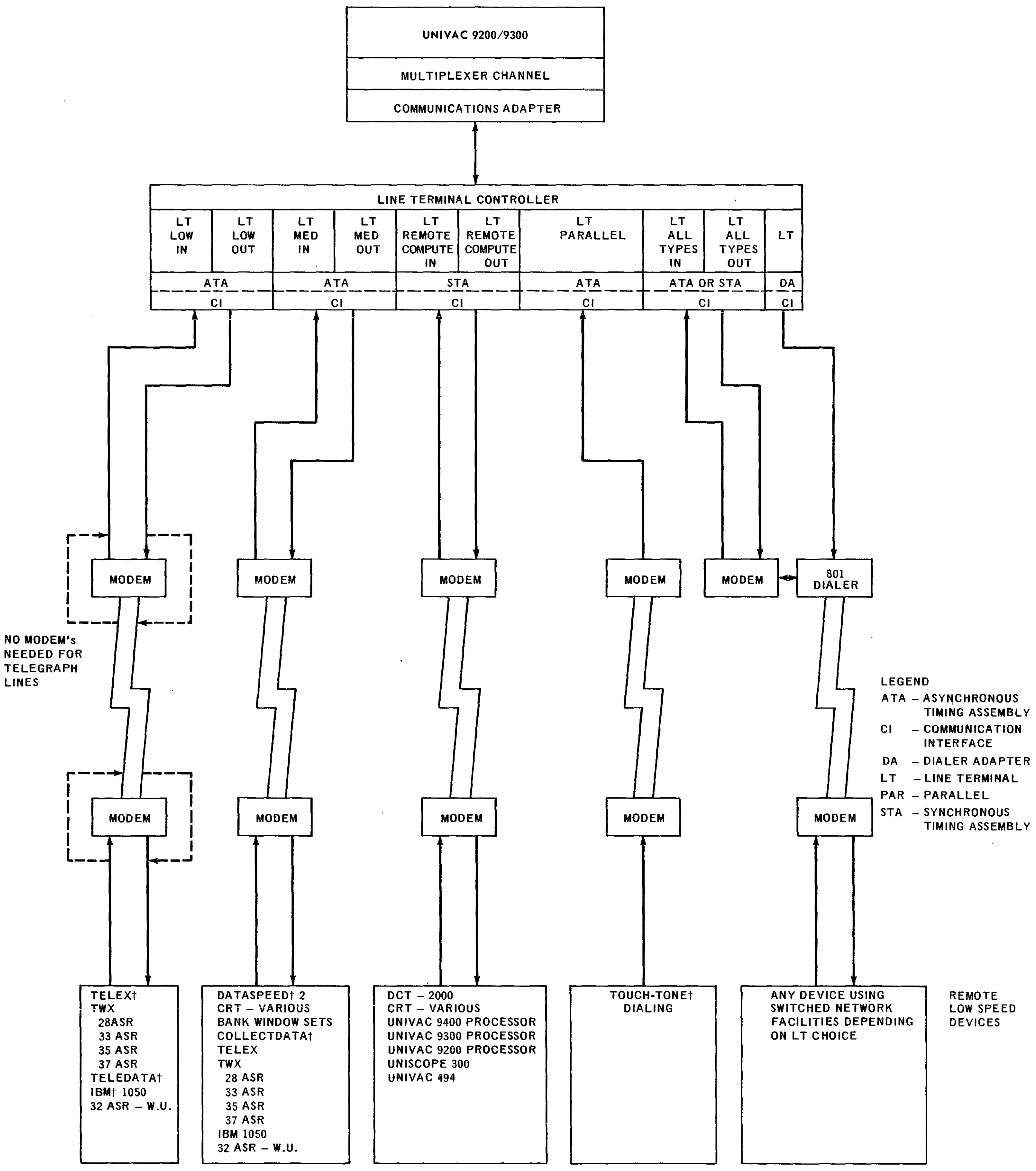
  Characteristics of the UNIVAC 9300 System are given in 3.5.3.4. Detailed information may be found in *UNIVAC 9300 System System Description Manual, UP-4119* (current version).

3.6.3.3. Data Communication Subsystem

The Data Communication Subsystem (DCS) provides the UNIVAC 9200/9300 Systems with a versatile communication capability. The subsystem will interface with private wire telegraph, public network and private wire voice grade, and broad band facilities at all standard speeds, including Bell System Telpak speeds.

The versatility of the DCS allows the UNIVAC 9200/9300 Systems to communicate with the UNIVAC 494 computer in virtually any manner desired by the user. The UNIVAC 9200 or UNIVAC 9300 may be used as a simple remote unit record device for remote job or data entry; or, in a more sophisticated application, the UNIVAC 9200 or UNIVAC 9300 System may be used to concentrate and preprocess remote inputs (such as those from the UNISCOPE 300) at a remote site before transmission to the UNIVAC 494.

The DCS configuration may be varied to suit the particular installation. The modular elements comprising the subsystem are the Line Terminal Controller (LTC)/(LT), the Timing Assemblies (TA), and the Communication Interface Units (CIU). Configurations with some of the more commonly used remote devices are shown in Figure 3–25.


UNIVAC 9200/9300
MULTIPLEXER CHANNEL
COMMUNICATIONS ADAPTER
LINE TERMINAL CONTROLLER
LT LOW IN
LT LOW OUT
LT MED IN
LT MED OUT
LT REMOTE COMPUTE IN
LT REMOTE COMPUTE OUT
LT PARALLEL
LT ALL TYPES IN
LT ALL TYPES OUT
LT
ATA
ATA
STA
ATA
ATA OR STA
DA
CI
MODEM
801 DIALER
NO MODEM's NEEDED FOR TELEGRAPH LINES
LEGEND
ATA – ASYNCHRONOUS TIMING ASSEMBLY
CI – COMMUNICATION INTERFACE
DA – DIALER ADAPTER
LT – LINE TERMINAL
PAR – PARALLEL
STA – SYNCHRONOUS TIMING ASSEMBLY
TELEX† TWX 28ASR 33 ASR 35 ASR 37 ASR TELEDATA† IBM† 1050 32 ASR – W.U.
DATASPEED† 2 CRT – VARIOUS BANK WINDOW SETS COLLECTDATA† TELEX TWX 28 ASR 33 ASR 35 ASR 37 ASR IBM 1050 32 ASR – W.U.
DCT – 2000 CRT – VARIOUS UNIVAC 9400 PROCESSOR UNIVAC 9300 PROCESSOR UNIVAC 9200 PROCESSOR UNISCOPE 300 UNIVAC 494
TOUCH-TONE† DIALING
ANY DEVICE USING SWITCHED NETWORK FACILITIES DEPENDING ON LT CHOICE
REMOTE LOW SPEED DEVICES

† TELEX – TRADEMARK OF WESTERN UNION TELEGRAPH CO.
TELEDATA AND COLLECTDATA – TRADEMARKS OF FRIDEN, INC.
IBM – REGISTERED TRADEMARK OF INTERNATIONAL BUSINESS MACHINES CORP.
DATASPEED – TRADEMARK AND SERVICE MARK OF A.T. & T. CO.
TOUCH-TONE – REGISTERED SERVICE MARK OF A.T. & T. CO.

Figure 3–25. DCS Configurations

- Line Terminal Controller

  The Line Terminal Controller (LTC) interfaces the UNIVAC 9200/9300 Multiplexer Channel with one or more Line Terminal (LT) devices. The LTC controls and coordinates data transfers between the UNIVAC 9200/9300 Systems and the LT's.

- Line Terminal

  Line Terminals (LT's) provide the logical interface between the communication facility and the 9200/9300 LTC by the system. The LT's are simplex units which may be interconnected to create half duplex or full duplex communications environments. Several types of LT's are available to provide low and medium speed asynchronous operation or to meet synchronous high speed requirements. Data characters may range from four to eight bits in size (level) depending upon the model and mode of LT used.

  Transfer rates of the various LT's are as follows:

– Low Speed:	75–300 bits per second
– Medium Speed:	300–2400 bits per second
– Synchronous	
Voice Grade:	2000–4800 bits per second
– Broad Band:	40,800 or 50,000 bits per second
– Telpak C:	230,400 bits per second

- Timing Assemblies

  Timing Assemblies (TA's) provide a clock source for asynchronous line terminals. Synchronous TA's are also available for operation with asynchronous modems or where there is no external synchronizing clock.

- Communication Interface Unit

  The Communication Interface Unit (CIU) is the electrical interface between the line terminals and the common carrier lines. The available CIU meets both the EIA RS-232B (Industry Standard Interface) and the MIL-STD-188B (Electrical Circuit Compatibility – Government) specifications. Each input/output line pair requires one CIU.

3.6.4. UNIVAC 1004 System

The UNIVAC 1004 System, shown in Figure 3–19, is a complete data processing unit which may be used as an online peripheral subsystem (see 3.5.3.3) or as a remote peripheral subsystem to the UNIVAC 494 System.

3.6.4.1. Remote UNIVAC 1004 System

The UNIVAC 1004 System may be used for printing, reading, and punching (optional) of 80-column cards in remote operation; paper tape handling facilities are not offered. Interface to the communication lines is made by attaching a Data Line Terminal (DLT) to the remote UNIVAC 1004. Synchronous transmission is used at speeds of 2000, 2400, or 40,800 bits per second depending upon the type of DLT and communication facility employed.

A special plugboard, which is also dependent upon the type of DLT used, is required when the UNIVAC 1004 is to be used as a remote subsystem to the UNIVAC 494 computer. The UNIVAC 1004 System retains its processing capability when used in this configuration, and can be switched to an offline mode at any time for operation as an independent computer.

■ Characteristics

Characteristics of the remote UNIVAC 1004 Subsystem are the same as given for the online UNIVAC 1004 Subsystem in 3.5.3.3, with the exception that the paper tape parameters are not applicable. The reading, printing, and punching speeds reflect the capability of the UNIVAC 1004 System; speeds attained in actual remote use may be somewhat lower due to the transmission facility employed. Detailed information may be found in *UNIVAC 1004 Card Processor 80 Column Reference Manual, UT-2543* (current version) and in *UNIVAC 494 Real-Time System Remote UNIVAC 1004 System*, P. I. E. Bulletin 12, UP-4121.12.

■ Data Line Terminals

A variety of Data Line Terminals (DLT's) are available for connection with a remote UNIVAC 1004 Subsystem depending upon the transmission facilities and speed required by the user. All DLT's employ synchronous transmission modes.

– DLT Type 1

The Type 1 DLT interfaces with the Bell System 201A or 201B Data Set or equivalent for half duplex transmission over voice grade communication facilities on either private line or the switched telephone network. Transmission speeds are 2000 bits per second with the 201A, and 2400 bits per second with the 201B. Character and message parity are checked and generated by the DLT.

– DLT Type 1B

The Type 1B DLT interfaces with the Bell System 301B Data Set and Telpak facilities for half duplex transmission over broad band lines at a rate of 40,800 bits per second. Parity checking and generating is the same as for the Type 1 DLT.

– DLT Type 3

The Type 3 DLT interfaces with the 201A or 201B Data Set or equivalent for half duplex transmission on the switched telephone network or a private line. Parity features are included. In addition, the Type 3 DLT allows the remote UNIVAC 1004 System to overlap processing with data transmission.

4. OPERATING SYSTEM SOFTWARE

4.1. SOFTWARE DESIGN CONCEPTS AND CAPABILITIES

The UNIVAC 494 Operating System is a comprehensive library of integrated programs comprising a flexible and powerful executive control system and a collection of programming languages, utility routines, and application packages. Through a versatile and effective control language, the executive routine organizes and directs basic computer operations and system activities to achieve maximum utility of computer facilities with great system economy.

4.1.1. Software Design Concepts

The executive routine is a master control program which has been designed and implemented to establish and to operate the efficient multiprogramming environment needed for utilizing the full capabilities of the UNIVAC 494 Real-Time System.

The speed and hardware capabilities of the UNIVAC 494 System are used to maximum advantage, and a given hardware configuration is used most effectively in the complex internal operating environment created by the executive routine. This environment must allow for the concurrent operation of many programs; for immediate reaction to the inquiries, requests, and needs of many different users at remote and local stations under the stringent demands of real time application; for storage, filing, retrieval, and protection of large blocks of data; and for optimum use of all available hardware facilities while minimizing job turnaround time.

The executive routine and other software of the UNIVAC 494 Operating System are discussed in the following sections. Detailed information may be found in *UNIVAC 494 Real-Time System Operating System Programmers Reference, UP-7504* (current version).

Through central control of all of the activities of the UNIVAC 494 System, the combined hardware and software capabilities are fully established and maintained to satisfy the requirements of all applications. The responsibility for efficient and flexible centralized control is borne by the executive routine. The comparatively simple interface presented to the programmer by the executive routine allows the programmer to use the system with relative ease, while relieving him of concern for the internal interaction between his program and other coexistent programs.

Design capabilities of the Operating System span a broad spectrum of data processing activities. No penalties for inefficiency are imposed upon any of the activities by the support provided for the others. Specific capabilities which are not desired by a particular installation may be eliminated at systems generation time.

Ease of use by the programmer or the casual user is emphasized in the system. Work to be performed by the system is described on control cards to minimize job turnaround time, operator intervention, and decision requirements. At his desk, the user may construct any logical combination of programs for a particular job by inserting the proper control cards in his job deck.

Job decks can be collected and entered into the system from many sources, remote or central. The executive routine controls the loading, allocation, and execution of the described programs once they are entered into the system. Jobs which cannot be completed because of program error are automatically deallocated and purged from the system with appropriate diagnostic information. The console operator is, in effect, responsible only for mounting and labeling tapes under direction of the executive routine.

4.1.2. System Capabilities

In summary, the UNIVAC 494 Operating System provides the following major features. Particular functional capabilities of each feature are presented in subsequent sections of this manual.

- Real Time/Online Processing

  The executive routine efficiently responds to the demands of real time processing, and gives preference to the operational needs of a real time program, these being the most critical requirements of the system. Executive services which are appropriate to the construction and execution of real time programs are provided which permit a real time program to exercise critical control over system service. The contingencies of real time are supported by nonstop operation with procedures for rollout of conflicting user programs, system restart, and other necessary functions.

- Batch Processing

  Facility of job preparation and submissions, with minimization of job turnaround time, is a design feature of the system. A priority specification provides preferential service for batch runs submitted by remote operation or where turnaround time is critical.

  The Operating System provides for high volume job shop operation. All jobs entering the system are described by a control language. The user may specify preferred service for certain jobs with no responsibility in planning schedules to achieve machine optimization. Job descriptions are accepted from any specified source and may be preregistered in referencing standard jobs. Automatic job-to-job transition, communication within jobs, and associated services, such as logging and accounting, are provided by the system.

- Program Development

The Operating System interfaces an inclusive set of source code language processors, enabling the programmer to use the languages COBOL, FORTRAN IV, 494 ASM (procedure-oriented assembler), and 494 SPURT. Independently processed program elements are collected and combined into an integrated object program. The collection provides an efficient and flexible facility for developing and maintaining a complex program.

The system provides standardization of common functions, eliminating duplication of these functions in separate user programs, and establishing a common program and operator interface. Standardization contributes to installation efficiency by accommodating changes in machine configuration and operating procedures without direct impact on user programs. Changes in one user program which tend to infringe on other user programs are similarly minimized.

A test system provides the user with complete control over programs in the debugging process and allows extraction and display of run-time information. An object time, source-level, debugging mechanism common to all programs is provided which eliminates the need for source-time planning of debugging strategy. The system significantly reduces the time and expense associated with program checkout.

- Automatic Operation

System operation is defined through a control language providing powerful and flexible user direction. The control language is a formal description of the functions preparatory to execution of a program. Operator participation is explicitly defined and is minimized as much as possible.

Utilization of random access storage is the primary method for eliminating the delays and errors attendant upon operator intervention and for increasing overall system efficiency. Random access storage is used as a system buffering for the job backlog accepted from system input devices and for the resultant images for system output devices shared among the executed jobs. The buffering allows the system to operate independently of the essentially low-speed peripheral devices. All executable programs are obtained from random access storage through a system of libraries maintained by monitor routines. Temporary intermediate files required in operation of a program are generally assigned to random access storage rather than tape, to facilitate automatic operation.

A catalog of random access files, the Master File Directory (MFD), is maintained by the system for files which transcend a particular job. The MFD facilitates automatic operation and provides permanent file storage to a collection of individual and independent users.

- Integrity

  Complete system integrity is effected through hardware memory lock-out, guard mode, and software validation of service requests. An errant program cannot destroy either the system or other programs in the multiprogram environment. Random access files are protected through the use of file codes and logical file addressing. Comprehensive contingency procedures facilitate recovery from error conditions.

  Programmers, users, and engineers are essentially prevented from changing basic options and from specifying other options which may influence operation of the total system.

- Modularity

  The Operating System is explicitly modular to facilitate future extensions, expansion of particular functions, or selection of available variants of a basic function. Modularity can be exercised during generation of a system so that each user can create versions of the system which will operate more efficiently for the system needs and configuration. A simple and flexible means of complete systems generation and maintenance can be utilized at each installation.

4.2. EXECUTIVE SCHEDULING AND CONTROL

The integrated routines of the Operating System provide basic control for coordinating and executing Univac- and user-provided programs, and for furnishing a flexible and reliable foundation upon which the installation environment can build. The Operating System is dependent upon random access storage as an operating base and upon a primary input device as a source for work definitions of jobs to be run. The basic output generated during execution of jobs is provided on appropriate system output devices.

As the heart of the Operating System, the executive routine is the vehicle for interfacing and controlling the systems environment. The executive routine contains elements for the selection and activation of tasks (job control); elements for the control of programs operating in a multiprogram environment (task/activity control); elements to provide run-time service to operating programs (service control); elements for the assignment, access, and manipulation of data files (data management); and elements to provide for the assignment and access of remote devices. Figure 4–1 shows the relationship of the major components of the executive routine.

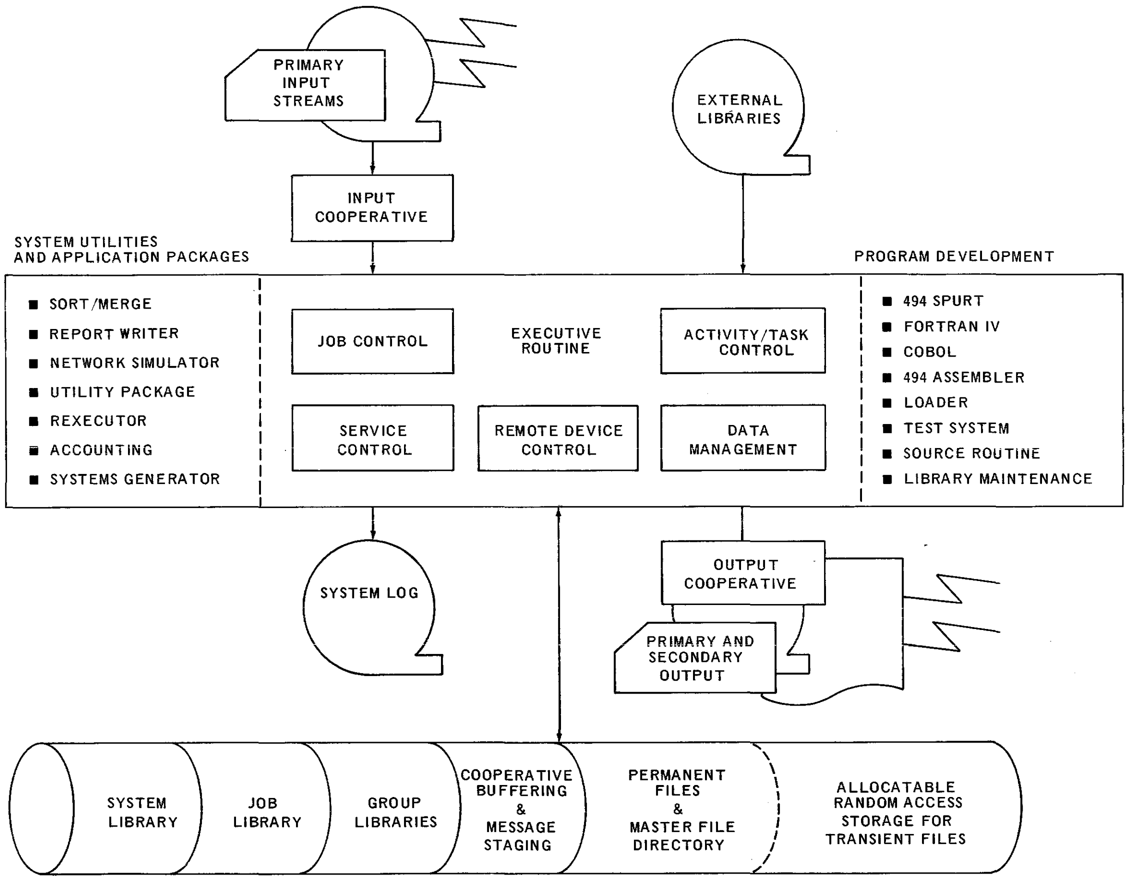
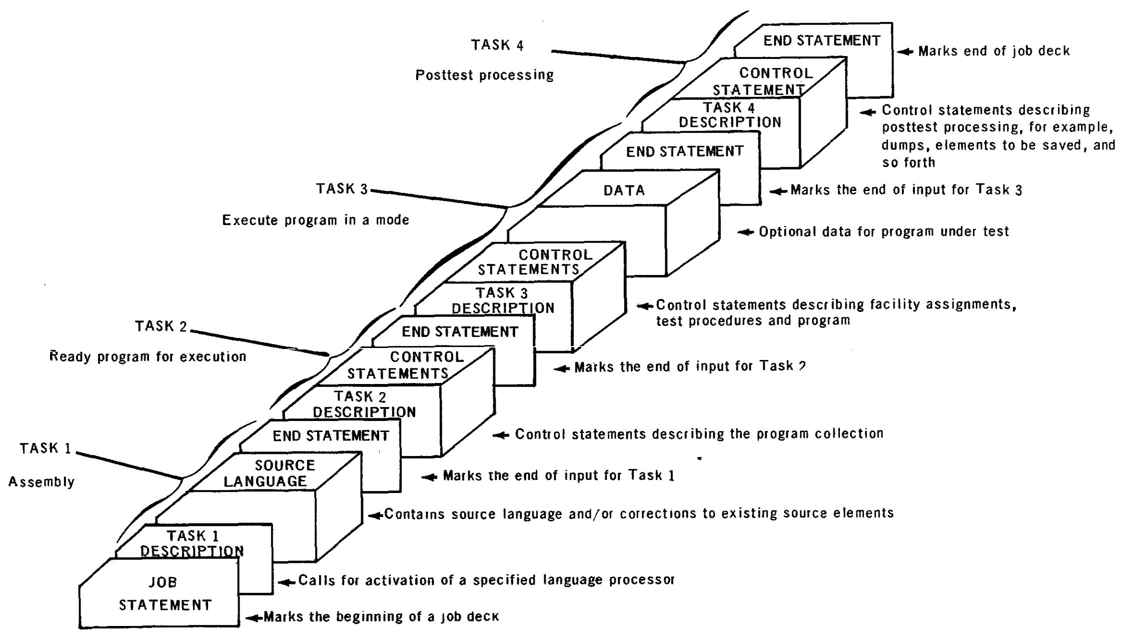


Figure 4–1. Operating System Executive Routine

4.2.1. Primary Input Stream

The primary input stream conveys information to the system through scheduling routines (input/output cooperative mechanism). This input includes control statements specifying operations to be performed, limited data, source code for the language processors, program parameters, and other information pertinent to a desired job. The information is formed into job decks composed of an ordered sequence of tasks. Each task is a logical step within the processing of the job and consists of a number of interrelated activities. As an example, the job deck for assembly and testing of an element of a program may be as represented in Figure 4–2.



