UNIVAC

DATA PROCESSING DIVISION

494

REAL-TIME SYSTEM

SYSTEM DESCRIPTION

REFERENCE MANUAL

UP-4032

This manual is published by the UNIVAC Division of Sperry Rand Corporation in loose leaf format as a rapid and complete means of keeping recipients apprised of UNIVAC ® Systems developments. The UNIVAC Division will issue updating packages, utilizing primarily a page-for-page or unit replacement technique. Such issuance will provide notification of hardware and/or software changes and refinements. The UNIVAC Division reserves the right to make such additions, corrections, and/or deletions as in the judgment of the UNIVAC Division, are required by the development of its respective Systems.

PRINTED IN U.S.A.

1. CONTENTS

2. INTRODUCTION

The UNIVAC 494 Real-Time System is a high-speed, user-oriented computing system which has, as its outstanding capability, the ability to efficiently respond to the demands of real-time processing with a priority structure which controls allocation of processing time as required by the operational needs of real-time programs. The system is modular, in construction, to facilitate future extensions, expansion of particular functions, and selection of variations of a basic function. This modularity permits each user to create a system specially geared to particular system requirements. Users of the UNIVAC 490, 491, and 492 systems require little effort to switch existing programs to the UNIVAC 494 system. Outstanding features of the 494 system include:

- Real-time/on-line operational environment
- Multi-programming capability for mix of real-time and batch processing
- Collection of independent program elements into one integrated object program
- A complete set of programming languages including COBOL, FORTRAN IV, and assembly languages
- Utilization of random access storage for program construction and storage
- Flexible priority structure balancing service for standard production and remote programs
- Random access buffering of input-output information
- On-line maintenance tests to minimize periodic preventive-maintenance procedures
- Complete system integrity achieved by memory lock-out, guard mode, and software validation of service requests

A. CENTRAL PROCESSOR

The central processor has the following capabilities:

- Compatability with UNIVAC 490, 491, and 492 programs and peripheral equipment
- Program protection preventing interaction of unrelated programs
- 12 full-word (30-bit) send/receive simplex input-output channels, expandable in increments of 4, to 24 input-output channels
- Multiplexing (time-sharing) on the simplexed communication channels, permitting interleaving of messages on same channel
- Direct coupling to a series of standard peripheral devices
- Time orientation-to a day clock, enabling conditioning of computer operations by the time of day; to an interval timer, for precision timing of computer operations with unconditional interrupt; to a real time clocks, for timing without interrupt
- 750-nanosecond cycle-time ferrite core memory with capacity of 16,384 30-bit words (plus internally-generated parity bit per half-word), field-expandable to 131,072 words
- Relative addressing providing relocation anywhere within memory, without changes to program
- High-speed processing due to *memory overlap* (permitting next instruction to be read from memory, simultaneously with last operand of current instruction), 14 index registers for effective addressing with *effective and relative addressing arithmetic circuits independent of main arithmetic circuits*, and *internal parallel bit transfer of data*
- Arithmetic operation in fixed-point binary mode (both single and double precision) with optional fixed-point binary-coded-decimal (BCD) mode easily programmed for multi-precision operations, and double-precision floating-point binary mode
- Representative times for a series of arithmetic instructions:

Fixed-point add to accumulator	750 nanoseconds
Fixed-point multiply	7.3 microseconds
Fixed-point divide	7.4 microseconds
Floating-point add to AQ register	3.2 microseconds
Floating-point multiply	12.5 microseconds
Floating-point divide	13 microseconds

B. OPERATING SYSTEM

The Operating System for the UNIVAC 494 Real-Time System is a comprehensive set of integrated programs to encompass the significant common tasks and functions met by the user. The operating system through its basic concepts meets the objective of providing a flexible and reliable foundation for the installation to build upon. The system is an open-ended structure which will be added to by the user and by the Systems Programming staff of UNIVAC.

The major elements constituting the initial system are listed below.

- The Executive System

  The Executive System is that set of routines which provide the basic control and fundamental mechanisms of the operating system. Major elements of the operating system such as compilers, generators, and input/output control routines are executed and controlled by the Executive System. The Executive System is responsible for the maintenance of a multi-program environment which makes the most efficient use of the facilities available at any time according to a priority structure which favors real time processing.

- A Symbolic Language Assembler (SPURT)

  The assembler accepts source programs in the SPURT II Assembler language, a continuation of the UNIVAC 490 SPURT Assembler language, and produces object programs in UNIVAC 494 relocatable binary (RB) code format. This is standard program input to the UNIVAC 494 Real-Time System. Existing UNIVAC 490 SPURT programs may be reassembled for compatible operation.

- FORTRAN IV

  The FORTRAN language will conform to the ASA standard. The compiler will operate under executive control in a multi-program environment, and will produce the standard object code format suitable for collection with previously compiled FORTRAN program elements or other elements produced according to system standards.

- COBOL

  The COBOL Compiler will be in accord with UNIVAC Standard COBOL and will produce the standard object code format. This compiler will also operate under executive control in a multi-program environment.

- Interpretive/Supervisor

  The Interpretive/Supervisor provides complete control over programs in the debug stage. Run time parameters control information extraction and display at a symbolic level during operation of the program. An important feature is the protection of operating programs in the multi-program environment.

- Library Editor

  The Library Editor provides the installation with the ability to establish and alter libraries of programs, Programs may be entered into the library in source language or relative and absolute object code. Library Control will modify programs within a library or control the transfer of programs from one library to another.

- File Control

  File Control consists of a File Processor and the File Control sub-library. An input/output control language is provided which defines buffer and file structure to the system. References to files may be made in source code by using macro directives associated with file name. The File Control Processor establishes resident programs to link the user program with the basic executive input/output routines. Users who do not wish to use the File Control Processor may write calls in their source code using standard parameters.

- Systems Library

  A collection of routines which provide the user with a source of standard operations. The library includes mathematical, statistical, editing, data communication handling and miscellaneous general routines.

- On-Line Maintenance

  A collection of hardware checking routines run by the executive in a multi-program environment. These tests could reduce the fixed preventive maintenance period normally associated with operation of an installation.

- PERT/COST

  This applications tool is applicable to research and development projects in which time and cost estimates cannot be predicted with adequate certainty. Probabilistic concepts are used for time and cost reporting and control. Specifications are based upon the framework provided by DOD/NASA Guide to PERT/COST System Design.

- Linear Programming

  Linear Programming is an operations research technique widely used in manufacturing and government to minimize costs or maximize efficiency in the production and distribution of products. The algorithm in this application employs the ''product form of the inverse'' method improved with an advanced path selection technique.

- Network Simulator

  This program provides a general class of communications networks out of which the user can create a model to solve his specific problem. Models in this class solve such problems as computer design and switching network.

- New Assembly Language

  A new language will be developed specifically for the UNIVAC 494 Real-Time System. This language will take full advantage of the power and flexibility of the system.

C. PERIPHERAL SUBSYSTEMS

A subsystem consists of one or more peripheral units of the same type connected to an available input/output channel. Each subsystem is controlled by a channel synchronizer/ control unit that interprets the control signals and instructions issued by the Central Processor, effects the transfer of data to or from the selected unit and the Central Processor, indicates to the Central Processor the status of the available peripheral units, and informs the Central Processor when errors or faults that affect the operation of the subsystem occur. The Central Processor and the subsystems have capabilities which lead to great efficiency in their mutual operations.

When the main program requires that the Central Processor employ a subsystem, the Central Processor issues control signals which select the proper subsystem and initiate the desired action. Once this is done the execution of the main program automatically continues until the subsystem has completed the required action. At this point the subsystem signals the Central Processor that the action is complete and the Central Processor now deals with the results of the action taken: for example the processing of data transferred from the subsystem.

In similar manner a subsystem signals the Central Processor its state of readiness to require action on the part of the Central processor, such as response to an inquiry, and it also signals the Central Processor when its requirements have been met. These characteristics not only provide almost instantaneous availability of the services of the subsystems to the Central Processor, and those of the Central Processor to the subsystems, but they also reduce to at most a few thousandths of a second those Central Processor delays ordinarily associated with drum latency periods, magnetic tape reading or writing, or the employment of printing, punched card, or communications systems.

During the execution of input/output instructions the Central Processor proceeds with the main program, taking action on results of the operation only on receipt of a signal from the subsystem indicating that the operation is complete.

The following peripheral subsystems are available for use with the UNIVAC 494 Real-Time System:

- Random Access Storage

  Random access storage subsystems which offer a variety of access times, transfer rates, and storage capacities are available to meet the individual needs of the user.

- Magnetic Tape Subsystems

  The magnetic tape subsystems that are available have the ability to read or write tapes in binary coded decimal format.

- Card Subsystem

- High Speed Printer Subsystem

- Paper Tape Subsystem

- Communication Subsystem

  The communication subsystem allows the user to utililize a wide variety of communication equipment.

- Univac 1004 Subsystem

- Other Data Processing Systems

  Capabilities exist for on-line or remote linkage with other systems for time sharing operations.

3. CENTRAL PROCESSOR

The central processor receives data and instructions, processes the data in accordance with the program of instructions, and either stores the result for further processing or delivers the result to an external output device. The principal elements of the central processor are the core storage element, the control and interrupt processing element, the arithmetic unit, and the input/output processing unit. This chapter describes the repertoire of instructions, instruction word formats (including a sample instruction), core storage memory and addressing, the control section, input/output section, operator's console, and maintenance panel.

A. INSTRUCTION REPERTOIRE

The instruction repertoire (Table 3-1) of the UNIVAC 494 central processor consists of 99 basic instructions which are listed by function and numeric order of instruction code, together with the "class" of instruction. The class notation refers to the interpretation of the "k" designator within the instruction word. This k designator may have as many as eight (0–7) different values and partially determines the operand to be used for that particular instruction. (The 77 xx instructions do not use k designators and, therefore, require no class notation.) The instruction is further modified by the "j" designator within the instruction word which is usually used to describe the conditions for skipping the next sequential instruction. An operation initiated by a sample instruction is described in this section.

The principal registers referred to in Table 3–1 are the following:

The accumulator ("A" register), a 30- bit register used in all arithmetic and internal transfer operations. This register has incrementing and bit complementing properties and may also be used as a shift register.

The Q register, a 30-bit auxiliary arithmetic register, used, mainly, to aid the accumulator in multiplication, division, and logical operations. The Q register also has incrementing and bit complementing properties and may be used as a shift register.

The AQ register, a 60-bit register formed by the combined operation of the accumulator and Q register. The AQ register is used in arithmetic operations involving double precision, floating point, and decimal operands. The most significant portion of the 60-bit word is contained in the accumulator.

The index registers ("B" registers) of which there are 14. These registers may be incremented or decremented. The contents of an index register are added, when called for by the instruction, to the address in the instruction word to determine the "effective" address.

Unless Table 3-1 indicates otherwise, an instruction is compatible with both UNIVAC 490 and 494 operation. Where differences are indicated, the 490/494 mode switch on the maintenance panel must be appropriately set.

Instructions in Table 3-1 are grouped by function in the following sequence:

- Shift Instructions

  A shift instruction shifts the word stored in a selected register a specified number of bit positions. In a right-shift, the bits shifted out are lost; in a left-shift, the bits shifted out are returned, in turn, to the vacated positions (circular shift) unless specified otherwise.

- Transfer Instructions

  A transfer instruction either transfers data contained in memory to a register or transfers the contents of a register to a memory location. Unless otherwise specified, the original source remains unchanged.

- Arithmetic Instructions

  An arithmetic instruction performs an arithmetic operation upon a number in standard binary code (single and double precision), floating point mode (double-precision), or decimal mode. Included in this set of instructions are specific instructions for comparing or testing the result.

- Jump Instructions

  Jump instructions transfer program control to other sections of the program.

- Sequence-Modifying Instructions

  Sequence-modifying instructions enable repetition of an instruction a specified number of times, or skipping of the next instruction, or jumping to another section of the program.

- Logical Instructions

  Logical instructions enable operations to be performed only upon selected bits of a word. The term ''logical product'' associated with some of these instructions means that only if both bits involved in the operation are 1's the result is a 1, otherwise the result is a 0.

- Input/Output Instructions

  Input/output instructions are associated with the movement of data between the central processor and external equipment such as the standard communication subsystem and standard input/output devices.

- Interrupt Instruction

  The interrupt instruction enables programmed generation of an interrupt to synchronize operations with another computer or peripheral device.

