SPERRY RAND

UNIVAC

494

REAL-TIME SYSTEM

CENTRAL PROCESSOR UNIT

PROGRAMMER/OPERATOR REFERENCE MANUAL

UP-4049 Rev. 2

CONTENTS

TABLES

1. INTRODUCTION

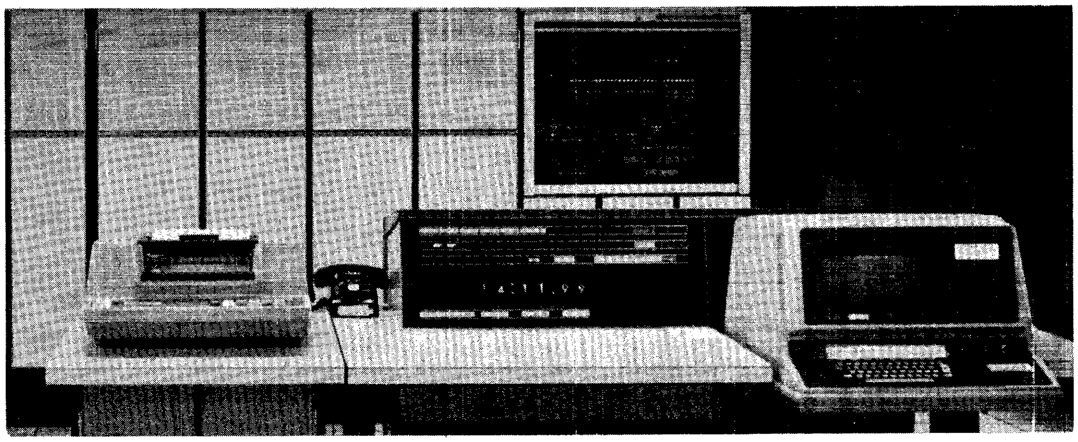


Figure 1-1. UNIVAC 494 Central Processor Unit, with Operator's Display Console and Primary Storage Unit

1.1. GENERAL

The UNIVAC 494 Real-Time System is a large scale information handling system with proven outstanding capabilities in multiprogram operations involving real time communications and data processing activities. As an enhanced successor to the UNIVAC 490 and 491/492 Real-Time Systems, the UNIVAC 494 System combines advanced system hardware and software design concepts and characteristics to fulfill the demands of real time applications while performing background batch processing jobs efficiently. The system is designed around the capabilities of the UNIVAC 494 Central Processor Unit (CPU), shown in Figure 1-1, which is a high speed, powerful digital computing unit. The CPU operates with a flexible framework of interrupts and a comprehensive repertoire of instructions under executive control by the UNIVAC 494 Operating System. The CPU is equipped to handle all of the message, and data transfers and manipulations, including logic, arithmetic, and queue processing called for by the system software; and controls all systems activities and functions. In cooperative interaction with primary storage providing 65 to 131K word capacity, the CPU utilizes a wide variety of peripheral devices and subsystems, including other CPU's and satellite computer systems, in on-site and remote applications to furnish various system configurations and functions according to the specific needs of a particular installation. Inherent system modularity and compatibility in both hardware and software facilitate future additions, expansions, and variations of functions.

Through executive control provided by the system software, the CPU and other hardware components, functioning on a multilevel priority scheme, achieve an optimum program mix in which noncritical programs may be interrupted to permit concurrent execution of real time programs in which time is more critical. All functions associated with selection, initiation, and termination of program elements are also performed under executive control, achieving maximum utility in the system.

Overall system characteristics for both hardware and software are discussed in *UNIVAC 494 Real-Time System System Description, UP-4032* (current version).

System software characteristics and logic are discussed in detail in *UNIVAC 494 Real-Time System Operating System Programmers Reference, UP-7504* (current version).

Characteristics of peripheral hardware components and subsystems are discussed in detail in Systems Programming Library Service publications covering the specific components or subsystems of interest.

1.2. CENTRAL PROCESSOR UNIT CHARACTERISTICS

The CPU provides a variety of features and capabilities which are of advantage to the system user:

- compatibility with an inclusive complement of peripheral subsystems.
- a minimum of 12 input/output (I/O) channels, field-expandable (in increments of four) to a maximum of 24, each of which may carry a complete peripheral subsystem, including I/O devices, communications equipment, or other computers.
- input/output channel operation in the Internally Specified Index (ISI) or Externally Specified Index (ESI) mode for all channels, with the exception of channel 0 which accommodates the operator's console and Day Clock.
- compatibility with UNIVAC 490/491/492 programs and programming techniques.
- read/restore cycle time of 750 nanoseconds.
- interface with a random access ferrite core primary storage (memory) facility having a minimum capacity of 65K 30-bit words (plus a parity bit for each half-word), in two 32K modules which may be cycled independently, expandable in 65K increments to 131K maximum, and with cycle time of 750 nanoseconds and read access time of 400 nanoseconds.
- memory overlap and interleaving capability, permitting the next instruction to be read from memory simultaneously with the reading from or writing into memory of the last operand, for reduced instruction read times.
- relative addressing capability, enabling flexible placement and relocation of programs anywhere within memory.
- buffered input/output, providing capability for program execution concurrently with I/O transfers.

- time orientation to a 24-hour Day Clock and an 18-bit Real-Time Clock, which generate interrupts at specified intervals for program monitoring, loading of instructions, and dealing with contingencies.
- direct arithmetic operations on fixed-point binary single and double precision operands, fixed-point binary coded decimal (BCD) double precision operands, and floating-point binary double precision operands.
- priority control network within the computer I/O logic which determines the order for honoring data transfer requests (function priority) and for responding to system contingencies (interrupt priority).
- response to a flexible and efficient Operating System, comprising the executive routine, utility routines, application packages, assemblers, and compilers.

2. CENTRAL PROCESSOR OPERATION

2.1. GENERAL

The UNIVAC 494 central processor unit (CPU) is the prime component of the UNIVAC 494 Real-Time System. The CPU has the responsibility for accepting data and jobs from input/output equipment, queuing and executing jobs in conformance with the demands of a real-time system, and executing and returning the processed results to the system user. The interface between the CPU and the external peripheral equipment is described in this section together with the functional operation and unique processing features of the CPU. These features include the Day Clock and Real-Time Clock, memory overlap and interleaving organization, dual configuration of index registers, and a system of interrupts for control of I/O operations and contingencies.

2.2. EXTERNAL INTERFACE

The CPU is designed to operate with a variety of I/O devices and may be equipped with a maximum of 24 I/O channels. A standard subsystem may be tied directly to each channel of the CPU for operating in the Internally Specified Index (ISI) mode, or a communications subsystem may be used between a channel and the I/O subsystem for operating in the Externally Specified Index (ESI) mode (ESI and ISI operations are discussed in subsequent sections of this manual). Each channel has a set of functional lines connected to the external equipment. The external interconnecting lines constitute the external interface, as indicated in Figure 2–1, which implements the exchange of data and control signals.

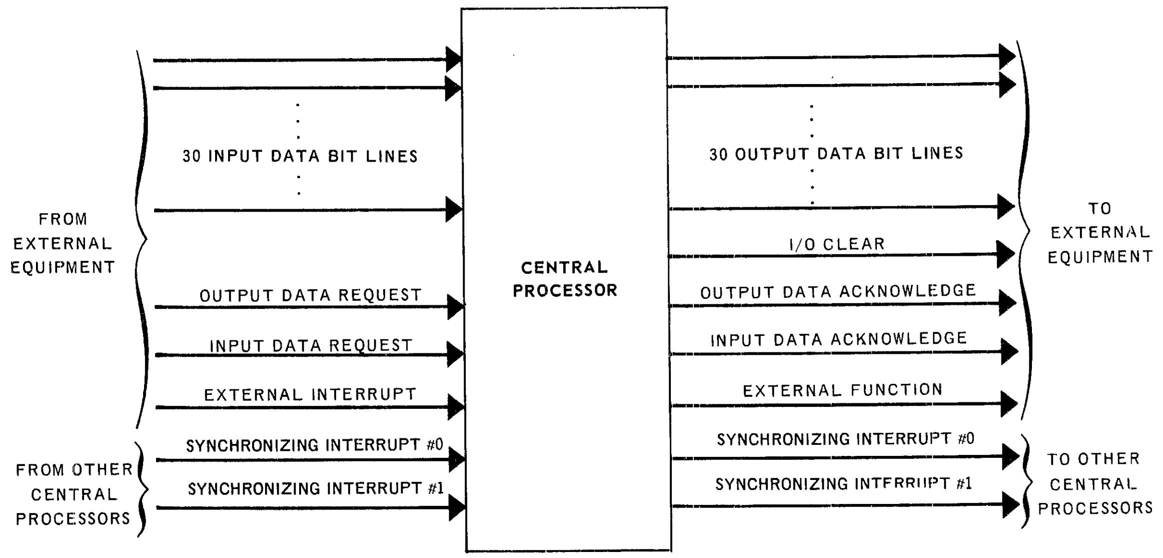


Figure 2–1. Central Processor Unit Interface

The I/O controls and their functions are as follows:

The *Synchronizing Interrupt* signal, which is programmed, causes an interrupt (at the receiving CPU) which transfers control to an interrupt routine. A maximum of two output and two input Synchronizing Interrupt lines may be used per CPU.

The *Output Data Acknowledge* signal indicates that the CPU has placed a data word on the output word lines.

The *External Function* signal informs the peripheral device that the output word is a function word rather than a data word. A function word is decoded at the output device to initiate a particular action (read a block of data from magnetic tape, for example). Provision is made within the repertoire of instructions for sending the External Function signal even though an Output Data Request signal (see following paragraph) may not be present.

The *Input Data Acknowledge* signal indicates that the central processor has stored an input word.

The *I/O Clear* signal is a master clear instruction transmitted by all I/O channels to the external devices, and is generated when a control at the CPU is manually pressed while the CPU is stopped.

The *Output Data Request* signal indicates to the CPU that the output device is ready to receive an output data word.

The *Input Data Request* signal indicates that the input device has placed a data word on the input data lines.

The *External Interrupt* (EI) signal indicates that the input device has placed a status word on the input word lines, rather than a data word The channel number is immediately stored in a five-bit Interrupt Address Storage register (IASR) and an interrupt is generated transferring control to an executive routine. This routine normally includes an instruction (Store Channel instruction) to store the status word in core memory so that it can be analyzed.