TASK 4
Posttest processing
END STATEMENT
Marks end of job deck
CONTROL STATEMENT
TASK 4 DESCRIPTION
Control statements describing posttest processing, for example, dumps, elements to be saved, and so forth
END STATEMENT
Marks the end of input for Task 3
TASK 3
Execute program in a mode
DATA
Optional data for program under test
CONTROL STATEMENTS
TASK 3 DESCRIPTION
Control statements describing facility assignments, test procedures and program
TASK 2
Ready program for execution
END STATEMENT
Marks the end of input for Task 2
CONTROL STATEMENTS
TASK 2 DESCRIPTION
Control statements describing the program collection
TASK 1
Assembly
END STATEMENT
Marks the end of input for Task 1
SOURCE LANGUAGE
Contains source language and/or corrections to existing source elements
TASK 1 DESCRIPTION
Calls for activation of a specified language processor
JOB STATEMENT
Marks the beginning of a job deck

Figure 4–2. Job Deck Composition

To each user, the system presents the singular function of performing the job submitted by him as described by the executive control language.

4.2.2. Input Cooperative

The input cooperative is a collection of scheduling routines which operate through the input/output cooperative mechanism (see 4.4.6) to accept the primary input stream from system input devices. The primary function of the input cooperative is to employ random access storage for buffering, to balance intermittent system utilization with the slow rate of peripheral devices. The buffering also permits parallel utilization of the input stream by jobs executed concurrently in the multiprogram environment.

The input cooperative feeds multiple streams of primary input to the system (see Figure 4–3). Access to the primary input stream data is by service request from the active tasks. Supplementary streams may be introduced from specified auxiliary sources. The supplementary streams may be used to merge and correct source code, be extended by an installation to serve the need of user programs, or be used to enter supplementary control statements for job description. This feature is especially advantageous to remote users of the system.

Each unit input routine scans the input stream to identify job decks which are queued for job control upon entry to the system. The input queue is maintained in a common storage pool (cooperative buffering) used by all system input/output cooperatives. The common storage pool dynamically expands or contracts depending upon the demands of the environment. The remote device scheduler accepts a control stream through communications facilities in the same manner as locally submitted jobs. The remote device scheduler provides the interface between the input/output cooperatives and remote equipment, and takes the place of an input/output routine to the unit record devices.

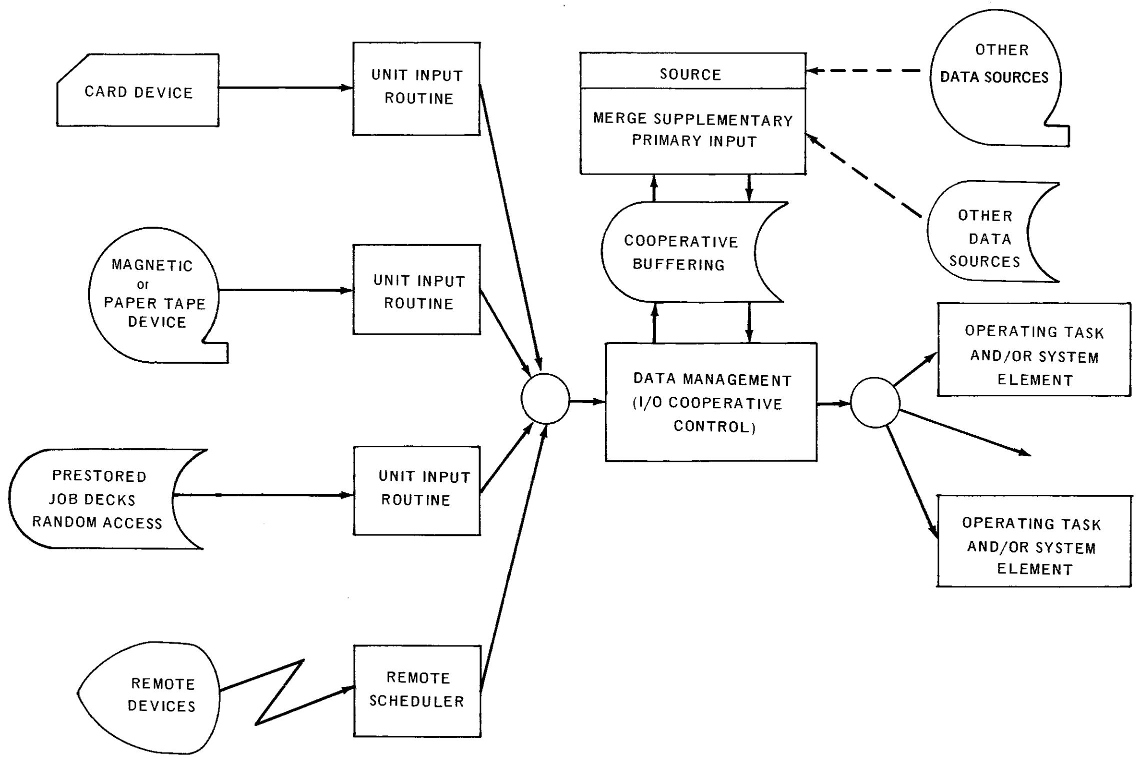

CARD DEVICE
UNIT INPUT ROUTINE
MAGNETIC or PAPER TAPE DEVICE
UNIT INPUT ROUTINE
PRESTORED JOB DECKS RANDOM ACCESS
UNIT INPUT ROUTINE
REMOTE DEVICES
REMOTE SCHEDULER
SOURCE
MERGE SUPPLEMENTARY PRIMARY INPUT
OTHER DATA SOURCES
OTHER DATA SOURCES
COOPERATIVE BUFFERING
DATA MANAGEMENT (I/O COOPERATIVE CONTROL)
OPERATING TASK AND/OR SYSTEM ELEMENT
OPERATING TASK AND/OR SYSTEM ELEMENT

Figure 4–3. Input Stream Cooperative Control

4.2.3. Job Control

Job control is a scheduling function which provides effective allocation of the hardware configuration, and minimizes turnaround of batch programs operating as background to the high priority real time program(s).

Each job deck recognized by primary input control is entered into the job stack on a first in, first out basis as determined by the priorities allowed for standard production and remotely originated jobs (see Figure 4–4). The job stack is processed by a selection routine which determines which job is to be introduced next into the multiprogram environment and which triggers preparatory functions for execution of that job.

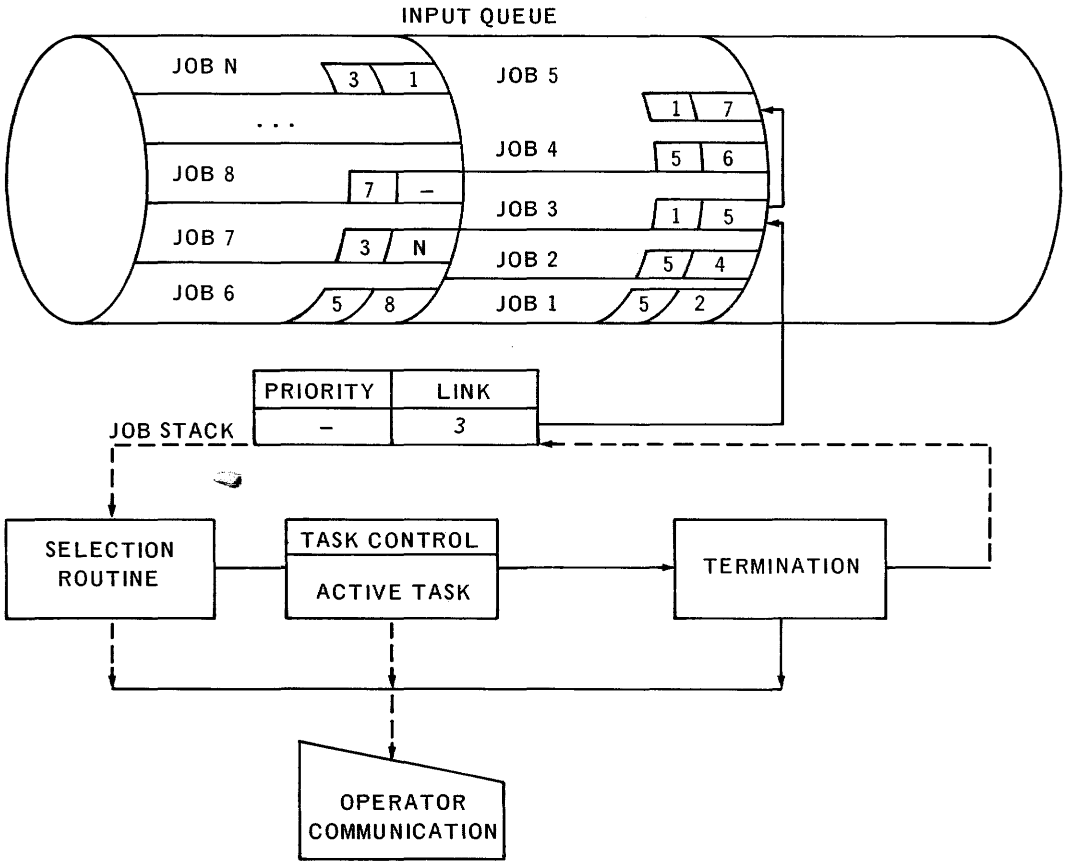


Figure 4–4. Control of Job Stack

4.2.3.1. Selection Routine

The selection routine has responsibility for determining which task from the job stack is introduced into the mix of active programs.

Selection is initiated in the following sequence:

(1) An input stream job card is detected by the input cooperative: the job card is entered into the job stack and selection is activated.

(2) An operating task within a job has terminated: the task is returned to the job stack and selection is activated.

(3) An operating task requests that a preformed job stream be entered into the job stack. Preformed jobs are source elements containing complete job streams and which may reside in a system library or in storage allocated to the requester.

(4) An operating task releases a facility (peripheral device, core, and similar hardware): selection is activated to ascertain if the enlarged facility pool will accommodate selection of another task.

In processing the job stack, the selection routine analyzes the job description to determine the facility requirements of the next task within each job. Requirements for system processors (such as Loader, library maintenance) are known to the system and do not require format expression by the user. Facility requirements are either expressed in the input stream or embedded as an information block associated with the absolute program referenced.

The selection routine ascertains which tasks may be activated by criteria based upon priority and the available facilities. The prime criterion is priority. Within a priority class, preference is given to tasks of a partially completed job, and to tasks which involve the best utilization of available facilities and core. Tasks which are bypassed in their normal order for lack of assignable facilities are assigned an improved preference for subsequent selections. No task of a job is initiated until all prior tasks of that job have been completed. Selection activates as many tasks into the multiprogram environment as can be accommodated by the existing system status, and continues until the job stack is exhausted.

The selection routine accesses the job input stream, and responds to system control cards by performing the described function or by submitting a service request to perform the function.

The selection routine unstrings the system control statements so that options and entries in the specification list can be conveyed to the task being activated. Each activated system or user task performs its function to completion. A completion status is declared by a task termination entry or may be imposed by some non-recoverable hardware or program contingency. Termination reactivates the job stack for continued interpretation of the remaining job input stream. At this point, the input stream may have been depleted of data by operation of the system or by the user task. Cards of a noncontrol nature, acquired by an active task through an I/O entry, may exist in the input stream, representing information processed by the user task, source language utilized by language processors, or secondary language used by utility programs. A task may be terminated without using all of the non-control cards available to the task through an end of file (EOF) condition imposed by the program. When a requester encounters a system control card in the job deck, an EOF condition is returned, determining the end of input of task data, and no image is presented. The task cannot force further input once the EOF condition is reached. The EOF condition may be used by the program to determine the end of task data. Reactivated selection simply ignores or discards the outstanding data until a control card is detected.

Each successive task has independent facility requirements which must be evaluated by selection; all facilities of the completed task are released except those explicitly required for subsequent tasks.

When the end of the job input stream is detected, a system element for post-job processing is initiated. The responsibilities of the post-job processing element include production of an accounting record; deallocation of all temporary files, records, and peripheral subsystems; and notification of the operator as to job status, with directions for demounting of tapes and performing other operator-controlled functions.

4.2.3.2. Task Execution

All system functions are integrated to effect complete execution of user or system tasks. Execution is accomplished through a priority queuing function which provides for time sharing of available facilities to achieve effective machine utilization and optimized throughput of batch and real time programs.

Queues are maintained for CPU control, I/O control, peripheral and core assignments, time intervals, shared programs, and other system needs. The queues are processed in a manner responsive to both program priorities and system optimization procedures, providing for maintenance of work backlogs to permit maximum parallel use of key facilities.

A task is activated and controlled through a task addendum which is used by all system elements in performing functions for the task. The operating task may define parts of itself, called activities, to be executed concurrently in the multi-program environment. An activity is controlled by an activity addendum which is linked to the task addendum. Although the activity is in large part independent from other activities of the task, the activity shares peripheral allocation, file reference, and other facilities with other activities through a link to the task. An activity may be executed asynchronously or in parallel mode with other activities of the task.

Task control accepts service requests defining a function to be performed during execution of a task. Service routines are characterized by direct control of the computer. In general, the routines execute privileged instructions such as input/output, operate without memory lockout, and access and directly manipulate the entire storage. The routines are responsible for the integrity of the system and for preventing interprogram destruction or conflict. A request initiated under control of an activity is either satisfied immediately, with control returned to the requesting activity, or queued for later service and subsequent reactivation of the requester.

Interrupt response and software/hardware contingencies are handled without mandatory user participation. User specified alternatives to the standard fault procedures may be employed by the system. Interrupts are processed as they occur and generally result in the queuing, for CPU control, of an activity which analyzes the interrupt. The critical functions for maximizing channel throughput are performed by the executive routine at the time of interrupt. Critical functions include buffer swap, initiation of the next function to effect continuous mode, and similar operations.

Immediate response to interrupts achieves a smoothing effect such that interrupts are actually processed at different priority levels and no interrupt precludes registration of another. Since the executive routine operates under the priority of the requesting activities, control may generally be switched to a high priority activity subsequent to the occurrence of an interrupt.

4.2.4. Element Libraries

The library concept is basic to the system. The smallest logical unit of information which may be entered into the system is an element. A collection of elements is called an element library. Three logical levels of libraries are referenced: job, group, and system. The levels are referenced in the stated order so that override may be controlled and predicted.

- Table of Contents (TOC)

  To identify and locate the elements within the library, a table of contents (TOC) is created and maintained by the system for each element within a library. The TOC identifies elements by name, version, and type, and by control information unique to each particular type. The name is symbolic and is associated with the element at the time of creation. A version may be associated with each relocatable element to differentiate between variations of the same element, and to provide a convenient reference when checking out large programs. A library may contain alternate versions of the same program, with one version being outdated but workable, and the other being in a test stage.

  Elements within the library complex are manipulated in the system by the Program Library Editor (PLE) which is a collection of maintenance routines (see 4.7).

4.2.4.1. Element Types

Three basic types of elements may comprise an element library:

- Source Element

  A source element may be any collection of source statements. In addition to the programming languages processed by COBOL, 494 SPURT, 494 Assembler, and FORTRAN IV, the control languages processed by the system may be stored as an element. The source element may contain any set of statements which may appear in the primary input stream. Thus, job control statements, secondary control statements, and limited data sets may be included as source elements.

- Relative Binary (RB) Element

  Relative or relocatable binary (RB) elements are intermediate output codes produced by the language processors, COBOL, 494 SPURT, 494 Assembler, and FORTRAN IV, and represent processed source language programs or subprograms which may be complete or which may be dependent upon collection with other RB elements before utilization in a program. An RB element must be processed into load element by the Loader before it is executable as a program.

- Load Element

  Absolute and relative load elements are produced by the Loader through the collection and code modification process which combines RB elements. An absolute element is an entity with all external references connected and interconnected, cross references resolved, and relative locations assigned. The absolute element can be entered into any continguous area of primary storage by, essentially, a direct read operation from random access storage, and is directly executable on machines operating in the 494 mode (with the RIR) under control of the Operating System. A relative load element, which may be modified as necessary at execution time, may be produced for machines which operate in the 490 mode (without the RIR).

4.2.4.2. Library Types

The characteristics of the hierarchy of libraries are determinants of their functional relationship to the system. Three basic types of libraries are used in the system.

- Job Library

  The job library is a collection of elements created for a particular job. The library is built as the job is executed, and is interrogated only to satisfy requests associated with the job. The job library supports the continuity between tasks within the job.

  A job library is established by a library maintenance function (IN), or by elements generated or referenced during job execution. Control language statements, therefore, explicitly and implicitly contribute to a job library. One job library exists for each active job. A given job library is transient and remains only as long as the job is active. Elements which are to be preserved must be output from a job library by a library maintenance function (OUT) as part of the job.

- Group Library

  The group library is a bridge between the impermanence of a job library and the permanence of the system library. In effect, the group library is an independent file established in the Operating System through the LINK statement and the Master File Directory (MFD). For example, a set of elements unique to a particular programming or applications group at an installation may be established in a group library with the Operating System rather than be repeatedly introduced into job libraries. Since the group library need exist only once for the multiprogram execution of a series of jobs, both time and storage are conserved. A group library may be registered with the Operating System, either permanently or immediately before the group's machine utilization. The MFD provides a convenient mechanism for registering group libraries for availability to the system.

  Once a group library is established through the LINK statement, the group library remains within the library complex until all the group library has been released by the user. Even if a group library is inactive (no users), it is maintained until released.

- System Library

  The system library is an integral and permanent part of the Operating System and is resident on random access storage. The system library contains standard RB elements, such as mathematical, utility, input/output, and editing routines, which may contribute to construction of a program; standard absolute programs, including system utility routines, transient elements of the Operating System, internally registered job streams, standard production job descriptions, and miscellaneous data file elements.

4.2.5. Output Cooperatives

Output cooperatives are a collection of routines which accept output images from operating tasks and provide for eventual writing on the appropriate output device. An intermediate random access buffer is utilized by the system to serve the double function of accepting output, independent of the writing rate of the output device, and independent of its present allocation.

The primary output cooperatives accept print images from the Operating System or user tasks, and buffer these to random access storage. Each job has a unique output stream of chained random storage modules containing a variable number of print images. The modules are retrieved by the appropriate unit output routine and either printed, transmitted to a remote station, or recorded on magnetic tape.

The secondary output cooperative functions similarly to the primary output cooperative with the exception that card images are accepted from operating tasks and are ultimately processed by a card punch output routine or are recorded on magnetic tape.

The routines which control the input/output devices utilize device level services provided by the system. Devices for input, primary output, and secondary output are defined when the system is generated. Figure 4–5 illustrates output cooperative control.

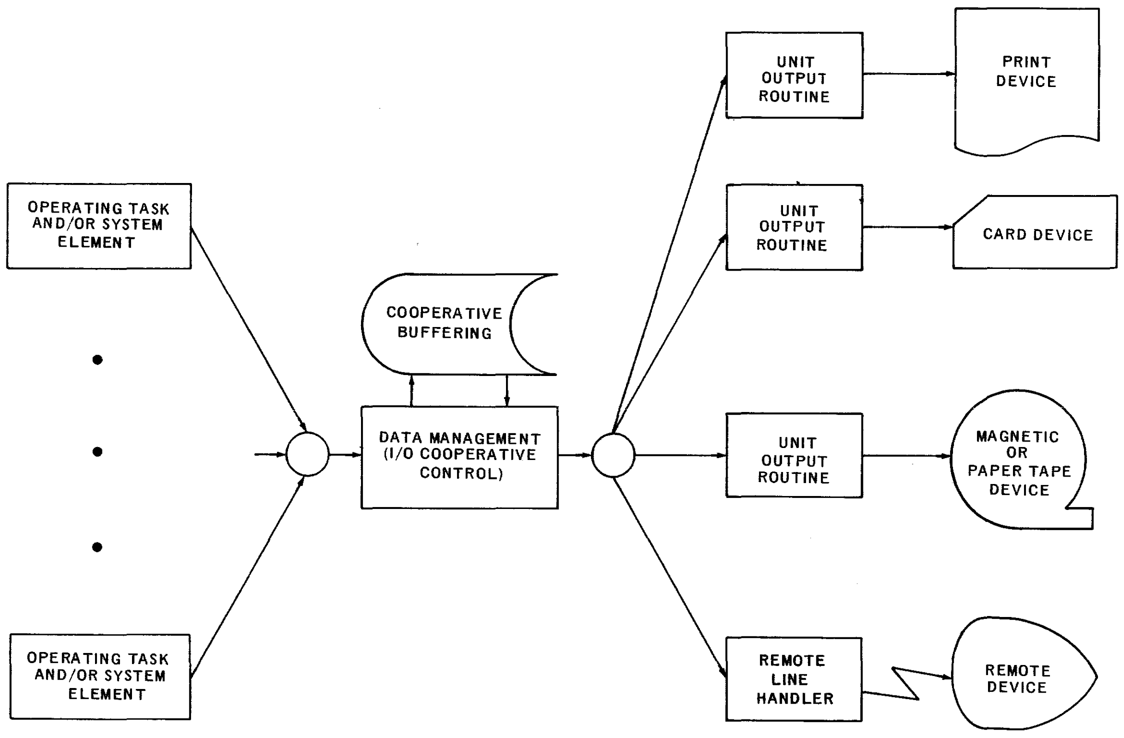

OPERATING TASK AND/OR SYSTEM ELEMENT
OPERATING TASK AND/OR SYSTEM ELEMENT
COOPERATIVE BUFFERING
DATA MANAGEMENT (I/O COOPERATIVE CONTROL)
UNIT OUTPUT ROUTINE
UNIT OUTPUT ROUTINE
UNIT OUTPUT ROUTINE
REMOTE LINE HANDLER
PRINT DEVICE
CARD DEVICE
MAGNETIC OR PAPER TAPE DEVICE
REMOTE DEVICE

Figure 4–5. Output Stream Cooperative Control

4.3. EXECUTIVE SERVICES

Through the executive services, the Operating System furnishes to the user the means for directing the following functions:

Job Execution (Executive Control Language)

Job Communication (Service Control)

Activity Control

Environmental Control

Core Allocation

Console Control

4.3.1. Executive Control Language

The executive control language provides the user with the means for directing the execution of the individual tasks of a job and for relaying operational information concerning a job to the system. The language is open-ended and easily expanded, so that features and functions may be easily added, as dictated by the specific needs of different installations.

The construction of a job deck is performed by the user and may include supplementary cards representing data, source code, or object code.

The basic form of the executive control language is the control statement which is quite simple in format and is amenable to a large number of input devices. Control statements are in card-image format, and are submitted through the primary input stream, or, in some cases, internally as service requests. Each statement consists of a leading character for recognition purposes, a function which categorizes the statement's basic operation, options as desired, and a variable number of specifications. Normally, for card input, the end of a statement is signified by the end o. a card, or by a carriage return or its equivalent for other types of input devices.

The executive control language can be divided into four general categories: organizational control, input/output control, task activation, and systems utility control. The control statements are described in detail in *UNIVAC 494 Real-Time System Operating System Programmers Reference, UP-7504* (current version).

4.3.1.1. Organizational Control Statements

Organizational control statements are used to activate and control a job stream. In general, this category of statements describes the job decks to be activated; supplies informational data to the system, the operational personnel, and the program; or in some manner activates a systems process not constituted as a task. These statements are processed upon their occurrence in the job deck.

Organizational control statements include:

■ JOB	Identifies the job, specifies parameters, and delineates tasks; begins job deck.
■ START	Schedules the execution of a job stream.
■ COR (Correction)	Applies corrections to an absolute element or to memory locations.
■ PRAM (Parameter)	Submits operational parameters to a task at execution time.
■ LOG	Enters accounting information into the systems log.
■ DUMP	Makes diagnostic printouts or postmortem dumps of primary and mass storage areas.
■ MSG	Communicates with and conveys instructions to the console operator.
■ SOURCE	Enters supplementary source images, control statements, or data to the primary input stream.

■ CALL	Schedules nonstandard output cooperative action; overrides normal routing of output.
■ READY	Identifies remote terminal requesting service; precedes job runs from terminal.
■ END	Marks end of primary input for a task.
■ FIN	Indicates termination of input stream, end of tape, end of transmission, and like conditions, from unit record equipment.

4.3.1.2. Input/Output Control Statements

Input/Output control statements assign and release peripheral unit record devices, random access storage, and communication facilities to a task.

Input/Output control statements include:

■ ASG	Assigns an I/O or mass storage device to a task or logical reference in a program.
■ FREE	Releases an assigned facility between job tasks to general facility pool or MFD.
■ SWITCH	Establishes equivalence between file codes for successive tasks using same facilities.
■ MFD	Catalogs, assigns, and releases files with the Master File Directory (MFD).
■ LASG	Assigns a communications line to a task
■ LFREE	Releases a file code and closes input and output queues on a communications line.

4.3.1.3. Task Activation Control Statements

Task activation control statements are primary control statements which describe an absolute program, produced by the Loader, and call for activation or initiation of a routine for execution of the program.