	CODE	CLASS	FUNCTION
SHIFT	01	Read	Right-shift content of Q register by specified number of bits and fill vacated positions with sign bit.
	02	Read	Right-shift accumulator by specified number of bits and fill vacated positions with sign bit.
	03	Read	Right-shift AQ register (accumulator and Q register treated as one 60-bit register) by specified number of bits and fill vacated position with sign bit.
	05	Read	Left-shift, circularly, contents of Q register by specified number of bits.
	06	Read	Left-shift, circularly, contents of accumulator by specified number of bits.
	07	Read	Left-shift, circularly, contents of AQ register by specified number of bits.
	7730**	–	Left-shift, circularly, the accumulator until the two most significant bits are unequal and record the number of shifts required in the Q register.
TRANSFER	10	Read	Load contents of specified source into Q register.
	11	Read	Load contents of specified source into accumulator.
	12	Read	Load contents of specified source into specified index register.
	14	Store	Store contents of Q register at specified location.
	15	Store	Store contents of accumulator at specified location.
	16	Store	Store contents of specified index register at lower half of specified location, filling upper half with 0's.
	7721**	–	Load the AQ register with the contents of the two specified consecutive locations, the first location into the accumulator.
	7725**	–	Store the contents of the AQ register at the specified two consecutive locations, with the accumulator stored at the first location.
	7731**	–	Load the accumulator in consecutive six-bit positions starting from the left with bits 00-05 at each of the specified five consecutive locations.
	7732**	–	Load the accumulator in consecutive six-bit positions starting from the left with bits 20-15 of the contents at specified five consecutive locations.
	7735**	–	Store successive six-bit accumulator characters, one-by-one, starting from left of accumulator, into bit positions 00-05 at each of five selected consecutive addresses, clearing bit positions 06-14 to 0's at each of these addresses.
	7736**	–	Store successive six-bit accumulator characters, one-by-one, into bit positions 15-20 at each of five selected consecutive addresses, clearing bit positions 16-29 to 0's at each of these addresses.
	7761**	–	Load contents of specified address into Internal Function Register (IFR) to facilitate returning to a repeat sequence in case this sequence has been interrupted, to select a particular type of Guard mode, to indicate a decimal overflow and/or a carry from a decimal operation, and/or to condition operation of the index registers.
	7762**	–	Transfer contents of specified address to Memory Lock-In Register which, by use of Guard mode designator in the Internal Function Register, specifies the memory lock-in area.
	7765**	–	Store contents of Internal Function Register at specified address.
	7766**	–	Transfer bits 06-17 of designated memory address to Relative Index Register (RIR). If IFR designator is set, the contents of the RIR bias all core memory address references.
	7771**	–	Transfer the least significant contents of seven successive addresses (starting from the specified address) to the seven "worker" index registers B1-B7 in succession.
	7775**	–	Transfer the contents of the seven "worker" index registers to the seven successive core memory locations, starting from Index Register B1 and the memory location specified in the instruction.

Table 3-1. Instruction Repertoire

	CODE	CLASS	FUNCTION
ARITHMETIC	20	Read	Add contents of specified·address to accumulator and store sum in accumulator.
	21	Read	Subtract contents of specified address from accumulator and store difference in accumulator.
	22	Read	Multiply contents of Q register by contents of denoted address and store product in AQ reqister.
	23	Read	Divide contents of AQ register by contents of denoted address. Store quotient in Q register and remainder in accumulator.
	24	Replace	Add the contents of a selected address to the contents of the accumulator and store the sum in the selected address and the accumulator.
	25	Replace	Subtract the contents of a selected address from the contents of the accumulator and store this difference in the accumulator and the selected address.
	26	Read	Add contents of specified address to Q register and store sum in Q register. Accumulator is undisturbed.
	27	Read	Subtract contents of specified address from Q register and store difference in Q register. Accumulator is undisturbed.
	30	Read	Add contents of Q register to contents of specified address and store sum in accumulator. Q register is undisturbed.
	31	Read	Subtract contents of Q register from contents of specified address and store difference in accumulator. Q register is undisturbed.
	32	Store	Add contents of Q register to accumulator and store this sum at specified address.
	33	Store	Subtract contents of Q register from contents of accumulator and store this difference at specified address.
	34	Replace	Add the contents of a selected address to the contents of the Q register and store this sum in the accumulator and the selected address.
	35	Replace	Subtract the contents of the Q register from the contents of a selected address and store this difference in the accumulator and the selected address.
	36	Replace	Add one to the contents of a selected address and store this sum in the accumulator and the selected address.
	37	Replace	Subtract one from the contents of a selected address and store this difference in the accumulator and the selected address.
	7701**	–	Add the floating point number in the AQ register to the floating point number contained in the specified two consecutive locations and store the normalized sum in the AQ register.
	7702**	–	Subtract contents of specified two consecutive locations from contents of AQ register and store this difference in AQ register.
	7703**	–	Multiply the signed floating point number in the AQ register by the floating point number in the selected two consecutive addresses.

Table 3-1. Instruction Repertoire (continued)

	CODE	CLASS	FUNCTION
ARITHMETIC	7705**	–	Divide the normalized floating point number in the AQ register by the floating point number in the selected two consecutive addressed locations and store the normalized quotient in the AQ register, discarding the remainder.
	7706**	–	Transfer the unsigned exponent part contained in a selected address to the AQ register to make up a floating point number with the unnormalized fixed-point part contained in the AQ register.
	7707**	–	Store the absolute value of the exponent part contained in the AQ register in a specified location and fill these bit positions of the AQ register with the sign bit of the fixed-point part.
	7711**	–	Add the 10-digit decimal number in the specified two consecutive locations to the decimal number in the AQ register and store the sum in the AQ register.
	7712**	–	Subtract the 10-digit decimal number in the specified two consecutive locations from the decimal number in the AQ register and store the difference in the AQ register.
	7714**	–	Change the decimal number in the AQ register to its decimal complement. The decimal complement is the difference formed by a digit-by-digit subtraction where the least significant decimal digit is subtracted from 10, (10 minus 0 = 0, with no carry), and all higher-order digits from 9.
	7715**	–	Add the decimal number in the specified two consecutive locations to the decimal number in the AQ register including, in this addition, any carry from a previous decimal operation.
	7716**	–	Subtract the decimal number in the specified two consecutive locations from the decimal number in the AQ register including, in this subtraction, any borrows from a previous decimal operation.
	7722**	–	Add the contents of the two specified consecutive locations to the contents of the AQ register and store this sum in the AQ register (fixed-point binary).
	7724**	–	Complement each of the bits in the AQ register.
	7726**	–	Subtract the contents of the two specified consecutive locations from the contents of the AQ register and store this difference in the AQ register (fixed-point binary).
	04	Read	Compare signed value of selected 30-bit operand with contents in accumulator or Q register and skip next instruction if result of comparison meets specified requirements.
	7710**	–	Test decimal number in AQ register for sign, overflow, and/or digit values to determine if the next instruction shall be skipped.
	7713**	–	Compare decimal number in AQ register and decimal number in specified location; skip next instruction if AQ is equal.
	7717**	–	Compare decimal number in AQ register and decimal number in specified location; skip next instruction if AQ is less.
	7723**	–	Compare contents of AQ register with contents of specified location; skip next instruction if both are equal.
	7727**	–	Compare contents of AQ register with contents of specified location; skip next instruction if AQ is less.

Table 3-1. Instruction Repertoire (continued)

	CODE	CLASS	FUNCTION
JUMP	60	Read	Jump to specified location for next instruction if conditions of "j" designator and either Q register or accumulator are satisfied; otherwise execute next sequential instruction.
	61	Read	Jump to specified location under conditions set by the manually set JUMP or STOP controls; otherwise execute next sequential instruction.
	64	Read	Store address of next sequential instruction at specified location and jump to this specified location plus one for next instruction if jump conditions are satisfied; otherwise execute next sequential instruction.
	65*	Read	Store address of next sequential instruction at specified location and jump to specified location plus one for next instruction if manual controls are set; otherwise execute next sequential instruction.
SEQUENCE MODIFYING	7734**	–	Interrupt program and jump to location 000007 enabling the execution program to store the next instruction address of the program which is interrupting.
	7737**	–	Execute the instruction stored at the address specified by this instruction and, if that instruction does not skip or jump, return to the next instruction of the main program.
	70	Read	Repeat the next programmed instruction a specified number of times.
	71	Read	Compare the contents of a specified index register ("j" designation) with the operand. If they are equal, skip the next instruction and clear the index register; if unequal, add one to the contents of the index register and perform the next instruction.
	72	Read	If the content of the specified index register is non-zero, jump to specified address and decrement content of index register by one; if zero, proceed to next sequential instruction.
LOGICAL	40	Read	Form bit-by-bit logical product of Q register contents and contents of a specified location and store logical product in accumulator. (Wherever 1's are present in both corresponding bit positions, a 1 is stored; wherever a 0 is present in either corresponding bit position, a 0 is stored.)
	41	Read	Add the initial contents of the accumulator to the bit-by-bit logical product of the contents in the Q register and a specified location and store this sum in the accumulator.
	42	Read	Subtract the logical product of the contents in the Q register and a specified location from the initial contents of the accumulator and store this difference in the accumulator.
	43	Read	Form logical product of Q register and contents of a specified instruction, subtract this logical product from accumulator, and use this difference as a criterion for skipping the next instruction, leaving accumulator unaltered.
	44	Replace	Store the logical bit-by-bit product of the contents of a selected address and the Q register in the accumulator and the selected address.
	45	Replace	Add the logical product of the contents of a selected address and the Q register to the initial contents of the accumulator and store this sum in the specified address and accumulator.
	46	Replace	Subtract the logical bit-by-bit product of the contents of the Q register and the contents of a selected address from the initial contents of the accumulator and store this difference in the accumulator and in the specified address.
	47	Store	Store bit-by-bit logical product of contents in Q register and accumulator in specified storage address.
	50	Read	For each 1 bit of the selected word, force a 1 bit into the corresponding position of the accumulator, leaving other positions of the accumulator unaltered.
	51	Read	For each 1 bit of the selected word, complement the corresponding bit of the accumulator, leaving other positions of the accumulator unaltered.
	52	Read	For each 1 bit of the selected word, force a 0 bit into the corresponding position of the accumulator, leaving other positions of the accumulator unaltered.
	53	Read	For each 1 bit in the Q register, force the corresponding bit of a selected word into the corresponding bit position of the accumulator, leaving the other contents of the accumulator unaltered.
	54	Replace	For each 1 bit in the contents of a selected address force a 1 bit into the corresponding bit position of the accumulator, leaving the other accumulator bits unaltered, and store this result in the accumulator and the selected address.
	55	Replace	For each 1 bit in the contents of a selected address complement the corresponding accumulator bit and store the result in the accumulator and the selected address.
	56	Replace	For each 1 bit in the contents of a selected address force a 0 into the corresponding bit position of the accumulator, leaving the other accumulator bits unaltered and store this result in the accumulator and the selected address.
	57	Replace	For each 1 bit in Q register, transfer the corresponding bit at a selected address into corresponding bit position of the accumulator. Leave other bits of the accumulator undisturbed. Retain result in accumulator and store at the selected address.

Table 3-1. Instruction Repertoire (continued)

	CODE	CLASS	FUNCTION
INPUT/OUTPUT	13*	Read	Transmit control word over channel specified by Channel Select Register to peripheral equipment for control of I/O action.
	17*	Store	Store, at designated address, the input word on specified channel and send Input Acknowledge signal.
	62*	Read	Jump to specified location if the input buffer of a specified input channel is active; otherwise execute next sequential instruction.
	63*	Read	Jump to specified location for next instruction if specified buffer output channel is active; otherwise execute next sequential instruction.
	66*	Read	Terminate transfer of data to specified input buffer.
	67*	Read	Terminate transfer of data from specified output buffer.
	73*	Read	Activate input buffer without monitor on channel number as specified.
	74*	Read	Activate output buffet or external function buffer on specified channel number.
	75*	Read	Activate input buffer with monitor on specified channel number.
	76*	Read	Activate output buffer or external function buffer with monitor on specified channel number.
	7772**	–	Store channel number of interrupt being processed by I/O section at indicated address.
	7773**	–	Transfer the least significant five bits of the contents of the specified address into the Channel Select Register (CSR) which indicates the channel to be activated, deactivated, or tested.
INT.	7770**	–	Send synchronous interrupt to another computer or external device.

Table 3-1. Instruction Repertoire (continued)

NOTE: Instruction code 00 causes a jump to address 000000 and initiation of the fault interrupt mode.

* *Indicates slight differences between 490 and 494 instructions.*

** *Indicates instructions unique to 494.*

1. Instruction Word Formats

Three different instruction word formats are used as shown below.

f	j	k	b	y
x x x x x x	x x x	x x x	x x x	x x x x x x x x x x x x x x x
29	23	20	17	14 00

f	g	b	y
I I I I I I	x x x x x x	x x x	x x x x x x x x x x x x x x x
29	23	17	14 00

f	$\hat{j}$	$\hat{k}$	b	y
x x x x x x	x x x x	x x	x x x	x x x x x x x x x x x x x x x
29	23	19	17	14 00

The first instruction word format is the format most generally used. The *function code designator, f,* indicates the operation to be performed. The *y designator* is a binary number which is added to the contents of the index register indicated by the *b designator.* If the b designator is a zero, then the y designator is used as is, since there is no Index Register 0. This sum is either the effective operand (to be used directly in the operation) or it is the effective address, indicating a location in core memory which contains the operand to be used. Determination of whether this sum is the effective operand or the effective address is made by the combination of k designator and class of instruction. Interpretation of the *k designator* falls into one of three classes (read, store, and replace classes), described in Table 3-2. For instance, a read class instruction with k designator of 0 or 4 uses the effective operand in the operation; a read class instruction with j designator of 1, 2, 3, 5, or 6 uses the word contained in the effective address memory location; a read class instruction with j designator of 7 would use the word in the accumulator. Table 3-3 describes the use of the *j designator.* All designators are described herein using octal notation, to facilitate descriptions.