The I/O channels for the CPU are numbered 0–23. Channel 0 can operate in the ISI mode only. The other channels may operate in either ISI or ESI mode and may be changed from ISI to ESI operation or from ESI to ISI operation by a physical change to an I/O printed circuit card. Channel 0 accommodates the operator's console and the day clock. Each group of eight channels (0–7, 8–15, 16–23) can operate in either the compatible mode or in the normal mode. A channel in the normal mode operates at an I/O transfer rate of 555 kHz (kilocycles per second). A channel in the compatible mode operates at a transfer rate of 250 kHz for compatibility with UNIVAC 490 peripheral equipment.

2.3. INTERNAL OPERATION

An instruction requires two steps: the interpretation step and the execution step. During the interpretation step, an instruction word is analyzed to set up the logic circuits for the specific functions called for. During the execution step, the specified functions are performed.

The basic substeps involved in an interpretation step are as follows:

(1) A 30-bit instruction word is read from memory.

(2) Portions of the instruction word are analyzed to determine the operand.

(3) Portions of the instruction word are analyzed to determine the operations to be performed and to condition the logic circuits involved.

The basic substeps involved in an execution step are as follows:

(1) The operand is read from memory (if required).

(2) The indicated logic or arithmetic operation is performed.

(3) The result is written back into memory (if required).

The functional blocks of the CPU which carry out the steps of interpretation and execution steps are shown in Figure 2–2.

2.3.1. Interpretation Step

As par t of the initial loading procedure, the address of the first instruction is placed in the Program Location Counter. The contents of the Program Location Counter are transferred to the 17-bit Program (P) register and then to primary storage. The instruction word is read from the indicated address of primary storage into the Instruction register, the word contents are analyzed for operand and function, and the execution cycle is started. Normally, as the execution step starts, the number in the Program Location Counter is increased by 1 and this new value is inserted into the P register. If a skip is indicated during execution, the Program Location Counter will again be incremented by 1 in time to change the contents of the P register before the next interpretation is initiated. If the instruction is a Jump instruction and conditions are satisfied, the new absolute address will be transferred to the P register. (If the instruction is a Repeat instruction, the 17-bit effective address of the instruction is stored in the Internal Function register (IFR) and address transfers to the IFR will be inhibited until the Repeat instruction is completed; if the instruction is a Jump instruction, the 17-bit P value will be stored in the IFR.) Instruction word analysis and conditioning is accomplished as follows:

(1) The basic address contained in the instruction word is added to the contents of an index register, specified by a designator in the instruction word, to form the relative address. Depending upon the activated index register configuration and the index register chosen (see 2.7), the relative address may be either 15 bits or 17 bits.

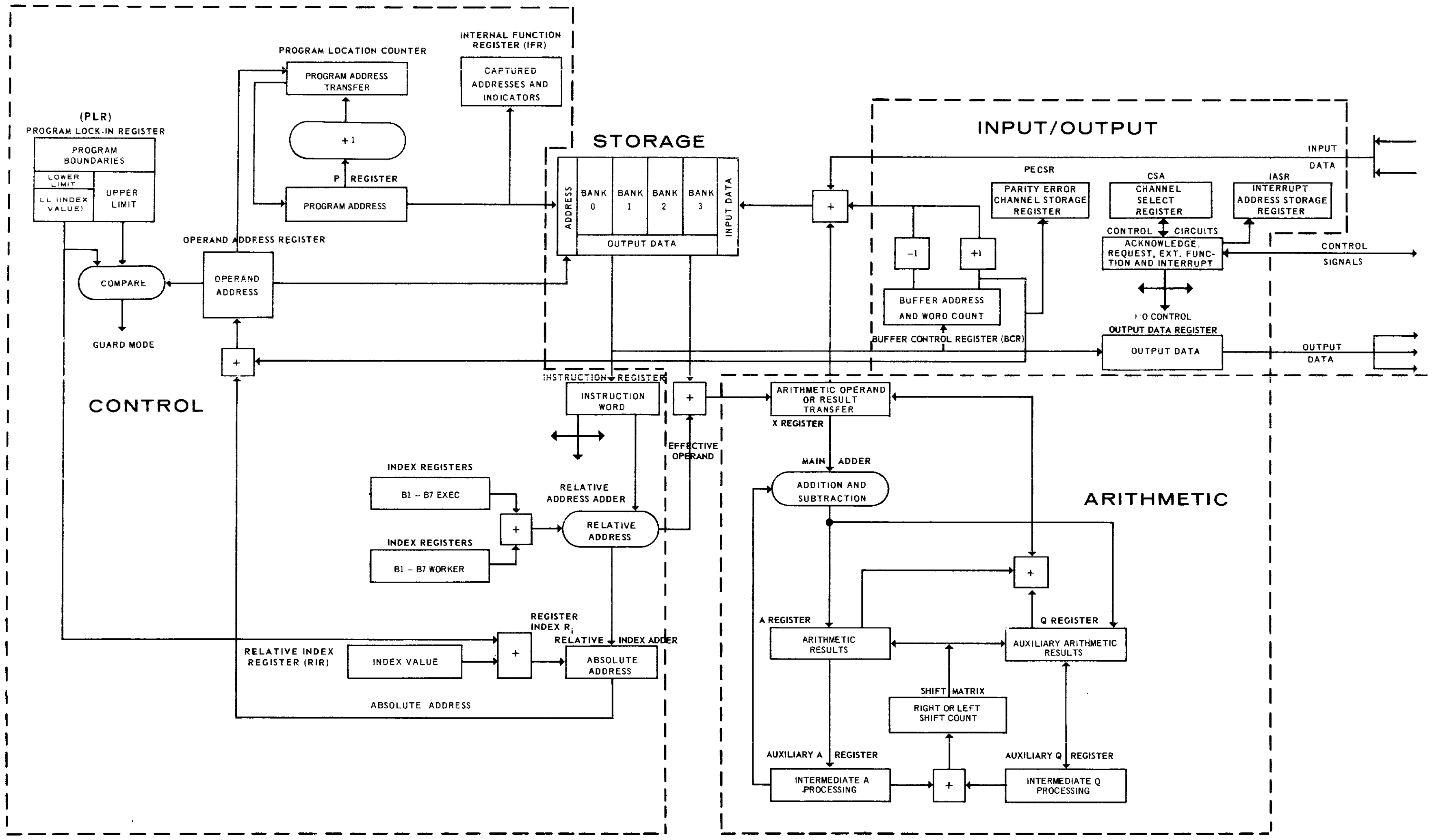


Figure 2–2. Central Processor Operation, Block Diagram

(2) The relative address is added to the relative index (R_i) to form the 17-bit absolute memory address containing the operand. The relative index is either the contents of the Relative Index register (RIR) or the lower boundary or lower lock limit (LL) in the Program Lock-In register (PLR). Bit 27 in the IFR and the b designator of the instruction determine the choice.

(3) The absolute address is loaded into the Operand Address register and is checked to determine that it lies within the program boundaries.

2.3.2. Execution Step

The operand is determined by the 30-bit contents of the Instruction register. In some cases, the operand is part of this instruction word and, if this is so, the operand is sent directly to the X register of the arithmetic circuits, after modification by an index register. In other cases, the instruction word specifies the memory location which contains the operand, and the operand is obtained in the following sequence after the conditioning described in 2.3.1 has been completed:

(1) The absolute address is sent to the memory and the operand is read from memory into the applicable register.

(2) The logic circuits, conditioned by timing and execution controls, process the operand, usually retaining the result in one of the arithmetic registers.

(3) If required by the instruction, the result is also stored in a memory location specified by the instruction word.

All primary storage address references pass through the relative addressing circuits except those references generated by normal sequential operation (including skips) of the P register and those which reference the "fixed" addresses (see 2.10), a group of addresses set aside for interrupt operations. The IFR establishes program protection to ensure that programs do not operate outside the boundaries defined in the Program Lock-In register (PLR), and also establishes Guard Mode to ensure that certain privileged instructions are not executed.

The interpretation and execution steps are essentially limited by the primary storage cycle time. With memory overlap, the last operand of the *current* instruction may be read simultaneously with the *next* instruction, eliminating one memory cycle time between instructions and significantly decreasing the running time required for a program.

The *X register* is a 30-bit register, handling all communication between arithmetic circuits and primary storage. The register is also capable of furnishing the ones complement (changing 1 bits to 0 bits, and vice versa) of data which it stores.

The *accumulator* (A register) is the principal arithmetic register and is used in most arithmetic operations. The accumulator is a 30-bit register with complementation capability.

The *Q register* has the same capability as the accumulator. Q assists A in multiplication, division, and logical operations. The A and Q registers are sometimes combined to form one 60-bit register, the AQ register, for double precision and BCD arithmetic operations. Auxiliary 30-bit registers are associated with the A and Q registers, and are used for storage of intermediate results in arithmetic operations.

The shift *matrix* (K register) enables the contents of the A and/or Q registers to be shifted, at one time, as many positions as required, instead of one bit position at a time.

The *Output Data registers* hold data on the data output lines long enough to meet requirements of comparatively slow speed I/O devices.

The *Buffer Control register* (BCR) with one input and one output for each channel minimum, holds the Buffer Control word (BCW) and controls I/O transfers between primary storage buffers and external equipment. The Channel Select register, the Interrupt Address Storage register, and the Parity Error Channel Storage register are concerned with I/O operations; their uses are described in 2.12.

The *Internal Function register* (IFR) is a 30-bit register used principally to facilitate executive routines. The IFR performs the following functions:

- captures the P register value for Jump instructions;
- captures the relative address of a Repeat instruction and its j designator (see Section 3);
- determines the particular type of program protection to be used;
- indicates overflow or carry for decimal arithmetic;
- specifies whether the executive set or the worker set of index registers is the active index register set;
- determines the bit capacity of the active index register configuration;
- activates one of the two relative addressing modes.

The Program Lock-In register (PLR) defines the upper and lower memory address limits to be used in the particular type of program protection selected by the IFR. The lower limit (LL) may be used as the Relative Index for operands.

The Relative Index register (RIR) is used as a dynamic bias for operand and/or instruction addresses to enable the execution of programs anywhere within memory without modification to these programs. The IFR, PLR, and the RIR are loaded by programmed instructions under executive control.