Two forms of task activation statements are recognized by the system:

- System program call statements (see 4.3.1.4) – Activate system processor utility routines.
- GO – Initiates the execution of a user absolute program prepared by the Loader, and is the normal method for activating user-developed programs.

4.3.1.4. System Program Call Statements

System program call statements are task activation statements which call for the execution of a system routine contained in the system library or user job library. Each system routine is selected as a task and, in general, recognizes a secondary control language and/or source code.

System program calls available are as follows:

■ IN, OUT, PRT, LINK, DEL	Structure and access the job, group, and system libraries.
■ TEST	Calls the test system for testing and debugging program areas.
■ LOAD	Calls for the collection and allocation of an executable program from RB elements.
■ UTL	Calls the utility system for distribution or collection of data.
■ ELM	Calls element library maintenance.
■ REX	Calls and activates the UNIVAC 490 REXecutor.
■ REPORT	Calls and activates the UNIVAC 494 Report Writer.
■ SPURT	Calls and activates the UNIVAC 494 SPURT Assembler.
■ ASM	Calls and activates the UNIVAC 494 ASM Assembler.
■ FOR	Calls and activates the FORTRAN Compiler.
■ COB	Calls and activates the COBOL Compiler.

4.3.2. Service Control

Service control is the interface by which an operating task communicates and requests services from the executive routine. An operating task requests service by a sequence of instructions which submits a parameter packet appropriate to the request and generates an interrupt signal to the executive routine.

Since hardware guard mode is enforced against operating tasks, the special Executive Entry instruction (EXRN) is used to submit a request.

The EXRN instruction causes an interrupt and, through its 15-bit field, identifies the function requested. In many cases, operational registers A, Q, and B7 are also used in communication of parameters for the request, and show the status or condition of the system upon completion of the function. The calling sequences are consistent with re-entrant programs since the sequences are restricted to the operational registers.

The routines which perform the requested service are logically executed as an extension of the requesting activity. This extension maintains CPU service at the priority level assigned to the requester and provides for direct logging of CPU time to the requesting task used in the execution of the function. Organization, which furnishes interrupt capability for the extent of the function, is also achieved. Service routines which are not permanently resident will be called and controlled by the executive routine. Some functions are performed by re-entrant routines which service simultaneous requests from several user or systems elements, such as input/output handlers and the segment Loader.

Since service requests are executed as extensions of the calling activity, the caller, under the current activity addendum, is delayed until completion of the request. If the task has been fragmented previously, the task may be eligible for program control under another activity addendum.

Essentially all executive routine functions called upon during execution of a task are service requests such as activity control, contingency control, and input/output. Many of these requests which are related to these functions are explained in detail in other sections of this manual. Service requests, which are contained in this section, are miscellaneous functions that are not covered elsewhere.

4.3.2.1. Activity Control Service Requests

An activity is established by definition of an operating task or by another activity. The function allows a dynamic declaration of parallel parts of a task, thereby achieving a multiprogram, multiactivity environment within the task. Activity control of this type is particularly appropriate for use in real time processing where activities are selected on the basis of the priority of data (transaction) being processed. The batch programs may utilize activities to regain CPU control during input/output waits for instruction executions, to decrease task turnaround time and fully utilize the available CPU time.

Activities are established at execution time through the use of fragmentation service requests by the operating task. Three forms of fragmentation requests are provided by the system: Standard Activity Registration, Queue Process, and Fork/Join.

Once established, all activities may make the same service requests as the task from which they emanate, and share operational identity, primary cooperative streams, facility allocation, logging and accounting with the task.

- Standard Activity Registration

  Standard activity registration is normally used to register and activate an independent program or subprogram When the program is activated, little or no communication or synchronization exists between the requester and the requested activity.

  The standard activity registration service request defines a point of program control within a task which is to be registered with the executive routine for CPU control. Once the service request is executed by the task, the registered activity is eligible for program control.

Optional parameters may specify the data area to be locked within the task, priority, and abnormal index register setting. The activity may voluntarily terminate through use of the RETURN, ERROR, or ABORT service requests. At termination, the activity is deallocated and purged from the system.

- Queue Process Activity Registration

  A queue process activity provides a method for controlling access to an independent subprogram or process. The Queue Process Activity Registration request is generally useful where the subprogram or process is to be performed serially or is not re-entrant because of complexity of code or data such as, tables, files, and buffers.

  Queue processing of an activity is a means for utilizing a task-permanent activity to respond to a series of events. Transactions are accepted and queued by the system, and the activity is executed when a queue entry exists and the activity is dormant. The activity signals completion of a given transaction by return of control through the RETURN service request. The executive routine re-executes the activity if transactions remain on the queue.

  Use of the queue process activity registration function allows the scheduling of events at the time that the events occur. The function is appropriate where no advantage can be gained from registration of concurrent executions by re-entrant code. Two operators are associated with use of this function. The first operator defines the activity; the second supplies the data to be queued and causes activation of the fragment.

- FORK and JOIN

  A FORK service request provides a method for establishing two points of program control, and furnishes the ability for synchronizing a completion point for the two, through the JOIN service request.

  The FORK operator is applicable to general batch processing programs more than to real time transactions, since it provides a higher level interface, and synchronization is on a gross basis. An activity established by a FORK may, in turn, establish other forks which provide additional levels on controlling parallel paths. All FORK activities are considered integral to the requester, and retain the same RIR, PLR, and index register modes of the requester.

  The JOIN operator requests a wait for completion of all parallel activities previously established by the FORK operator as indicated by the activity completion operator, RETURN. A JOIN requests completion of all activities directly emanating from the requesting activity, and when given by the originating task activity, waits on all outstanding forked activities within the task since the original task activity is the base of all forking. A JOIN request by subsequent activities waits on only those activities established either directly by the activity or indirectly by forks from activities which are themselves direct forks from the requester.

The following illustration, Figure 4–6, shows the sequence of three forks executed from base activity A, causing the activation of activities B, C, and D, and the subsequent points of synchronization. The code within activities is described by a_1, a_2, a_3, a_4 and (←) shows the point at which the code is eligible for execution.

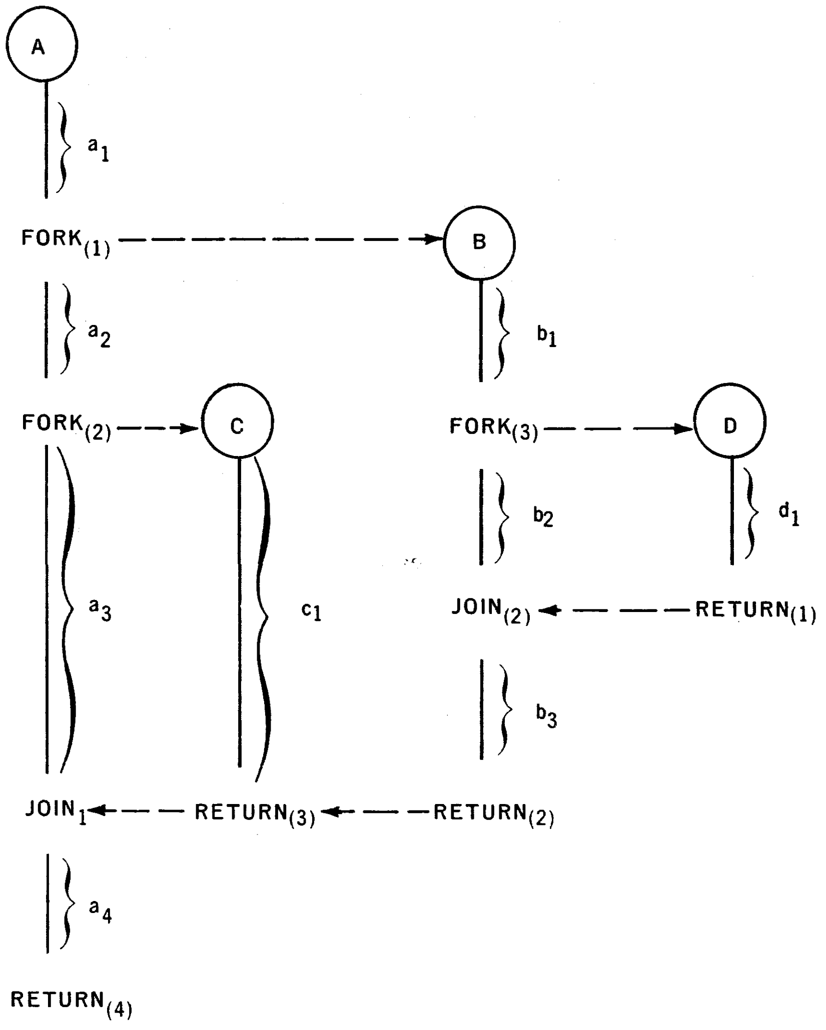


Figure 4–6. Example of Activity FORK Sequence

In the above illustration, coding contained in a_4 is not executed until coding contained in a_1, a_2, and a_3 is completed, and activities B, C, and D have executed a RETURN signal request.

4.3.2.2. Termination Service Requests

Termination service requests are the formal means for termination of an activity, task, or job. At completion of a scheduled operation, or if, for some reason, the operation must be discontinued, the service requests are used to institute an exit or termination procedure and to return control to the executive routine.

Termination service requests include:

- RETURN – Relinquishes program control at the conclusion of an activity.
- ABORT – Causes voluntary release of the CPU by the operating task/activity. The system purges all references to the task, including outstanding I/O and service requests, with the exception of primary and secondary output which are processed in the normal manner to the point of the ABORT request. This entry is an indication of abnormal operation and implies that the entire job is to be terminated.
- ERROR – Same as ABORT, except that only the task and its activities are terminated.
- RETURN1 – Same as RETURN, except that, when given by registered Queue Process Activity, the activity together with all outstanding references is deallocated.

4.3.2.3. Environmental Control Service Requests

Environmental control service requests apply to basically task related functions and may be made by an activity or task during its execution.

Environmental control service requests include:

- Segment Call – Requests loading of a program segment or overlay defined during collection by the Loader (see 4.6.4).
- Subroutine Call – Requests loading of a named absolute library program at a specified location.
- Common Subroutine Linkage – Requests forms of dynamic segmentation which allows concurrent tasks to utilize the same code.
- SEND/RECEIVE – Requests storing and transferring of limited data sets between independent tasks and/or activities within a job.

4.3.2.4. Date and Time Operations

The Operating System provides date and time operations for both internal and external uses. Timing functions for activity control and for activating programs at a specific time of day are also provided. The following service requests are used:

- Time of Day
- Elapsed Central Processor Time
- Date and Time
- Delay

4.3.2.5. Logical Switches

A set of logical switches is maintained by the system for each job entering the system and the switches are referenced by each task within a job deck. A logical switch provides a simple on/off parameter to be conveyed to each task, which can be set externally to the program. The switches A through E are initially set by the JOB control statement, from which point they may be changed or tested dynamically by a task through service requests. The switches may be reset or altered by the GO control statement. The on/off condition of the switches is represented by binary 1 for on and binary 0 for off. The following requests are used:

- Set Off
- Set On
- Test Switches

4.3.2.6. Primary Storage Allocation

The primary storage which is allocated for any one task is always contiguous. While the address range for a program is 32 K, the Relative Index Register (RIR) allows the program to operate in any contiguous area in a 131 K memory.

All primary storage assigned to a task element and extensions are in multiple units of 100_8 words, allowing the Program Lock Register (PLR) to be effective for the total area assigned to one task.

The initial limits for a task program are determined at selection time of the task. The limits are specified in the preamble of the task element and in any optional CORE statements used to extend the initial allocation. The preamble limits of the task code are determined by the Loader at collection time. The optional CORE statements may be collected with the task element or may be contained in the control stream.

Once activated, the task program may dynamically expand and/or contract primary storage assigned to it by a service request to the executive routine. Dynamic expansion requests may cause either compaction of storage area and/or roll out of a low priority task to provide the added area.

The Relative Index Register (RIR) makes it feasible to dynamically relocate program elements. This relocation allows compacting or reordering of programs so that a contiguous area of primary storage can be made available for selection of a task or for expansion of an operating task. Compacting is only performed when necessary, since the operation requires that the task(s) to be moved is temporarily stable. Stability implies that all I/O transfers into the program code, common subroutine, or service requests are completed.

4.3.2.7. Task Storage Extensions/Contractions

Four service requests are provided for the control of additional assignments of contiguous primary storage to the task element. The CORE statement may be submitted at the task selection time to attain optional storage without invoking compaction procedures. The MADD and MREL service requests are provided to allow dynamic expansion and/or contraction of primary storage assigned to the task. The fourth service request is the TCORE operator which allows the operating task to obtain its physical location in memory and the amount of storage assigned to the task.

4.3.3. Console Control

The Operating System has been designed to minimize the extent to which the computer operator must communicate with the system. However, certain operational information must be made known to the operator, while another class of information must be made known to the system by the computer operator.

The console printer is used by the Operating System and by operating activities to inform the computer operator of an event or to solicit a reply from the operator as to the completion of a requested action. Representative operator service requests are included in the following list:

- MOUNT
- DEMOUNT
- CHANGE
- Unsolicited Task Input Messages
- Scheduler Display
- Interlock
- Central Processor Time Overflow
- Output Overflow

4.4. DATA MANAGEMENT SYSTEM

The Data Management System exercises centralized control over all peripheral resources available on the UNIVAC 494: their assignment, usage, and access, as the basis for efficient multiprogram operation. In addition, centralized control of facilities establishes the procedures necessary for providing programmers and operational personnel with the tools necessary for storage, retrieval, and manipulation of the large volume of data and programs involved in utilization of the computing system.

To effectively utilize all hardware resources in the multiprogram environment, the Data Management System performs three major roles: Assignment, File Access, and File Manipulation. One of the major features of the Operating System is the efficient assignment of the system resources, which is performed in such a manner that a relatively simple interface is provided between the user and the physical device while a high degree of device independence is achieved. Device independence permits flexibility in the choice of peripheral devices assigned to the program at execution time without need for changing the program.

References to random access storage or to peripheral devices within a user program are symbolic so that the program may be compiled, collected, entered, and rolled out, independent of assignments. The association of physical device or area with symbolic reference is made when the task programs are set up, without modifying the text of the program.

4.4.1. Data Access Methods

Centralized control over data access provides the coordination necessary for full utilization of the system by concurrent user programs. This control includes three levels of user interface: device control, cooperative control, and file control. The user of either device control or file control acquires assignment of required peripherals or random access storage prior to access. Assignment associates the physical device or area with a unique alphabetic character (file) code as the symbolic reference for each data source contained in the program. The file code, in turn, is presented with each data access either explicitly through device control or implicitly through file control, and establishes the link between the task and device or area.

■ Device Control

The device control level of interface provides for functional control of a particular type of device or random storage area. The user submits parameter packets describing the functions to be performed, and assumes the responsibility for buffer, item, and device strategies.

Packet requests are formatted for execution by common subroutines referred to as input/output handlers. The Random Storage File Handler is required by the system and is a permanent resident. Other device handlers are maintained in the systems library and are called into memory only when an assignment for the type of associated device is made. All executive routine elements utilize this level of data access.

■ Cooperative Control

Cooperative control is inherent in the Operating System and is available for use through direct service requests. Functions performed are requests for primary input images, normally cards, included in the job stream; and submission of print images and card images for processing by system output routines. No assignment is required for utilization of cooperative control. Data requested or conveyed to cooperative control is buffered to random access storage, and is collected and distributed to system-allocated peripheral units, determined at systems generation time.

■ File Control

File control is a group of standard elements providing data handling operations at the block or item level. These elements provide a high level, device-independent interface which manages blocking and buffering while utilizing and augmenting the system facility for storing and retrieving data. File control utilizes device level input/output to perform its functions.

4.4.2. Maintenance Functions

The UNIVAC 494 Operating System contains elements used in the maintenance and manipulation of data files or program libraries, and elements used to perform general utility functions. These elements are provided with the Operating System, but are not an intrinsic part of the system. In general, they are system processors and utility routines activated by control card or service request to perform an explicit function with minimal interface with the executive routine.

Data management maintenance routines fall into three categories: maintenance of program libraries, maintenance of the file directory, and routines used for utility functions.

4.4.3. Assignment of Input/Output Subsystems

The function of the assignment elements is to maintain the status and availability of assignable peripheral subsystems attached to the UNIVAC 494. To perform this function, facility assignment maintains the peripheral and random access storage requirements of all active tasks and elements by responding to their static and dynamic, peripheral, and random access storage assignment requests.

The association of peripheral device or area of random storage to a task is specified by the ASG control statement. The ASG statement contains specifications and options for selecting and initializing a specific device, or a device from a general class of subsystems, dependent upon availability. In addition to describing the desired subsystem, the ASG statement specifies the symbolic link which the operating task uses at access time. The ASG statement is of the normal primary control statement format and is amenable to many types of peripheral subsystems. The variety of input/output devices available for the UNIVAC 494 makes it necessary to describe the ASG statement according to the class of subsystems; for example, random access storage, tape units, unit peripherals, and remote devices. The ASG statement contains a peripheral code indicating the unit or type to be assigned, a file code by which the unit is to be referenced, assignment specifications appropriate to the type of device being assigned, and options indicating initialization parameters.

4.4.3.1. Peripheral Code

The peripheral code is the mnemonic name of the requested peripheral unit or random storage assignment (for example, TAPE, UN6C, F880, and such other devices). The permissible mnemonics for this field are determined by the names applied by the installation at systems generation time to represent a particular configuration and order of assignment. The choice of mnemonics is completely open-ended. Several names may describe a single peripheral subsystem or a unique name may specify a particular unit on a specific peripheral subsystem. The assignment routines provide a mapping function of mnemonic names through which a specific name may contain a number of alternate choices for assignment in a preset order.

A request for a tape file (TAPE) limits the choice to magnetic tape units. Through the specification of additional mnemonic names, the generality of assignment may be further limited to a specific type of tape unit (UNISERVO VI C or UNISERVO VIII C); to request a unit from a specific channel (UN6CA or UN6CB); or, to request a specific unit on a channel (SPT). The assignment mapping element could be adjusted at systems generation time to allow the use of alternates to satisfy the request. Alternates could specify that units from the UN8C and/or UN6CB group can be assigned if units in the UN6CA group were not available.

In addition to specifying a physical device, peripheral code also implies the device handler which is used in processing I/O requests by the operating task to the assignment. Input/output handlers, for infrequently used devices or nonstandard I/O processing, are entered into primary storage and initialized only when the device is assigned or when nonstandard processing is to take place. The I/O handlers provide the following features:

- The ability to conserve primary storage during periods when a particular I/O handler is not required because of lack of assignment.

- The ability to utilize nonstandard I/O handlers for the shared usage of the subsystem or a particular unit on the subsystem. That is, two or more tapes or random access storage handlers may utilize a physical channel concurrently. This is particularly advantageous to the installation which requires special or additional processing for a class of I/O access (for example, double queuing of file updates and audit trails). Use of a nonstandard I/O handler also provides for the integrated subsystem test to operate concurrently with production tasks, with little or no impact upon their functional requirements.

4.4.3.2. File Code

File code is the symbolic bridge by which an operating task accesses or references the physical device or area assigned to it. Once the choice is made through the ASG statement, the user conveys the file code with each device level I/O request or other reference to the assigned peripheral device or area of random access storage. This establishes a mapping arrangement whereby the task code does not require modification at execution time for I/O access. File code also affords the user a procedure through which a task cannot inadvertently access a unit or area of random access storage which has not been previously equated to it through the ASG statement.

File codes are established at the task level. That is, each task currently operating within the system has a complete set of file codes eligible for its use. Hence, no programming conventions are required by the user for specifying file codes, other than those conventions required for intratask control of assignments. However, it must be noted that all activities emanating from the task have shared usage of all facilities assigned to the task.

Each task addendum is provided with a basic set of 25 file codes to which the user may assign a peripheral device, area of random access storage, or communication lines. These are symbolically referenced by an alphabetic character from the set A through Y. In cases where 25 file codes are inadequate for the task, a user may specify that a designated file code be fragmented into an additional set of 26 file codes which have the same characteristics as the original set. For example, if the file code B were fragmented, the new set would be referred to as BA, BB. . .BZ.

4.4.3.3. Random Access Storage Assignment

A request for the assignment of random access storage can be satisfied on any one of several subsystems available in the system. These subsystems include the magnetic drums FH-880, FH-432, FH-1782, and the FASTRAND II and FASTRAND III. Each assignment request is satisfied by a multiple of the block size used to map the subsystem. The block size for any one type of subsystem is a practical minimum number of words consistent with the characteristics of the device. On FASTRAND, for example, the block size is a multiple of a track.

The physical description of an assigned random access file may consist of discrete noncontiguous areas across available blocks of random access storage. The noncontiguous areas may cross drums, channels, and even types of drum subsystems. As an example, a file may be composed of blocks from FH-880 drum and FASTRAND. This may occur either through planned partitioning of a file or as the result of successive extensions to a sequential or random file. However, to the operating task, all random access storage assigned to a file code is word-addressable and logically continuous.

File routines and the Random Storage File Handler provide the user with a word-addressable interface for all types of random access storage through the use of file code and logical increment. The logical increment is essentially a pseudo drum address relative to the base of the assigned area. At execution time, the task code submits the logical increment of the file segment being accessed along with the desired I/O function. The Random Storage File Handler then maps the logical increment to the physical address to perform the I/O function.

To summarize, random access assignment and data access elements provide the user with the following important features necessary in a multiprogram environment:

- File Protection

  The ability to protect random storage files assigned to a task from inadvertent destruction by concurrently operating tasks. At the same time, through use of the file directory, two or more concurrently operating tasks and/or activities may share usage of common files.

- File Expansion

  Random storage files may be dynamically expanded and/or contracted without the need for compaction to develop a physically contiguous area.

- Device Independence

  This logical independence allows the program to be assigned word-addressable drum or sector-addressable FASTRAND without impact upon code logic.

- Partitioned Files

  Portions of a particular file which are frequently accessed may be assigned to magnetic drum, whereas portions of the file which are infrequently accessed may be assigned to FASTRAND, thereby minimizing task turnaround time within the constraints of the configuration.

4.4.3.4. UNISERVO Tape Assignment

A request for the assignment of a tape is satisfied by the compatible magnetic tape units, UNISERVO VI C and UNISERVO VIII C. The request for tape unit assignment is submitted by the ASG statement. The ASG statement contains the necessary specifications required to ready the unit for operation. The specifications include: parity and density in which the unit is to be used, file code of assignment, and operator mounting instructions.

A feature of the magnetic tape handler is automatic block numbering. Each block of data recorded on a magnetic tape, load point being 0, is affixed with a sequential binary number as its first word. The block numbering is performed without assistance from the user task. No allowance for increased buffer specification is required to accommodate the block number. The magnetic tape handler effects the reading and writing of each tape through scatter read and gather write techniques to separate block number from data. On each read request, the block number is returned to the requesting activity in its binary form through the Q register.

The block numbering option is considered the normal mode of operation for all system processors and user programs operating under control of the system. However, when tapes recorded on other computers, or going to other computers, are processed by routines under control of the executive routine, the block numbering option may be disabled. This is because the block number is the first physical word on each tape block. Otherwise, the source and/or object computer would have to be cognizant of the block number feature and compensate for it through programming.

Block numbering provides the ability to verify block continuity on a tape, thereby eliminating the possibility of undetected data loss during process functions. In addition, block numbering simplifies error recovery and provides for a more efficient checkpoint and restart procedure.

■ Translate Feature

The Translate feature is optional hardware on the UNISERVO VI C and VIII C control units. The feature provides for hardware translation of complete data blocks transferred from tape to primary storage (read operations) and/or primary storage to tape (write operations). The translation is from one six-bit code set to another six-bit code set, normally Fieldata to/from Binary Coded Decimal (BCD).

The Translate option is activated through an option on the ASG statement. Once assignment is set for translation, all data directed to/from the assigned unit undergoes translation.

■ 9-Track Format Feature

The optional 9-Track Format feature for the UNISERVO VI C and VIII C tape units provides the ability to read and/or write tapes prepared on or for the UNIVAC 9200/9300 computers or the IBM 2400 Series Magnetic Tape Subsystem. The main difference between the normal 7-Track Format and the 9-Track feature is the method of recording the tape frame. The 9-Track feature provides for nine bits in each tape frame while the 7-Track unit writes seven bits in each frame. The nine bits in each frame consist of eight data bits plus a parity bit which provides odd lateral parity for the frame.