The second instruction word format is used with the "77" instructions. In these instructions f = 77_8 (111111) and the two-digit code of the sub-function designator, g, actually designates the function to be performed. The b and y designators have the same functions as in the first instruction word format. No j or k designator can be used with a 77 instruction.

The third instruction word format is the I/O instruction word format, used with instructions 13, 17, 62, 63, 66, 67, 73, 74, 75, and 76. The $\hat{j}$ designator is a four-bit code specifying the I/O channel references. The $\hat{k}$ designator is a two-bit code controlling the source of the operand, storage of the operand, or both.

	READ CLASS	STORE CLASS
k or $\hat{k}$ = 0	Upper half of operand = 0's; lower half is effective operand.	Store contents* in Q register, filling any remaining Q register positions with 0's.
k or $\hat{k}$ = 1	Upper half of operand = 0's; lower half is lower half of memory word at effective address.	Store contents of index register or lower half of accumulator or Q register at lower half of effective address, leaving upper half of effective address contents undisturbed.
k or $\hat{k}$ = 2	Upper half of operand = 0's; lower half is upper half of memory word at effective address.	Store contents of index register or lower half of accumulator or Q register at upper half of effective address, leaving lower half of effective address contents undisturbed.
k or $\hat{k}$ = 3	Operand is memory word at effective address.	Store contents of accumulator, Q register, selected channel, or index register at the effective address.
k = 4	Lower half of operand is bits 00-14 of effective operand; upper half is filled with bit 14 of effective operand.	Store contents* in accumulator.
k = 5	Lower half of operand is bits 00-14 of word at effective address; upper half is filled with bit 14 of this word.	Store bit-by-bit complementation of index register or lower half of the accumulator or Q register in lower half of word at effective address, leaving upper half undisturbed.
k = 6	Lower half of operand is bits 29-15 of memory word at effective address; upper half is filled with bit 29 of this word.	Store bit-by-bit complementation of index register or lower half of the accumulator or Q register in upper half of word at effective address, leaving lower half of word undisturbed.
k = 7	Operand is word in accumulator.	Store bit-by-bit complementation of index register, accumulator, or Q register into word at effective address. If index register is stored, fill upper half of affected word with sign bit of original index register word.

	REPLACE CLASS
k = 0	Not used.
k = 1	Only the low-order 15 bits of a register or memory location are used to form the replacement word which is stored in the low-order portion at the register or effective address, leaving the high-order portion undisturbed.
k = 2	The word used in forming the replacement word is made up of the addressed high-order portion shifted to the low-order portion with zero-fill in the high-order portion. The replacement word is stored in the high-order portion at the register and effective location, leaving the low-order portion undisturbed.
k = 3	The full 30-bit words of a register or effective address are used to form the replacement word which is then stored at an effective address or register.
k = 4	Not used.
k=5	The word used in forming the replacement word is made up of the low-order addressed word with sign-fill (bit 14 of the addressed word) in the high-order positions. The low-order 15 bits of the replacement word are stored in the low-order positions of the addressed word, leaving the high-order positions undisturbed.
k=6	The word used in forming the replacement word is made up of the addressed high-order bits shifted to the low-order positions and the high-order positions filled with the sign of the original word. The low-order bits of the replacement word are stored in the low-order position of the address.
k=7	Not used.

Table 3-2. Interpretation of "k" Designator.

** Store class exceptions:*

1. In the "14" instruction, "k" = 0, the bit-by-bit complementation of the Q register is stored.

2. In the "15" instruction, "k" = 4, the bit-by-bit complementation of the accumulator is stored.

j	FUNCTION
0	Perform next sequential instruction.
1	Skip next sequential instruction.
2	Skip next sequential instruction if Q register is positive.
3	Skip next sequential instruction if Q register is negative.
4	Skip next sequential instruction if accumulator is zero.
5	Skip next sequential instruction if accumulator is not zero.
6	Skip next sequential instruction if accumulator is positive.
7	Skip next sequential instruction if accumulator is negative.

(Contents of this table describe most common use of j designator, applicable to instructions 01-03, 05-11, 14, 15, 20-25, 30-37, 41-43, 45-47.)

*Table 3-3. Interpretation of "J" Designator**

2. Sample Instruction

A sample instruction may be coded as follows:

f	j	k	b	y
2 0	6	3	3	0 0 5 0 1
29	23	20	17	14 0

a. The function code "f" is 20_8, instructing the computer to add the operand to the contents of the accumulator and store the sum in the accumulator.

b. The "y" designator indicates that the unmodified address of the operand is 00501_8 .

c. The "b" designator is a 3, indicating that the contents of index register 3 must be added to "y" to find the effective address of the operand. If we assume that index register 3 contains the number 631_8, the effective address is 1332_8, which (we assume) contains the value + 1353535353_8.

d. The "k" of 3 indicates that the operand is the entire contents of the effective address. (See Table 3-3 for interpretation of "k" designator in read class instructions.) Therefore, if we assume that the accumulator initially contained + 0535353535_8 , the sum is + 2111111110_8 and this sum is stored in the accumulator.

e. The ''j'' designator of 6 initiates a skip of the next sequential instruction since the accumulator content is positive and we proceed to the instruction following the skipped instruction. (See Table 3-3 for interpretation of ''j'' designator.)

B. MAIN MEMORY AND ADDRESSING

The main memory is a 32-plane, coincident-current, ferrite core memory with maximum storage of 131,072 30-bit words (plus one parity bit per half-word). Memory transactions may transfer 15-bit half-words, 30-bit words, 60-bit double-precision words, or (by means of the logical functions) program-selected bits of a word.

1. Memory Layout

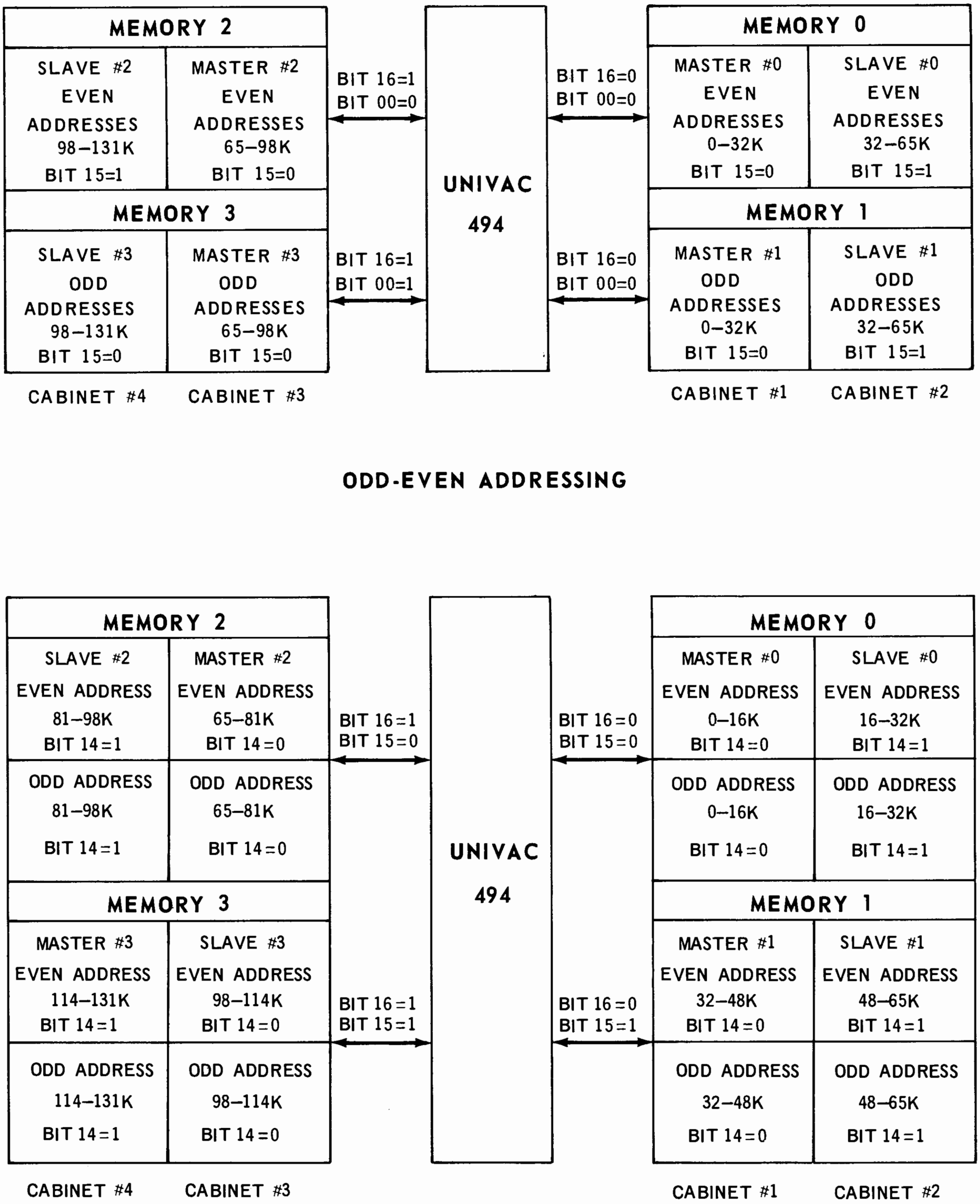
Minimum memory capacity consists of one master unit containing 16,384 words using continuous addressing. To obtain the advantages of the memory overlap feature the addition of at least a second master unit containing 16,384 words is required so that the odd-even addressing mode (Figure 3-1) can be used. Since the master units can operate independently, it is possible for one master unit to read the next instruction from memory while another master unit is simultaneously reading the last operand of the current instruction from memory. Thus; the average time per instruction for a series of one-operand instructions approaches one memory cycle time. The memory can be expanded, in 16K increments to a maximum of 131K with memory overlap. For each 16K increment, two 8K modules are added. Each added 8K module is either one half of a master unit or one half of a slave unit. The odd-even addressing structure is the normal mode for expanded memory capacity. The continuous addressing mode (selected manually for expanded memories) is used when memory faults are detected.

2. Relative Addressing

Relative addressing permits a program to be moved about anywhere in memory and then rerun without modification to the program. Another advantage of relative addressing is that it places a program in memory starting at an address specified by the executive routine. The Relative Index Register contains the reference for all instruction addresses and is loaded by instruction 7766 (see Instruction Repertoire). An address within memory is first defined by the effective address (addition of the contents of an index register to the address specified within the instruction word) and this sum is then added to the contents of the Relative Index Register to determine the absolute location within core memory to be used. This means that, though a program is restricted to 32K locations, it is not restricted to any one of the 32K memory banks and may even be located in two different banks. The contents of the Relative Index Register do not modify the effective operand.

3. Preassigned Storage Addresses

A set of fixed core storage addresses are reserved for interrupt locations or associated functions as listed in Table 3-4. These fixed locations are under control of the executive routine and are programmed to contain entrances to interrupt subroutines. (If the system is operating in the UNIVAC 490 mode, core storage addresses 00070 – 00077 and 00130 – 00137 are also used as interrupt subroutine entrances.)



ODD-EVEN ADDRESSING

CONTINUOUS ADDRESSING

Figure 3-1. Memory Address Layout

OCTAL ADDRESS	FUCTION
000000	Illegal Instruction
000001	Guard Mode
000002	Power Loss
000003	Parity Error-Memory Bank 0
000004	Parity Error-Memory Bank 1
000005	Parity Error-Memory Bank 2
000006	Parity Error-Memory Bank 3
000007	Executive Return
000010	Floating Point Underflow
000011	Floating Point Overflow
000012	External Synchronizer #1
000013	External Synchronizer #2
000014	Interval Timer Interrupt
000015	Day Clock Interrupt
000016	Day Clock Time
000017	Real Time Clock/Interval Timer Update
000020	External Interrupt ESI
000021	Input Monitor ESI
000022	Output Monitor ESI
000023	Buffer Control Word Parity Error
000024	External Interrupt ISI
000025	Input Monitor ISI
000026	Output Monitor ISI
000027	I/O Data Parity Error
000040-000067	Output Buffer Control Word Channels 00-23
000100-000127	Input Buffer Control Word Channels 00-23

Table 3-4. Preassigned Memory Addresses

4. Parity Error Protection

A parity bit is internally generated and inserted into core memory for each half-word written into core memory. Parity is checked each time data is read from core memory. A parity error interrupts the program to a preassigned memory location where appropriate action can be initiated under program control. The system fault alarm is activated. If recovery is unsuccessful, the addressing structure can be manually switched from the odd-even addressing structure of Figure 3-1 to the continuous addressing structure, so that a faulty memory unit will not be used, and switch the preassigned interrupt locations to another module.

C. CONTROL SECTION

The control section of the UNIVAC 494 Central Processor interprets and sequences computer instructions, with each instruction involving a basic operation (add, subtract, etc.) and modifiers which may specify one or more auxiliary operations to modify or extend the basic operation. Other functions included within the control section are program protection by the guard mode and fault interrupts.