2.4. INPUT/OUTPUT OPERATIONS

Input/output operations effect the transfer of data and control signals between the CPU and peripheral equipment. A buffer is a specified number of sequential core storage addresses, starting from a specified address, set aside as required for storing input or output data before it is processed by the CPU or sent to the peripheral device. The 24 (maximum) I/O channels can operate with standard peripheral devices (drums, tapes, printers, etc.) or with multiplexed communication subsystems.

Transfers with multiplexed communication subsystems use the ESI mode. Channel 0 is reserved for, and can only be used through, the operator's console. Transfers with standard peripheral devices use the ISI mode. An ISI channel has one input buffer control register (BCR) and one output BCR (located at the fixed addresses listed in Table 2–1) per channel. For ESI operation, each of the multiplexed I/O devices on a channel is assigned its own input and output ESI location.

All I/O operations are governed by a multilevel priority scale which resolves situations when two or more channels attempt to communicate with the CPU simultaneously. The priority scale uses the fixed priority assigned to each request or interrupt. Function priority determines which of the data requests has the highest priority; interrupt priority determines which of the various I/O interrupts has the highest priority. Function priority and interrupt priority operate independently of each other and each one is a primary level of priority. Both function and interrupt priorities are shown in Table 2–2. Within each of these primary levels there is a secondary level which resolves priority when the same situation exists in two or more channels of the same channel group (15–8, 23–16, and 7–0). In such a case, the highest numbered channel within the group has the highest priority. (The interrupts shown in Table 2–2 are the I/O (conditional) interrupts described in 2.12.)

ADDRESS	FUNCTION	
	490 MODE	494 MODE
000000	FAULT ADDRESS	ILLEGAL INSTRUCTION
000001	MEMORY PROTECTION	PROGRAM PROTECTION OR TIMEOUT
000002	POWER LOSS	POWER LOSS
000003	MEMORY PARITY BANK #0	MEMORY PARITY BANK #0
000004	MEMORY PARITY BANK #1	MEMORY PARITY BANK #1
000005	BCR PARITY ERROR	BCR PARITY ERROR
000006	I/O DATA PARITY	I/O DATA PARITY ERROR
000007		EXECUTIVE RETURN
000010		FLOATING-POINT UNDERFLOW
000011		FLOATING-POINT OVERFLOW
000012		SYNCHRONIZING INTERRUPT #0
000013		SYNCHRONIZING INTERRUPT #1
000014	REAL TIME CLOCK INTERRUPT	REAL TIME CLOCK INTERRUPT
000015	DAY CLOCK INTERRUPT	DAY CLOCK INTERRUPT
000016	DAY CLOCK TIME UPDATE	DAY CLOCK TIME UPDATE
000017	REAL TIME CLOCK UPDATE	REAL TIME CLOCK UPDATE
000020	EXTERNAL INTERRUPT CHANNEL 0	EXTERNAL INTERRUPT ESI
000021	EXTERNAL INTERRUPT CHANNEL 1	INPUT MONITOR ESI
000022	EXTERNAL INTERRUPT CHANNEL 2	OUTPUT MONITOR ESI
000023	EXTERNAL INTERRUPT CHANNEL 3	MEMORY PARITY BANK #2
000024	EXTERNAL INTERRUPT CHANNEL 4	EXTERNAL INTERRUPT ISI
000025	EXTERNAL INTERRUPT CHANNEL 5	INPUT MONITOR ISI
000026	EXTERNAL INTERRUPT CHANNEL 6	OUTPUT MONITOR ISI
000027	EXTERNAL INTERRUPT CHANNEL 7	MEMORY PARITY BANK #3
000030	EXTERNAL INTERRUPT CHANNEL 8	TEST AND SET
000031	EXTERNAL INTERRUPT CHANNEL 9	
000032	EXTERNAL INTERRUPT CHANNEL 10	
000033	EXTERNAL INTERRUPT CHANNEL 11	
000034	EXTERNAL INTERRUPT CHANNEL 12	
000035	EXTERNAL INTERRUPT CHANNEL 13	
000036	EXTERNAL INTERRUPT CHANNEL 14	
000037	EXTERNAL INTERRUPT CHANNEL 15	
000040	INPUT INTERRUPT CHANNEL 0	OUTPUT BCR CHANNEL 0
000041	INPUT INTERRUPT CHANNEL 1	OUTPUT BCR CHANNEL 1
000042	INPUT INTERRUPT CHANNEL 2	OUTPUT BCR CHANNEL 2
000043	INPUT INTERRUPT CHANNEL 3	OUTPUT BCR CHANNEL 3
000044	INPUT INTERRUPT CHANNEL 4	OUTPUT BCR CHANNEL 4
000045	INPUT INTERRUPT CHANNEL 5	OUTPUT BCR CHANNEL 5
000046	INPUT INTERRUPT CHANNEL 6	OUTPUT BCR CHANNEL 6
000047	INPUT INTERRUPT CHANNEL 7	OUTPUT BCR CHANNEL 7
000050	INPUT INTERRUPT CHANNEL 8	OUTPUT BCR CHANNEL 8
000051	INPUT INTERRUPT CHANNEL 9	OUTPUT BCR CHANNEL 9
000052	INPUT INTERRUPT CHANNEL 10	OUTPUT BCR CHANNEL 10
000053	INPUT INTERRUPT CHANNEL 11	OUTPUT BCR CHANNEL 11
000054	INPUT INTERRUPT CHANNEL 12	OUTPUT BCR CHANNEL 12
000055	INPUT INTERRUPT CHANNEL 13	OUTPUT BCR CHANNEL 13
000056	INPUT INTERRUPT CHANNEL 14	OUTPUT BCR CHANNEL 14
000057	INPUT INTERRUPT CHANNEL 15	OUTPUT BCR CHANNEL 15

Table 2–1. Fixed Address Locations (Octal-Coded)
(Part 1 of 2)

ADDRESS	FUNCTION	
	490 MODE	494 MODE
000060	OUTPUT INTERRUPT CHANNEL 0	OUTPUT BCR CHANNEL 16
000061	OUTPUT INTERRUPT CHANNEL 1	OUTPUT BCR CHANNEL 17
000062	OUTPUT INTERRUPT CHANNEL 2	OUTPUT BCR CHANNEL 18
000063	OUTPUT INTERRUPT CHANNEL 3	OUTPUT BCR CHANNEL 19
000064	OUTPUT INTERRUPT CHANNEL 4	OUTPUT BCR CHANNEL 20
000065	OUTPUT INTERRUPT CHANNEL 5	OUTPUT BCR CHANNEL 21
000066	OUTPUT INTERRUPT CHANNEL 6	OUTPUT BCR CHANNEL 22
000067	OUTPUT INTERRUPT CHANNEL 7	OUTPUT BCR CHANNEL 23
000070	OUTPUT INTERRUPT CHANNEL 8	
000071	OUTPUT INTERRUPT CHANNEL 9	
000072	OUTPUT INTERRUPT CHANNEL 10	
000073	OUTPUT INTERRUPT CHANNEL 11	
000074	OUTPUT INTERRUPT CHANNEL 12	
000075	OUTPUT INTERRUPT CHANNEL 13	
000076	OUTPUT INTERRUPT CHANNEL 14	
000077	OUTPUT INTERRUPT CHANNEL 15	
000100	INPUT BCR CHANNEL 0	INPUT BCR CHANNEL 0
000101	INPUT BCR CHANNEL 1	INPUT BCR CHANNEL 1
000102	INPUT BCR CHANNEL 2	INPUT BCR CHANNEL 2
000103	INPUT BCR CHANNEL 3	INPUT BCR CHANNEL 3
000104	INPUT BCR CHANNEL 4	INPUT BCR CHANNEL 4
000105	INPUT BCR CHANNEL 5	INPUT BCR CHANNEL 5
000106	INPUT BCR CHANNEL 6	INPUT BCR CHANNEL 6
000107	INPUT BCR CHANNEL 7	INPUT BCR CHANNEL 7
000110	INPUT BCR CHANNEL 8	INPUT BCR CHANNEL 8
000111	INPUT BCR CHANNEL 9	INPUT BCR CHANNEL 9
000112	INPUT BCR CHANNEL 10	INPUT BCR CHANNEL 10
000113	INPUT BCR CHANNEL 11	INPUT BCR CHANNEL 11
000114	INPUT BCR CHANNEL 12	INPUT BCR CHANNEL 12
000115	INPUT BCR CHANNEL 13	INPUT BCR CHANNEL 13
000116	INPUT BCR CHANNEL 14	INPUT BCR CHANNEL 14
000117	INPUT BCR CHANNEL 15	INPUT BCR CHANNEL 15
000120	OUTPUT BCR CHANNEL 0	INPUT BCR CHANNEL 16
000121	OUTPUT BCR CHANNEL 1	INPUT BCR CHANNEL 17
000122	OUTPUT BCR CHANNEL 2	INPUT BCR CHANNEL 18
000123	OUTPUT BCR CHANNEL 3	INPUT BCR CHANNEL 19
000124	OUTPUT BCR CHANNEL 4	INPUT BCR CHANNEL 20
000125	OUTPUT BCR CHANNEL 5	INPUT BCR CHANNEL 21
000126	OUTPUT BCR CHANNEL 6	INPUT BCR CHANNEL 22
000127	OUTPUT BCR CHANNEL 7	INPUT BCR CHANNEL 23
000130	OUTPUT BCR CHANNEL 8	
000131	OUTPUT BCR CHANNEL 9	
000132	OUTPUT BCR CHANNEL 10	
000133	OUTPUT BCR CHANNEL 11	
000134	OUTPUT BCR CHANNEL 12	
000135	OUTPUT BCR CHANNEL 13	
000136	OUTPUT BCR CHANNEL 14	
000137	OUTPUT BCR CHANNEL 15	

Table 2–1. Fixed Address Locations (Octal-Coded)
(Part 2 of 2)

I/O FUNCTION PRIORITY	INTERRUPT PRIORITY
Output Transfer Channel 15	Buffer Control Word Parity Error
Input Transfer Channel 15	I/O Data Parity Error
13 or 17 Instruction	Power Loss
Output Transfer Channels 14-08	External Interrupt (ESI)
Input Transfer Channels 14-08	Input Monitor (ESI)
Output Transfer Channels 23-16	Output Monitor (ESI)
Input Transfer Channels 23-16	External Interrupt (ISI)
Output Transfer Channels 7-0	Input Monitor (ISI)
Input Transfer Channels 7-0	Output Monitor (ISI)
Day Clock Update	Day Clock Interrupt
Real Time Clock Update	Real Time Clock Interrupt
	Synchronizing Interrupt #0
	Synchronizing Interrupt #1

Table 2–2. Input/Output Priority

The assignment of buffers is programmed by a buffer instruction (73, 74, 75, or 76), except that an ESI input operation must always be initiated with monitor (75). These instructions activate a buffer and place a buffer control word (BCW) into a BCR. The channel number is obtained from the five-bit Channel Select register (CSR) which was previously loaded by an Enter CSR (7773) instruction. When the buffer is activated, control of I/O transfers is taken over by the BCW so that the CPU is free to process its own program. The BCW defines the first address in the buffer (see Figure 3–2) and specifies the number of consecutive addresses allotted to the buffer. With each transfer, the current buffer address is increased by 1 and the word count is decreased by 1. When the word count reaches 0, the buffer is terminated.