4.4.4. Master File Directory

The Master File Directory (MFD) provides for registration of files which transcend jobs and for acquisition of these files by subsequent jobs. The directory itself and most registered files are stored on random access storage. Registered files may also be contained on magnetic tape.

The MFD contains a physical description of each file cataloged. In the case of random access storage files, several physically separate areas may constitute the file. When a file is acquired from the directory, the description is read to memory for reference by device I/O control during the period that the file is active.

Files are entered in the MFD by a user number and file number. The definition of permissible users and the number of files to be accommodated for each user is supplied by the installation. Each user may invoke a keyword protection mechanism to establish privacy and access constraints for these files. Acquisition of a file then requires as parameters the user number, the file number, and the keyword. Cataloging and acquisition of files in the MFD is effected by executive routine service requests. For magnetic tape files, the directory identifies the reels of the file and the recording characteristics. All files, when cataloged, are associated with a retention date to allow automatic discard or purge.

Access to a specific file may request logical lockout to prevent conflicting use by another job in the multiprogram operation. Alternately, by logical lockout, the file can be opened by coexistent programs which protect against conflicting use of individual records or blocks. By this means, coexistent routines may reference, and even update, a file without mutual interference.

Special utility routines are provided to perform installation management services on the MFD, including listing, purging, compacting, deleting, copying, renaming, and other functions.

4.4.5. Device Control

Device control is a basic access method which is applicable to both randomly and sequentially arranged data. Through the device control method, physical characteristics of specific peripheral equipment may be recognized for more efficient utilization than is generally possible with higher level access methods. In addition, a degree of device-independence may be maintained to the extent determined by the programmer. Detailed knowledge of specific I/O devices by the programmer is not required.

Data access using the device control method is accomplished by specifying macro instructions, or macros, within the object program. These macros cause the generation of a parameter list and a function code which completely define the I/O requests. The parameter list may be changed dynamically by the user to effect a variation in blocking, buffering, and other operations.

The I/O operation is scheduled by executive routine at the time that the request is received from the user program. The parameter list is checked for validity by the system and, if all is found to be in order, the operation is performed. At the completion of the operation, control is returned to the requester with an indication of the success or failure of the operation. Supplementary information may also be available at this time, depending upon the function performed.

The system attempts to recover from all error conditions encountered during the performance of an I/O request. The data block is reread (or rewritten) a number of times in an attempt to perform the operation without error. Only when all system recovery attempts fail to produce a normal response from the device is an error condition reported to the requester. The user may then take any action desired (that is, attempt his own recovery, use the data block in spite of the error, ignore the block, abort, or perform some other function).

4.4.5.1. Data Format Considerations

The device control method of data access deals with physical data blocks rather than with individual logical records. The data blocks may consist of any number of fixed- or variable-size records. Initial data block size is specified by the user through the macro instructions; however, block size may be varied during operation. For output operations, the block size represents the number of words to be transferred to the output device. For input operations, the block size represents the maximum number of words to be accepted from the input device. The system informs the user of the number of words transferred so that short blocks may be successfully processed.

Reference may be made to the peripheral unit only by file code, never by channel and unit. Use of the file code procedure makes it impossible for a task to reference a peripheral which has not been validly assigned to the task through the Operating System. Random access storage is referenced through a file code and also through a logical address. The file code refers to an area (or areas) of storage that has been assigned to the task, and the logical address refers to the position within the assigned area. Thus, two different tasks operating in a multiprogrammed environment may be using the same file code and logical address, and yet be referencing two completely separate and distinct areas.

4.4.5.2. Device Control Macros

The device control macros are source statements placed within the program by the user. From the information contained in the macro statement, a macro call packet, comprising a parameter list and the necessary control instructions, is generated to perform the I/O operation specified. The general format of the macro statement comprises an operator and an operand specification list. The operator defines the function to be performed, and the specification list contains the necessary information for performing the function, such as file code, buffer location and size, logical address, and search identifier.

4.4.5.3. Status Codes

At the completion of each I/O operation, control is returned to the requester at the line of coding following the Executive Entry (EXRN) instruction of the macro call packet. At this time, the results of the operation are presented in the form of a status code in the A register. The number of words transferred is also indicated in the A register. The Q register contains supplementary information dependent upon the type of peripheral referenced.

If the I/O operation is not completed successfully, or if an abnormal condition exists at the end of the operation, an abnormal status code is returned to the requester through the A register.

For magnetic tape read functions, an abnormal frame count is not considered an error condition; the magnitude of the frame count error is always returned in the Q register. The number of words transferred, as contained in the A register, includes the partial word.

4.4.5.4. Random Access Storage Macros

The random access macros may be used for I/O operations on the magnetic drums, FH-432, FH-1782, FH-880, and FASTRAND II and FASTRAND III. The system provides for word addressability of random files, although the user may orient himself to the 33-word sector size for write operations to minimize latency if the file is allocated to FASTRAND unit. When I/O functions are initiated, the data transfers continue until the request is fulfilled according to the given specifications or until an unrecoverable error occurs. The activity is terminated and a status code, with other pertinent information as applicable, is returned to the requester.

4.4.5.5. Magnetic Tape Macros

The magnetic tape macros may be used to effect I/O operations on the UNISERVO VI C/UNISERVO VIII C Subsystems. All data transfers continue until the requested activity is completed according to the given specification, or until an unrecoverable error occurs. At completion, appropriate status information is returned to the requester.

4.4.5.6. Unit Record Macros

Device level control of unit record equipment may be exercised by the user, provided that the peripheral has been duly assigned to the user by the system. Assignment can be made if the unit is not currently being used for cooperative input and/or output.

For the purpose of device control, unit record equipment is classified on the basis of data rather than peripheral type; that is, card input may be specified and the peripheral assigned may be either an online UNIVAC 1004 or a Punched Card Subsystem, whichever is available. The classifications made are card, printer, and paper tape. I/O operations are completed according to specifications or carried through until an unrecoverable error occurs. At termination of the activity, the appropriate status information is returned to the requester.

- Punched Card Operations

  Punched card operations may be effected through the Punched Card Subsystem or through the online UNIVAC 1004 System. Available card processors accommodate 80-column cards which may be read and punched in either translate mode (card code to Fieldata) or in column binary mode.

- Printer Operations

  Print requests may be directed to either a High Speed Printer Subsystem or to an online UNIVAC 1004 System. The system routines handling the UNIVAC 1004 printer have been designed to duplicate the functions of the High Speed Printer Subsystem. Vertical form control is specified on the ASG statement in the form of a top margin, bottom margin, and the number of printable lines per page. This information is used to exercise basic form control during subsequent print applications.

- Paper Tape Operations

  Paper tape read and punch requests may be directed either to a standard Paper Tape Subsystem or to an online UNIVAC 1004 containing the appropriate paper tape equipment.

  Any code (5, 6, 7, or 8 Level) may be read or punched by the user with optional parity.

4.4.6. Cooperative Control

Cooperative control is a collection of systems elements which retrieve and coordinate all scheduling information in the form of control streams, and which submit accounting and actions taken by the executive routine as the result of processing schedule parameters.

The cooperative control mechanism, because of its scheduling role, is inherent to the system and is responsible for controlling three data streams defined as follows:

- Primary Input

  Primary input is used to contain system schedule parameters, limited user data, source code to system compilers and assemblers, program parameters, and other information.

- Primary Output

  The primary output stream is a series of print images containing information pertaining to each job. The system uses the primary output stream for printing copies of such information as assembler and compiler output, accounting, control cards, test mode dumps and traces, and limited data sets. This method of printing is also available to the user. An image submitted to primary output is placed in the stream as it occurs.

  Full words of trailing spaces are deleted from the image by the system. The various unit record routines create a complete print line by adding the necessary number of trailing spaces to the image. The user need not specify a full print line when submitting an image to the primary output stream; the remainder of the line will be space filled by the unit record routine.

  An image may be submitted to the primary output stream through a macro call.

- Secondary Output

  The secondary output stream is an optional stream which may be employed for user data, assembler/compiler output, and other library output. An image submitted to secondary output is placed in the stream in the same manner as for primary output.

  Full words of trailing spaces are deleted from the image by the system. The various unit record routines create a complete image, if required, by adding the necessary number of trailing spaces. The user need not submit a complete image to the secondary output stream; the remainder of the image is space-filled by the system when, and if, required.

An image may be submitted to the secondary output stream through a macro call.

The cooperative mechanism is composed of input unit record routines, which are responsible for accepting primary input streams from systems input devices; input/output cooperative control, which performs staging for all three streams; and output unit record routines, which are responsible for submitting primary and/or secondary output streams to designated system devices. Figure 4–3 and 4–5 depict the relationship and interaction between these three sets of elements.

4.4.6.1. Input Unit Record Routine

The input unit record elements are individual routines which accept the primary input streams from their assigned devices and submit the I/O streams to cooperative control for staging. Any number of type of unit record routines may concurrently operate under control of the executive routine, allowing multiple streams to enter the system. Each unit record routine scans the input stream to identify job descriptions which are, in turn, queued for selection and activation.

Access to primary input data is by service request through I/O cooperative control from the active task. Supplementary streams may be introduced from specified auxiliary sources. The supplementary streams may be used to merge and correct source code and may be used also to enter supplementary control statements for job descriptions which are resident to the system.

A standard set of unit record routines is provided with the system. However, because of their modularity and relatively simple interface, any device is an eligible candidate as a unit record device.

4.4.6.2. Input/Output Cooperative Control

Input/Output cooperative control is a basic system element which is responsible for staging of I/O streams to and from random storage as presented or requested by unit record routines. In performing this function, the control element must recognize and control overflow conditions which may occur. The second function of I/O cooperative control is to process service requests for operating tasks or systems elements for primary input, or for submission of primary or secondary output.

Cooperative control utilizes random access storage as a staging area for the streams, providing the system and user the following advantages and features:

- The staging of low speed input/output data to random storage to balance intermittent system utilization with the slow rate of peripheral devices. Staging allows the device to operate at full capacity within the controlled constraints of the staging area. The buffering to mass storage permits the parallel utilization of low speed devices by operating tasks in the multiprogram environment.

- Provision of a consistent mechanism to compilers and/or assemblers, system processors, and user programs, which is independent of device characteristics, for obtaining and submitting data. The mechanism feature purges system elements of redundant code required to assign, recognize, and handle varied devices.

- Maintenance of the staging area cooperative library is performed by cooperative control, allowing multiple streams of primary and secondary data streams to utilize the pooled library. Data for any one stream is linked by chaining techniques to the job. Cooperative control expands and contracts the cooperative library as required to maintain the system. In addition to expansion of random storage, cooperative control invokes temporary suspension to control overflow conditions.

  The user may obtain card image input to his program through the cooperative control mechanism. This data must be contained in the primary input stream for his job and located at the correct point within the stream. For example, the images in the control stream follow the control card which starts the user program (for example, the GO card).

  The end of the user's image set may be determined by a unique image which the user recognizes as a signal to stop requesting input, or the user may continue requesting images until an end of file or end of input status is received as the result of detecting an executive control statement.

4.4.6.3. Method of Operation

Primary input is entered into the system by the input unit record routines from a variety of devices, including 80-column cards, paper tape, magnetic tape, and random access storage (prestored job streams).

The appropriate unit record routine is activated in response to a keyboard entry by the computer operator.

The input unit record routine begins reading images from its assigned facility. When a JOB statement is encountered, a task addendum for the job is set up. Subsequent images for this job are then staged to random access storage by the cooperative control mechanism and are linked to the job through the task addendum. When another JOB statement is encountered, the previous input stream is closed off and another task addendum is created for the new job. This process continues until a FIN statement is reached. At this time, the input stream is closed off, the unit record routine is terminated, and its facility is released.

As soon as a task addendum has been created for a job, the job is eligible for selection although its complete job stream has not as yet entered the system. When the job has been selected, the system then performs the functions specified by the control images in the order determined by the sequential arrangement of the images. At this time, the primary (and possibly secondary) output stream is initiated.

The primary output images generated for a job are staged to random access storage by the I/O cooperative and are linked through the job's task addendum. Any secondary output generated by the job is staged and linked in the same manner as primary output.

Primary (and secondary) output images are allowed to accumulate on random access storage until one of two contingencies occurs, as described below. At this point, the appropriate output unit record routine is activated to retrieve the output stream by cooperative control and to output the images to the chosen device (such as, printer, magnetic tape, paper tape, or punched cards). When the output unit record routine senses that the end of an output stream has been reached, a search is made to find another output stream which is ready to be processed. If another stream for this unit record routine is not ready, the routine is deallocated and its facility is released.

The two contingencies which trigger the activation of an output unit record routine are:

- The complete job has been terminated.
- A preset output threshold has been exceeded. This threshold is an installation parameter.

4.5. REMOTE DEVICE CONTROL

The Operating System provides a flexible environment for activating and controlling the flow of remote communications data to and from the UNIVAC 494. This environment is structured so that tasks may utilize remote communication facilities concurrently in the multiprogram system.

- Scheduling Elements

  Job control streams originating from a remote site are accepted and processed in the same manner as streams originating from on-site equipment. Subsequent primary and secondary output streams from the scheduled job are automatically transmitted to the remote user, providing the remote site user with the same scheduling abilities as that provided to an on-site user.

- Batch Programs

  Regular batch programs may utilize one or more remote devices for input/output. This is provided by a high level interface which permits the program to use simple acquire, read, and write service requests, with the executive routine performing required buffering, conversion, and transmission of data.

- Online Application Programs

  The system provides an interface through which remote data based control programs can operate without impact on other remote application programs or on other users of remote site data. The type of application tasks assumed here include message switching, transaction control, inventory control, and airline reservations requiring a control program for the loading, activation, and termination of a small data-dependent worker routine (where the called worker program is dependent upon the content of the message).

4.5.1. Device Control Elements

The Operating System provides a remote communications interface which satisfies all of the needs of the various program types. Essentially the interface is composed of Externally Specified Index (ESI) channel control, remote line handlers, the Communications Director, and Remote Facility Assignment. (See Figure 4–7 which illustrates the logical flow of messages to and from the system through the various elements.)

4.5.1.1. Externally Specified Index (ESI) Channel Control

The Externally Specified Index (ESI) channel control is composed of the basic executive elements which control the physical hardware channel. The elements are responsible for sending functions to the communication subsystem as directed by a remote line handler; for allocating and switching buffers when an interrupt occurs; and for activating remote line handlers on predetermined interrupt.

4.5.1.2. Remote Line Handlers

The remote line handlers operate as the interface between the ESI channel control and the user task or Communications Director. Each handler is responsible for directing hardware control of communication lines, and for accepting and transmitting resultant data. Remote line handlers are written to control a particular type of remote unit and, in general, perform the following functions in one form or another:

– Initiate input transmission through polling techniques or by establishing an input data buffer and for acknowledging receipt of input data from the remote device.

– Perform, or request the performance of, required process functions to convert, pack into message staging buffers, and detect end of message (EOM) on all incoming communication data.

– Submit messages or message segments to the Communications Director or user task.

– Accept from the Communications Director or user, task messages for output transmission to a remote site. The handler unpacks and converts data contained in staging buffer to communication buffers, and submits communication buffers to ESI channel control for transmission.

Handlers will be supplied for remote UNIVAC 1004, DCT 2000, UNIVAC 9200/9300, and UNISCOPE 300. User handlers can be written for special purpose devices or to control remote devices in an installation prescribed manner. Remote handlers can be written as re-entrant or not re-entrant, depending upon whether they are registered with multiple activity addendums.

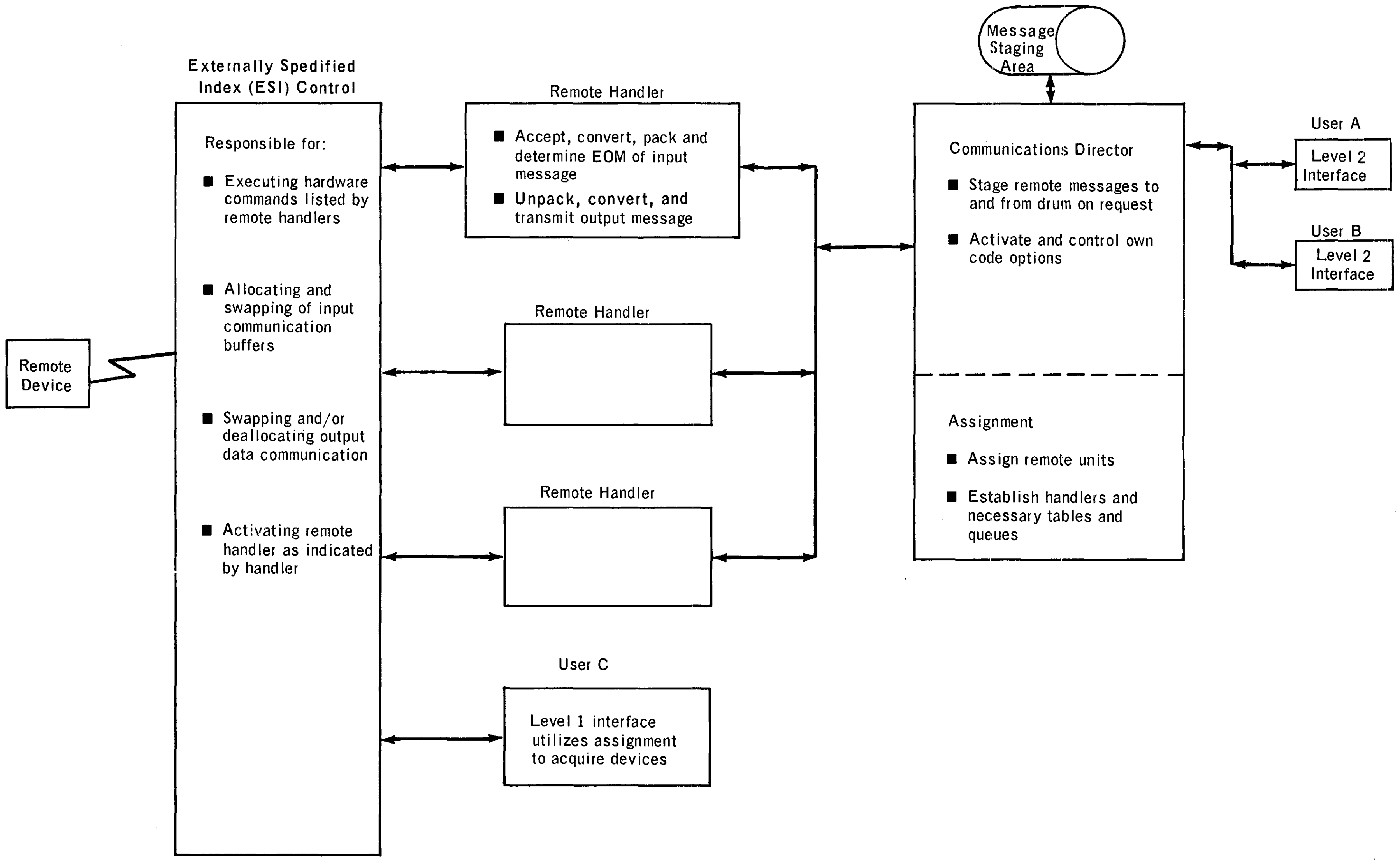

Externally Spedified Index (ESI) Control
Responsible for:
■ Executing hardware commands listed by remote handlers
■ Allocating and swapping of input communication buffers
■ Swapping and/or deallocating output data communication
■ Activating remote handler as indicated by handler
Remote Device
Remote Handler
■ Accept, convert, pack and determine EOM of input message
■ Unpack, convert, and transmit output message
Remote Handler
Remote Handler
User C
Level 1 interface utilizes assignment to acquire devices
Message Staging Area
Communications Director
■ Stage remote messages to and from drum on request
■ Activate and control own code options
Assignment
■ Assign remote units
■ Establish handlers and necessary tables and queues
User A
Level 2 Interface
User B
Level 2 Interface

Figure 4–7. Remote Device Control Elements

4.5.1.3. Communications Director

The Communications Director provides a high level interface between multiple user tasks and assigned remote devices. The functions of the Communications Director are as follows:

- To accept messages from remote line handlers, stage them to random access storage, and, on a READM service request, to transfer messages to a user task from a communication line assigned to the task. Own code options are available as data is transferred from handler to random access storage and from random access storage to a user task.

- To accept messages from the user task on WRITEM service requests, stage them to random access storage, and activate and/or transfer to remote line handler messages which are transmitted across communication lines assigned to a task. Own code options are available as messages flow from user to random access storage and from random access storage to remote line handler.

4.5.1.4. Remote Facility Assignment

Remote facility assignment (see 4.5.3) is responsible for maintaining the status of a user task, and for assignment of remote devices to the Communication Terminal Module (CTM). Assignment is performed through the normal assignment control statements and acquire service requests. The assignment statement dedicates the physical CTM, forms applicable tables used in the interface, establishes the link between the line handler and the CTM, and establishes output own code options. The acquire service request activates the handler, and establishes input own code options and input queues.

4.5.2. Levels of Interface

The Operating System provides the user task with two interface points in the control of his remote environment as shown in Figure 4–7. For ease of description in subsequent paragraphs of this section, the interface points are referred to as Level 1 (interface at ESI Control) and Level 2 (interface at the Communications Director).

Independent tasks operating under control of the executive routine may interface at either of the points concurrently when remote facility assignment routines and table structure are utilized.

- Level 1 Interface

  The Level 1 interface allows the task to interface directly with ESI channel control. The interface assures the user task of a predictable elapsed time period between the time when data enters the machine from communication lines and when the task has control of the data. The user task has complete control of the hardware by commands, which are given to ESI channel control which performs the actual I/O hardware instructions as directed by the task.

The Level 1 user task performs all editing, translating, packing, unpacking, and staging of messages to random access storage. The task is also responsible for forming poll output messages, monitoring input poll replies, and effecting dial connections. Any special control requirements of the various devices must be recognized and provided for in the task program. In installations where both Level 1 users and Level 2 users coexist, the Level 1 user is responsible for obtaining communications from Remote Facility Assignment and utilizing the standard CTM control blocks.

The Level 1 interface essentially fulfills the needs of specialized communication tasks such as real time programs with time-critical response constraints, which cannot be guaranteed through normal interface.

- Level 2 Interface

  The Level 2 interface allows the task program to utilize simple READ/WRITE macros to handle communication traffic. The system handles the details of message buffering, translating, packing, unpacking, polling, and establishing remote connections, as needed.

  The user may include his own remote line handler for special units or networks and still use the structure of the Level 2 interface. The user may also include optional own code routines, which are activated by the system as messages that are written and/or read from random access storage.

  The flow of data through the Level 2 interface is illustrated in Figure 4–8.

4.5.3. Remote Facility Assignment

The remote facility assignment elements perform the necessary functions for assigning remote units to a task and for forming the tables and linkages required by the task to access its assigned units. Remote facility assignment functions are:

- To maintain status and availability of remote units and/or lines. Remote facility assignment maintains a map of all units eligible for direct access to the UNIVAC 494, together will all dial type lines. As units or lines are assigned to tasks or marked as inoperable, the map is changed to reflect their current status. The user is provided with the ability to load and activate multiple tasks which utilize remote devices, with the executive routine applying the same task selection rules in regard to remote units as those provided for on-site peripherals.

- To assign remote units or lines to a task by CTM. Remote facility assignment dedicates the CTM's and establishes necessary linkage to the assigned line prior to activation or during initialization of the requesting task.

- To load into core and register the remote line handler required for controlling the assigned CTM.

- For users of Level 2 interface, remote facility assignment establishes random storage staging queues to contain messages prior to processing (input) or prior to transmission (output); and loads and establishes linkages for user own code routines to edit or divert messages prior to processing or transmission.