1. Guard Mode

Program protection is afforded by the guard mode to ensure maximum protection for the executive routine and prevent interaction of unrelated programs. Under program control the Internal Function Register (IFR) can be loaded so that any of the following three guard modes can be enabled (instruction 7761):

- Guard Mode-Read and Write.
- Guard Mode-Write with Instruction Protection.
- Guard Mode-Write without Instruction Protection.

The Guard Mode-Read and Write with Instruction Protection prevents reading, writing, and jumping outside a fixed (lock-in) area within core storage. This mode is the normal operating mode for all worker and customer-written programs. In this mode all input/output instructions, and instructions with function codes in the range 776X and 777X cannot be performed. Reading, writing, or jumping outside the program lock-in area results in an interrupt to memory location 00001 where program control is returned to executive routines for appropriate action. This guard mode is normally enabled when leaving the executive routine and disabled during performance of the executive routine.

The Guard Mode-Write with Instruction Protection guarantees that the worker program does not modify the contents of addresses outside its limits but does permit it to perform common subroutines shared by several programs. This mode differs from the previous guard mode in that the program is permitted to read or jump to any location in memory. Writing outside the lock-in area results in an interrupt to location 00001, switching control to the executive routine. This mode prevents execution of the instructions described in the preceding paragraph.

The Guard Mode-Write without Instruction Protection is similar to the preceding guard mode in operation except that in this mode all instructions in the repertoire can be executed.

2. Fault Interrupts

The fault interrupts are interrupt signals with first priority initiated as a result of a fault, a programmed interrupt, or an arithmetic decision. A second group of interrupts, the I/O interrupt group, is described in the Input/Output section. The fault interrupts include the following:

- Illegal Instruction Interrupt
- Guard Mode Interrupt (already described in the section on guard modes).
- Executive Return Interrupt.
- Parity Interrupt.
- Floating Point Overflow Interrupt.
- Floating Point Underflow Interrupt.

The Illegal Instruction interrupt is initiated when the instruction word has an operation code of 774X, or 775X. If the option for decimal arithmetic is not installed, operation code 771X will also initiate the Illegal Instruction interrupt. If the option for floating point arithmetic is not installed this interrupt will also be initiated by operation code 770X. All other interrupts are locked out and the program is sent to address 000000 for executive action.

The Executive Return Interrupt is initiated by the 7734 instruction, routing the program to address 000007. Guard Mode Protection, Relative Addressing, etc., are rendered inactive due to initiation of the interrupt sequence. All other I/O type interrupts are locked out.

The Parity Interrupt is the result of a parity error in a word or half-word read from memory. The interrupt will route the program to an executive routine at address 000003, 000004, 000005, or 000006. (For further details, see Parity Error Protection, Main Memory and Addressing Section.)

The Floating Point Overflow Interrupt is initiated when the result of a floating point operation has an exponent part greater than 3777_8 or 4000_8. This interrupt is also caused if division by floating point number ± 0 is attempted. The program is routed to address 000011.

The Floating Point Underflow Interrupt is initiated when the exponent part of the result in a floating point add, subtract, divide, or multiply operation is less than zero, as stored in the accumulator. The program is then routed to memory address 000010 which leads to an executive routine.

D. ARITHMETIC SECTION

The arithmetic section performs the arithmetic, logical, and compare instructions required in a program. Arithmetic operations may be performed in fixed point binary or binary-coded-decimal modes, double-precision,floating point mode, or double precision binary mode.

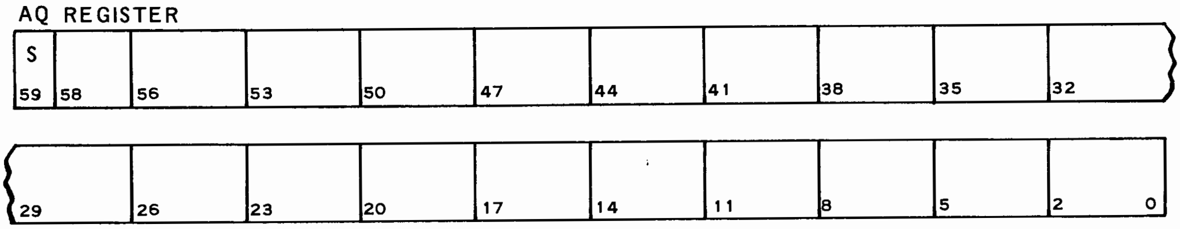
1. Fixed Point Formats

The format for a binary coded single precision, fixed point data word, as shown below,

S										
29	28	26	23	20	17	14	11	8	5	2 0

assigns three bit positions per octal digit, except for the most significant digit which has two bit positions. The sign, contained in bit position 29 is a 0 for a positive number, a 1 for a negative number. The order of digits is from 00 to 28, with position 00 containing the least significant digit.

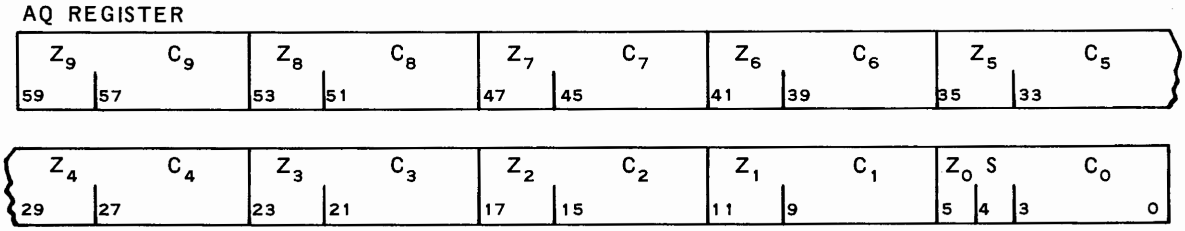
The format for a binary coded, double precision, fixed point data word, shown below,



uses 20 octal coded digits and requires two consecutive memory locations for storage and the use of the accumulator and Q register for operations. The most significant portion of the word (including sign) appears in the accumulator and/or the first of the two consecutive memory locations.

2. Decimal Format

Decimal operations may be performed upon data words stored in any six-bit binary-coded-decimal form, such as Fieldata code. Table 3-5 shows the acceptable coding of the first four bits for binary-coded-decimal digits. No other characters are considered valid for decimal operations. Because arithmetic operations are performed directly upon the decimal characters, no programmed coding conversion is necessary to convert binary-coded-decimal characters to straight binary characters for arithmetic processing. Ten decimal characters, including sign, may be stored and operated upon, requiring two consecutive memory locations and/or the accumulator and Q register placed in combination as shown below.



NO.	CODE	NO.	CODE
0	0000	5	0101
1	0001	6	0110
2	0010	7	0111
3	0011	8	1000
4	0100	9	1001

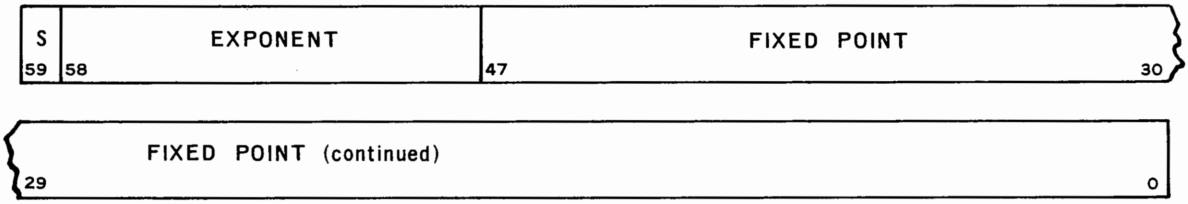
Table 3-5. Binary-Coded-Decimal Digits

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 of the number- a 1 for positive and 0 for negative. The word is right-justified with zero-fill for unused portions (both characters and zones). In accordance with standard Fieldata code, the zone bits are both 1's. The zone bits are ignored in all decimal operations and are the same as the original zone bits in the AQ register.

3. Floating Point Format

The floating point word format uses 60 bits to express a number as a fixed-point part (a fraction between .5 and 1) multiplied by a power of two (the exponent part). Two consecutive memory locations and/or the AQ register store the floating point number, as shown below.

AQ REGISTER



The sign, S (which is the sign of the fixed point part), the exponent part and the most significant part of the fixed point part are stored in the accumulator and/or the first of the two consecutive locations. A 1 bit is always added to the most significant bit of the exponent part so that the exponent part is always biased by 1024_{10}, obviating the need to sign the exponent part (which must lie within the limits of + 1024_{10}). The fixed point part is normalized (left-justified until a 1 appears as the most significant bit) for processing so that the fixed point part is a fraction between .5 and 1, due to the binary coding used.

4. 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 done in a separate unit so as not to tie up arithmetic operations. Due to the memory overlap feature, the average time per operation for a series of basic fixed point arithmetic operations approaches the cycle time of the main memory (750 nanoseconds).

E. INPUT /OUTPUT SECTION

Input/output channels are field-expandable, in number, from the basic 12 to a maximum of 24, in modules of four each. The 24 channels are divided into three groups of eight channels each. Each group may be operated at a 30-bit word transfer rate of 250 kc/s or 555 kc/s. Operation at the compatible rate ensures compatibility with all standard peripheral units. Channels are numbered from 00 to 23 (decimal). A unique feature of the central processor is that input/output loading and unloading operations, once initiated, do not disturb main program operation.

1. Index Modes

All input/output operations use either one of the two following index modes:

- Internally Specified Index (ISI) mode, used with standard peripheral devices.
- Externally Specified Index (ESI) mode, used with communication equipment.

The ISI mode provides a unique buffer for each channel, which is, in turn, serviced by a standard input/output peripheral subsystem. Since only one input/output subsystem is assigned to a channel the data to and from the peripheral will be stored in continuous locations within the buffer, as shown in Figure 3-2. Buffer control of input/output operations is maintained by the BCW (Buffer Control Word) loaded into a buffer control location, by the input/output initializing instructions, in the following format:

- Bits 00-17 indicate the address of the current data transfer. As each data transfer takes place, this address is incremented. Since 18 bit positions are available, all memory locations are accessible.
- Bits 18-29 indicate the word count. With each data transfer, this count is decremented. The 12 bits allotted for the count permit a maximum buffer length of 4096_{10} words. When the count is equal to zero, following a transfer, the buffer is terminated.

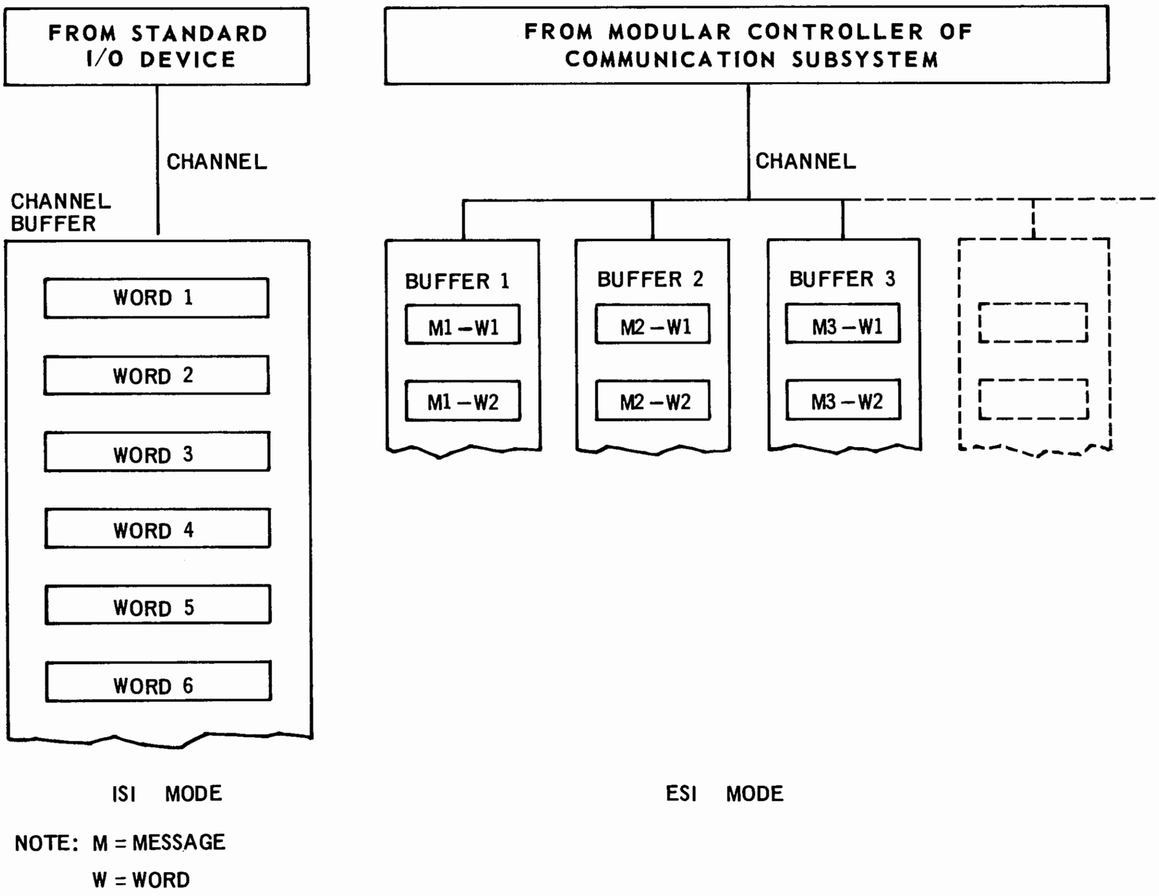


Figure 3-2. Input Word Storage, Internally Specified Index (ISI) Mode vs. Externally Specified Index (ESI) Mode.