In an input buffer operation, each transfer (updating the BCW and loading the input data into a buffer location) is initiated by the Input Data Request signal. In an output buffer operation, the transfer is initiated by the Output Data Request signal.

When an External Interrupt signal is received on a channel, the channel number is loaded into the five-bit Interrupt Address Storage register (IASR), but the status word on the data lines is not automatically stored. An interrupt is generated and the interrupt routine references the IASR, rather than the CSR, for the channel number. A Store Channel (17) instruction must be used to transfer the status word from the input data lines to primary storage and to send an Input Acknowledge (IA) signal to the subsystem as recognition of the External Interrupt (EI).

Basically, four modes of normal I/O data transfer operation are available:

- ISI mode (input)
- ISI mode (output)
- ESI mode (input)
- ESI mode (output)

The ISI modes use fixed addresses (see Table 2–1) for BCR's; the ESI modes can use any memory address as an ESI location (but once specified and wired for a particular I/O device it remains fixed for that I/O device). Actually, the number of ESI buffers which may be active on any one channel is restricted by the amount of primary storage available, the maximum allowable I/O cable length, and the overall transfer rate (which cannot exceed that of the CPU I/O logic).

On ESI input, the peripheral equipment sends an Input Data Request (IDR) along with a 30-bit word in the following format:

5, 6, 7 or 8-bit character (right-justified)	ESI address
29 15	14 00

The I/O logic of the CPU uses the ESI address (plus the two bits provided by the MSR) as the ESI location. The BCW is extracted from the ESI location, has its current address (see Figure 3–2) incremented, its word count decremented, and is replaced in the ESI location. Then the I/O logic stores the data on the upper 15 data input lines into the last previous current address (right-justified in the upper 15 bits with zero-fill; the lower 15 bits are undisturbed and may be used for any purpose).

On ESI output, the peripheral device must be connected to both the input and output channels. When the peripheral device sends the computer a request for an output data word, the peripheral device places the address of the corresponding output ESI BCW on the lower 15 bits of the input data lines. I/O logic uses this address (plus contents of the MSR) as the location of the assigned output ESI location. The BCW is fetched, updated, and replaced. Then the word at the last previous current address is sent by way of the 30 output data lines.

Upon termination of an ESI output, the ESI location address is automatically stored in the lower 15 bits of the ISI output BCR for that channel (see Table 2–1). An example of ESI output termination would be: xxxxx 00542 located in address 056. This implies that the peripheral device on position 42 of a communication multiplexer presented ESI address 00542 on channel 14_{10} (16_8) along with the Output Data Request (ODR) signal. The data transfer took place, the word count was decremented to 0, and BCR 056 received the ESI location address as an output termination of an ESI channel.

The termination of an ESI input operation differs from the ESI output operation in that the ESI location address is not automatically stored. Upon termination, a monitor interrupt must occur to initiate a subroutine which stores the ESI location address at address Y by means of a Store Channel (17) instruction.

2.4.1. ISI Input Mode

The basic sequence for ISI input is:

(1) Program activates input buffer for channel specified by CSR (73 or 75 instruction).

(2) Peripheral unit places data on the 30 input data lines along with IDR signal.

(3) CPU responds to IDR as soon as possible.

(4) CPU updates BCW in assigned BCR.

(5) CPU stores a 30-bit input word at last previous 17-bit current address and sends Input Acknowledge (IA) signal.

(6) Peripheral unit senses IA signal and drops IDR and data signals.

Steps (2) through (6) are repeated for every data word until the number of words specified by the BCW have been transferred. The channel is deactivated during the transfer of the last word of the buffer.

2.4.2. ISI Output Mode

The basic sequence for ISI output is:

(1) Program activates output buffer for channel specified by CSR (74 or 76 instruction).

(2) Peripheral unit sends ODR, indicating that peripheral is ready to receive data.

(3) CPU responds to ODR when possible.

(4) CPU updates BCW in assigned BCR.

(5) CPU reads out data from the last previous current address in the BCR.

(6) CPU transmits 30-bit data along with Output Acknowledge (OA) signal.

(7) Peripheral equipment senses OA signal and stores 30-bit data.

(8) Peripheral equipment drops ODR signal.

Steps (2) through (8) are repeated for every data word until the number of words specified in the BCW have been transferred. The channel is deactivated when the CPU receives the ODR following the last data transfer of the buffer.

2.4.3. ESI Input Mode

The basic sequence for ESI input is:

(1) CPU activates input buffer for given channel (75 instruction).

(2) Peripheral unit places data on input lines along with IDR signal.

(3) CPU responds to IDR signal by updating BCW at location specified in input (plus MSR).

(4) CPU stores input data at upper half of last previous current address in ESI location and sends IA signal.

(5) Peripheral unit responds to IA signal by dropping IDR and input data signals.

Steps (2) through (5) are repeated for every data word until the number of words specified in the BCW have been transferred. The channel is deactivated as the last word is read into the CPU.

2.4.4. ESI Output Mode

The basic sequence for ESI output is:

(1) CPU activates output buffer for given channel (74 or 76 instruction).

(2) Peripheral unit sends address of output ESI location on lower 15 input data lines along with ODR signal.

(3) CPU responds to ODR signal when possible.

(4) CPU updates BCW at output ESI location.

(5) CPU reads out data from last previous current address in ESI location.

(6) CPU transmits 30-bit output data along with OA signal.

(7) Peripheral unit senses OA signal and stores data.

(8) Peripheral unit drops ODR signal.

Steps (2) through (8) are repeated for every data word until the number of words specified in the BCW have been sent. The channel is deactivated as the last word of the buffer is transferred out. At step (6) when the word count in the BCW is 0, the ESI location address on the lower 15 data input lines will be stored in the output BCR for that channel.

2.5. EXECUTIVE CONTROL

Executive control is maintained by a set of routines, the UNIVAC 494 Operating System, which is concerned with selection and execution of worker programs. The executive and other routines control the basic machine facilities and provide for utilization by the worker program within a multiprogram environment. The executive routine is loaded into the CPU as part of the system-initializing procedure, and, from that point on, may be regarded as a programmed extension of the CPU. The routine enforces such basic control as program protection, facility allocation, relocation in memory, etc., and employs essential features of the computer design in effecting its functions.

An executive routine is activated by the occurrence of an interrupt. A special interrupt is generated by the Executive Return instruction enabling a worker program to request service provided by the executive. Interrupt processing is facilitated by the automatic deactivation of RIR and IFR. With IFR inactive, no guard mode or write protection is effective and the executive index registers are active in the 17-bit mode. Executive index registers are used to retain values used in controlling the worker program. With relative addressing inactive, the executive routine can span a 131K memory through the 17-bit index registers. During execution of the interrupt routine, transfers of P to the IFR are inhibited. A Store IFR instruction within the interrupt routine will capture the IFR contents for restoration when the worker is reactivated.

An executive routine required in processing a service request will be activated from an interrupt routine or scheduled for subsequent activation. The interrupt processing is therefore usually limited to a switching or scheduling function so that interrupt processing of input/output, etc., will not be delayed. The interrupt routine will activate RIR, IFR, and release interrupt lockout when control is either switched or returned to the interrupted program. The Enter IFR instruction loads IFR and RIR from successive memory locations Y and Y+1. The relative index established will affect operands of following instructions so that a Jump instruction will normally be direct by an executive index register. Write protection limits established by a previous Load Program Lock-In register instruction and Guard Mode become effective with the activation of IFR. The index register configuration is active prior to the Enter IFR instruction and is effective during the execution step of the next instruction; the interpretation step, however, will be controlled by the new IFR. RIR may be activated independently by IFR by the Enter RIR instruction in special switch situations effected by the executive.

2.6. PROGRAM PROTECTION

Program protection prevents interference between worker programs or program elements. The IFR may select one of three different modes of program protection, or no program protection, to govern the operation of worker programs. (Program protection is automatically disabled for executive routines.) The protection modes are:

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

The Guard Mode is concerned with illegal instructions, and the program protection mode deals with program overextensions, or violations of program limits defined in the PLR.

2.6.1. Guard Mode Protection

The Guard Mode ensures that certain instructions cannot be executed in the course of a worker program. These instructions are: all I/O instructions (13, 17, 62, 63, 66, 67, 73, 74, 75, and 76), and instructions 7760 through 7767, 7770, 7772, 7773, 7774, 7776, and 7777 (instructions restricted to executive routines such as loading of the RIR or IFR). In the Guard Mode, Stop instructions will be executed with any applicable jump, but the stop condition will be inhibited. If the Guard Mode is enabled, any attempt to read the privileged instructions will cause an Illegal Instruction interrupt. The Guard Mode also checks to assure that interrupts are not locked out for a period exceeding 100 microseconds.

2.6.2. Program Protection Modes

Read and Write Protection with Guard Mode uses all the protective features of the Guard Mode and, in addition, will cause a Program Protection or Timeout interrupt if reading, writing, or jumping outside the limits set by the PLR is attempted.

Write Protection with Guard Mode also uses all the protective features of the Guard Mode and will also cause a Program Protection interrupt if writing is attempted outside the limits set by the PLR. However, reading and jumping to any area in primary storage are permitted and are not limited by the PLR.