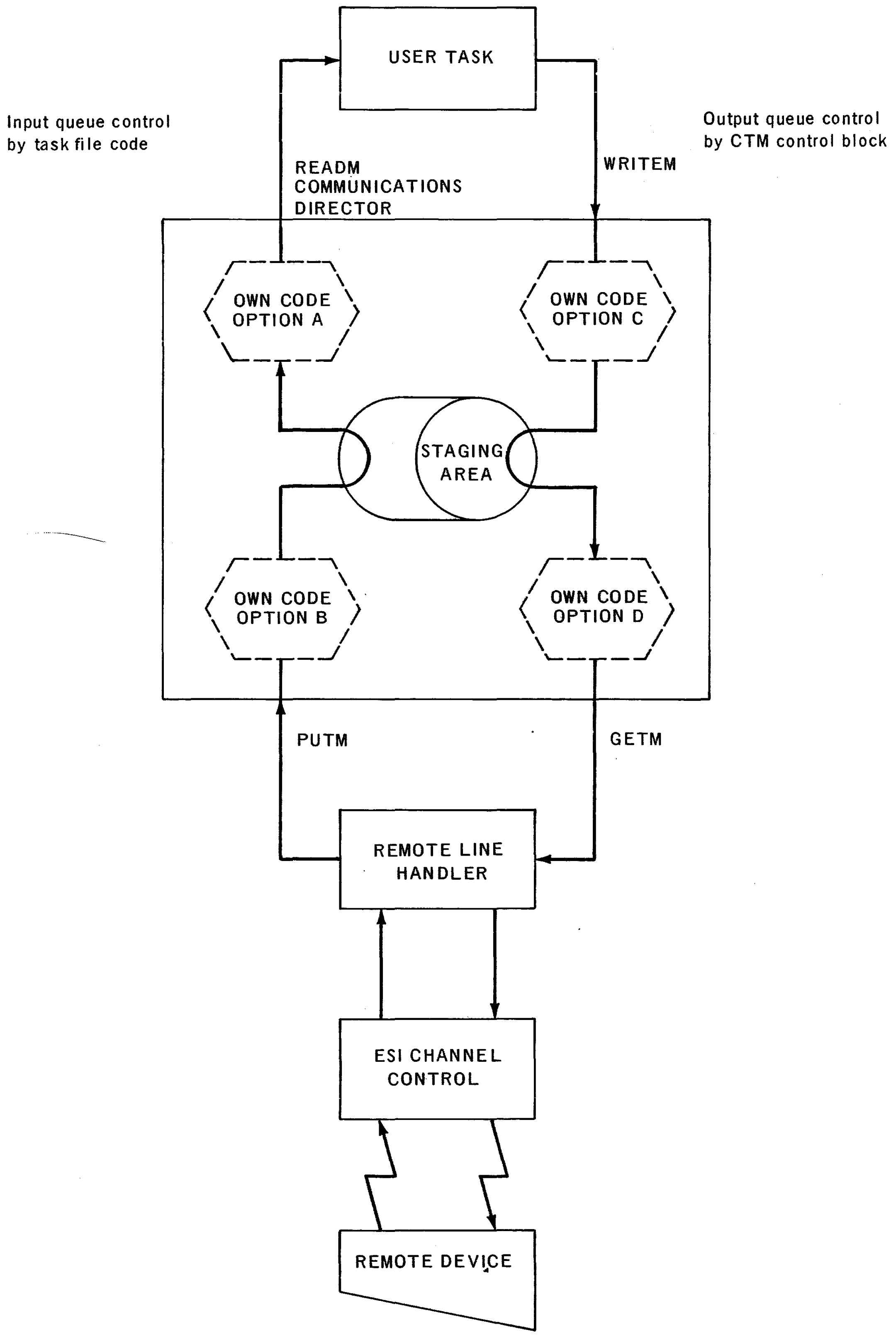


Figure 4–8. Direction of Message Flow

In general, the user is responsible for performing the following functions required to ready a unit or a line for access:

- Systems Generation Time

  Participation in forming the remote facility map used to list all units eligible for access by the system. Developed and/or included in the systems library are all remote line handlers and own code routines required by the installation.

- Task Selection Time

  Submission of LASG control statements as required to assign CTM's, establish remote line handlers, and specify optional output own code routines.

- Task Initialization

  Submission of the LACQ service request by users of Level 2 interface, during the initialization phase, to activate the handler, establish the input queue, and specify optional input own code routines.

- Task Termination

  Release of the line by the LFREE control statement for use by the system or other tasks when the user task has no further need for the CTM.

The following service requests and control statements are provided by remote facility assignment:

- LASG (Line Assign) Statement
- LACQ (Line Acquire) Service Request
- LFREE (Line Free) Statement
- CTMFREE Service Request
- Remote Facility Update
- LUP (Line Up) Service Request
- LDOWN (Line Down) Service Request

4.5.4. Remote Data Access Service Requests

The following service requests are provided for transferring messages to and from the UNIVAC 494. Service requests included are those used by the task program, remote line handler, and own code routines. See Figure 4–8, which illustrates the direction of message flow upon submission of the various service requests executed by the task program and remote line handler.

- Read Message

  The read message macros request that the next message contained in the specified input queue be transferred from random access storage to the task program. The own code routine, if specified by the LACQ request, is automatically activated upon completion of message transfer from random access storage to primary storage, but prior to return of program control to the requesting task activity.

  Three read service requests are provided by the executive routine:

  - READM Read
  - READMW Read and Wait
  - LOOKM Look Ahead

- Write Message

  The write message macro requests the transfer of a message from the primary storage buffer of the task to the indicated random access storage queue. The random access message queue is associated with the CTM control block. The output own code routine, if specified on the LASG statement, is activated prior to the transfer of the message to the queue. Once the message transfer to the output queue is completed, the message is eligible for retrieval and transmission by the remote line handler.

- Get Message

  The get message macros request the next message contained in the specified output queue to be transferred from random access storage to the remote line handler. The own code routine, if specified by the LASG statement, is automatically activated upon completion of the message transfer from random access storage to primary storage, but prior to return of control to the handler.

  Two Get service requests are provided by the executive routine:

  - GETM Get
  - GETMW Get and Wait

- Put Message

  The put message macro requests the transfer of a message or message segment from the specified handler's primary storage buffer to the indicated random access queue. The random access message queue is associated with the CTM control block. The own code option, if specified on the LACQ request, is activated prior to the transfer of the message to the queue. Once a complete message is transferred to the queue, the message is eligible for retrieval and transfer to the worker task.

4.6. PROGRAM DEVELOPMENT

The Program Development System encompasses a variety of programs, utilities, and procedures to provide the programmer and operational personnel with the tools necessary for the development, checkout, and execution of programs. A variety of of programming languages, including FORTRAN, COBOL, 494 SPURT, and 494 ASM, are offered to provide the user a choice of the language best suited to programming his specific problem.

To enhance the modularity of the Operating System, a standard set of routines is provided; these routines interface the language processors with the executive routine. Standardization provides the following features not otherwise obtainable:

- The ability to form programs as a combination of elements produced by various language processors such as, 494 SPURT, COBOL, and FORTRAN.

- The ability to incorporate additional systems or language processors into the Operating System with little or no change to the system software.

- The elimination of redundant elements used in the control of individual translators, thereby simplifying usage and eliminating many idiosyncrasies inherent to individual compiler or assembler control elements.

- Enhancement of installation efficiency by accommodating changes in machine configuration and/or operating procedures without direct impact on user programs. Changes in one user program which impact on other user programs are similarly minimized.

The elements of the Program Development System are summarized below:

- The Source Routine

  The Source Routine provides the capability for generating and/or updating source elements.

- The Language Processors

  Language Processors are system components which translate programming languages into machine usable form. From source language statements, the Language Processors produce intermediate output codes, or relative binary (RB) elements, which may be collected and allocated with other program elements by the Loader prior to execution, thus forming executable absolute or relative load programs.

- The Loader

  The Loader is a systems routine which collects, allocates, and links the RB output of the various language processors into an executable program.

- The Program Library Editor

  The Program Library Editor (PLE) is a collection of file maintenance routines activated by control statements or service requests and is responsible for maintenance and manipulation of the user job library. The editor routines also coordinate and control elements in the group and system libraries which are used by the job library.

- The Test System

  The Test System provides the user with complete control over programs in the debug process and allows run time information extraction and display. The test package provides an object time source-level debugging mechanism common to all programs, and eliminates the need for source-time planning of debugging strategy. This system significantly reduces the time and expense associated with program checkout.

4.6.1. Steps in Program Development

Programs are composed of one or more Relocatable Binary (RB) elements. Each element or subprogram is processed by a Language Processor as a separate entity having its own and distinct source code input, call for translation, and subsequent RB output and print listings. An RB element is not executable as such, and may be thought of as a subprogram requiring the joining together (collection) with other subprograms before the complete and executable program (object program) is obtained.

At collection time, subprograms may be contained in any of the random access storage libraries recognized by the system, placed there as a result of language translation or entered through the PLE. The joining together of subprograms provides the user with two important aids in the program development: the capability for compiling or assembling only those parts of his program which are in error or require updating; and the ability to develop a program which is composed of subprograms produced by different processors such as FORTRAN, COBOL, and 494 SPURT.

4.6.2. Source Routine

The source routine is called upon to introduce a supplementary input stream (control statements and/or source code, data, and similar information) into the primary input stream. The function includes a merge against subsequent data from the original input stream as a mechanism for correcting source code input to language processors.

Source elements are entered from the user's job library by the SOURCE control statement in the source deck. The SOURCE statement defines the element through specifications and options which describe and locate the desired element in the job library; indicates the corrections in the primary input stream which are to be applied to the source element; calls for an updated source element to be entered into user's job library; and indicates the point within the deck at which the element is to be merged. The SOURCE statement may also be used to enter additional control stream or user data.

Any source element within the job or system library may be corrected or modified by the source image corrector. This processing may include correction of the source element, submitting the source element to the primary input stream, constructing a new updated source element, and other such functions. All corrections to the source element are done by means of correction statements which immediately follow the SOURCE statement. Any number of corrections may be made.

A source code element can be merged into the primary input stream from the job library at translation time only if the element has been entered previously into the job library. Individual elements can be entered into the job library through the SOURCE statement. Once it is merged into the input stream, the source element may be deleted from the job library through an option on the SOURCE statement or by the use of the DELETE control statement following the language translation task.

4.6.3. Language Processors

Language processors are the system components which translate programming languages into machine amenable form. The primary output of a language processor is a relative binary (RB) element which is an intermediate, relocatable element used to form the object program. The language processor accepts source language, or source element input, and produces RB code output through the retrieval, control, and storage functions of the executive routine. The cooperative action of the language processors and the executive routine allow access, update, and storage of source language and object code within the library structure of the system.

Calls for activation of the language processors have similar formats which specify the particular language processor to be used, the elements to be processed, and the storage location, with various options for processing and modification of the element as necessary. Within the framework provided by the Operating System, new processors may be appended by specifying their names and characteristics during systems generation. Language processors are discussed in more detail in 4.10 of this manual and in *UNIVAC 494 Real-Time Operating System Programmers Reference, UP-7504* (current version). Concurrently available language processors for the 494 System include the following:

- COBOL

  The COBOL compiler was developed in accord with the extended COBOL-61 language, which permits program specification in relatively machine-independent, human language-oriented terms. The compiler accepts source programs in COBOL language and outputs intermediate object programs in RB code format, with source elements defining the collection procedure for the RB elements in making up the load or executable object program.

- Symbolic Language Assembler (494 SPURT)

  The 494 SPURT assembler was developed from the SPURT II language, an extension of the 490 SPURT language, which permits program specification in simple mnemonic or symbolic terms. The assembler accepts source programs in SPURT language and outputs intermediate object programs in RB code. Existing UNIVAC 490/491/492 SPURT programs may be converted or reassembled for compatible operations by the 494 System utility routines.

- 494 Assembler (ASM)

  The 494 ASM assembler is an outgrowth of the 1107 SLEUTH II language, which permits succinct, yet mnemonic expressions, and which provides directives permitting the programmer to define symbols which perform operations and define other operative symbols, building a powerful programming structure. The assembler accepts source code in the ASM language and outputs RB code.

- FORTRAN IV

  The 494 FORTRAN compiler was developed in accord with the USASCII standards for FORTRAN IV, with certain extensions, which permits program expression in relatively machine-independent, problem-oriented language. The compiler accepts source code in FORTRAN language and produces RB output.

4.6.4. Loader

The Loader is a system component which provides a flexible and efficient means for synthesis or collection of independent relative binary (RB) elements into an object program for execution as a task. An RB element normally contains references (external references or XREF's) to other RB elements and may itself contain definitions (external definitions or EDEF's), which are referenced by another RB element. The Loader joins RB elements generated from source statements expressed in FORTRAN, COBOL, 494 ASM, and 494 SPURT in the collection process. The Loader does not actually load a program into memory for execution, but constructs the entity which may be read and executed. The collection process facilitates compilation and debugging of small parts of a total program and combination of these individual parts for execution without recompiling the entire set of parts. The collected absolute program is an entity with no unresolved references and may be read into any primary storage area for execution without modification of instructions. The relocatability of the program is inherent from the Relative Index Register (RIR) and is device independent with regard to system references.

Separate elements existing in the job, group, or system libraries are collected in constructing an object program. Elements are collected on the basis of XREF's in elements which can be satisifed by EDEF's within other elements. The Loader may be directed to include or exclude specific elements by secondary control statements.

The basic output of the Loader is an absolute object program. The program is entered into the job library with the name specified by the user. Optional output includes a list of labels and tags contained in the program for utilization in testing procedures. Error messages and/or a storage layout listing may be obtained as a hardcopy record of the collection process. The Loader can also transfer the secondary control language as a job library element for subsequent reference.

Interlanguage compatibility is provided by the ability to mix, at collection time, subroutines which may have been written in any set of languages. The languages available, COBOL, FORTRAN, 494 SPURT, and 494 ASM, have differences in terminology and functions which are passed on into the generated subroutines.

Basic compatibility between object subroutines is provided by the Loader which can integrate any routines or subroutines produced in relative binary form, which is the output form of the language processors.

Using a called program in one language, with a calling program written in another, normally requires the transfer of data from one routine to the other. When transferring data between assembly language routines, the programmer has complete and direct control over the methods and conventions employed. The programmer can specify a calling sequence between the routines; or he can allocate memory in any manner which he desires, and make reference to items directly, using his knowledge of the correct locations. In the case of compiler generated subprograms, the programmer has only an indirect control over the machine instructions generated. Awareness of the conventions followed by the compilers is necessary for the interchange of data.

4.6.4.1. LOAD Statement

The Loader is scheduled and activated in response to a LOAD control statement in a job input stream. Information on the Load card is comprehensive enough to direct the collection and loading of most programs and may consist of the following items:

- The name and version of the RB element to be collected.
- The name and version to be given to the absolute element formed by the collection process.
- The name and version of a source element containing secondary control statements necessary to direct the collection process.

4.6.4.2. Secondary Control Statements

Construction of segmented programs or particular collections are described by a secondary control language which normally follows the LOAD statement.

The secondary control language recognized by the Loader allows description for the most complex programs. The user can enter these control statements with the input stream for each collection or he can reference a library element of control statements.

4.7. LIBRARY MAINTENANCE

Maintenance of the system libraries provides capability for establishing, altering, and preserving libraries. The Program Library Editor (PLE) is a systems element with the ability to read, write, list, and otherwise manipulate job or group libraries and their associated tables of contents. The PLE provides the mechanism for storage and retrieval of elements from the external devices, primarily tape, which support the random access job and group libraries. Under direction of the library control language, elements in source, relative binary, and absolute formats may be entered into a library, modified, or transferred from one library to another.

The PLE functions through the IN, OUT, PRT (print), LINK, and DEL (delete) control statements which specify the operations and elements involved, input/output source or media as necessary, and optional parameters, such as disposition in case of error. The PLE is discussed in detail in Section 4.7 of *UNIVAC 494 Real-Time System Operating System Programmers Reference, UP-7504* (current version).

4.8. TEST SYSTEM

The test system is a utility function designed to provide the programmer with a basic set of procedures to aid program development and testing.

4.8.1. Test Procedures

The test package interpretively executes programs placed in a test mode and provides a flexible means of dynamic control of test procedures through control statements and symbol tables. Symbol tables generated by compilers are accepted to provide symbolic reference of test points and data areas. Relative reference is also provided for testing of collected programs. Either means of reference allows definition of test procedures, external to the program being tested. Test procedures can, therefore, be employed at object time in contrast to source level, allowing the programmer to vary test strategies without costly recompilations and collections.

Test procedures provided fall into two general categories: conditional procedures which regulate logical switches that are used to control activation and frequency of functional procedures; and, functional procedures, themselves, used to perform diagnostics. Functional procedures provided in the basic package are: snapshot dump of primary storage and/or peripheral areas; store statements used either for patches or to set test values; printing trace display; trap display; and exit used for early termination.

4.8.2. Logical Switches (Conditional)

The test system contains a set of 26 logical switches, A through Z, which may acquire a value of on or off. A conditional procedure logically operates the state of a particular switch. The effect of successive conditional procedures is cumulative. The purpose of conditional procedures and the logical switches is to establish whether or not a functional procedure, contingent upon the state of the switches, will be executed. This allows the user to employ debugging techniques responsive to the dynamic execution of a program. Through use of multiple switches, the programmer may nest procedures to provide additional flexibility.

4.8.3. TEST Statement

The TEST statement is the primary control statement for the Test package. The TEST statement activates the Test routine, specifies the program area and values to be tested, and indicates the logical switches on the JOB statement which are to be tested. These logical switches, A–E, are not the same logical switches for establishing conditionality as mentioned in the previous paragraph.

4.8.4. Secondary Control Language

The secondary control language describes functions to be performed by the system and immediately follows the TEST control statement.

4.9. UTILITY PACKAGES

Utility programs are auxiliary routines incorporated into the UNIVAC 494 Operating System to provide ancillary support to the system and to provide a wider range of data processing and information handling capabilities. The routines perform such functions as file handling and manipulation, program updating and conversion, report generation, and accounting.

4.9.1. File Control

File control is performed by the Basic File Handler (BFH), a high level, device independent routine. The BFH provides form independent access to both random and sequential files. Significant features of the routine include block or item level access, blocking/deblocking, overlap buffering, and label and sentinel processing. The BFH serves as an intermediary between the worker program and the device control elements of the executive routine; requests read and write operations whenever required; and performs automatic label checks, checksum computations, unit swapping, and other functions. These services are performed for files residing on random access storage, magnetic tape, punched cards, and online printers. The BFH is discussed in detail in *UNIVAC 490/491/492/494 Real-Time System Basic File Handler Programmers Reference Manual, UP-7573* (current version).

Through the BFH, characteristics of the file, which is to be processed, are defined by the user in a File Description and Usage (FDU) Table. The table contains information necessary for identifying and processing the file, such as file name and file code, buffer location and usage, block and item structure, and similar parameters. The BFH operates under control of the executive routine through basic file, read, and write instructions.

4.9.1.1. File Organization

Data files are constructed in a variety of formats because of the differences in problems, the conflicting interests of space versus time, and real time access applications. File control provides conventions through which a variety of files may be described and handled. The conventions must be selected for describing both blocks of data and items of data.

A data file consists of a collection of items which may be of either fixed or variable size. Variable items contain a length field within the first word to allow file control to handle the items. Except for this field, file control makes no reference to the data within an item.

When variable-length items are necessary, fixed-length blocks may be retained by using padded blocks or spanned files. Padded blocks waste space, and some items may grow larger than the selected size for blocks. Such blocks have the advantage that processing of the blocks in place is possible. Spanned files save space, eliminate all conflict of item size and block size, but do not permit processing in place.

The addressing of a random file requires relative item addresses. The addresses imply a user provided routine which transforms keys into relative file addresses, either by formula manipulation or by privately maintained indices. Random files are essentially unblocked, although a user imposes pseudo blocking by the buffer sizes selected. For spanned files, items to be processed must be transferred to a work space; they may not be processed in place.

Spanning is avoided on fixed item files by adopting buffers which are multiples of item size. However, if any single item exceeds the buffer size, spanned files must be used. Padding may be employed as a user option for nonspanned files.

4.9.1.2. File Read

The file read instruction repertoire of the BFH provides methods for transferring data from input buffers to the worker program and also for automatic buffer monitoring, block reading, sentinel testing, and unit swapping. The worker program may use these services at the block and/or item level as desired.

4.9.1.3. File Writing

The write instructions of the BFH provide methods for loading output buffers with data, and automatically perform buffer monitoring, block writing, sentinel insertion, and tape unit swapping operations. The worker program may use these services at the block and/or item level.

4.9.2. Report Writer

The UNIVAC 494 Report Writer is a system utility program which scans a file from beginning to end, and prints a report conveying one set of information for each accepted record in the file. The Report Writer processes files residing on magnetic tape, random access storage, and punched cards. The Report Writer is discussed in detail in *UNIVAC 490/491/492/494 Real-Time Systems Report Writer Programmers Reference, UP-7650* (current version).

The Report Writer is designed for use with the UNIVAC 494 Basic File Handler (see 4.9.1) which performs read, write, and other services in manipulation of the file. The Report Writer is activated by the REPORT control statement as a task under control of the executive routine. The user describes the file and the desired report input by means of parameter cards. The program obtains records one by one from the file and processes each record through a predetermined series of steps. Consequently, user programming experience is not necessary, although some familiarity with card and tape files is helpful.

The file which is to be processed must be composed of records that are recognizable by the BFH. The records may be of variable size. The program handles those fields in the records which are at fixed positions relative to the record start.

4.9.2.1. Capabilities

The Report Writer is designed to provide power, flexibility, and ease of use while conforming to the concept that the user "describes the input file and describes the report which he wants from it." Capabilities and restrictions of the Report Writer are as follows:

- The prime utility of the Report Writer is its ability to move and manipulate fields as designated by the user in generation of a report.
- The program makes one pass through the input file and produces one report which is generated in input file sequence.

- The file may reside on magnetic tape, mass storage, or cards. The program provides an option permitting a card file to be read through the primary input as well as from a secondary card reader.

- The input file is composed of records which are recognizable as such by the BFH and are transferred one by one to a work space. In the case of primary stream card file input, the record size may comprise 1 to 33 sixteen-word card images.

- The program references fields within records. The locations of such fields are specified by stating the number of bits of the record which must be skipped. The sizes of the fields are specified by stating the number of bits to be used.

- Sequence checking is provided. One field of the record is checked against the corresponding field of the previous record. If the specified sequence fails, the program is terminated. The sequence may be specified as ascending or descending.

- A selective feature limits the report to acceptance of only those records which pass certain tests which compare fields of the record to values provided by the user. Eligibility of the record is determined on the basis of the results of the comparison. The user may "and" several tests to form one rule. He may also "or" several rules. On each test the user may specify:

  - equal to or not equal to;
  - less than or not less than;
  - greater than or not greater than.

- When a record has been accepted for inclusion, a series of tests may be made for breakpoints. A breakpoint is defined for the Report Writer as a condition which occurs when a field is not equal to the same field of the previous record. When a breakpoint occurs, a subtotal line/lines is printed, indicating the amounts accumulated in one or more numeric columns. A maximum of 15 sets of totals may be printed, allowing for 14 breakpoint fields (subtotals) plus the grand total line. When an intermediate breakpoint is reached, all lower level subtotals are automatically printed. A hierarchy of breakpoints may be arranged. For example, a hierarchy might be region, state, and county. A state break would cause county totals to be printed; then, state totals would be printed. Accumulators are reset to zero after printing. All levels of subtotals, plus the grand total, are made available for printing at end of file.

- Accumulation is done only on columns being printed in the "detail" lines of the report.

- After printing any totals lines caused by breaks, the program prints a user-specified detail group from the current record by directing the line number and the column locations in which fields are to be printed. Additions are made to accumulators, if any, and processing advances to the next record.

- One group of detail lines per accepted record is the rule. The detail group format is not variable during the run.

- Fields may be edited on the way to the print buffer by truncation, roundoff, leading zero suppression, and decimal point insertion.
- The user may input page heading lines containing report title, column headings, time information, and date/page package information as provided under the High Speed Printer basic write package of the BFH. Line spacing and lines per page information are also selectable.
- User input is kept to a minimum. For example, absence of acceptance specifications means that all records are accepted; and absence of breakpoint specifications means that no breakpoints are desired, and no accumulator totals are required.

4.9.2.2. REPORT Statement

The Report Writer Program is activated in response to the REPORT statement. Options specified in the control statement determine the disposition of errors encountered in the parameters and in the processing of the file.

4.9.2.3. Parameter Statements

Parameter statements describe the input file and its fields to be processed, page and line format of the report, the nature of the report information, and special options desired by the user. The parameter statements must follow the REPORT statement in the control stream.