The ESI mode provides different buffer areas for a number of input/output devices that are connected to the channel by a multiplexer. Characters of different messages are assigned continuous memory addresses within the buffer area allotted to each message. Buffer control is maintained by a BCW similar to the BCW of the ISI mode. Each data word loaded into the buffer uses bits 00-14 to specify the core memory address of the BCW and bits 15-29 for the actual data. Each time data output is unloaded from the buffer, the data is contained within bits 00-29 of the data word and the memory address of the BCW is sent in on the lower 15 *input* channel lines.

2. External Equipment Control

External peripheral equipment control (by the central processor) is attained by function (''control'') words and interrupt words. A function word is sent by the central processor to initiate a particular operation at a peripheral unit. The function word is distinguished from a data word by activation of the External Function control line. The use of function words rather than a set of instructions makes the central processor independent of the characteristics of the input/output unit, enabling connection of a variety of peripheral equipment with no modification to computer logic.

Interrupt words are sent to the central processor by peripheral equipment to inform the computer program of the status of the peripheral equipment, readiness to initiate or terminate data transfer, etc. An interrupt is distinguished from a data word by activation of the External Interrupt line. Activation of an External Interrupt line initiates an unconditional jump to a main memory address where, at the programmer's discretion, the program may begin a subroutine to analyze the interrupt word and take appropriate action.

3. Input/Output Priority

The priority network of the central processor determines which input/output requests must be honored immediately and which requests are to be honored when time is available. Input/output requests are assigned priorities in a two-level system, by function and by interrupt, as shown in Table 3-6. Transfer of a word, to or from the central processor, once initiated, is completed before the priority system is re-examined.

The Real Time Clock is a free-running oscillator which updates address 000017 of memory once every millisecond, and decrements the Interval Timer as well. The Real Time Clock may be used for a variety of program-timing purposes as logging of periodic messages, preparation of statistical data dealing with the frequency of certain transactions, etc. When the count in the Interval Timer is zero (a maximum of approximately 4.1 seconds) the Interval Timer Interrupt is generated. The time for the Interval Timer to interrupt can be controlled by the insertion of a word into bits 18-29 of address 000017.

An External Interrupt is a program-interrupt originating in the peripheral device and is usually accompanied by a status code which defines the reason for the interrupt.

A Monitor Interrupt is caused when an active buffer has the word count portion of its BCW set to zero, routing the program to the routine specified at the appropriate fixed memory location.

I/O FUNCTION PRIORITY	INTERRUPT PRIORITY
Output Transfer Channel 15	Buffer Control Word Parity Error
Input Transfer Channel 15	I/O Data Parity Error
Output Transfer Channels 14-08	Power Loss
Input Transfer Channels 14-08	External Interrupt (ESI)
Output Transfer Channels 23-16	Input Monitor (ESI)
Input Transfer Channels 23-16	Output Monitor (ESI)
Output Transfer Channels 7-0	External Interrupt (ISI)
Input Transfer Channels 7-0	Input Monitor (ISI)
Real-Time Clock Update	Output Monitor (ISI)
	Day Clock
	Interval Timer Interrupt
	External Sync. #1
	External Sync. #2

Table 3-6. Input/Output Priority

The External Synchronizer interrupts are used when two or more central processors are used in parallel within a system. These interrupts synchronize the timing signals of the units.

F. OPERATOR'S CONSOLE

The Operator's Console contains indicators and switches, a keyboard, and a printer furnishing hard copy. The indicators and switches are associated with the day clock fault, abnormal temperature, stop, run, fault, automatic recovery, keyboard status, and various program jumps and stops. The keyboard is a standard four-bank keyboard. Also present on the operator's console is a group of indicators that informs the operator of the status of the program in progress and a group of switches permitting various manual operations to be performed. Keyboard, day clock, and printer share channel 00.

G. MAINTENANCE PANEL

The maintenance panel contains a series of indicators showing the status of various registers within the central processor and a series of switches that control the program-step execution time, disconnect the Real Time Clock, and furnish other aids in localization of a malfunction.

H. UNIVAC 490 PROGRAM OPERATION

Most UNIVAC 490 programs will operate with but few changes. The UNIVAC 494 Real-Time System provides increased capacity, higher speeds, and additional features. A new executive routine will be written for the UNIVAC 494 Real-Time System. The UNIVAC 490/494 Mode switch has been included on the maintenance panel to insure that the UNIVAC 494 Real-Time System reacts as closely as possible for programs originally written for the UNIVAC 490 Real-Time System. The following changes must be made to UNIVAC 490 programs in order to run them on the UNIVAC 494 Real-Time System.

1. Day Clock

The day clock interrupt occurs every six seconds to a fixed address (000015). The time is not stored by the program during interrupt subroutine but is automatically stored every 600 milliseconds at address 0000016. The format of the time has been changed to the following:

HOURS		MINUTES		HUNDREDTHS			
TEN	UNIT	TEN	UNIT	TEN	UNIT	0 0	0 0 0 0 0 0
29	27	23	19	15	11	7	5 0

2. Initial Load

The initial load is prewired so that it is not a customer option. A bootstrap function is sent to a peripheral device on a channel selected by switches on the maintenance panel. A 200 word block of data is then read into the computer under hardware control.

3. Buffer Control Words

The Buffer Control Word of the UNIVAC 490 Real-Time System consisted of 15 bits for for a current address and 15 bits for a terminating address. The Buffer Control Word in the UNIVAC 494 consists of a 12 bit word count and an 18 bit current address. A common subroutine in the Input/Output handler of the UNIVAC 494 can easily convert UNIVAC 490 Buffer Control Words in the new format.

4. Real Time Clock

The lower half of address 0000017 remains unchanged, but the interval timer is changed from a 15-bit incrementing count to a 12-bit decrementing count.

4. OPERATING SYSTEM

The UNIVAC 494 Operating System is a comprehensive library of integrated programs featuring powerful programming languages and a flexible control language for directing basic computer operations. The system achieves maximum utilization of the computer and peripheral devices, automatically executes programs, and organizes system tasks by combining or utilizing elements from various sources.

The elements of the system are designed to permit each user to create a version of the basic system that will be consistent with the needs of the individual installation. System input is in the form of a control stream containing primary input, which defines and initiates a job, and secondary input, which provides parameters and supplementary data relevant to the execution of a job and its tasks.

An example of a job is the collection, loading, and execution of the programming units (tasks) required to process a FORTRAN source program, execute the object program that is produced in a test environment, and then include the program in a library. Tasks will be reduced to smaller elements to take advantage of the multi-processing capabilities of the system.

Figure 4-1 shows the relationship of some of the major elements in the system.

A. OMEGA

That set of routines which provide the basic control and fundamental mechanisms of the operating system will be called Omega. As the heart of the operating system, Omega is the vehicle for the implementation of the concepts of the operating system. The elements of Omega may be classified into external and internal control sections.

1. External Control

This set of executive elements is concerned with job-to-job transition, entry of tasks into the system, and control and utilization of other elements of the operating system. External control is based upon a scheduling function which provides for effeotive allocation of the facility configuration and minimizes turn-around time, time between submission and completion, of batch programs operating as background to high priority real-time processing.

External control accepts control language input streams from multiple devices. Input/Output cooperatives are the system elements which effect the buffering of system input and system output (primary and secondary) between the low speed external devices and the system. Random access storage is used as the buffering medium. Input/Output cooperatives include modular routines to interface directly with the particular external devices in use. These routines buffer and format data from the particular device and pass resultant images to a common control routine which enters each job entity onto a job queue.

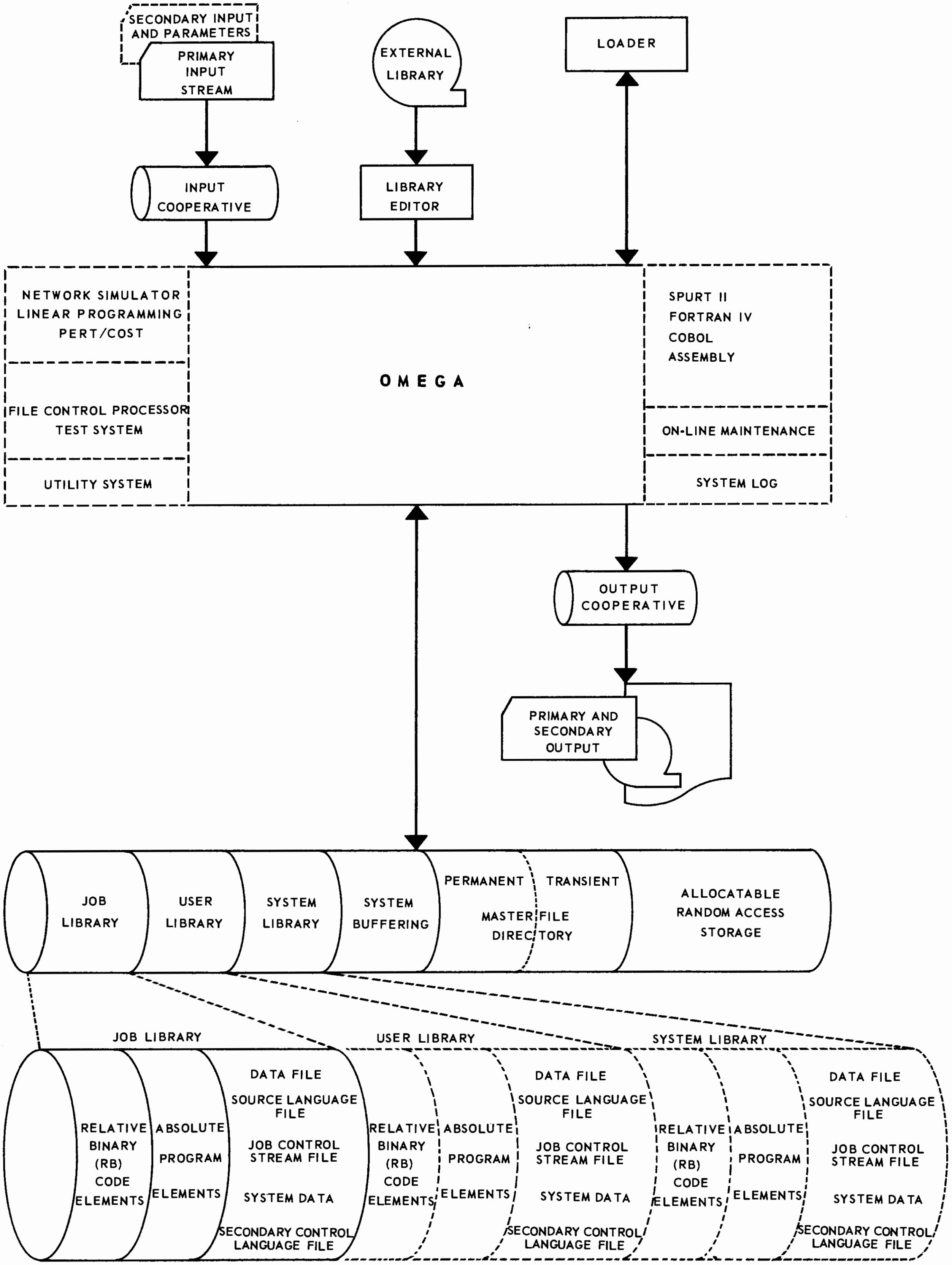


Figure 4-1. Major System Elements

Jobs are queued for selection pending introduction into the multi-program environment for execution by internal control. Each successive task is introduced to internal control when its characteristics satisfy the criterion utilized by selection. Activated jobs are sequenced as described by the job description by integrating and utilizing other elements of the operating system to perform functions required for the next specific task. The system elements employed in execution of the job are involved with compiling and constructing the task to be executed, satisfying facility requirements, entering library elements to be referenced by the task and initiating auxiliary system processors.

As each task completes its processing, the job is requeued with selection for return submission of the next task cycle. When all tasks of a job are complete, post-job processes are initiated including accounting output, facility deallocation, operator communication, compacting of core or random access memory as appropriate. Primary and/or secondary system output generated during operation of the job are initiated when there exist sufficient and complete task files and the output device is available. Input/Output cooperatives control the retrieval of output from drum and submission to and interface with the particular output devices.

2. Internal Control

This set of executive elements is concerned with control of the actual execution of user or system tasks, cognizance and supervision of equipment status, and schedule communications with operating personnel. Internal control is based upon a queuing function which provides for time-sharing of available facilities for effective machine utilization and through-put optimization for batch and real time programs in a multi-program environment. Internal control routines are characterized by the direct control of the computer. In general, they will execute privileged instructions, such as input/output, operate without memory lockout, and access and directly manipulate the entire storage. These routines are responsible for the integrity of the system and for preventing inter-program destruction or conflicts.