Write Protection without Guard Mode will cause an interrupt only if writing is attempted outside the limits set by the PLR. Reading and jumping to any location in primary storage are permitted. All instructions in the repertoire can be executed. The Guard Mode is not activated.

Read and/or Write Protection is not applicable on references made by way of the P register. Therefore, a normal P advance or skip outside the limits set by the PLR is not detected. Attempts to reference an instruction outside the lock-in area by a Jump or Execute Remote instruction will be detected and any such reference will not be executed. Also, Read and/or Write Protection is not applicable to I/O data transfers between the CPU and peripheral equipment, since this type of storage protection is effected by Buffer Control words.

2.7. INDEX REGISTERS

Fourteen addressable index (B) registers are available. These registers provide for operand address modification, index codes, counters, and modifier incrementation. These registers are divided into two groups so that each group consists of seven registers. The intention is that one set be used by executive interrupt answering routines and the other set by all other programs.

Only one set of B registers is active at any one time and is selected by the IFR. The executive set is activated when the processor is master cleared and whenever an interrupt occurs. Under program control the executive routine must load the IFR to activate the worker set when leaving the executive routine to enter a worker program. The worker program, in turn, returns to an executive routine by way of an interrupt, thereby activating the executive set.

Each set of B registers can operate in one of two modes. In one mode all seven registers will operate as 15-bit registers. In the other mode three registers (B1, B2, and B3) will operate as 15-bit registers and four registers (B4, B5, B6, and B7) will operate as 17-bit registers. To determine which mode is active, a second bit in the IFR is used.

When the CPU is master-cleared or an interrupt occurs, the CPU will operate with the executive set of registers and will have three 15-bit registers and four 17-bit registers. As the executive routine jumps to a worker program, the IFR register is loaded with the appropriate bits to select the worker set of index registers and to select the desired bit mode.

Normal operation for a worker program is the 15-bit length for all index registers, permitting complete access to all data within a 32K memory area. This natural limitation is imposed because 15-bit half-words are normally used for address generation. Any given program including instructions and data will then be contained within a 32K memory area. Using the RIR, the absolute starting address of the 32K area can be varied throughout the 131K memory in increments of 64 words.

The purpose of the 17-bit B registers is to allow a program to reference data outside the 32K limit. The instructions of the program are still limited to the 32K words following the value in RIR, but the data can be stored anywhere in the 131K memory and can then be referenced directly by the program.

When transferring data into and out of the 17-bit B registers, full-word transfers should be used. To enter a 17-bit B register, the lower 17 bits of a 30-bit location are transferred while a full-word store instruction stores the 17-bit value into the lower 17 bits of a 30-bit location with the upper 13 bits cleared to 0's. Half-word transfers into or out of a 17-bit register cause the highest order bits to be set to 0.

Instructions 7741 through 7747 transfer (P-RIR) to B and jump to Y. This is normally a 15-bit value and it can be entered into any B register.

In the instructions which enter and store the worker set of index registers (7771, 7775), a full-word of memory is used; that is, 15 bits are transferred to and from B1, B2, and B3 and 17 bits are transferred to and from B4, B5, B6, and B7 with remaining upper bits written to 0 on a Store instruction. The executive routine uses this instruction whenever it is necessary to save a program environment.

2.8. MEMORY OVERLAP

The memory overlap mode capability is achieved by independent operation of the memory banks which make up the primary storage. For a series of one-operand instructions, the time per instruction approaches one memory cycle time, since successive memory references can use alternate memory banks. A manually controlled selector switch (MSR) provides either odd-even addressing or continuous addressing. In the odd-even mode, odd memory references use one memory bank; even memory addresses use another memory bank, minimizing the programming effort necessary for obtaining maximum memory overlap. Odd-even addressing is the normal mode of operation. The continuous mode is used to avoid storage in a particular memory bank (in case of a memory fault, for example). Memory overlap can be obtained in the continuous mode but requires greater programming effort.

2.9. RELATIVE ADDRESSING

Relative addressing permits access to memory locations on the basis of a relative index value, R_i. Two modes of relative addressing are available for worker programs; interrupt routines do not use relative addressing (or, in effect, R_i is +0). In the dual relative index mode, either the RIR or the lower limit (LL) of the program boundary in the PLR may be used for R_i for operand addresses; in the RIR relative index mode, only the RIR may be used for R_i for operand addresses. In both modes, instruction address memory references shall be relative to the RIR (except for fixed addresses). The dual relative index mode facilitates programming where instructions are located in one primary storage area and operands are located in another. An example of dual indexing is the common subroutine which is entered into primary storage only once but which may operate on data (operands) in several primary storage areas.

Bit 27 of the IFR determines the relative index mode. If bit 27 is 0, the RIR relative index mode is activated; if bit 27 is 1, the dual relative index mode is activated. When an interrupt occurs, the IFR and RIR are deactivated. When the executive routine transfers control to the worker program, the Enter IFR instruction may activate either the dual relative index mode or the RIR mode. The RIR must be entered anew before the jump to the worker program.

In the dual relative index mode, the b designator (see Section 3 for instruction word formats) determines the R_i as follows:

- If the b designator is 0, 1, 2, or 3, R_i is the contents of the RIR.
- If the b designator is a 4, 5, 6, or 7, R_i is the LL in the PLR.

An Enter B and Jump instruction (774-) or the capture of a relative address by a Return Jump instruction (64 or 65) will store the value of P minus the value of the RIR. Therefore, P values must be relative to the RIR. For example, an executive routine is usually terminated by an arithmetic jump instruction (60) which activates the IFR after the indexing operation. If the jump is direct (the k designator is 0 or 4), any b designator may be used although the dual relative index mode is activated, and P values will be relative to the RIR. However, if the jump is indirect (the k designator is not a 0 or 4) and the b designator is 4, 5, 6, or 7, the RIR would be used to obtain the "jump to" address, the LL would be used to obtain the first worker instruction, and all subsequent values of P would be relative to the LL. In the latter case, the captured relative address (P minus RIR) would not be the true relative address.

2.10. FIXED ADDRESSES

Addresses 000000-000137$_8$ are reserved for entrances to specific interrupt routines and I/O buffer control. Entrance to these addresses is the result of an interrupt or is accomplished automatically during I/O operations. Reference to these addresses is obtained by an original 15-bit address which is modified by the Memory Select register (MSR) to form a 17-bit address. The MSR can be manually set on the maintenance panel to 00, 01, 10, or 11 to select any of the four primary storage modules in the two memory banks. Normally this register is set to 00, and is set to another position usually as the result of a fault in the first memory bank, which setting must be accompanied by a change to the continuous addressing mode instead of the odd-even addressing mode if an MSR setting of 01 or 11 is to be used. Odd-even addressing can be retained if the MSR is set to 10 and the entire second bank is being used.

Relative addressing circuits do not modify fixed address references in the same way as the circuits modify other primary storage addresses. Assignment of fixed addresses is shown in Table 2–1. The MSR is used for bank selection when referencing fixed addresses, for address bias when relative indexing is deactivated, and for bank selection during the initial loading of the CPU.

2.11. DAY AND REAL-TIME CLOCKS

The CPU has two clocks that may be used for program timing: the Day Clock and the Real-Time Clock. The Day Clock is a twenty-four hour clock that records the time of day in hours, minutes, and hundredths of minutes. The Real-Time Clock is an 18-bit counter that is incremented every two hundred microseconds.

2.11.1. Day Clock

The Day Clock time is stored in primary storage at address 000016 in the following format:

TENS ⋮ UNITS	TENS ⋮ UNITS	TENS ⋮ UNITS	
HOURS	MINUTES	HUNDREDTHS	0 0 0 0 0 0 0 0
29 28 ⋮ 27 24	23 20 ⋮ 19 16	15 12 ⋮ 11 8	7 0

Time is displayed in hours, minutes, and hundreths of a minute on the operator's console. Every 600 milliseconds, the Day Clock sends a data request signal through the lowest priority input/output channel (channel 00) to update the time contained at address 000016 (bits 08–29). During each input transfer, the data is placed on the 22 upper input data lines. Every six seconds, a Day Clock interrupt occurs, routing the program to main memory address 000015, where the system may start a subroutine conditioned by the time of day.

2.11.2. Real-Time Clock

The Real-Time Clock updates the word at the fixed memory address 000017 every 200 microseconds. The word stored at memory address 000017 has the following format:

0 0 0 0 0 0 0 0 0 0 0 0	REAL-TIME CLOCK COUNT
29 … 18	17 … 00

Each automatic update adds 1 to the low order 18-bit portion of the word. The high order 12-bit portion of the word will be all 0's after each update operation. As the Real-Time Clock count is recycled from its maximum (all 1 bits) to its minimum (all 0 bits), the Real-Time Clock interrupt is generated, sending the program to fixed memory address 000014 for a subroutine to determine further action. The Real-Time Clock interrupt will be generated approximately every 52 seconds unless the contents of memory address 000017 are changed by a programmed instruction. The contents of this word may be used, for example, as an interval timer. To set the Real-Time Clock for a desired interval (in multiples of 200 microseconds), a 30-bit word is entered into address 000017 having, in bits 0–17, the ones complement of the desired interval. Thus, for a 200-microsecond interval, the word will be xxxxxxxxxxxx111111111111111110; for a 1-second interval, xxxxxxxxxxxx111110110001110111.

2.12. INTERRUPTS

Interrupts are signals which temporarily suspend operation of the program being processed and initiate processing of a routine which has an entrance at one of the fixed addresses. Interrupts are generated by programmed instructions, faults, clock operation, or I/O transfer operations.

An interrupt is either unconditional, requiring immediate attention, or conditional, which may not require immediate attention. An unconditional interrupt is never locked out and always locks out conditional interrupts. An interrupt lockout is defined as the condition whereby conditional interrupts may be locked out. This condition can be set up by execution of a Return Jump with j designator of 0 oı 1 (SIL) or by processing of any interrupt. The interrupt lockout is in effect until released by a Jump with j designator of 0 or 1 (RIL) or until the machine is master-cleared. There are also temporary I/O interrupt lockouts imposed by the hardware during these instructions. The lockouts are:

- The period between reading of a Repeat instruction from memory and initiation of the first execution of the repeated instruction. This is a period of two memory cycles.
- During all cycles (except the last) of instructions requiring more than one memory cycle. As an example: the Enter B-Worker (7771) requires seven operand references and locks out I/O interrupts for six memory cycles.