4.9.3. Utility Generator

The Utility Generator is a system utility program which provides a flexible and device-independent means of generating and manipulating data files. Data may be transferred to primary storage from a file or transferred from file to file. The Utility Generator is a useful adjunct to the Test System, since it provides procedures for the distribution of test data as a prelude to debugging, or for the collection and display of data manipulated by the test run.

The Utility Generator operates as a task under control of the executive routine through the UTL control statement. Secondary control statements describe the type of file manipulation to be performed. These statements are processed by the Utility Generator, and an absolute element is formed, capable of performing the actions described. The element generated is placed in the job library with the name and version specified by the user. The element may then be activated by means of the GO statement, and/or the element may be retained for subsequent use through the OUT statement. Files to be referenced must be assigned by the user prior to activation of the generated utility element. As an option, files that follow UNIVAC 490/494 data file conventions may be processed.

4.9.3.1. UTL Statement

The Utility Generator is activated in response to the UTL control statement. Options on the UTL statement determine the disposition of illegal control statements and errors encountered during execution. The UTL statement also specifies the name and version of the utility routine to be generated.

4.9.3.2. Secondary Control Statements

The secondary control statements of the Utility Generator describe the actions to be performed by the generated element and are of two types: action statements and test statements. The action statements provide for transfer of data, iteration, and unconditional transfer of control. The test statements provide for conditional transfer of control based on predetermined contingencies. User own code subroutines may also be called to process file data. System control statements may be included in the secondary control language; these statements are submitted internally during execution.

4.9.4. Logging and Accounting

The Operating System maintains a log for the collection of information pertinent to a task or job and to the operation of the computer complex as a whole. This information is gathered and logged by executive elements and is later processed by a utility routine to provide accounting statements, on the UNIVAC 494 complex under the user operational load.

Accounting information is sorted by charge number, summarized, and extended for billing purposes by the installation. The accounting statement contains, by task, the following pertinent information:

- Sign-on and sign-off time, date and charge number
- Amount of primary storage utilized and duration of time used
- Amount of CPU time used
- Peripheral Units: number and amount of time utilized
- Magnetic Tape Units: number and amount of time utilized
- Random access storage: amount, weighted by grade, and time utilized
- Communication Lines: number and length of time assigned for each line; including the number of input/output messages
- Number of drum modules used for primary and secondary input/output
- Optional summary of number and length of I/O transfers to random access storage and magnetic tape units
- Number of activities within the task
- List of program and hardware contingencies experienced during execution of the job
- List of all LOG statements submitted by the user task and control statements

Other messages entered to the log are: hardware contingencies, console messages, conventionalized user messages, and similar information.

4.9.5. REXecutor

The REXecutor is a utility package which is designed to load and execute REX 490 (UNIVAC 490 Real-Time Executive Routine) oriented programs under the UNIVAC 494 Operating System. The REXecutor loads and executes programs which have been assembled by SPURTFD (490 SPURT) with simple relative (SPURT output 321) or complex relative (SPURT output 322) object coding and standard REX request packets.

The REXecutor is composed of two common subroutines; one of which is re-entrant and the other is not re-entrant, and a 400_8 word interface which is attached to each task. This organization allows concurrent REXecutor execution of several independent programs.

The REX 490 worker program is loaded into primary storage by the REXecutor. Once the worker program is loaded and initiated, the program is allowed to run free. The REXecutor is not an interpretive executor. The REXecutor is activated by the SILRJP instruction to the jump table at 140–146 (the RIL operation is performed immediately by the REXecutor to prevent interference with a real time environment). The request is interpreted and either reformatted for submission to the executive routine or executed by the REXecutor itself, whichever is appropriate. CKSTAT's (status checking) and TAKEOVER's are executed in the same way as in REX. Proper REX status words will be returned to the requester. All return points are under REX conventions.

4.9.5.1. Restrictions

The REXecutor is able to operate most UNIVAC 490 worker programs; however, some restrictions are imposed:

- The REX request packets must conform to standard 490 REX specifications. Any packet written to conform to special site modifications of REX will not be executed properly, if at all.
- The REXecutor is not designed to execute real time communications programs.
- Execution of hardware level I/O instructions (privileged instructions) will cause a fault interrupt.
- Input/output facilities cannot be acquired during execution time and must be assigned prior to the selection of the REXecutor task. The facilities will be held by the REXecutor until requested by a REX internal facility request. Once a facility is released, it cannot be acquired again.
- The binary time at which the REXecutor is initiated will be in word 147. This time is in UNIVAC 494 format and will not be updated by the REXecutor. The date is in word 135.
- Only five I/O operations, with or without CKSTAT, are allowed to be outstanding at one time. Requests in excess of this limit will be treated as a REX addendum overflow.
- Because of the disparities between the REX addendum and the REXecutor program interface, the 490 program should not contain instructions which modify the REX addendum.
- The following REX loader requests will be ignored and marked as completed by the REXecutor. Where possible, the request will be marked as not having been fulfilled. All other REX loader service requests, including site utility, will be considered errors; the REXecuted program will be terminated.

REX Service Request

- Real time initialization
- Start slave
- Rerun dump
- Subroutine load
- Real time extension
- Internal load request
- Core release
- Random storage release

- The EXCHANGE request will be considered to be an error and will cause the program to be terminated.

- Any subroutine employed by a REX 490 program must be defined by a PROG statement and loaded at the same time as the primary program. The subroutines must be activated by a jump or return jump. As noted below, subroutine load packets will be ignored. This limitation does not apply to the loading of secondary segments.

4.9.5.2. Control Cards

Execution of a program under the REXecutor is very similar to execution of any other normal task under the executive routine. The REXecutor is activated by the REX control statement.

When the task is selected and primary storage is allocated, the system will insert the 400_8 work interface into the beginning of the area for the exclusive use of the REXecutor. All worker programs will be loaded above the interface. Control is given to the interface, which will, in turn, call in the REXecutor proper.

When the REXecutor is activated, the program begins the reading of secondary control cards through primary input.

- The MEANS Statement

  All peripheral and random access storage assignments are made prior to task activation by means of ASG control statements. There is no load time allocation of peripherals as under REX. The assignment is equated to a specific peripheral device in the worker program. The peripheral device is a unique channel/ unit as defined by the SPURT MEANS, ASSIGN, and FACIL statements. Since the channel and unit are not altered when the worker program is loaded, the channel/unit specification appears as originally specified in the SPURT MEANS and ASSIGN statements. The REXecutor does not examine the contents of the previously submitted ASG control statement. This allows a degree of flexibility. For example, a 490 program written for UNISERVO II A magnetic tape units, can now be run on a UNIVAC 494 System with UNISERVO VIII C magnetic tape units without alteration or reprogramming. The UNISERVO II A packet will be properly submitted for UNISERVO VIII C magnetic tapes by the REXecutor.

- The PROG Statement

  The PROG statement loads the REX 321 or REX 322 program into primary storage. Any number of PROG statements may be submitted to the REXecutor, one for each program to be loaded. All subroutines are loaded before the primary program is activated.

- A Card

  The A card is used for errata in the same way that it is used in REX, and the format is identical to the format used for the A card in REX. The addresses are relative to the task base. This includes the 400_8 word interface area.

- B and C Cards

  The B and C cards are used for parameters exactly as they are used in REX. Parameters are stored beginning at the address specified in the Executive Information Region (EIR). The REXecutor supplies the count of the number of parameter words. B and C cards must follow the PROG card which loads the program that is receiving the parameters.

- The PS Statement

  The PS statement is used to activate the worker program. All images in the primary input stream between the PS statement and the END control statement will be considered data for the worker program.

4.9.6. CONVAID

CONVAID is a system utility routine which assists in the conversion of a SPURTFD (REX-Oriented) program to a SPURT4 (494 SPURT) program. CONVAID accepts a source program written in the SPURTFD language and outputs a source element in 494 format with a flagged listing of the source element. Optional "own code" elements may be used with CONVAID.

The SPURTFD source language input may be contained in a symbolic card deck, a SPURTFD 301 output tape, or on a library tape created by the REX utility routine, RMASL. Provision has been made for multiple reel input. Card input may be entered through the primary input stream or through a nonprimary card reader.

The output of CONVAID is a source element in the 494 Operating System format. The element is placed in the job library and may be retained for further use by the OUT control statement, and/or the element may be assembled by 494 SPURT through the SOURCE control statement. A listing of the newly created source element is also submitted to the primary output stream. Items in the element which will cause 494 SPURT assembly and/or run-time errors are flagged in the listing.

Own code routines may be collected with CONVAID to assist in the conversion process. The own code routines may be used to insert, replace, or delete source images and to modify the print images concerning the processed images.

Each individual installation has certain features common to many of the programs to be converted. The own code routine may be written to take advantage of these features, and can change REX code automatically to 494 executive routine code in an optimum fashion. CONVAID is described in detail in *UNIVAC 490/491/492/494 Real-Time Systems CONVAID, P.I.E. Bulletin 4, UP-7505.4.*

4.9.7. Random Storage File Handler

The UNIVAC 494 Random Storage File Handler (RSFH) is a routine which provides the programmer with an efficient means of generating and utilizing files placed on random storage. Through collection (a Systems Processor (Loader) function) with the worker program, the RSFH offers a complete repertoire of re-entrant functions to generate any number of files, and to search and update each file without programming the extensive bookkeeping and I/O operations involved. Files of virtually any size, with fixed- and/or variable-length records, containing keys (used for element recognition) of any size, can be optimally handled through judicious selection of the available services. Both searchable and nonsearchable files can be handled by the system. Searchability is restricted to files composed of fixed-length records. Files composed of variable-length records cannot be searched, but can be tied to searchable files to obtain the related services. The worker program must establish for each file a short descriptive table, work space reserves, and buffer reserves. Consequently, primary storage utilization outside the RSFH program is dependent only upon the particular user's facilities, needs, and applications.

The speed of the RSFH system is similarly dependent on user applications. The basic factor for consideration is the average access time for any given record within a file. This is governed primarily by the number of records entered into core storage per input request, and by the I/O cycle time and latent access time of the peripheral random storage device used. Since I/O operations are major time consumers, the system will function faster if it requires fewer I/O requests for accessing a given record. For this reason, the system provides two methods for handling searchable files, the key table method and the pyramid method. The selection of the method to be utilized is determined by the user and is based on key size, file size, and available facilities.

The key table method is faster than the pyramid method, but for large files requires considerable core storage to maintain a searchable table of keys. Through this method, a key in core storage references each block of records on random storage. Consequently, every record is available through only one I/O request.

The pyramid method, on the average, must issue more than one I/O request to access a given record on random storage, but will handle files of any size without the need for additional core storage. Through this method, records in higher level blocks point to groups of records in lower level blocks, necessitating several I/O requests for obtaining records in the lower level blocks of the file. The fundamentals to be considered in selecting and utilizing the RSFH services for optimization of user applications are discussed in detail in the *UNIVAC 494 Real-Time System Random Storage File Handler, P.I.E. Bulletin 10, UP-4121.10.*

4.10. APPLICATION PACKAGES

Application packages are ancillary programs which may be used by an installation to perform specific functions or applications as extensions of the basic computer software. When packages have been implemented in the UNIVAC 494 System to provide greater system usage potential for the system user, the application packages operate under control of the executive routine designed to be easily integrated into the 494 Operating System. For each package, detailed discussion has been made in a separate reference document.

4.10.1. Sort/Merge

The UNIVAC 494 Sort/Merge package is a generalized subroutine which is utilized by a user's own code at load time to form a particular Sort program. The subroutine approach is a new concept in sort design, giving the user greater control over both the data to be sorted and the sorting process itself, and substantially increasing his flexibility in the use of the sort. The Sort/Merge is designed for use with all of the assembler and compiler languages of the system, including 494 SPURT, 494 Assembler, COBOL, and FORTRAN. The unordered input file is read by the user's own code and transmitted in memory to the Sort subroutine, one record at a time. The Sort/Merge is discussed in greater detail in *UNIVAC 494 Real-Time System SORT/MERGE Programmers Reference, UP-7538* (current version).

4.10.1.1. Basic Concepts and Operations

Since the interface between own code and the Sort subroutine is strictly internal, the external storage device and the external format of both the original input and final output files is completely controlled by the user's own code. The user may handle input/output by any means that he desires. The user's own specialized input/output routines may be applied to formats unique to his installation; or, for standard UNIVAC 494 formats, the UNIVAC 494 Basic File Handler is available (see 4.9.1). Communication between the Sort subroutine and the own code routine is achieved by standard program linkages and by an internal parameter table in the own code routine. The parameter table defines the format of the data to be sorted and the hardware configuration available to the Sort subroutine. The linkages are similar to those required for using a file control routine.

For a typical sort, only a few basic parameter entries and program linkages are required for communication with the Sort subroutine. For more sophisticated usage, additional parameters and linkages may be used to achieve greater control over the sorting process. For example, one additional parameter to the basic set, together with a few additional linkages, allows the user to have each pass own code for data reduction. Another parameter allows the user to provide his own comparison subroutine for sorting data on unusual key types which could not be handled by the Sort subroutine. Other parameters allow splitting of a very long sort process into several smaller runs.

The Sort subroutine is in no way fixed to a particular hardware configuration. Any combination of drums, FASTRAND, and tapes may be used. Depending upon record size and hardware facilities assigned, a variable amount of primary storage is required. Up to 32K, less the own code/Sort program, may be assigned. Although use of the drum is optional, its utilization as an intermediate merging media is recommended for achieving maximum speed. In general, sorting speed will increase with the amount of primary storage and drum storage assigned.

For tape sorting, the Sort subroutine requires a minimum of three units for its own use. Up to 14 units can be used. Tape units assigned to the Sort subroutine are not available for use by own code; that is, units used by own code for reading initial input and writing final output must be other than those assigned to the Sort subroutine. For convenience to the user, Sort/Merge timing tables have been derived for various equipment configurations and volumes of data. Also given are formulas for deriving sorting times for configurations and volumes of data not presented on the tables. This information may be found in *UNIVAC 494 Real-Time System Sort/Merge Timing Tables, P.I.E. Bulletin 2, UP-4121.2.*

4.10.1.2. Capabilities

The Sort/Merge is a highly flexible subroutine providing features desired by very sophisticated users, yet it is simple and easy to use for those sorting on a one-time basis or whose requirements do not include a sophisticated own code. The significant features of the Sort/Merge are as follows:

- The subroutine concept allows the sort to be called by a large and complex own code in which the sort acts as a true subroutine. On the other hand, the sort may be called by a few lines of standard coding which cause the sort to be executed essentially as an independent program.
- Multicycle capability allows an unlimited volume to be sorted, either automatically or in separate phases.
- Record size is limited only by the amount of primary storage available.
- Either fixed- or variable-length records may be sorted.
- Five different types of keys may be specified.
- Key fields may appear anywhere in the record for fixed size records. For variable size records, all keys must be contained in the first link. There is no limit on the number of fields which may be specified.
- Key fields may be sorted in ascending or descending sequence.
- The user may specify his own collating sequence.
- Both rerun and restart ability are provided.
- Data reduction own code may be executed.
- The user may provide his own comparison routine to handle special key types.
- A Merge subroutine, which is provided, will perform an internal merge of up to 26 files.

4.10.2. Network Simulator

The UNIVAC 494 Network Simulator (UNS) is a package designed to allow simulation of communication networks, especially those networks built around a computer system for operation of real-time systems. UNS is particularly applicable to the simulation of reservation systems, communication systems, and so forth. UNS requires the building of a logical model of a real or proposed network of interconnected components (nodes and trunks). These components may create, modify, move, or remove the system work load (transactions), in accordance with the users instructions.

4.10.2.1. Design and Capabilities

The input required to build the internal model for the UNS is provided through a self-documenting series of English language sentences and parameters. The input consists of a network configuration description specifying the various nodes and the trunks connecting them, the capacities and other characteristics of the nodes and trunks, the volume and other characteristics of the transactions (work load), the logical decisions to be made, routing to be taken, and the statistical output desired.

The simulation uses Monte Carlo techniques to generate transactions and initiate various activities. Each transaction enters the system and, going from node to node, proceeds through input, processing, output, and transfer stages. The input and output stages are essentially measuring points for queuing and waiting. The transaction which is to be processed depends upon priority, available processing capacity, and similar factors. Processing time is based upon transaction length processing node characteristics, or originating node characteristics. Transfer of a transaction to another node depends upon completion of processing in the present node, priority, available trunk capacity, and available next node capacity. The next node may be selected in several ways depending upon the termination node. The transaction may be forced into the next node, may ask permission to transfer to it, or may wait for the next node to request it. The transfer time is based upon the transaction length, trunk length, and trunk speed. System performance is recorded with every change of state. Queuing and waiting statistics are collected for the input and output stages. Utilization statistics are collected for the processing and transfer stages. In addition, statistics on the total system performance are collected.

The output of the simulator includes total system statistic tables showing distribution of response times, waiting times, waiting frequencies, and activity levels. Individual statistics are collected for each node and trunk. These tables show distributions of input and output waiting times and frequencies, queuing, and processing and trunk utilizations. The average and maximum response times of each node, which originates transactions, are also given.

Special features of UNS include a stability check, decision rules, and a transaction history. The stability check will determine if the system has reached, and is maintaining, a certain level of activity. Statistics prior to stability are discarded. Statistics collected after stability are checked for excessive fluctuations and cycles.

UNS has seven decision rules available which allow the user to select a range of values for each parameter (length, priority, processing time, and so forth) and also allow UNS to make choices between alternative routings and courses of action. These decision rules allow selections from a constant round robin sequence, discrete distribution, or from a uniform, continuous, normal or negative exponential distribution. The transaction history option provides a complete node to node trace of every transaction entering the system and of any event, such as waiting or processing, which will increase the transaction response time. Detailed information on UNS can be found in *UNIVAC 494 Real-Time System Network Simulator Programmers Reference, UP-7548* (current version).

4.10.3. Transaction Control System

A number of applications require a real time transaction-oriented system rather than batch-oriented processing. These applications include airline reservation systems, stock inquiry systems, message switching, management information systems, and similar operations. In these types of applications, each message (transaction) must be processed as soon as possible after it is received by the system. A transaction-oriented system considers each message a discrete entity in itself and activates segments to process the data. This is the main difference from batch-processing systems in which programs are loaded in a determined sequence and the data is passed against the program.

The main task of a transaction-processing system is accepting message input from remote sites, processing this data, accessing and updating master files residing on random storage as required, and generating a response to the message to be output to one or many remote sites. A major part of the above process is involved in protecting the integrity and providing recovery of the user's master files.

The UNIVAC 494 Transaction Control System (TCS) is a random storage, transaction-oriented system made up of modular routines that have been designed to run under the executive routines. In providing the services above, TCS is the controlling agent between Transaction Processing Segments (TPS's) written by the user and the executive routine. TCS operates as a normal task, and communicates with the system in the same manner as a batch program. This provides the ability for an installation to mix transaction- and batch-oriented processing on the same computer at the same time.

4.10.3.1. TCS Control

TCS control of message flow through the system encompasses three main areas: message startup, processing, and termination. The file recovery and restart feature are integrated into the main areas and occur automatically unless specifically inhibited.

The message startup routine is the first part of the TCS. Initialization of TCS provides for user own code to assign and request terminal lines and to allow the installation to call the various functions necessary for a particular configuration.

Control and queuing of messages is performed by system communication routines. The system provides entries for user own code routines to classify messages and assign priorities for queuing. The extent of the processing done at this level will depend upon the needs of the installation. A user that interfaces directly with Level 1 communication routines may also use TCS. This will require the particular user to provide all functions necessary for moving message data to the TCS input queue buffers and to or from the transaction area when necessary.

Message processing is the second part of TCS and is concerned with processing of messages, updating of files, and obtaining information from the files to process the messages. The tool used in processing is a Transaction Processing Segment (TPS) which is a program that runs as an activity under the TCS task. The TPS is generated using any of the language processors available and is collected through the standard procedures. Together with the TPS is a Transaction Area (TAR) which is used as a control and scratch area for the transaction to hold the input message, perform calculations, store intermediate results for subsequent TPS's, and build output messages.

Message termination is the third part of TCS and is concerned with transaction termination, which may come to an orderly completion or an abnormal completion. In normal completion, the TPS and TAR associated with the transactions are deallocated and the starter routine is entered to search for a new transaction to process. In abnormal termination, the TPS and TAR are deallocated and any files updated by the aborted transactions are restored to their previous state. The abnormal termination method is discussed in subsequent paragraphs with file recovery procedures.

4.10.3.2. Message Processing

In message processing, the message enters the system from a line terminal, and is handled by the system communication routines and an optional user own code routine. The own code routine may determine which random storage input queue of the Communications Director will store the message.

At fixed-time intervals, and whenever a transaction is terminated, the starter searches the input queues, beginning at the highest priority, for a queue that is not busy and which contains a message. The starter activates the transaction, and provides for a user own code routine whose responsibility will be to assign a starter value to the message.

The TCS starter is responsible for loading the correct TPS as indicated in the starter control table, establishing the TAR to be used with this TPS, setting the RIR and PLR limits, and finally registering the TPS and TAR as an activity. The processing of each transaction is registered as a separate activity under the TCS task, permitting multiple transactions to be processed concurrently.

The TPS acquires the message through a READM service request which transfers a specified amount of the message into the requesting TAR working storage area. Subsequent READM's will transfer the remaining part of the message into the TAR until an end of message is reached.

The TPS processes the message using whatever service requests it needs. If another TPS is required to process the message, the current TPS gives the required request for calling the next TPS for processing.

When the processing of a message has been completed, the TPS initiates a function (CLEAN UP) which ensures a proper completion of the transaction.

When the TPS has finished the processing and an output is required, a WRITEM service request passes the output message from the TAR to a random storage output queue. The WRITEM request also sets up a control block for this message indicating the destination of the message and links the message to the other control blocks for recovery. The message is transferred from the output queue to the output device through an optional user own code routine. The amount of processing by the user own code routine depends upon the needs of the particular installation.

4.10.3.3. File Processing

The TCS file accessing subroutines provide the user with the mechanism for accessing file records and for deleting, creating, and expanding files. The created files are registered with the system MFD as permanent files, and are available through TCS by reference to their user number/file number. Records are addressed by specifying the user number/file number and logical increment. The file accessing subroutines maintain controls for the recovery subroutines.

- Duplex Files

  User files which are assigned through TCS may be designated as duplex files. TCS will then maintain two identical copies of the same file on two different subsystems. If one subsystem fails or a section of one file becomes unreadable, the correct information will be obtained from the alternate file. Write operations to the file are directed to both copies by TCS.

- Logical Lock

  A logical lock list is maintained by TCS to prevent different TPS's from accessing and updating the same file area concurrently. Whenever a TPS reads a file area, that area is made unavailable to other TPS's. Any other TPS attempting to reference the locked area is delayed until the area has been released. The areas locked in this manner by a transaction are maintained until the transaction terminates, and are then made available to other TPS's.

4.10.3.4. Recovery

Two types of recovery are possible in TCS: file recovery and TCS system restart.

- File Recovery

  The basic file recovery system used is to capture dumps of the files onto tape and to record images of any portions of the files that have changed by writing "after-look records" onto an audit trail tape. The file recovery procedure is a combination of reloading the files up to a certain point and updating them with any changes made after that time.

File recovery is also concerned with portions of the files that have become unreadable. In the case of duplex files, much of the recovery procedure is automatically built into the error routines, where portions of files are re-created as they are determined to be unreadable. In certain cases the unreadable files are treated as simplex files and recovery procedures are initiated from the audit trail.

The "before-look" file is another method of file recovery. An image of each area of a file changed by a transaction is saved before the change is made. When a transaction is aborted, the transaction "before-look" file is checked and all modified areas are rewritten with the initial data.

- TCS Restart

  TCS restart accomplishes program recovery through the use of checkpoint files. As each transaction is started, its beginning TAR is written to the checkpoint file, and as a transaction terminates, its final TAR is written to the checkpoint file. When a contingency occurs requiring TCS to be restarted, the checkpoint file is referenced to determine which transactions were in progress when the contingency occurred. The transactions are then aborted, their files restored, and new transactions are started.