Internal control accepts service requests during execution of a task defining the function to be performed by Omega. A ''service request'' is an entry to the executive conveying parameters to describe the requested function. The request is either satisfied immediately and control returned to the requestor or queued for later service and subsequent reactivation of the requestor. An active task is either in control of the central processor, subject to interruption by internal control, or queued at some service point. Queues are processed in a manner responsive to both program priorities and procedures optimizing machine utilization. Maximum parallel use of facilities is dependent on queues to maintain a work backlog for key facilities. Queues are maintained for central processor control, peripheral and core assignments, time intervals, shared programs, and other functions.

Interrupt response and contingencies are handled by internal control without mandatory user participation. Standard fault procedures or a user specified alternate will be employed by the system. Interrupts are processed as they occur and generally result in the queuing with central processor control of an independent activity which will analyze the interrupt. This immediate response achieves a smoothing such that interrupts will actually be processed at different priority levels and no interrupt will preclude registration of another. Since Omega in general is a permanent task composed of independent activities, it may generally be ''switched'' to a high-priority task subsequent to an interrupt occurrence.

Schedule communications allow the operator to alter internal functions, relay messages, accept unsolicited operator requests, such as equipment status inquiry or declaration, and task termination, suspension, or abortion. Logging of peripheral usage and operational errors encountered is performed cumulatively for tasks of a job as input to the accounting procedure.

B. CONTROL LANGUAGE

The construction and execution of a job is described by a series of control statements prepared by the user. Control statements with supplementary cards representing data, parameters, source code or object code, constitute the control stream. The control stream is interpreted by Omega to schedule, sequence and operate the necessary elements of the system. Control language statements can be divided into two categories. Category one describes job organization and execution. Category two describes control of system processors such as the File Control Processor or Test System.

1. Operational Control

These statements suffice to describe and activate a job which exists in the system library. Control statements are interpreted and executed as a basic executive function. Each statement is complete and is processed independently of other control statements. Examples of category one statements are listed below.

JOB	Identifies start of job description.
CALL	Names a job stream to be executed.
–	
LOAD	Directs construction of a program.
GO	Initiates execution of existing program.
–	
ASSIGN	Associates internal input/output reference with physical unit.
RELEASE	Release input/output from job duration assignment.
–	
SOURCE	Identifies source of supplementary control statements.
MSG	Controls operator direction and system logging.

2. System Control

System control statements must be employed in conjunction with category one statements when a job must be constructed prior to execution, or established in a user or job library, or when particular system procedures must be defined. The format and content of a secondary control language is defined by the processor. A representative job sequence containing various statements is shown in Figure 4-2.

STATEMENT	COMMENTS	TASK NO.	NOTES
JOB	Identifies start of job description.		
ASSIGN	Associates logical unit A with tape containing current versions of RB elements required by program in test.	1	
IN	Requests input of RB elements from logical unit A into JOB library.		
SPURT	Requests Assembly of source code for program in development.	2	
SOURCE	Identifies source code to be used from an external medium if other than primary input stream.		
(DATA CONTAINING SOURCE CORRECTIONS)			
TEST	Requests processing of secondary language describing test strategy.	3	Tasks #3 thru #7 may be bypassed if error in assembly with option to abort job specified.
(SECONDARY LANGUAGE STATEMENTS)			
LOAD	Constructs absolute program by combining RB element just assembled with RB elements in JOB, USER or SYSTEM library as required.	4	
ASSIGN	Associates logical unit with tape containing test data.	5	Tasks #5 thru #7 may be bypassed if construction encounters error flagged elements and option to abort job is specified.
ASSIGN	Associates logical unit with random access storage to be initialized with test data.		
UTL	Requests processing of secondary language describing distribution of test data.		
(SECONDARY LANGUAGE STATEMENTS)			
GO	Requests execution of existing absolute program.	6	
ASSIGN	Associates logical unit A with tape to receive the RB element for development program.	7	Task #7 may be bypassed by operation of interpretive supervisor (test system) declaring an ABORT condition.
OUT	Requests output of RB element from JOB library.		
JOB	Identifies start of next job description.		

Figure 4-2. Representative Job Deck Sequence

C. SYSTEM LIBRARIES

The availability of random access storage in the system is exploited by a hierarchy of libraries. These libraries allow run-time construction and structure of object programs while avoiding the inefficiencies of set-up, searches, and implied order associated with such external files as magnetic tapes and cards. These manipulative techniques are also extended to data files which include internally registered job streams, secondary language elements, and source programs. The operating system is therefore oriented toward exclusive processing of random access storage files which are established by control language direction from external media, primarily magnetic tape and cards.

The smallest logical unit of information that may be entered into the system is an element and the collection of elements available to the user is called the element library. The element library is divided into three logical levels, the Job Library, the User Library and the System Library. These libraries are searched in the stated order whenever an element is referenced in execution of a job so that override may be controlled and predicted.

The User and Job Libraries are established from external peripheral files. Statements in the Library Maintenance Language provide for the entry of elements in the library and recording of new library elements on external storage so that they will be present at the next establishment of the library. The Systems Library may not be manipulated by Library Maintenance Control statements.

1. Library Elements

Elements within a library are identified by their name, version and type. The name is a symbolic name associated with the element at time of creation. A version may be associated with each locatable element to differentiate versions of an element when checking out large programs. A library may contain alternate versions of the same program, one outdated but workable, the other in a test stage.

The basic types of elements composing an element library are:

a. Relative Binary (RB) Element

These elements are produced by the system compilers (SPURT, COBOL, FORTRAN), and represent a processed source language program which may be an entity or dependent on collection with other RB elements to become an operable program.

b. Absolute Element

These elements result from the collection and loading process performed by the Loader. An absolute program is an entity with all external references collected and interconnected, cross references resolved, and relative locations assigned. It can be entered into any contiguous core area without scanning the text.

c. Data Files

These elements conform to a standard data file format and include source language (referenced by any compiler), job control streams (may be activated externally or internally), secondary language elements (referenced by appropriate system processor), installation data files (manipulated by a task or file control) and miscellaneous data files (retrieved at program execution time).

2. System Library

The System library is an inherent part of the operating system which is resident on random access storage at all times. The system library includes standard RB elements (mathematical, utility, input/output, editing,) which may contribute to construction of a program, standard absolute programs (including system processors, transient elements of Omega, internally registered job streams and standard production job description, and miscellaneous data file elements including a master file directory and files. Only the system may modify this library.

3. Job Library

The Job library is a collection of elements created for each job. It is built as the job is executed and is interrogated only to satisfy requests associated with the job. The Job Library is established by a library maintenance function (IN) or by elements generated or referenced during execution of the job. Control stream directives, therefore, explicitly or implicitly contribute to this library.
The Job Library is transient and remains only as long as the job is active. Elements to be preserved may be output by a library maintenance function (OUT).

4. User Library

A User library may be established by the installation or by a particular programming or applications group within the installation. The User Library for each group has a separate identity. A User Library may be used as a repository of programs or data files developed by the programming group which have utility for the members of the group. A Job Library may be linked to the User Library so that elements from the User Library can be collected at load time or referenced during execution. In effect the User Library is an extension of the Job Library which is shared by simultaneous users.

A User Library is established in the storage by a library maintenance statement (IN) contained in the job control stream. All of the jobs associated with the programming group can reference one User Library through use of a control statement (LINK). If the library is already resident, the call is considered already fulfilled. A statement linking a Job Library and an existing User Library sanctions references to the User Library and provides the mechanism for maintaining the User Library in storage. Once established, the library remains resident until there are no links to it; at that time it becomes non-permanent and may be deleted. Deletion is not forced until a requirement exists for the storage.

5. Library Structure

Each library contained in the system is described by a table of contents. The table of contents resides on random storage, and is a composite of tables which serve various purposes in the manipulation of the library.

An element table contains one item for each element in the particular library. It contains the name, version, type (relocatable, absolute, or data file) and link to its text for each element. The linked text to an element may be broken into several distinct pieces, each of which occupies a distinct but contiguous area of drum. For example, an RB element consists of a preamble table, a text file and a symbol table. For each distinct list, the element table contains an increment to the list on drum.

An entry point table contains all externally defined symbols in all elements contained in a library. The entry point table is used by the collector to simplify the collection process.

6. The Library Editor

The Library Editor is a systems element with the ability to read, write, list and otherwise manipulate the Job or User Library and their associated table of contents. Library maintenance represents the mechanism for recording on external devices processed elements from the Job or User Library.

Library maintenance functions are initiated through the control stream (IN, OUT, LINK, PRINT). All control statements along with any diagnostic messages are submitted to the primary output stream for subsequent printing. The library maintenance function operates as a task of the job and references external devices, through logical units previously defined by ASSIGN or HOLD statements.

D. LOADER

The Loader is a system processor which provides a flexible and efficient means of collecting independent relocatable elements to produce an absolute object program for execution as a task under control of Omega.

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.

Construction of segmented programs or particular collections are described by a secondary control language which normally follows the LOAD statement. The loader or utility package can be directed, however, to place a set of statements into a job library as an element. Subsequent execution of the Loader may utilize this element as directed by the LOAD control statement.

Relocatable elements existing in the Job, User, or System Libraries are collected in constructing an object program. Elements are collected on the basis of an external reference in one element which can be satisfied by an external definition within a second element. The Loader may be directed to include or exclude specific elements by secondary control statements. Since all system compilers generate the relative binary (RB) code, the Loader serves as a common focus which enables combination of coding expressed in various program languages.

The basic output of the Loader is an absolute object program. The program is entered into the Job Library by 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 printed record of the collection process. The Loader can also transfer the secondary control language as a Job Library element for subsequent reference.

E. FILE CONTROL

The control stream may include a secondary control language which defines to the File Control Processor the nature of the file to be processed. File description is a task level activity; the description is available to all activities of the task. A file used by more than one task in a job sequence need be described only once, at the point of first use.

Data files are constructed in a great variety of formats due to the inherent differences in problems, the conflicting interests of space saving versus time saving, and the urgency of ready access for real time applications. It is not the intent of file control to limit services to files constructed to a single convention, but rather to provide conventions by which a variety of files may be described and handled. These conventions must be selected for describing both blocks of data and items of data.

1. Tape Files

Tape files may have fixed item sizes and fixed blocks, fixed item sizes and variable blocks, variable item sizes within fixed blocks, or variable item and variable block sizes. The description of the file will designate the file organization.

The item and block services provided are intended to cover a variety of situations. However, it is probable that a particular data processing center will wish to standardize on certain conventions and avoid needless complications.

2. Random Files

There are a number of cross classifications for files which are maintained on random access storage. These files may be sequential or random, semi-permanent or transient. Semi-permanent files are considered to be resident in storage beyond the activity range of any task or activity which refers to them. Transient files are files established by the control stream of a particular job and exist only for the duration of the job. Such transient local files are available only to tasks associated with the job control stream. Permanent files are listed in a master file directory and may be accessed by programs in independent job stream. Provision is also made for the allocation of ''scratch'' drum area. File control tables for permanent files are maintained in residence with the files.

A random file may be addressed by item key or by relative item address. Reference by use of a key implies an index block associated with the file which relates the key to a specific address or address range. Relative addressing implies a user provided routine which transforms keys into relative file addresses, either by formula manipulation or by privately maintained indices.

F. INPUT/OUTPUT COOPERATIVES

Input/Output cooperatives are a collection of routines which control multiple system peripheral devices. Their primary function is to utilize intermediate random access storage buffers to balance the intermittent data requirements of the system with the slow but continuous transfer rate of the peripheral device. The cooperatives also enable tasks which have apparently conflicting need for a peripheral device to operate concurrently. This is accomplished by the intermediate buffering which functionally makes the device simultaneously addressable. Each cooperative maintains its own control for buffering on random access storage. A common storage pool is used by all cooperatives which dynamically extends or contracts depending on the demands of the environment. Counts of drum modules used by a cooperative are kept for job accounting.

The input cooperatatives supply multiple streams of primary input to the system. Primary input is used for control information and to provide limited quantities of data. References to data within the input stream are made by a service request entry submitted for system or user programs. Provision is made to merge supplementary data into a primary input stream from an auxiliary source. This feature is normally used to merge and correct source code submitted to the system processors. The function may be extended by an installation to serve the need of user programs and may also be used to enter supplementary control statements for job description.

The primary output cooperatives accept print images from 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 fixed 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 in a similar manner except that card images are accepted from operating tasks and ultimately processed by a card punch output routine, or recorded on magnetic tape.

G. COMPILERS AND ASSEMBLERS

These are the major system components which translate the programming languages accepted by the system into a machine amenable form. A primary output of compilers and assemblers will be relative binary (RB) elements acceptable as input to the Loader. These routines exist in the system library in absolute form.

The source language input and object code output by the compilers and assemblers are handled through the program storage and retrieval functions of the system. These elements may be retrieved or manipulated by library maintenance. The cooperative action of the compilers and assemblers and the system allow the updating and storage of source language and object code within the system storage area.

H. TEST SYSTEM

The test system is a processor which provides a set of basic functions designed to aid the programmer during program development and test.