- The execution of an Enter IFR and RIR instruction (7761) locks out I/O interrupts for six memory cycles.
- The execution of an Enter RIR instruction (7766) locks out I/O interrupts for two memory cycles.

When an interrupt sends the program to a fixed address, the P register is not affected by the fixed address. The P register will contain the 17-bit absolute address of the next instruction in the interrupted program, except in the case of a Memory Parity interrupt. Because the Memory Parity interrupt can be generated at any time in the operation of an instruction, the P register will contain either the address of the instruction being performed or the address of the next instruction. It is important to note that if a Guard Mode fault or a Memory Parity fault occurs as a result of an operand read operation for a Jump instruction, the absolute address of the Jump instruction plus 1 will have been captured by the IFR.

The fixed address will usually contain either a Return Jump-Arithmetic or an Enter B and Jump (774-) instruction which captures P minus RIR, a relative address, for use in resumption of the interrupted program. If the relative address is a 17-bit value, the value must be stored in a 17-bit index register, otherwise the two highest order bits will be lost. If the program was interrupted in a Repeat sequence, the 17-bit current relative address of the repeated instruction with the j designator of the Repeat instruction will automatically have been stored in the IFR, together with a designator indicating the Repeat mode. For Jump instruction of the worker program, the 17-bit P value will have been stored in the IFR. On nonjump instructions, the value in the R register is stored in the IFR; the R register value will be a 17-bit absolute operand address or will be an operand dependent upon the operand source of the current instruction.

If an interrupt occurs during a Repeat sequence, address transfers to the IFR are inhibited until the interrupt is processed. The hardware retains this address in the IFR so that a Store IFR (7765) instruction can be executed. The IFR will remain undisturbed until an Enter IFR (7761) instruction is executed. Since the IFR may contain a Repeat designator and address, another Repeat instruction must not be attempted until the Store IFR instruction is executed. (See Repeat instruction, Section 4, for further details.)

2.12.1. Unconditional (Fault) Interrupts

Unconditional interrupts include:

- Memory Parity
- Program Protection or Timeout
- Executive Return
- Test and Set
- Floating-Point Underflow
- Floating-Point Overflow
- Illegal Instruction

2.12.1.1. Memory Parity Interrupt

The Memory Parity interrupt is caused by an error in a read or write operation. As a 30-bit word is written into primary storage, two parity bits (one for each half-word) are written along with the word. As a 15-bit half-word is written into primary storage, a parity bit is automatically generated and written into primary storage along with the half-word. When the contents of a primary storage address (either half-word or full-word) are read, the parity bit(s) are checked to assure that an odd number of bits have not been added or lost from the half-words originally written into memory. As a half-word is written into memory, the other half-word at the same address is automatically checked in a test read operation. Thus, a memory parity error can be detected in a memory read operation of a half-word write operation. A Memory Parity Error interrupt can occur in the interpretation or execution cycle of an instruction, resulting in the "uncertain" P value. Parity errors associated with I/O operations cause conditional interrupts and are discussed further in this section. Parity errors associated with internal transfer of data cause the Memory Parity interrupt, and a different fixed address is assigned to each of the four memory banks.

When a memory parity error occurs, the data is written back into the same address with the parity bit(s). Therefore, further references to the same address will produce another parity interrupt. If the error indicates a malfunction in a memory bank, the fixed locations can be moved to another bank and operation switched to continuous addressing by manual controls.

2.12.1.2. Program Protection or Timeout Interrupt

The Program Protection interrupt is generated when the program protection mode in effect (see 2.6) is violated. A Timeout interrupt is generated if a worker program, running with $IFR_{21} = 1$, has been locked-out through a Set Interrupt Lockout (SIL) instruction for more than 100 microseconds.

2.12.1.3. Executive Return Interrupt

The Executive Return interrupt is caused by the Executive Return (7754 or EXRN) instruction, which sends the program to a fixed address leading to a routine enabling capture of the P value of the program which is interrupting. The P register will contain the address of the instruction following the EXRN in the program when the interrupt is generated. The IFR will contain a (y+b) value produced by the executive routine.

2.12.1.4. Test and Set Interrupt

The Test and Set interrupt is the result of a Test and Set instruction (7752) which tests bit 14 that is stored at a selected primary storage location. If bit 14 is already set when the test is made, the interrupt is generated and an interrupt routine takes over; if bit 14 is not set, bits 0 through 14 are then set, no interrupt is generated, and the next sequential instruction in the program is executed. The P register will contain the address of the instruction following the Test and Set instruction in the program when the interrupt is generated.

2.12.1.5. Floating-Point Underflow Interrupt

The Floating-Point Underflow interrupt is generated when the result of the exponential arithmetic operation has an exponent less than -1024_{10} (see arithmetic instructions in Section 4). For multiplication and division, this exponent is also checked in the operands.

2.12.1.6. Floating-Point Overflow Interrupt

The Floating-Point Overflow interrupt is generated when the result of an exponential arithmetic operation has an exponent greater than $+1023_{10}$ or if exponential division by 0 is attempted, or if any of the exponential operands in multiplication or division has an exponent greater than $+1023_{10}$.

2.12.1.7. Illegal Instruction Interrupt

The Illegal instruction interrupt is generated if the Guard Mode is activated and an attempt is made to violate the Guard Mode by privileged instructions. The interrupt is also generated if an attempt is made to execute an instruction with a function code of 00 or 7700. Instructions using unassigned function codes are, in effect, No Operation instructions. No interrupt will be generated; these codes should not be used. If 77xx instructions are attempted with the CPU in the 490 mode, the interrupt will be generated.

2.12.2. Conditional Interrupts

Conditional interrupts are associated with equipment faults, and clock, synchronizing, and I/O operations. Where two interrupts occur simultaneously, channel priority (where applicable) is used as a secondary level in the following order: 15---8, 23---16, 7---0, with channel 15 having highest priority and channel 0 having the lowest priority. Conditional interrupts include:

- BCW Parity
- I/O Data Parity
- Power Loss
- External Interrupt (ESI)
- Input Monitor (ESI)
- Output Monitor (ESI)
- External Interrupt (ISI)
- Input Monitor (ISI)
- Output Monitor (ISI)
- Day Clock Interrupt
- Real-Time Clock Interrupt
- Synchronizing Interrupt #0
- Synchronizing Interrupt #1

2.12.2.1. Buffer Control Word (BCW) Parity Interrupt

The BCW Parity interrupt is generated to indicate a parity error in the BCW as it is read from a BCR or ESI location. Every data transfer between a CPU buffer and external equipment involves reading of the BCW, updating the BCW and writing the BCW back into memory. Only during the read cycle can a parity error be detected and the interrupt be generated. If the parity error is detected, the write cycle will write all 0's (parity incorrect) in the BCW. In a buffer input operation, no data word will be stored. The Input Data Acknowledge is sent only on an ESI channel. In a buffer output operation, the data will be sent as all 1's together with the Output Data Acknowledge signal. If the error occurs on an ISI channel, the channel will be deactivated; if an error occurs on an ESI channel, the channel will not be deactivated.

2.12.2.2. I/O Data Parity Interrupt

The I/O Data Parity interrupt indicates a parity error during I/O operations when reading an output data word or writing half-word data (ESI input) as part of I/O operations. Transfers of the data will be completed and channels will not be deactivated. The BCW will be updated and the Output Data Acknowledge signal will be sent.

2.12.2.3. Power Loss Interrupt

The Power Loss interrupt indicates that the input line voltage has dropped below 80 percent of nominal value (which still permits operation). The interrupt routine then has a fixed time of 5 milliseconds to store the status of the program so that when power is restored, the program is supplied with an entry point and can recover in an orderly manner. If the program is operating after this interval, the loss was transient and the program can be resumed with no loss of data.

2.12.2.4. External Interrupt

The External interrupt (EI) is a signal sent from external equipment to CPU together with a status word on the data input lines. The interrupt routine will contain a Store Channel (17) instruction for storing the status word in primary storage. The channel number will automatically be stored in the Interrupt Address Storage register. After the status word has been stored, the Input Data Acknowledge signal will be sent as a result of execution of the Stop Channel instruction.

2.12.2.5. Monitor (Internal) Interrupt

The Monitor interrupt (also termed internal interrupt) occurs when a buffer, activated with monitor, has been either filled or emptied. The interrupt is generated after the write operation (output) or read operation (input) with this one exception: the buffer must receive the next Output Data Request signal before generating the interrupt on an ISI output channel.

2.12.2.6. Day Clock Interrupt

The Day Clock interrupt is generated every 6 seconds, enabling processing operations to be conditioned by the time of day.

2.12.2.7. Real-Time Clock Interrupt

The Real-Time Clock interrupt is generated every time the contents of the Real-Time Clock recycle from all 1's to all 0's and may be used for interrupt routines which perform statistical analyses of CPU operations, or for time-outs to prevent program "looping", etc.

2.12.2.8. Synchronizing Interrupt

There are two Synchronizing Interrupt signals: Synchronizing Interrupt #0 and Synchronizing Interrupt #1. The Synchronizing Interrupt signals are sent from one CPU (as the result of a programmed instruction) to another CPU, and cause an interrupt at the receiving CPU. When a Synchronizing Interrupt #0 occurs, the CPU program takes the next instruction from location 012_8; if a Synchronizing Interrupt #1 occurs, the CPU program takes the next instruction from location 013_8.

2.12.3. I/O Interrupt Registers

The registers which are used by I/O interrupt routines are the Channel Select register (CSR), the Interrupt Address Storage register (IASR), and the Parity Error Channel Storage register (PECSR).

In normal operations, I/O instructions refer to the CSR to determine which channel to activate, deactivate, or test. (The CSR must have been loaded previously by an Enter Channel Select register instruction (7773).)

The IASR contains the number of the channel on which a monitor interrupt signal or an External Interrupt (EI) signal occurs. During the interrupt subroutine, the IASR is used in place of the CSR to specify the channel. When the Store Channel Number (7772) is executed, the number in the IASR is stored. The use of the IASR does not alter the CSR; after the subroutine is completed, I/O instructions again refer to the CSR. The use of the IASR is enabled from the time the interrupt is honored (enter subroutine) until the interrupt lockout is released. It should be noted that if interrupts are locked out under program control, the CSR will be referenced.