4.10.4. Critical Path Method

The Critical Path Method (CPM) package provides a mathematically ordered system for scheduling and coordinating complex projects such as large scale construction, scientific research and development, introduction of new products, and analysis of business control systems. The CPM is a management technique which permits specification and subsequent analysis of a project in terms of a network of activities, each of which has a defined event as its starting point and a defined event as its end point or result. The technique considers each activity of the project on the basis of time, costs, and other factors required to complete the activity, providing a base for evaluation of the effect of the activity on the total time and effort required to complete the project. Through this analysis, the critical path of the project is determined, that is, which activities must be performed and when the activities must be performed for achieving optimum efficiency in completion of the project.

The CPM package consists of four separate programs. This separation permits greater flexibility in CPM programming and allows a given CPM application to be processed using less primary storage than would be needed if all four programs were required to be resident in primary storage simultaneously.

The first program, *Calculations*, computes durations, costs, and manpower requirements, and then calculates the free time in the network, as well as the earliest and latest start and finish times of each activity in terms of elapsed working days. These calculations are printed and recorded on magnetic tape for subsequent processing by a calendar dating program. The Calculations program also checks the data input to ensure that the data is in the prescribed order, that the number of activities in a network is not excessive, and that the network is complete.

The second program, *Calendar Dating,* uses the output of the Calculations program and an internal calendar to produce an output report showing the calendar date of the earliest and latest start and finish times of each activity and the "float" (free time) of that activity.

The third program, *Sequence,* is provided as a tool for dealing with contingencies. One contingency is that condition wherein the data violates the prescribed input order requirement of the Calculations program. Providing the data supplied is otherwise correct, Sequence renumbers the activities to meet the requirement and provides an input suitable for Calculation. Sequence is also useful when two or more networks are to be combined into one because it relieves the user of the necessity of completely renumbering one or more of the networks in order to fulfill the input data order requirement.

The fourth program, *Time-Sequence Plot,* furnishes a graphic representation of the interrelationships of the activities of the CPM network. Using information supplied by the Calculations program, the Time-Sequence Plot program generates a bar chart plot on a calendar day basis of the activity duration from scheduled early start to scheduled early finish and the amount of total float. The plot can be used to monitor project performance by drawing the actual start and completion dates of an activity below the plotted dates.

4.10.5. Linear Programming

The linear programming package, the UNIVAC LP49X System, solves the standard linear programming problem: minimization of a linear objective function subject to a set of linear constraint expressions that are represented by a matrix and a right-hand-side vector. A product-form revised composite simplex algorithm is used and all data is contained completely in core. A vocabulary of 62 mnemonic six-character words is used to control the execution of the LP49X System. The program operates under control of the executive routine of the UNIVAC 494 Operating System, and, with modifications, under the SEER 3 monitor routine of the UNIVAC 490/491/492 Systems.

The LP49X System operates in either of two modes: (1) CARD mode, in which control word and data input are by cards, or card images on magnetic tape; and (2) CALL mode, in which the LP49X System becomes a subroutine that is called from a user's independent FORTRAN program, with control words input by subroutine call arguments and data input by cards. The LP49X System permits the user to change from one mode to the other by control word to suit his needs.

When used as an independent program (in CARD mode), the LP49X System for the UNIVAC 494 will attempt to solve any problem which has no more than 80 rows, no more than 320 columns, and no greater than ten percent matrix density. When the LP49X System is to be run on any other computer of the UNIVAC 490 Series, the system must be configured by Systems Programming personnel in St. Paul to make use of all available memory in a specific computer at a specific installation.

When used as a subroutine called by a user's independent program, two key elements, subroutine LP and block data subprogram LPBD, may be used either without change, if the user's independent program is less than 2350_8 core locations in length, or with changes, as needed, to the dimension of array A and values of parameters MRLIM, the maximum number of rows, MCLIM, the maximum number of columns, and ITOTAL, the number of *core locations* in array A.

The LP49X System is a modification of a system named MFOR, which was developed by DuPont to run on a UNIVAC 1107. The DuPont version is a modification of an earlier MFOR system that was developed by the Rand Corporation to run on IBM equipment.

Modifications made in the LP49X System for operation on computers of the UNIVAC 490 Series are based on the following conditions:

(1) a word length of 30 bits instead of 36 bits

(2) use of two words for each floating-point number

(3) Operating System characteristics relating to the 494 clock and unsolicited console input messages

The LP49X System requires a minimum of one card reader, one card punch, and one magnetic tape unit for execution. Ordinarily the user will, if possible, use one tape unit for the Operating System or SEER 3, and up to four tape units for the LP49X System as determined by the user's requirements. These tapes are: (1) Alternate Input Tape; (2) Alternate Output Tape; (3) Problem Library Tape; and (4) Dump-and-Reload Tape.

Primary storage requirements for the control program and the LP49X System are 32K words.

Detailed information for the LP49X System may be found in *UNIVAC 490/491/492/494 Real-Time System LP49X Linear Programming System, P.I.E. Bulletin 5, UP-7505.5.*

4.11. LANGUAGE PROCESSORS

The Operating System includes a variety of language processors for the development of user programs. Two compilers (FORTRAN IV and COBOL) and two assembly languages (494 SPURT and 494 ASM) are available.

The language processors operate within the control environment of the executive routine. The processors are activated by control statements and obtain their input source language from the primary input stream. The source language is translated into one or more relative binary (RB) elements and placed into the user's job library. All printer listings and diagnostic messages are submitted to the primary output stream. The RB elements generated by the language processors are compatible; that is, the RB elements of the different processors may be collected together to form a single load element.

4.11.1. 494 SPURT Assembler

The 494 SPURT assembly language is a development from the SPURTFD or SPURT II languages used with the UNIVAC 490/491/492 computers. SPURT provides a simple mnemonic method for representing the computer instructions to be performed. The basic one-for-one code generation is enhanced by polyoperations, general assembler directives, and macro operations. Mnemonics for system service requests are also provided. The SPURT mnemonic code corresponding to the instruction repertoire of the 494 computer is given in Table A–1.

The 494 SPURT assembler utilizes two scratch files; one random access file, and one sequential file. The sequential file may be specified on either random access storage or on magentic tape. The assembler generates approximately 6600 instructions per minute.

The output of the 494 SPURT assembler is a RB element which is placed in the user's job library. A side-by-side listing of the source code and the object code is produced unless specifically inhibited by the use of an option on the SPURT control statement. Other options provide for reference dumps, table expansion, and various error recoveries.

Detailed information on the 494 SPURT assembler may be obtained in *UNIVAC 494 Real-Time System SPURT Programmers Reference, UP-4090* (current version).

4.11.2. 494 ASM Assembler

The 494 ASM assembler is an assembly language designed for the UNIVAC 494 Real-Time System and developed from the 1107 SLEUTH II language. The ASM assembly language is a set of concise mnemonic codes combined with assembler directives which the assembler translates from source images into RB elements. The RB elements may be collected, together with other RB elements, to produce an object code program for execution.

The user is given great freedom in the source image format; free field format is allowed, enabling the user to design a source image for his particular problem.

Assembler directives which the user might employ include the FORM directive which allows the user to describe a special word format. The word format may be composed of fields of variable length within the word.

The two most powerful assembler directives available to the user are the PROC and FUNC assembler directives. PROC and FUNC directives are used to define often used sequences of coding which are not necessarily identical, but which are similar enough so that repetition of the coding requires only the insertion of parameters when the sequence is called.

The PROC directive differs from the FUNC directive in that the PROC directive normally generates lines of object code at assembly time which are to be executed at object time. The FUNC directive is executed at assembly time and stores its results in the RB element produced. The FUNC directive calculates a value when referenced and does not cause generation of object code.

A paraform (parameter reference form) is the means whereby an operation within the PROC or FUNC can obtain values of parameters used in the operation from the call line or entry point. This enables the same PROC or FUNC to be used many times with the call line or entry point furnishing a different set of parameters to specialize the PROC or FUNC.

The entry point is a line within a PROC or FUNC which may be referenced from outside the PROC or FUNC. More than one entry point may exist in a PROC or FUNC, and a parameter may be specified on the entry line so that the actual entry point can be determined from the alternate entry points available.

The FUNC directive enables the user to obtain a value at assembly time contingent upon a set of parameters. The FUNC directive causes certain predetermined lines of coding to be saved when encountered during assembly, which, when referenced subsequently, causes a computation to be made. The evaluated quantity is then substituted for the reference call within the program.

In the case of the PROC directive, user defined and parameterized lines of coding will be substituted for the call to the PROC in the user's program.

The ASM mnemonic codes corresponding to the instruction repertoire of the 494 computer are given in Table A–1. Detailed information on the 494 ASM assembler may be found in *UNIVAC 494 Real-Time System Assembler Programmers Reference, UP-4133* (current version).

4.11.3. FORTRAN IV Compiler

The UNIVAC 494 FORTRAN compiler is an adaptation to the 494 System and an extension of USASCII standards for FORTRAN IV, with which the compiler is in full compliance. The compiler operates as a task under control of the executive routine, and is called and activated by the FOR control statement.

FORTRAN IV (FORmula TRANslation) is a problem-oriented programming language which provides a precise statement form for mathematical, scientific, engineering, and data processing applications.

A FORTRAN source program consists of a set of statements for handling communication with external devices, and for description, definition, manipulation, and computation of data. The 494 FORTRAN IV compiler translates FORTRAN source statements or elements, and produces RB elements. The programmer may think in terms of the problem to be solved and the method of solution, rather than think in terms of the computer which is used to solve the problem. While initially designed for scientific applications, FORTRAN has proved effective and convenient for many commercial and industrial applications.

The UNIVAC 494 FORTRAN IV compiler contains all the features of the USASCII FORTRAN. In addition, octal I/O editing codes, shift and bit manipulation, extensions to allow for more flexible data handling, and many other valuable extensions have been implemented to minimize user programming efforts, and to significantly increase the power and flexibility of the language.

4.11.3.1. Extensions

The significant additions to the 494 FORTRAN IV compiler are as follows:

- The 494 FORTRAN IV compiler utilizes the extended instruction repertoire and the 494 hardware double precision features within:

  (a) the 494 FORTRAN IV compiler itself

  (b) all mathematical function routines (where appropriate)

  (c) the I/O Conversion Program Package

- Mixed mode capability: variables of different types may occur in the same expression except that LOGICAL types can only be mixed with INTEGER types.

- I/O statements are extended to include PRINT and PUNCH statements.

- The DATA statement is extended to provide limited implied DO loops.

- A READ statement with no list may be used for skipping an input record.

- A logical unit 0 may be designated in READ statements. Such a statement provides reread capability on input utilizing several different formats, allowing inputs to dynamically designate the format specification to be used in interpreting the input.

- Debugging aids are provided, including a full label table, and run time error walk back procedures leading to the specific SOURCE statement causing a problem.

- Runs may be terminated optionally on any error, on I/O errors, on an unrecoverable error only, or on a legal stop.

- The same COMMON blocks (blank or labeled) need not be the same size in each of the program units. The size of a block is determined by the maximum required size in any of the program units as fixed by a specification statement.

- The use of arithmetic and logical arguments and operators has been extended to permit mixing of different types in a masking expression.

- Assignment statements have been extended in the General Assignment statement to permit use of masking expressions.

- The arithmetic IF statement has been extended to permit use of masking expressions.

- The logical IF statement has been extended to permit use of masking expressions.

- The LOGICAL type variable has been extended so that it may represent all data legal for the INTEGER type.

- The following I/O editing codes have been added to the FORTRAN repertoire:

  (a) The nOw code, for transfer of octal coded integers

  (b) The \$$h_1,h_2,\ldots,h_w$\$ code which is similar to the Hollerith code $wHh_1,h_2,\ldots,h_w$ except that the field width need not be specified.

- The nLw editing code has been extended so that, on output, TRUE or FALSE is transferred provided that w is greater than 3 or 4, respectively.
- The library of external functions has been expanded (exceeding USASCII requirements) to permit full range of bit/word manipulation capabilities, including:
  - (a) The FLD function for manipulation of a specified field within a computer word
  - (b) The logical OR, AND, COMPL ("NOT") and XOR (Exclusive OR) functions
  - (c) The LSH and RSH functions for a left shift or a right shift, respectively, of bits within a computer word.

4.11.3.2. Utilization

The basic design objective of the 494 FORTRAN IV compiler is to create a rapid processor which, at the same time, will produce an object program optimized with respect to both storage requirements and execution time. The compiler is drum oriented and achieves its performance without the use of magnetic tape.

The UNIVAC 494 FORTRAN IV compiler translates FORTRAN source statements into RB elements, which, as subprograms in the user's job library, under control of facilities for interprogram communications, permit separately compiled programs or subprograms to be linked together at load time. A job to be executed may be composed of many subprograms (or elements). The program may be constructed and executed, and any of its elements or constructions may be saved on appropriate peripheral devices for later or additional executions. The 494 FORTRAN IV RB is compatible with other processor output (such as, SPURT, ASM, COBOL), promoting subprogram generation in any language the user desires.

The 494 FORTRAN IV compiler utilizes the Elementary Mathematical Functions Library which includes an extensive collection of basic mathematical subroutines and functions. This collection includes all of the standard FORTRAN functions and has been expanded to give the programmer coverage of the more often used mathematical routines. Each of these mathematical routines has been carefully developed to offer the programmer maximum accuracy range with minimum routine size and execution time.

Full information concerning the UNIVAC 494 FORTRAN IV compiler and the Elementary Mathematical Functions may be found in *UNIVAC 490/491/492/494 Real-Time System FORTRAN IV Programmers Reference Manual, UP-4087* (current version) and *UNIVAC 490/491/492/494 Real-Time System Elementary Mathematical Functions, UP-4150 (current version).*

4.11.4. COBOL Compiler

The UNIVAC 494 COBOL compiler is an adaptation to the 494 System and an extension of COBOL-61 EXTENDED, as outlined by the CODASYL Maintentance Committee, with which the compiler is in full compliance.

The compiler operates as a task under control of the executive routine, and is called and activated by the COB statement.

COBOL (COmmon Business Oriented Language), provides a method for stating computer problems and solutions in English. The language comprises a basic set of English words and symbols used to define and create a program. COBOL further permits the definition and naming of data according to the individual dictates of the user rather than the peculiarities of the computer, and forms a functional language that is largely independent of the computer make or model.

The COBOL language program employs many types of words, including nouns which are established by the programmer to name the various data elements upon which the program will operate, verbs which are supplied by the COBOL system to direct the manner in which the data will be treated, and certain selected words to improve the readability of the language or to complete the meaning of a sentence or expression.

A COBOL source program consists basically of four divisions: the IDENTIFICATION DIVISION which identifies the program, programmer, security, date written, and any other information desired to describe the program; the ENVIRONMENT DIVISION which specifically names the source and object computers used as well as the peripheral devices necessary for execution (this section is *machine dependent* as it describes the interface between the compiler and the object program for a specific hardware configuration); the DATA DIVISION which contains a complete description of all data elements including files for communication with peripherals, working areas, constant areas, and common storage areas; and the PROCEDURE DIVISION which contains the program logic as described in the procedural statements. The statements are organized to form sections, paragraphs, and sentences which fully describe what the programmer intends to accomplish. As in English, COBOL verbs designate the action to be taken on the defined data to produce desired results.

Within the requirements of COBOL-61 EXTENDED, the UNIVAC 494 COBOL compiler adds and implements a considerable number of elective elements to aid the programmer. Among the electives are the SORT, COMPUTE, and ENTER verbs; the MOVE, SUBTRACT, and ADD verbs with CORRESPONDING option; the WRITE verb with ADVANCING option; and SIGN OVERPUNCH handling capabilities.

To enhance the speed of compilation, the basic compiler design stresses inline code rather than excessive subroutine linkages. Heavily used procedures, however, may be implemented as object time subroutines. When activated, the compiler requests additional primary storage and random access storage for tables and scratch files. Compilation speed is to some extent dependent upon the amount of additional core available and the grade of mass storage used, but, in general, compilation speed is about 700 source statements per minute.

Compiler output consists of RB elements placed in the user's job library and a source element describing the collection of the RB elements according to the user's segmentation scheme.

Additional information describing the 494 COBOL compiler may be found in *UNIVAC 490/491/492/494 Real-Time System COBOL Supplementary Reference, UP-7608* (current version).

APPENDIX A. INSTRUCTION CODES AND j-K DESIGNATORS

The following tables list the instruction repertoire for the UNIVAC 494 Real-Time System, and list the designators for modification of the instruction execution sequence and the instruction operand.

Table A–1 presents the instruction repertoire on the basis of instruction type, mnemonic codes for both 494 SPURT and 494 ASM assembly languages, octal function codes, operations performed, instruction class, execution time in nanoseconds, and variables or modifications.

Table A–2 presents the j designator, which specifies skip, jump, repeat, and other sequence modifications, for normal and exception interpretation, in terms of octal and mnemonic codes for the assembly languages, and shows the results of the operations.

Table A–3 presents the k designator, which specifies operand modification or formation of the operand, Y, for both normal and exception interpretation, on the basis of instruction class, portion of the operand affected, and the type of operation performed.

TYPE	MNEMONIC CODE		OCTAL CODE	OPERATION	CLASS	EXECUTION TIME (NANOSECONDS)				VARIABLES
	ASM	SPURT				ALTERNATE BANK	SAME BANK k = 0,4	SAME BANK k = 7	SAME BANK $k \neq 0,4,7$	
TRANSFER	LQ	ENT*Q	10	$Y \rightarrow Q$	Rd	750	750	750	1500	
TRANSFER	LA	ENT*A	11	$Y \rightarrow A$	Rd	750	750	750	1500	
TRANSFER	LB	ENT*B_j	12	$Y \rightarrow B_j$	Rd	750/857	750/857	750/857	1500	NO CFL/CFL①
TRANSFER	ZB	CL*B_j	12	$0's \rightarrow B_j$	Rd	750	750	750	1500	
TRANSFER	NOP	NO-OP	12	NO OPERATION	Rd	750	750	750	1500	
TRANSFER	SQ	STR*Q	14	$Q \rightarrow Y$	St	750	750	1500	1500	
TRANSFER	ZQ	CL*Q	ASM/SP 16/10	$0's \rightarrow Q$	St/Rd	750	750	–	–	
TRANSFER	NQ	CP*Q	14	$Q' \rightarrow Q$	St	750	750	–	–	k is 0
TRANSFER	SA	STR*A	15	$A \rightarrow Y$	St	750	750	1500	1500	
TRANSFER	NA	CP*A	15	$A' \rightarrow A$	St	750	750	–	–	k is 4
TRANSFER	ZA	CL*A	ASM/SP 21/11	$0's \rightarrow A$	Rd	750/857	–	750/857	–	
TRANSFER	SB	STR*B_j	16	$(B_j) \rightarrow Y$	St	750	750	1500	1500	
TRANSFER	SZ	CL*Y	16	$0's \rightarrow Y$	St	750	750	1500	1500	
TRANSFER	DPL	DPENT	7721	$(Y, Y+1) \rightarrow AQ$	–	1500	2250			
TRANSFER	DPS	DPSTR	7725	$AQ \rightarrow (Y, Y+1)$	–	1500	2250			
TRANSFER	CPL	CREL	7731	$(Y)_{5-0} \rightarrow A_{29-24}, \ldots, (Y+4)_{5-0} \rightarrow A_{5-0}$	–	3750	4500			
TRANSFER	CPU	CREU	7732	$(Y)_{20-15} \rightarrow A_{29-24}, \ldots, (Y+4)_{20-15} \rightarrow A_{5-0}$	–	3750	4500			
TRANSFER	CUL	CRSL	7735	$A_{29-24} \rightarrow (Y)_{5-0}, \ldots, A_{5-0} \rightarrow (Y+4)_{5-0}$	–	3750	4500			
TRANSFER	CUU	CRSU	7736	$A_{29-24} \rightarrow (Y)_{20-15}, \ldots, A_{5-0} \rightarrow (Y+4)_{20-15}$	–	3750	4500			
SHIFT	RSQ	RSH*Q	01	SHIFT Q RIGHT } WITH	Rd	750	750	750	1500	
SHIFT	RSA	RSH*A	02	SHIFT A RIGHT } SIGN	Rd	750/964	750/964	750/964	1500/1714	$j \neq 4, 5/j = 4, 5$
SHIFT	RSAQ	RSH*AQ	03	SHIFT AQ RIGHT } EXT.	Rd	964	964	964	1714	
SHIFT	LSQ	LSH*Q	05	SHIFT Q LEFT } BY	Rd	750/857	750/857	750/857	1500/1607	$j \neq 2, 3/j = 2, 3$
SHIFT	LSA	LSH*A	06	SHIFT A LEFT } CIRCULARLY } Y	Rd	750/857	750/857	750/857	1500/1607	$j \neq 6, 7/j = 6, 7$
SHIFT	LSAQ	LSH*AQ	07	SHIFT AQ LEFT } POSITIONS	Rd	1071	1071	1071	1821	
SHIFT	LRSQ	LRSQ	7751	SHIFT Q RIGHT } WITH	–	750	1500			
SHIFT	LRSA	LRSA	7755	SHIFT A RIGHT } ZERO	–	750	1500			
SHIFT	LRSAQ	LRSAQ	7756	SHIFT AQ RIGHT } FILL	–	857	1607			
COMPARE	TA	COM*A	04	COMPARE Y TO A AND SKIP PER j	Rd	750	750	750	1500	
COMPARE	TQ	COM*Q	04	COMPARE Y TO Q AND SKIP PER j	Rd	750	750	750	1500	
COMPARE	TR	COM*AQ	04	SKIP PER j IF Y RANGES BETWEEN A AND Q	Rd	750/964	750/964	750/964	1500/1714	$j = 1/j = 4, 5$
COMPARE	TLP	COM*MASK	43	COMPARE A TO LP(Y·Q) AND SKIP PER j	Rd	750/964	750/964	750/964	1500/1714	$j=4,5,6,7/j=0,1,2,3$
COMPARE	DPTE	DPCME	7723	SKIP IF AQ = (Y, Y + 1)	–	1500	2250			
COMPARE	DPTL	DPCML	7727	SKIP IF AQ < (Y, Y + 1)	–	1500	2250			
COMPARE	MATE	MACE	7753	SKIP IF LP(A·Q) = LP(Y·Q)(ALPHA TEST)②	–	750	1500			
COMPARE	MATL	MACL	7757	SKIP IF LP(A·Q) < LP(Y·Q) (ALPHA TEST)②	–	750	1500			
JUMP	J	JP	60 or 61	$Y \rightarrow P$ IF j CONDITION SATISFIED	Rd	IF JUMP CONDITION SATISFIED				
JUMP		RIL		RELEASE INTERRUPT LOCKOUT		750/1500	750	1500	1500	$k = 0, 4/k \neq 0, 4$
JUMP		RILJP		RELEASE INTERRUPT LOCKOUT AND $Y \rightarrow P$		IF JUMP CONDITION NOT SATISFIED				
JUMP						750	750	750	1500	
JUMP	SLJ	RJP	65	$P + 1 \rightarrow Y$, $Y + 1 \rightarrow P$ PER j AND KEY	Rd	IF JUMP CONDITION SATISFIED				
JUMP						1500/2250	1500	2250	2250	$k = 0, 4/k \neq 0, 4$
JUMP						IF JUMP CONDITION NOT SATISFIED				
JUMP						750	1500	750	1500	
JUMP	SLJT	SIL, RJP OR SILRJP	64	$P + 1 \rightarrow Y$, $Y + 1 \rightarrow P$ PER j; SIL PER j	Rd	SAME AS 65 INSTRUCTION				
JUMP	ER	XQT	7737	(Y) = NI; RETURN IF NO SKIP OR JUMP AT NI	–	750	750			
JUMP	LBPX	EBJP*Bx	774x	$P \rightarrow Bx$, $Y \rightarrow P$	–	750	750			

Table A–1. Instruction Repertoire
(Part 1 of 3)