The package interpretively executes programs in test to provide a flexible means of obtaining dynamic control of test procedures. Symbol tables generated by compilers are accepted to provide symbolic reference of test points and data areas. Relative reference is also provided for testing absolute programs. Either means of reference allows definition of test procedures external to the program to be tested. Test strategy can therefore be employed at object time in contrast to source level. Test procedures include snapshot dumps controlled by conditional procedures, a set value procedure for either patches or setting test condition values, and postmortem dumps of core or specific logical units. Each of the functions is activated and described by secondary control statements contained in the input stream and/or symbol tables generated by system processors.

I. ON-LINE MAINTENANCE

The operating system includes a set of integrated maintenance tests intended for automatic, and operator selected, execution to serve as an effective tool for the discovery and reporting of malfunctions. These tests provide the following significant advantages:

- on-line execution of maintenance tests with consequent reduction of fixed scheduled maintenance periods and down time from application or production
- automatic confidence checks as permanent background activity when customer or system is temporarily idle
- semi-permanent cumulative record of abnormal conditions for historical, statistical, and machine diagnosis
- single set of maintenance tests operated independently, in a controlled maintenance environment, or in the multi-program environment.

J. SYSTEM LOG

The System automatically maintains a log of accounting information and data acquired for use in preventive maintenance. Error information is submitted by the integrated maintenance routine or input/output handlers through the normal pressure of handling input/output requests. Accounting information is collected for each job as it progresses through the system. At job termination time, a facility usage statement is submitted to the primary output stream of the job and is entered into the log table.

K. UTILITY SYSTEM

This system processor provides a basic set of utility functions to facilitate the transfer of data from one peripheral medium to another. Options are provided to establish the transferred data with the system as a data file for access by file control.

Specifications for operation are given by a secondary control language. The control language and any diagnostic messages incurred during processing are recorded in the primary output stream for each job.

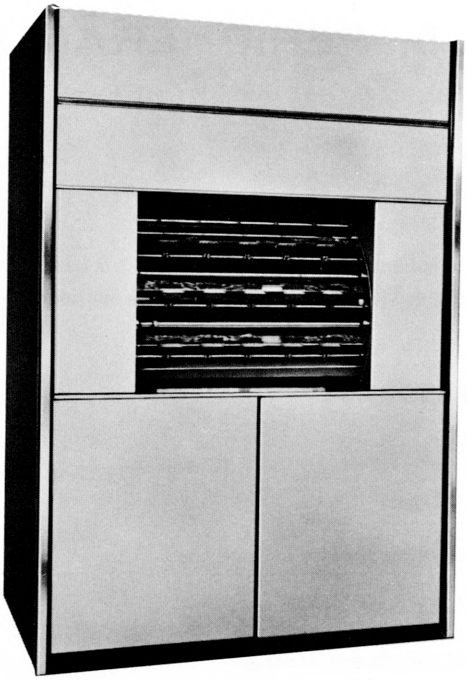
5. PERIPHERAL SUBSYSTEMS

A large variety of standard peripheral subsystems may be utilized to accomplish the transfer of data to or from the central processor. These subsystems may be grouped as follows:

- **RANDOM ACCESS STORAGE**
  - FH-880 Magnetic Drum
  - FH-432 Magnetic Drum
  - FASTRAND IA Mass Storage
  - FASTRAND II Mass Storage
  - FASTRAND Modular Mass Storage
- **MAGNETIC TAPE SUBSYSTEMS**
  - UNISERVO VIC
  - UNISERVO VIIIC
- **CARD SUBSYSTEM**
  - Card Reader
  - Card Punch
- **PRINTER SUBSYSTEM**
  - High Speed Printer
- **PAPER TAPE SUBSYSTEM**
  - Paper Tape Reader
  - Paper Tape Punch
- **UNIVAC 1004 SUBSYSTEM**
  - With appropriate channel adapter
- **COMMUNICATIONS SUBSYSTEM**
  - A large variety of communications equipment may be connected to the central processor. Input/Output from many units may be interleaved by the use of a channel multiplexer.

A. RANDOM ACCESS STORAGE

- Flying Head-880 Magnetic Drum Subsystem

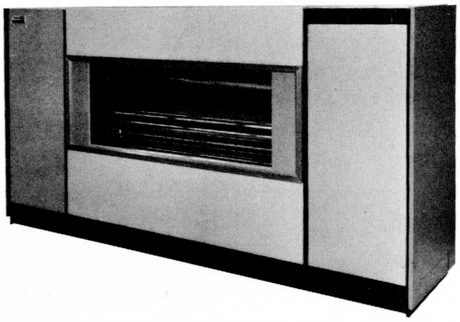


CHARACTERISTICS	
NUMBER OF UNITS PER PER SUBSYSTEM	1–8
STORAGE CAPACITY PER UNIT	786,432 30-bit computer words 3.9 million alphanumeric characters (approximately)
TOTAL STORAGE CAPACITY (maximum number of units)	6,291,456 30-bit computer words 31.4 million alphanumeric characters (approximately)
RECORDING DENSITY	518 bits per inch
AVERAGE ACCESS TIME	17 milliseconds
TRANSFER RATE	60,000 words per second 300,000 characters per second
DRUM SPEED	1,770 revolutions per minute
NUMBER OF READ/WRITE HEADS	880 (40 blocks with 22 heads in each block)

- Flying Head-432 Magnetic Drum Subsystem

CHARACTERISTICS	
NUMBER OF UNITS PER SUBSYSTEM	1–9
STORAGE CAPACITY PER UNIT	262,144 30-bit computer words 1.3 million alphanumeric characters (approximately)
TOTAL STORAGE CAPACITY (maximum number of units)	2,359,296 30-bit computer words 11.8 million alphanumeric characters (approximately)
RECORDING DENSITY	870 bits per inch
AVERAGE ACCESS TIME	4.25 milliseconds
TRANSFER RATE	240,000 words per second 1,200,000 characters per second
DRUM SPEED	7,100 revolutions per minute
NUMBER OF READ/WRITE HEADS	432 (9 blocks with 48 heads in each block)

- FASTRAND IA Mass Storage Subsystem



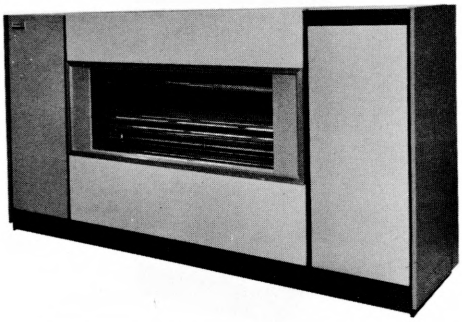
CHARACTERISTICS

NUMBER OF UNITS PER SUBSYSTEM	1–8
STORAGE CAPACITY PER UNIT	12,976,128 30-bit computer words 65 million alphanumeric characters (approximately)
TOTAL STORAGE CAPACITY (maximum number of units)	103,809,024 30-bit computer words 519 million alphanumeric characters (approximately)
RECORDING DENSITY	1,000 bits per inch
TRACKS PER INCH	53
AVERAGE ACCESS TIME	92 milliseconds
TRANSFER RATE	25,150 words per second 125,750 characters per second
DRUM SPEED	880 revolutions per minute
NUMBER OF READ/WRITE HEADS (moveable)	64

Optional Features:

FASTBAND (24 fixed read/write heads)	This option provides the unit on which it is installed with an additional 50,688 word storage area. The average access time to data in this area is 35 milliseconds.
LOCKOUT PROTECTION	This option consists of a key controlled switch which, when set, will prevent the writing of data in a specified area on the unit on which it is installed.

- FASTRAND II Mass Storage Subsystem



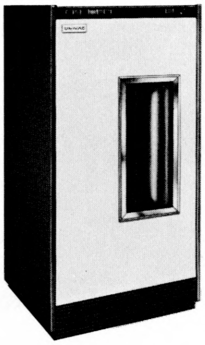
CHARACTERISTICS

NUMBER OF UNITS PER SUBSYSTEM	1–8
STORAGE CAPACITY PER UNIT	25,952,256 30-bit computer words 130 million alphanumeric characters (approximately)
TOTAL STORAGE CAPACITY (maximum number of units)	207,618,048 30-bit computer words 1 billion alphanumeric characters (approximately)
RECORDING DENSITY	1,000 bits per inch
TRACKS PER INCH	106
AVERAGE ACCESS TIME	92 milliseconds
TRANSFER RATE	25,150 words per second 125,750 characters per second
DRUM SPEED	880 revolutions per minute
NUMBER OF READ/WRITE HEADS (moveable)	64

Optional Features:

FASTBAND (24 fixed read/write heads)	This option provides the unit on which it is installed with an additional 50,688 word storage area. The average access time to data in this area is 35 milliseconds.
LOCKOUT PROTECTION	This option consists of a key controlled switch which when set will prevent the writing of data in a specified area on the unit on which it is installed.

■ FASTRAND Modular Mass Storage Subsystem

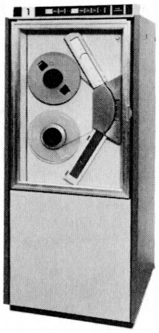


CHARACTERISTICS

NUMBER OF UNITS PER SUBSYSTEM	1–8
STORAGE CAPACITY PER UNIT	2,162,688 30-bit computer words 11 million alphanumeric characters (approximately)
TOTAL STORAGE CAPACITY (maximum number of units)	17,301,504 30-bit computer words 87 million alphanumeric characters (approximately)
RECORDING DENSITY	1000 bits per inch
TRACKS PER INCH	106
AVERAGE ACCESS TIME	67.5 milliseconds
TRANSFER RATE	25,150 words per second 125,750 characters per second
DRUM SPEED	1760 revolutions per minute
NUMBER OF READ/WRITE HEADS	16
Optional Features:	
FASTBAND (24 fixed read/write heads)	This option provides the unit on which it is installed with an additional 16,896 word storage area. The average access time to data in this area is 17.5 milliseconds.
LOCKOUT PROTECTION	This option consists of a key controlled switch when set will prevent the writing of data in a specified area on the unit on which it is installed.

B. MAGNETIC TAPE SUBSYSTEMS

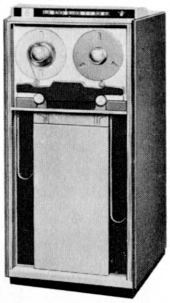
- UNISERVO VIC Magnetic Tape Subsystem



CHARACTERISTICS	
NUMBER OF UNITS PER SUBSYSTEM	1–16
RECORDING DENSITY	200,556, or 880 6 bit characters per inch
TRANSFER RATE	8,500, 23,700, or 34,200 characters per second
TAPE SPEED	42.7 inches per second
TAPE WIDTH	0.5 inch
TAPE LENGTH	2,400 feet
TAPE THICKNESS	1.5 mils
BLOCK LENGTH	variable
SPACE BETWEEN BLOCKS	0.6 inch
CHANNELS ON TAPE	7 channels 6 data 1 parity
READ/WRITE OPERATION	Reading in forward and backward directions; writing in the forward direction only.
COMPATIBILITY	The subsystem has the ability to read or write tapes in IBM* binary coded decimal format.
Optional Features:	
9-CHANNEL OPERATION 8 DATA 1 PARITY	Recording density 800 characters per inch Transfer Rate = 45,500 characters per second

* *Registered trademark of the International Business Machines Corporation.*

- UNISERVO VIIIC Magnetic Tape Subsystem

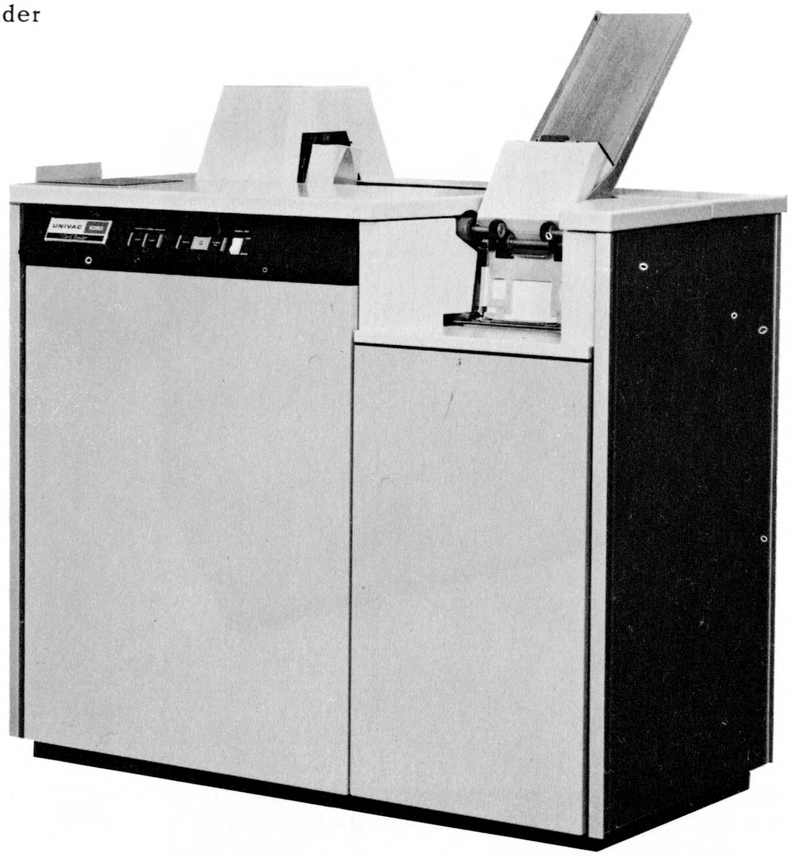


CHARACTERISTICS	
NUMBER OF UNITS PER SUBSYSTEM	1–16
RECORDING DENSITY	200,556, or 800 6 bit characters per inch
TRANSFER RATE	24,000, 66,720, or 96,000 characters per second
TAPE SPEED	120 inches per second
TAPE WIDTH	0.5 inch
TAPE LENGTH	2,400 feet
TAPE THICKNESS	1.5 mils
BLOCK LENGTH	variable
SPACE BETWEEN BLOCKS	0.6 inch
CHANNELS ON TAPE	7 channels 6 data 1 parity
READ/WRITE OPERATION	Reading in forward and backward directions; writing in the forward direction only.
COMPATIBILITY	The subsystem has the ability to read or write tapes in IBM* binary coded decimal format.
Optional Features:	
9-CHANNEL DATA OPERATION 8 DATA 1 PARITY	Recording density = 800 characters per inch Transfer rate = 128,000 characters per second