The PECSR is used in error routines initiated by a BCR Parity Error interrupt or I/O Data Parity Error interrupt. The channel number of the interrupt is automatically stored in the PECSR. When the Store Channel Number (7772) is executed, the contents of the PECSR will be stored. Within the BCR or I/O Data Parity Error routine, the CSR is used to specify which channel to activate, deactivate, or test.

3. WORD FORMATS

3.1. GENERAL

The CPU uses three types of words for internal manipulation: instruction words, register words (the data contained in the registers of Figure 2–2), and arithmetic operands. The variations within each of these types are described in the following paragraphs. Detailed descriptions, where applicable, are furnished in Section 4. (Abbreviations and special symbols used in the text are listed and defined in Table A–1.)

3.2. INSTRUCTION WORDS

The instruction word formats (Figure 3–1) used for internal operations are: the normal instruction word, the 77 instruction word, and the I/O instruction word. An instruction word indicates the operation to be performed and the operand to be used; and may also indicate the condition for a skip (skip next instruction in the program) or a jump operation. A NO OP instruction is a variation of an instruction in the repertoire and does not initiate any particular operation but permits a time delay between two instructions.

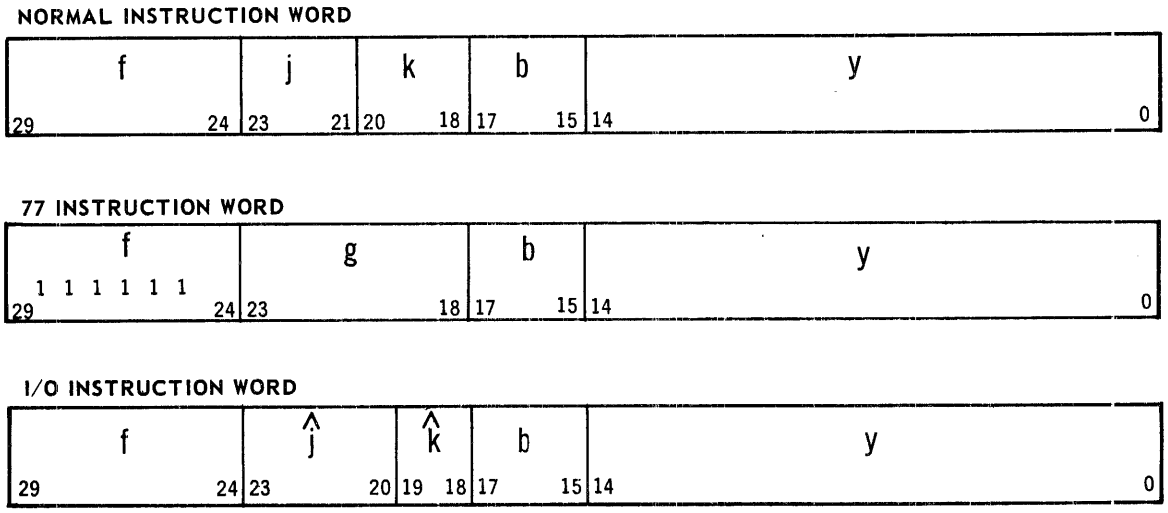


Figure 3–1. Instruction Word Formats

3.2.1. Normal Instruction Word

The normal instruction word format is applicable to all instructions of the instruction repertoire except the 77 and I/O instructions. The term "normal" is used to indicate that the word is applicable to a wide range of instructions. The instructions are further subdivided into "classes," as shown in the instruction repertoire, which condition interpretation of the k designator of the word. These classes are: *read, store,* and *replace.*

A read class instruction transfers data from primary storage to an appropriate register.

A store class instruction transfers data between registers or from a register to primary storage.

A replace class instruction replaces the data in primary storage with the result of an operation performed upon this data and is a combination of read and store class.

The repertoire of instructions, Table B–1, indicates the class of each instruction, where applicable.

The **f designator** is a six-bit code (two octal digits) that specifies the operation to be performed.

The **j designator** is a one-digit (octal) code that may be interpreted as a skip designator, register designator, repeat modification, or jump designator. The most common interpretation (instructions 01-03, 05-11, 14, 15, 20, 21, 24, 25, 30-37, 41, 42, and 45–57) is a skip designator (see Table 3–1) together with a special application for instructions, 04, 12, 16, 22, 23, 26, 27, 40, 43, 44, 60, 61, 64, 65, and 70.

The **k designator**, together with the class of instruction defines the source of the operand or its destination (see Table 3–2). The k designator permits use of half-word operands of 15 bits with the remaining 15 bits filled either with 0's (zero-fill) or with the highest order bit of the transferred half-word (sign-fill).

The **b designator** indicates an index register (1–7) whose contents (B_b) are added to the 15-bit **y designator** to form either a 15-bit or 17-bit effective operand or relative primary storage address which (when added to R_i) is the location of the operand or (as in a Jump instruction) is the location of the next instruction to be performed. If the b designator is a 0, then all 0's are used in the addition to form the relative address so that the y designator, in this case, is either the relative address or effective operand. As an example: a read class instruction with a k designator of 0 or 4 uses the effective operand in the instruction; a read class instruction with k designator of 1, 2, 3, 5, or 6 uses the word (or instruction) at the relative address; a read class instruction with k designator of 7 uses the word in the accumulator as the operand.

The relative address or effective operand is formed by the addition y + (B_b). Both numbers are treated as unsigned positive numbers. This addition is always performed in the end-around carry mode: any carry generated at the highest-order bit position is carried around for addition at the lowest-order bit position. If the 15-bit index register configuration is activated by the IFR, a 15-bit addition will occur. If the dual-index register configuration is activated and B4, B5, B6, or B7 is referenced, this addition will be a 17-bit addition; otherwise, a 15-bit addition is performed. It is not possible to generate an address (or operand) of all binary zeros unless y is all binary zeros and either B0 is referenced or the contents of the referenced index register is all binary 0's. The following examples illustrate both 15-bit and 17-bit additions with an end-around carry (using octal notation):

15-BIT ADDITION

$$\begin{array}{lcr} y & = & 77777 \\ (B_b) & = & \underline{00001} \\ & & 00000 \\ & & \underline{1} \\ \bar{y} & = & 00001 \end{array}$$

17-BIT ADDITION

$$\begin{array}{lcr} y & = & 77777 \\ (B_b) & = & \underline{300001} \\ & & 000000 \\ & & \underline{1} \\ \bar{y} & = & 000001 \end{array}$$

3.2.2. 77 Instruction Word

The 77 instruction words extend the range of functions of the 494 System above the functions of the 490/491/492 Systems, and also handle I/O functions.

In the format for 77 instruction words, the f designator is 77_8. The g designator (two octal digits) actually defines the operation to be performed. The b and y designators are similar in function to their counterparts in the normal instruction word. Since no k designator is used in the format, the term "class", which defines interpretation of the k designator, is not applicable.

3.2.3. I/O Instruction Word

The I/O instruction word is used for instructions governing transfer of data to and from peripheral equipment (tapes, drums, printers, etc.) and communication subsystems, and acts as the I/O function word for 490 mode operation. The I/O instruction word format is applicable to the 13, 17, 62, 63, 66, 67, 73, 74, 75, and 76 instructions.

The **f, b, and y designators** are functionally similar to their counterparts in the normal instruction word format. The **$\hat{j}$ designator** is a four-bit code limited to a value of 0 or 1 in the 74 and 76 instruction, and to a value of 0 (causing no modification) in the other I/O instructions. In the 74 and 76 instructions, a 0 indicates that data is to be sent; a 1 indicates that a function word is to be sent. The **$\hat{k}$ designator** (since these are read class instructions) interpretation is the same as k designator for read class instructions, but is limited to certain specific values presented in the detailed description of I/O instructions.

	INSTRUCTION						
j DESIGNATOR	NORMAL**	04	12, 72	16	22	23	26, 27
0	no skip	no skip	no op	Store 0's	no skip	no skip	no skip
1	Skip	Skip	B_j = B1	B_j = B1	Skip	Skip	Skip
2	Skip if sign in (Q) is +	Skip if Y≤(Q)	B_j = B2	B_j = B2	No Overflow	Overflow	Skip if sign in (A) is +
3	Skip if sign in (Q) is –	Skip if Y>(Q)	B_j = B3	B_j = B3	Overflow	No Overflow	Skip if sign in (A) is –
4	Skip if (A) is +0	Skip if (A)<Y≤(Q)	B_j = B4	B_j = B4	Signed prod. ≤ 31 bits	Remainder =+0	Skip if (Q) is +0
5	Skip if (A) is not +0	Skip if Y>(Q) or Y≤(A)	B_j = B5	B_j = B5	Overflow	Remainder ≠+0	Skip if (Q) is not +0
6	Skip if sign in (A) is +	Skip if Y ≤ (A)	B_j = B6	B_j = B6	Skip	Remainder=+or+0	Skip if sign in (Q) is +
7	Skip if sign in (A) is –	Skip if Y > (A)	B_j = B7	B_j = B7	no skip	Remainder= or-0	Skip if sign in (Q) is –

**Normal interpretation: 01–03, 05–11, 14, 15, 20, 21, 24, 25, 30–37, 41, 42, and 45–47 instructions.

	INSTRUCTION					
j DESIGNATOR	40, 44	43	60, 64	61, 65	70	71
0	no skip	no skip	no jump; 60–clear lock-out 64–lock-out interrupts	Jump	Next Y=Y	Skip if Y is +0
1	Skip	Skip	Jump; 60–clear lock-out 64–lock-out interrupts	Jump if key 1 is set	Next Y=Y+1	B_j = B1
2	Skip if (A) even no. of 1's	(Q) = +	Jump if sign in (Q) is +	Jump if key 2 is set	Next Y=Y–1	B_j = B2
3	Skip if (A) odd no. of 1's	(Q) = –	Jump if sign in (Q) is –	Jump if key 3 is set	Next Y=Y+(B_b)	B_j = B3
4	Skip if (A) is +0	(A)–LP(Y·Q)=+0	Jump if (A) is + 0	Jump and stop	Next Y=Y[+(B_6)]*	B_j = B4
5	Skip if (A) is not +0	(A)–LP(Y·Q)≠+0	Jump if (A) ≠ + 0	Jump; stop if key 5 is set	Next Y=Y+1[+(B_6)]*	B_j = B5
6	Skip if sign in (A) is +	(A)–LP(Y·Q)=+	Jump if sign in (A) is +	Jump; stop if key 6 is set	Next Y=Y–1[+(B_6)]*	B_j = B6
7	Skip if sign in (A) is –	(A)–LP(Y·Q)+–	Jump if (A) ≠ + 0	Jump; stop if key 7 is set	Next Y=Y+(B_b)[+(B_6)]*	B_j = B7

Table 3–1. Interpretation of j Designators

If instruction being repeated is Replace (Rp) class instruction, result is stored at Y+(B_6), where Y is the prevailing Y of the current execution.