TYPE		MNEMONIC CODE		OCTAL CODE	OPERATION	CLASS	EXECUTION TIME (NANOSECONDS)				VARIABLES
		ASM	SPURT				ALTERNATE BANK	SAME BANK k = 0,4	k = 7	k = 0,4,7	
SEQUENCE MODIFYING		R	RPT	70	REPEAT NI Y TIMES; $Y \rightarrow B7$	Rd	1285/1500	1285	1285	1500	$k = 0,4,7/k \neq 0,4,7$
		TBI	$BSK*B_j$	71	$IF (B_j) = Y$, 0's $\rightarrow B_j$ SKIP; $IF (B_j) \neq Y, (B_j)+1 \rightarrow B_j$	Rd	750	750	750	1500	
		JBD	$BJP*B_j$	72	$IF (B_j) \neq 0, (B_j)-1 \rightarrow B_j$ AND JUMP	Rd			JUMP		
					$IF (B_j) = 0$, NI		750/1500	750	1500	1500	$k = 0,4/k \neq 0,4$
									NO JUMP		
							750	1500	750	1500	
		TSET	TSET	7752	IF $Y_{14} = 0$, 1's $\rightarrow Y_{14-0}$; IF $Y_{14} = 1$, INTERRUPT	–	1821		1821		
		EXRN	EXRN	7754	INTERRUPT	–	1285		1285		
ARITHMETIC	FIXED POINT BINARY	A	ADD*A	20	$A + Y \rightarrow A$	Rd	750/857	750/857	750/857	1500/1607	$j \neq 6,7/j = 6,7$
		AQ	ADD*Q	26	$Q + Y \rightarrow Q$	Rd	750/857	750/857	750/857	1500/1607	$j \neq 6,7/j = 6,7$
		LAQ	ENT*Y+Q	30	$Y + Q \rightarrow A$	Rd	750/857	750/857	750/857	1500/1607	$j \neq 6,7/j = 6,7$
		SAQ	STR*A+Q	32	$A + Q \rightarrow Y, A$	St	1500	1500	–	2250	
		RA	RPL*A+Y	24	$Y + A \rightarrow Y, A$	Rp	1500	1500	1500	2250	
		RAQ	RPL*Y+Q	34	$Y + Q \rightarrow Y, A$	Rp	1500	1500	1500	2250	
		RI	RPL*Y+1	36	$Y + 1 \rightarrow Y, A$	Rp	1500	1500	1500	2250	
		AN	SUB*A	21	$A - Y \rightarrow A$	Rd	750/857	750/857	750/857	1500/1607	$j \neq 6,7/j = 6,7$
		ANQ	SUB*Q	27	$Q - Y \rightarrow Q$	Rd	750/857	750/857	750/857	1500/1607	$j \neq 6,7/j = 6,7$
		LANQ	ENT*Y–Q	31	$Y - Q \rightarrow A$	Rd	750/857	750/857	750/857	1500/1607	$j \neq 6,7/j = 6,7$
		SANQ	STR*A–Q	33	$A - Q \rightarrow Y, A$	St	1500	1500	–	2250	
		RAN	RPL*A–Y	25	$A - Y \rightarrow Y, A$	Rp	1500	1500	1500	2250	
		RANQ	RPL*Y–Q	35	$Y - Q \rightarrow Y, A$	Rp	1500	1500	1500	2250	
		RD	RPL*Y–1	37	$Y - 1 \rightarrow Y, A$	Rp	1500	1500	1500	2250	
		DPA	DPADD	7722	$AQ + (Y, Y + 1) \rightarrow AQ$	–	1500		2250		
		DPAN	DPSUB	7726	$AQ - (Y, Y + 1) \rightarrow AQ$	–	1500		2250		
		DPN	DPCP	7724	$CP (AQ) \rightarrow AQ$	–	750		1500		
		M	MUL	22	$Q \times Y \rightarrow AQ$	Rd	7277	7277	7277	8027	
		D	DIV	23	$(AQ/Y) \rightarrow Q$, REMAINDER IN A	Rd	7277	7277	7277	8027	
	FLOATING POINT BINARY	SFS	SGSH	7730	NORMALIZE A, SHIFT COUNT $\rightarrow Q$	–	857		1607		
		FA	FADD	7701	$AQ_{FP} + (Y, Y + 1)_{FP} \rightarrow AQ_{FP}$	–	2356/2998		2891/3641		$SUM = \pm 0/ \neq \pm 0$
		FAN	FSUB	7702	$AQ_{FP} - (Y, Y + 1)_{FP} \rightarrow AQ_{FP}$	–	2356/2998		2891/3641		$DIF. = \pm 0/ \neq \pm 0$
		FM	FMUL	7703	$AQ_{FP} \times (Y, Y + 1)_{FP} \rightarrow AQ_{FP}$	–	12093		12843		
		FD	FDIV	7705	$AQ_{FP} / (Y, Y + 1)_{FP} \rightarrow AQ_{FP}$	–	12200		12950		$\|M\|_{AQ} \geq \|M\|_{Y, Y+1}$
							12414		13164		$\|M\|_{AQ} < \|M\|_{Y, Y+1}$
							1607		2357		$\|(Y, Y + 1)\| = 0$
		FP	FPP	7706	$(Y)_C$ AND $AQ_M \rightarrow AQ_{FP}$	–	857/1500		1607/2250		$AQ = \pm 0/AQ \neq \pm 0$
		FU	FPU	7707	$AQ_{FP} \rightarrow Y_C$ AND AQ_M	–	750		1500		
	FIXED POINT DECIMAL	DT	DTEST	7710	SKIP PER Y	–	750		1500		
		DA	DADD	7711	$AQ_D + (Y, Y + 1)_D \rightarrow AQ_D$	–	2035/2249		2785/2249		SIGNS = /SIGNS ≠
		DAC	DADDC	7715	$AQ_D + (Y, Y + 1)_D$ + CARRY $\rightarrow AQ_D$	–	2035		2785		SIGNS = OR $\|AQ\| < \|Y, Y + 1\|$ ⑤
							2249		2999		SIGNS ≠ AND $\|AQ\| \geq \|Y, Y + 1\|$ ⑤
		DAN	DSUB	7712	$AQ_D - (Y, Y + 1)_D \rightarrow AQ_D$	–	2035/2249		2785/2999		SIGNS = / SIGNS ≠
		DANB	DSUBB	7716	$AQ_D - (Y, Y + 1)_D$ – BORROW $\rightarrow AQ_D$	–	2035		2785		SIGNS ≠ OR $\|AQ\| < \|Y, Y + 1\|$ ⑤
							2249		2999		SIGNS = AND $\|AQ\| \geq \|Y, Y + 1\|$ ⑤
		DN	DCP	7714	9's OR 10's COMPLEMENT (AQ) $\rightarrow$ AQ PER Y	–	1285		1500		
		DTE	DCME	7713	SKIP IF $AQ_D = (Y, Y + 1)_D$	–	1500		2250		
		DTL	DCML	7717	SKIP IF $AQ_D < (Y, Y + 1)_D$	–	1500		2250		
		DCL	DCVL.	7733	$(Y_{0-5}$ THRU $Y + 4_{0-5})_D \rightarrow AQ_{BINARY}$	–	3750		4500		
		DCU	DCVU	7734	$(Y_{15-20}$ THRU $Y + 4_{15-20})_D \rightarrow AQ_{BINARY}$	–	3750		4500		

Table A–1. Instruction Repertoire
(Part 2 of 3)

TYPE	MNEMONIC CODE		OCTAL CODE	OPERATION	CLASS	EXECUTION TIME (NANOSECONDS)				VARIABLES
						ALTERNATE BANK	SAME BANK			
	ASM	SPURT					k = 0,4	k = 7	k = 0,4,7	
LOGICAL	LLP	ENT*LP	40	LP (Y.Q) → A	Rd	750/1071	750/1071	750/1071	1500/1820	$j \neq 2,3/j = 2,3$
	SAND	STR*LP	47	LP (Q·A) → Y	St	750	750	–	1500	
	RLP	RPL*LP	44	LP (Q·Y) → Y,A	Rp	1500	1500	1500	2250	
	ALP	ADD*LP	41	LP (Y·Q) + A → A	Rd	750/857	750/857	750/857	1500/1607	$j \neq 6,7/j = 6,7$
	RALP	RPL*A+LP	45	LP (Y·Q) + A → Y,A	Rp	1500	1500	1500	2250	
	ANLP	SUB*LP	42	A–LP (Y·Q) → A	Rd	964	964	964	1714	
	RANLP	RPL*A-LP	46	A–LP (Y·Q) → Y,A	Rp	1607	1607	1607	4357	
	OR	SEL*SET	50	IF A_n OR $Y_n = 1$, $A_n = 1$	Rd	750	750	750	1500	
	ROR	RSE*SET	54	IF A_n OR $Y_n = 1$, $A_n = 1$; A → Y	Rp	1500	1500	1500	2250	
	XOR	SEL*CP	51	IF EITHER A_n OR $Y_n = 1$, 1 → A_n	Rd	750	750	750	1500	
	RXOR	RSE*CP	55	IF EITHER A_n OR $Y_n = 1$, 1 → A_n; A → Y	Rp	1500	1500	1500	2250	
	NOT	SEL*CL	52	IF $Y_n = 1$, CLEAR A_n	Rd	750	750	750	1500	
	RNOT	RSE*CL	56	IF $Y_n = 1$, CLEAR A_n; A → Y	Rp	1500	1500	1500	2250	
	SSU	SEL*SU	53	IF $Q_n = 1$, Y_n → A_n	Rd	750	750	750	1500	
	RSSU	RSE*SU	57	IF $Q_n = 1$, Y_n → A_n; A → Y	Rp	1500	1500	1500	2250	
INPUT/OUTPUT	LC	EXT-FCT *C0*Y	13	Y AND EXTERNAL FUNCTION → CH_{CSR}	Rd	750/2889 (3)	750/2889 (3)			$\hat{k} = 0,1,2/\hat{k} = 3$
	SC	STR*C0*Y	17	INPUT CH_{IASR} → Y	St	750/2884 (3)	750/2889 (3)			$\hat{k} = 0,1,2/\hat{k} = 3$
	JIC	JP*Y*	62	Y → P IF INPUT CH_{CSR} ACTIVE	Rd	IF JUMP CONDITION SATISFIED				
		C0*				750/1500	750/1500			$\hat{k} = 0/\hat{k} = 1,2,3$
		ACTIVE IN				IF JUMP CONDITION NOT SATISFIED				
						1500	1500			
	JOC	JP*Y*	63	Y → P IF OUTPUT CH_{CSR} ACTIVE	Rd	IF JUMP CONDITION SATISFIED				
		ACTIVE OUT				750/1500	750/1500			$\hat{k} = 0/\hat{k} = 1, 2, 3$
						IF JUMP CONDITION NOT SATISIFED				
						750	750			
	DIC	TERM*C0	66	TERMINATE INPUT CH_{CSR}	Rd	750	750			
	DOC	*INPUT TERM*C0	67	TERMINATE OUTPUT CH_{CSR}	Rd	750	750			
	LIC	OUTPUT IN*C0*Y	73	ACTIVATE INPUT CH_{CSR}	Rd	750/1500	750/2250			$\hat{k} = 0$,ESI/ $\hat{k} = 3$, ISI
	LOC	OUT*C0 OR C1*Y	74	ACTIVATE OUTPUT OR EF CH_{CSR}	Rd	750/1500	750/2250			$\hat{k} = 0$,ESI/ $\hat{k} = 3$, ISI
	LICM	IN*C0*Y *MONITOR	75	SAME AS 73, WITH MONITOR	Rd	750/1500	750/2250			$\hat{k} = 0$, ESI/ $\hat{k} = 3$, ISI
	LOCM	OUT*C0 OR C1*Y*MONITOR	76	SAME AS 74, WITH MONITOR	Rd	750/1500	750/2250			$\hat{k} = 0$, ESI/ $\hat{k} = 3$, ISI
TRANSFER (4)	LIFR	EIFR	7761	Y → IFR, (Y + 1) → RIR	–	1500	2250			
	LPLR	EPLR	7762	Y → PLR	–	750	1500			
	SIFR	SIFR	7765	IFR → Y	–	750	1500			
	LRIR	ERIR	7766	Y → RIR	–	750	1500			
	ISI	SSI	7770	SYNCHRONOUS INTERRUPT PER Y	–	750	750			
	LBW	EWB	7771	(Y) → B1,..., (Y + 6) → B7	–	5250	6000			
	SCN	SCHN	7772	I/O CH_{IASR} OR PECSR	–	750	1500			
	LCSR	ECSR	7773	Y_{4-0} → CSR	–	750	1500			
	SBW	SWB	7775	B1 → Y,..., B7 → (Y + 6)	–	5250	6000			

NOTES:

1 B CONFLICT ARISES WHEN B_j OF 12 INSTRUCTIONS $\neq$ 0 AND B_b OF NI = B_j OF 12 INSTRUCTION.

2 ALPHA TEST MEANS THAT SIGN BIT IS TREATED AS PART OF ABSOLUTE VALUE OF WORD.

3 TIME LISTED IS MINIMUM VALUE.

4 PRIVILEGED INSTRUCTIONS FOR USE IN 494 MODE.

5 $|x|$ INDICATES ABSOLUTE VALUE OF x.

Table A–1. Instruction Repertoire
(Part 3 of 3)

NORMAL j INTERPRETATION		
OCTAL CODE	MNEM. CODE	RESULT
0		NO SKIP
1	SKIP	SKIP
2	QPOS	SKIP IF (Q) POSITIVE
3	QNEG	SKIP IF (Q) NEGATIVE
4	AZERO	SKIP IF (A) = 0
5	ANOT	SKIP IF (A) ≠ 0
6	APOS	SKIP IF (A) POSITIVE
7	ANEG	SKIP IF (A) NEGATIVE

EXCEPTIONS

OCTAL CODE	TA/TQ/TR		J, SLJ		JT/SLJT		ASM
	COM*A/COM*Q/COM*AQ		JP,RJP		JP/RJP		SPURT
	MNEM.	RESULTS[5]	MNEM.	RESULTS	MNEM.	RESULTS	
0	x	x		JUMP	RIL/SIL[2]	RIL/SIL	
1	SKIP[6]	SKIP	KEY 1	JUMP IF KEY 1 SET	RILJP/SILRJP[1]	RIL,JUMP/SIL,RETURN JUMP	
2	YLESS	x/SKIP IF Y≤Q/x	KEY 2	JUMP IF KEY 2 SET	QPOS	JUMP IF (Q) POS.	
3	YMORE	x/SKIP IF Y>Q/x	KEY 3	JUMP IF KEY 3 SET	QNEG	JUMP IF (Q) NEG.	
4	YIN	x/x/SKIP IF A<Y≤Q	STOP	STOP	AZERO	JUMP IF (A) = 0	
5	YOUT	x/x/SKIP IF Y>Q OR Y≤A	STOP 5	STOP IF KEY 5 SET	ANOT	JUMP IF (A) ≠ 0	
6	YLESS	SKIP IF Y≤A/x/x	STOP 6	STOP IF KEY 6 SET	APOS	JUMP IF (A) POS.	
7	YMORE	SKIP IF Y>A/x/x	STOP 7	STOP IF KEY 7 SET	ANEG	JUMP IF (A) NEG.	

OCTAL CODE	D		R		LLP, RLP		ASM
	DIV		RPT		ENT*LP, RPL*LP		SPURT
	MNEM.	RESULTS	MNEM.[3]	RESULT	MNEM.	RESULT	
0		NO SKIP		$\bar{y}_{NE} = \bar{y}$		NO SKIP	
1	SKIP	SKIP	ADV	$\bar{y}_{NE} = \bar{y} + 1$	SKIP	SKIP	
2	NOOF	SKIP IF NO OVERFLOW	BACK	$\bar{y}_{NE} = \bar{y} - 1$	EVEN	SKIP IF (A)=EVEN NO. OF 1 BITS	
3	OF	SKIP IF OVERFLOW	ADDB	$\bar{y}_{NE} = \bar{y} + n \times (B_b)$	ODD	SKIP IF (A)=ODD NO. OF 1 BITS	
4	AZERO	SKIP IF (A) = 0	R	$\bar{y}_{NE} = \bar{y}$	AZERO	SKIP IF (A) = 0	
5	ANOT	SKIP IF (A) ≠ 0	ADVR	$\bar{y}_{NE} = \bar{y} + 1$	ANOT	SKIP IF (A) ≠ 0	
6	APOS	SKIP IF (A) POS.	BACKRNE/BACKR[4]	$\bar{y}_{NE} = \bar{y} - 1$	APOS	SKIP IF (A) POSITIVE	
7	ANEG	SKIP IF (A) NEG	ADDBR	$\bar{y}_{NE} = \bar{y} + n \times (B_b)$	ANEG	SKIP IF (A) NEGATIVE	

OCTAL CODE	AQ, ANQ		ASM
	ADD*Q, SUB*Q		SPURT
	MNEM.	RESULTS 7	
0		NO SKIP	
1	SKIP	SKIP	
2	APOS	SKIP IF $A_S = +$	
3	ANEG	SKIP IF $A_S = -$	
4	QZERO	SKIP IF Q = +0	
5	QNOT	SKIP IF Q ≠ +0	
6	QPOS	SKIP IF $Q_S = +$	
7	QNEG	SKIP IF $Q_S = -$	

NOTES:

[1] FOR SPURT, EQUIVALENT MNEMONICS ARE RILJP/SILRJP.

[2] RIL = RELEASE INTERRUPT LOCKOUT; SIL = SET INTERRUPT LOCKOUT.

[3] DESIGNATORS OF 0, 1, 2, AND 3 USED WITH READ OR STORE CLASS INSTRUCTIONS.
DESIGNATORS OF 4, 5, 6, AND 7 USED WITH REPLACE CLASS INSTRUCTIONS.
$\bar{y}_{NE}$ REPRESENTS 1/y OF NEXT EXECUTION.
RESULT OF REPEATED REPLACE CLASS USING $\bar{y}_{NE}$ IS STORED AT $\bar{y}_{NE} + (B_6)$.

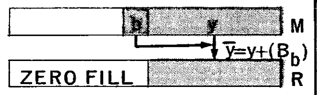
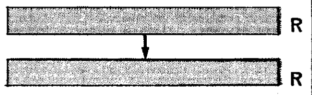
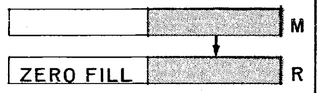
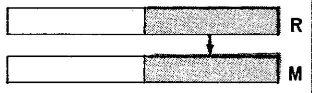
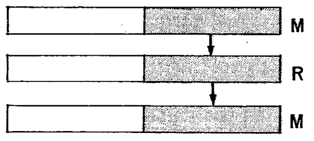
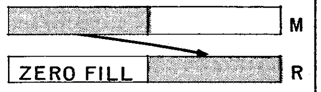
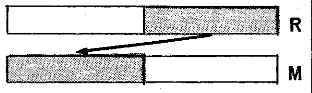
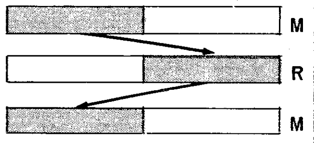
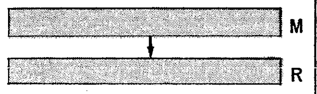
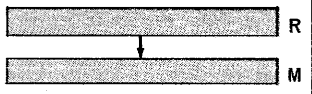
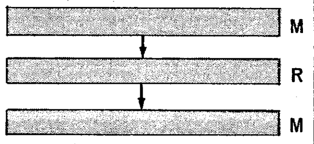
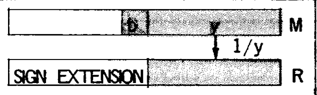
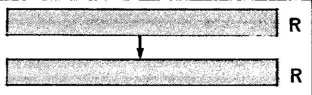
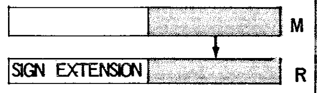
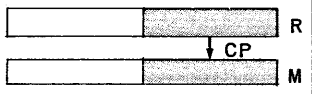
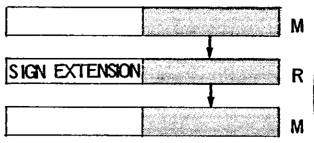
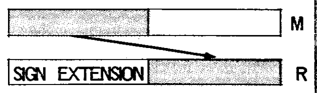
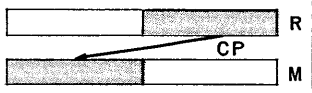
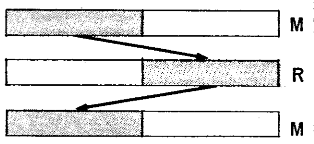
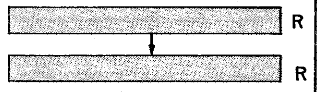
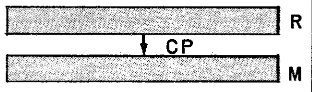
[4] ASM/SPURT

[5] X INDICATES ILLEGAL OR INVALID USE

[6] NOT USED IN SPURT

[7] A_S = SIGN BIT OF A: Q_S SIGN BIT OF Q (0 FOR POSITIVE, 1 FOR NEGATIVE)

Table A–2. Interpretation of j Designator

NORMAL Y FORMATION BY k DESIGNATOR						
k	READ CLASS		STORE CLASS		REPLACE CLASS	
0			Q			
1	L(x)		L(x)		L(x)	
2	U(x)		U(x)		U(x)	
3	W(x)		W(x)		W(x)	
4	X		A			
5	LX(x)		CPL(x)		LX(x)	
6	UX(x)		CPU(x)		UX(x)	
7	A		CPW(x)			

EXCEPTIONS

k	J, SLJ, SLJT, JBD	RSQ, RSA, RSAQ, LSQ, LSA, LSAQ	LB j = 4,5,6,7	LB j = 1,2,3	LB j=0	SA	SB	SQ	R	SAND	ASM
k	JP, RIL, RJP, SIL, BJB*B$_j$	RSH*Q, RSH*A, RSH*AQ, LSH*Q, LSH*A, LSH*AQ	ENT*B$_j$ j = 4,5,6,7	ENT*B$_j$ j = 1,2,3	ENT*B$_j$ j=0	STR*A	STR*B$_j$	STR*Q	RPT	STR*LP	SPURT
0	$Y=\bar{y}$	$Y=\bar{y}_{0-5}$	$Y=\bar{y}_{0-16}$	$\bar{y}_L$	NO OPERATION		$(B_j)\rightarrow Q_L$, ZERO FILL	$CP(Q)\rightarrow Q$	$Y=\bar{y}_{0-16}$	$LP\rightarrow Q$	
1	$Y=(\bar{y})_L$	$Y=(\bar{y})_{0-5}$	$Y=(\bar{y})_L$, ZERO FILL	$(\bar{y})_L$					$Y=(\bar{y})_L$	$LP_L\rightarrow(\bar{y})_L$	
2	$Y=(\bar{y})_U$	$Y=(\bar{y})_{15-20}$	$Y=(\bar{y})_U$, ZERO FILL	$(\bar{y})_U$					$Y=(\bar{y})_U$	$LP_L\rightarrow(\bar{y})_U$	
3	$Y=(\bar{y})_L$	$Y=(\bar{y})_{0-5}$	$Y=(\bar{y})_{0-16}$	$(\bar{y})_L$			$(B_j)\rightarrow(\bar{y})_L$		$Y=(\bar{y})_{0-16}$	$LP\rightarrow(\bar{y})$	
4	$Y=\bar{y}$	$Y=\bar{y}_{0-5}$	$Y=\bar{y}_L$, SIGN EXT.	$\bar{y}$		$CP(A)\rightarrow A$	$(B_j)\rightarrow A_L$, ZERO FILL		$Y=(\bar{y})_L$	$LP\rightarrow A$	
5	$Y=(\bar{y})_L$	$Y=\bar{y}_{0-5}$	$Y=(\bar{y})_L$, SIGN EXT.	$(\bar{y})_L$					$Y=(\bar{y})_L$	$CP(LP)_L\rightarrow(\bar{y})_L$	
6	$Y=(\bar{y})_U$	$Y=\bar{y}_{15-20}$	$Y=(\bar{y})_U$, SIGN EXT.	$(\bar{y})_U$					$Y=(\bar{y})_U$	$CP(LP)_L\rightarrow(\bar{y})_U$	
7	$Y=A_L$	$Y=A_{0-5}$	$Y=A_{0-16}$	A_L			$CP(B_j)\rightarrow(\bar{y}_L)$, SIGN EXT.		$Y=A_{0-16}$	$CP(LP)\rightarrow(\bar{y})$	

NOTES:

1. M indicates word in memory; R indicates word in register.
2. Designation $\bar{y}$ is sum of (B_b) and y; designation $(\bar{y})$ is contents of word at location $\bar{y}$.
3. Z_L is lower 15 bits of word Z; Z_U is upper 15 bits of word Z.

Table A–3. Interpretation of k Designator

UNIVAC

SPERRY RAND