** Registered trademark of the International Business Machines Corporation.*

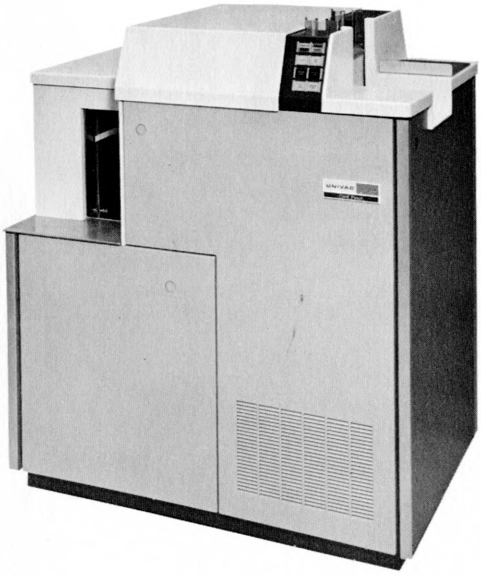
C. PUNCHED CARD SUBSYSTEM

- Card Reader



CHARACTERISTICS	
CARD READING SPEED	800 cards per minute (80 columns) 900 cards per minute (72 columns)
INPUT HOPPER CAPACITY	3,000 cards
OUTPUT STACKER CAPACITY	2,000 cards
REJECT STACKER CAPACITY	100 cards
READ MODES	Translation (The subsystem automatically translates the 12 bit card code to 6 bit Fieldata code before it is transferred.), column binary, or row binary.

- Card Punch



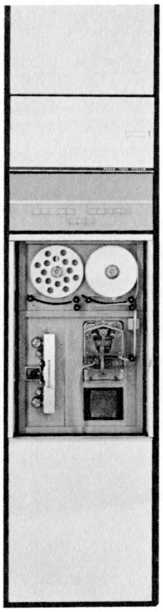
CHARACTERISTICS	
CARD PUNCHING SPEED	300 cards per minute
INPUT HOPPER CAPACITY	1,000 cards
OUTPUT STACKER CAPACITY	1,000 cards
SELECT STACKER CAPACITY	1,000 cards
PUNCH MODES	Translation (The subsystem automatically translates the 6 bit Fieldata code to 12 bit card code before it is punched), column binary, or row binary.

D. HIGH SPEED PRINTER SUBSYSTEM

CHARACTERISTICS	
PRINTING RATE	700/922 lines per minute
MAXIMUM NUMBER OF CHARACTERS PER LINE	132
NUMBER OF PRINTABLE CHARACTERS	63 (26 alphabetic, 10 numeric, and 27 special characters)
LINES PER INCH (VERTICAL)	6 or 8 (manually selected)
CHARACTERS PER INCH (HORIZONTAL)	10
PAPER STOCK	Any sprocket-fed paper from (up to and including card stock) 2.75 to 21.5 inches wide and up to 22 inches in length.
NUMBER OF COPIES	At least 5 carbons and an original with a pack thickness of 15.5 mils.

E. PAPER TAPE SUBSYSTEM

- Paper Tape Reader



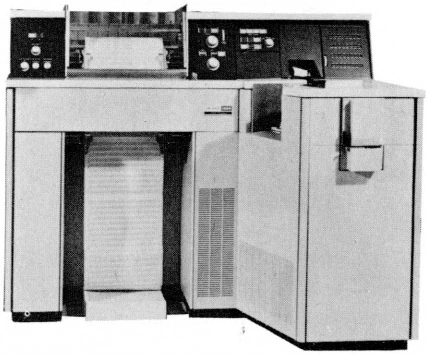
CHARACTERISTICS	
READING RATE	400 characters per second
NUMBER OF CHANNELS	5,6,7, or 8
CHARACTERS PER INCH	10
TAPE SPEED	40 inches per second
TAPE WIDTH	11/16, 7/8, or 1 inch

- Paper Tape Punch

CHARACTERISTICS	
PUNCHING RATE	110 characters per second
NUMBER OF CHANNELS	5, 6, 7, or 8
CHARACTERS PER INCH	10
TAPE SPEED	11 inches per second
TAPE WIDTH	11/16, 7/8, or 1 inch

F. UNIVAC 1004 SUBSYSTEM

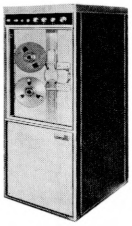
A channel adapter is used to effect interface with the central processing unit.



CHARACTERISTICS	
CARD READING SPEED	615 cards per minute
CARD PUNCHING SPEED	200 cards per minute
PRINTING RATE	600 lines per minute
MAXIMUM NUMBER OF CHARACTERS PER LINE	132
NUMBER OF PRINTABLE CHARACTERS	63 (26 alphabetic, 10 numeric, and 27 special characters)
LINES PER INCH (VERTICAL)	6 or 8 (manually selected)

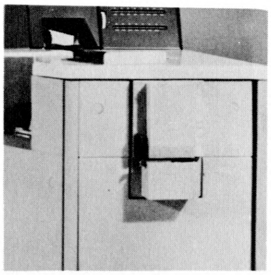
Optional Features:

- UNISERVO Magnetic Tape Units



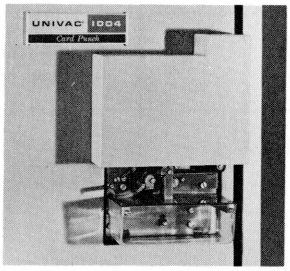
CHARACTERISTICS	
NUMBER OF UNITS	2 (with UNIVAC 1004 III Subsystem)
RECORDING DENSITY	200, 556, or 800 6 bit characters per inch
TRANSFER RATE	8,500, 23,700, or 34,200 per second
CODE CONVERSION	Any 6 bit code to any other 6 bit code

- Paper Tape Reader



CHARACTERISTICS	
READING RATE	400 characters per second
NUMBER OF CHANNELS	5,6,7, or 8
CHARACTERS PER INCH	10
TAPE SPEED	40 inches per second
TAPE WIDTH	11/16, 7/8, or 1 inch

- Paper Tape Punch



CHARACTERISTICS	
PUNCHING RATE	110 characters per second
NUMBER OF CHANNELS	5,6,7, or 8
CHARACTERS PER INCH	10
TAPE SPEED	11 inches per second
TAPE WIDTH	11/16, 7/8 or 1 inch

- Communications

The channel logic provides for linkage with communication equipment through the use of a Data Line Terminal, Type 1.

G. UNIVAC 494 COMMUNICATIONS SUBSYSTEM

The UNIVAC 494 Communication Subsystem enables the UNIVAC 494 Real-Time System to receive and transmit data via any common carrier in any of the standard codes and at any of the standard rates of transmission up to 4800 bits per second. It is the only communication system which can receive data from or transmit data to low speed, medium speed, or high speed lines in any combination.

The subsystem consists of two principal elements, the Communication Terminal Modules (CTM's),which make direct connection with the communication facilities, and the Communication Modular Controller through which the CTM's deliver data to and receive data from the Central Processor. A Modular Controller may be connected to any general purpose computer channel or two or more controllers may be connected to two or more channels. If required, a number of controllers may be connected through a Scanner Selector to the same general purpose channel. The total number of controllers which can be connected to a general purpose channel is dependent on the number and speed of the communication systems linked to the controller by their CTM's.

1. Communication Terminal Modules (CTM's)

There are three basic kinds of input and output CTM's: low speed (up to 300 bps*), medium speed (up to 1600 bps) and high speed (200 – 4800 bps). Each is easily adjusted to the speed and other characteristics of the type of line with which it is to operate. Each CTM accomodates two full duplex communication lines or two input and two output simplex communication lines. A CTM requires one position of the Communication Modular Controller.

2. Communication Line Terminals (CLT's)

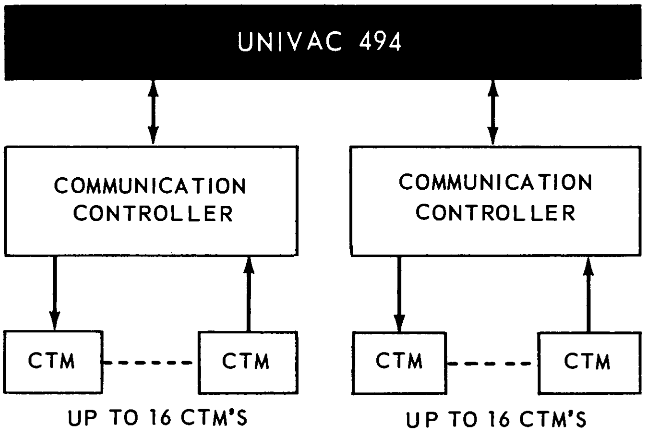
The CLT – Dialing is an output CLT which is employed to enable the Central Processor automatically to establish communications with remote points via the common carrier's switching network. In addition, a special parallel input and output CLT is provided for transmission of bit parallel data up to 75 cps.

3. Communication Modular Controller

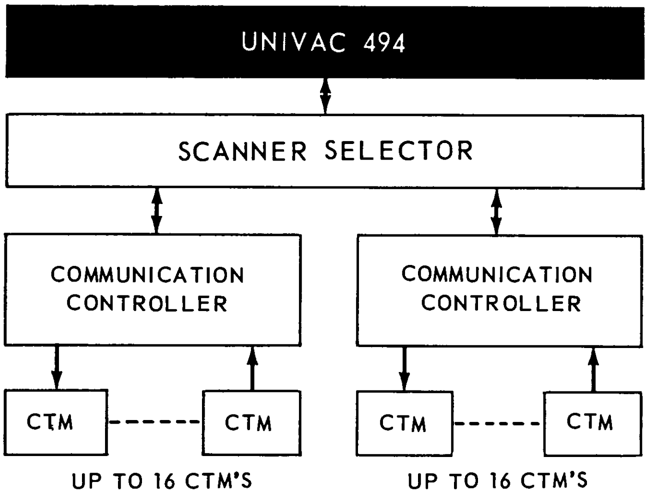
The Communication Modular Controller functions as the link between the processor and the CTM's. In each of the CTM modules, two input and output positions are provided. A Communication Multiplexer can accommodate up to 16 CTM's.

The CTM's may request access to the Central Processor via the Controller in random sequence, or several, or conceivably, all CTM's might request access simultaneously. The Communication Controller automatically assigns priorities among CTM's requesting access and identifies to the Central Processor the particular CTM granted access.

** bits per second*



One Communication Controller per General Purpose Channel



Multiple Communication Controllers per General Purpose Channel using Scanner Selector

CHARACTERISTICS

	COMMUNICATIONS TERMINAL MODULES (CTM'S)			COMMUNICATIONS LINE TERMINALS (CLT'S)	
NAME	LOW SPEED	MEDIUM SPEED	HIGH-SPEED	†CLT DIALING	CLT PARALLEL
CODE	5, 6,7, or 8 LEVEL	5,6,7, or 8 LEVEL	5,6,7, or 8 LEVEL	4 LEVEL	8 LEVEL
MODE	*ASYNCHRONOUS	ASYNCHRONOUS	**SYNCHRONOUS	***TIMING SIGNAL	TIMING SIGNALS
	BIT SERIAL	BIT SERIAL	BIT SERIAL	BIT PARALLEL	BIT PARALLEL
SPEED	UP TO 300 bps	UP TO 1600 bps	2000-4800 bps	VARIABLE	UP TO 75 cps

Types of Communication Service provided:

PRIVATE LINE TELETYPEWRITER	WIDE AREA TELEPHONE SERVICE (WATS)
TELEX	PRIVATE LINE TELEPHONE
TELETYPEWRITER EXCHANGE SERVICE (TWX)	DIRECT DISTANCE DIALING (DDD)

† *CLT—Dialing — This is an output CLT employed when the Central Processor is automatically to establish communications with remote points via the common carrier's switching network.*

* *ASYNCHRONOUS — Employs start and stop bit with each character to establish timing.*

** *SYNCHRONOUS — Uses timing characters at pre-determined intervals between data characters.*

*** *TIMING SIGNAL — Indicates the presence of a character at a Data Set.*

UNIVAC

DIVISION OF SPERRY RAND CORPORATION