3.3. REGISTER FORMATS

This paragraph describes the formats of those registers which condition the operating environment for worker programs—the IFR, the Program Lock-In register (PLR), and the RIR – and the Buffer Control register (BCR) which controls buffer operations between CPU and peripheral equipment (see Figure 3–2).

IFR FORMAT

f9	x	f8	f7	f6	f5	f4	f3	f2	0	f1
29	28	27	26	25	24	23	22 21	20 18	17	16 00

PLR FORMAT

0 0 0 0	UPPER LIMIT	0 0 0 0	LOWER LIMIT (LL)
29 26	25 15	14 11	10 00

RIR FORMAT

NOT USED	0	INDEX VALUE	NOT USED
29 18	17	16 06	05 00

BCR FORMAT (BCW)

ADDRESS COUNT	x	CURRENT ADDRESS
29 18	17	16 00

NOTE: X indicates not used.

Figure 3–2. Register Formats

3.3.1. Internal Function Register (IFR) Format

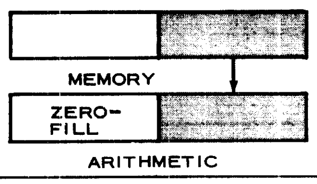
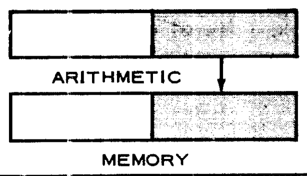
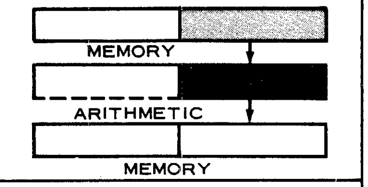
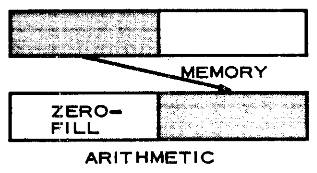
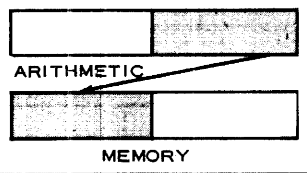
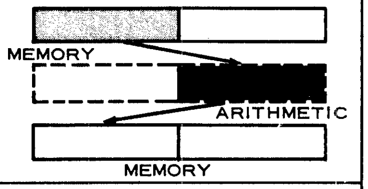
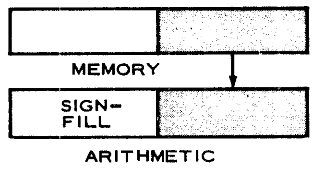
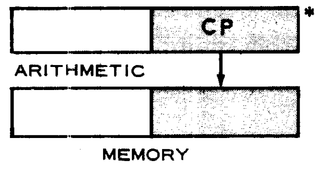
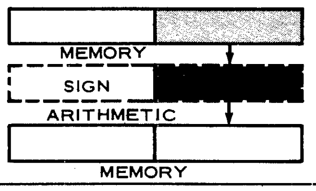
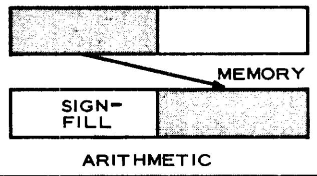
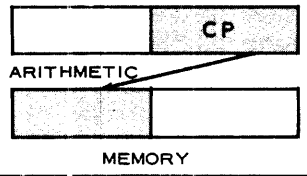
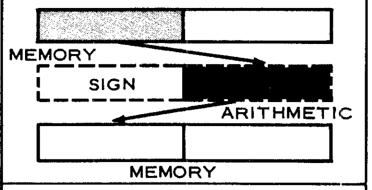
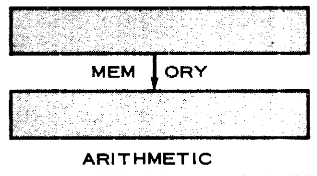
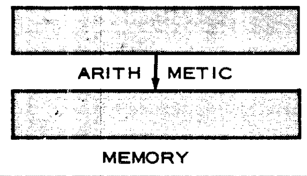
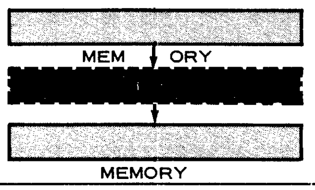
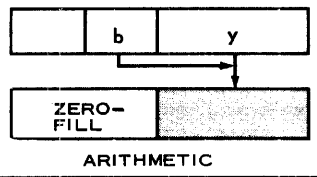
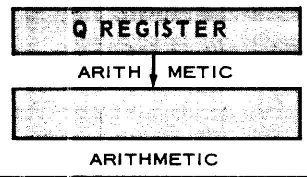
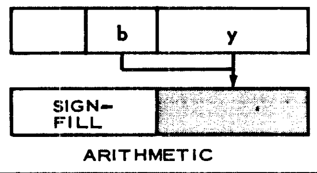
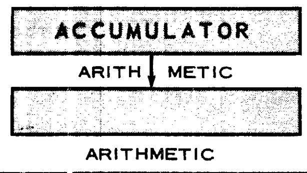
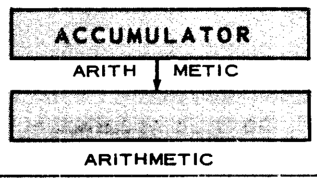
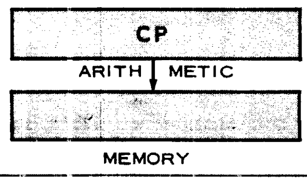
The IFR is loaded, under executive control, by an Enter IFR (7761) instruction to condition worker program environment and aid in recovery procedures.

The contents of **field f1** are determined by whether a repeat sequence is interrupted (field 9 is set) or whether a nonrepeat sequence is interrupted or being executed. On interruption of a repeat sequence:

(1) If a Fault interrupt occurs between the time that the operand for an Enter IFR (with bit 29 = 1) instruction is read from memory and the first execution for the reinitiated Repeat instruction, **field f1** is the **field f1** read from memory.

(2) If an I/O interrupt occurs during any cycle but the last of a repeated instruction, **field f1** contains the relative address, $\bar{y}$, to be used for the next execution of the repeated instruction.

(3) If a Fault interrupt occurs during any cycle of a repeated instruction, **field f1** holds either the absolute address, $\bar{y}$ + (RIR), on which the fault occurred or the effective operand, $\bar{y}$, depending upon the use of $\bar{y}$ in the instruction.

k	READ	STORE**	REPLACE
1			
2			
5			
6			
3			
0			
4			
7			

* CP is ones complement

** Exceptions:

1. f = 14, k = 0: CP(Q) $\longrightarrow$ Q
2. f = 15, k = 4: CP(A) $\longrightarrow$ A
3. f = 16, k = 0: $(B_j) \longrightarrow Q_L$, zero-fill
   k = 3: $(B_j) \longrightarrow (\bar{y})_L$
   k = 4: $(B_j) \longrightarrow A_L$, zero-fill
   k = 7: $CP(B_j) \longrightarrow (\bar{y})_L$, Sign-fill
4. f = 47, k = 0: LP $\longrightarrow$ Q
   k = 4: LP $\longrightarrow$ A
   k = 7: not used

Table 3–2. Interpretation of k Designators

On nonrepeat instructions:

(1) On all nonjump instructions, **field f1** contains either the effective operand or absolute address, $\bar{y}$ + (RIR), of the instruction. This value is retained upon occurrence of a Fault or I/O interrupt.

(2) On jump instructions, the absolute value of the next instruction (the absolute value of the current instruction plus one) is placed in **field f1** and retained if a Fault or I/O interrupt occurs. The one exception to this is a Jump following an Enter IFR with bit 29 = 1, which was previously described.

Field f2 is the j designator of a Repeat instruction when the Repeat sequence is interrupted. **Field f3** determines the program protection mode. If this field is 00, there is no Guard Mode and no write protection; if 01, there is Guard Mode with read and write protection; if 10, there is write only protection; if 11, there is Guard Mode with write protection but no read protection. **Field f4** indicates an end-off borrow from a previous decimal subtraction. **Field f5** indicates an end-off carry from a previous decimal addition. When **field f6** is a 0, the executive set of index registers is used by an instruction; when 1, the worker set is used. When **field f7** is a 1, the active Index registers 1, 2, and 3 act as 15-bit registers, and Index registers 4, 5, 6, and 7 act as 17-bit registers; when a 0, all active index registers are 15-bit registers. When **field f8** is a 1 and the b designator of an instruction is 4, 5, 6, or 7, the R_i uses LL to form the absolute address; for all other cases, the R_i uses the index value in the RIR. Bit 28 is not used. **Field f9,** when a 1, indicates that a Repeat sequence was interrupted and that field f1 contains an effective address; when a 0, it indicates that field f1 contains an absolute address. At the end of an executive routine when control is transferred back to the worker program, the Repeat will be reinitiated if this bit is a 1. When this field is clear, it indicates that field f1 contains an absolute value.

3.3.2. Program Lock-In Register (PLR) Format

The PLR defines the upper and lower memory limits for program protection and provides the lower limit (LL) for the R_i. The register is loaded by a Load PLR (7762) instruction under executive control. When this register is used in the worker program, bits 00 through 10 are the 11 high order bits of a 17-bit number for the LL; the six low order bits are assumed to be 0's. In the Upper Limit, the six lowest order bits are assumed to be 1's. The LL used for the R_i is also a 17-bit number formed by this procedure. If both the Upper Limit and the Lower Limit are the same as, for example, 1234_8, protection will be furnished for addresses 123400_8 through 123477_8. Bits 11–14 and 26–29 are always written to 0's.

3.3.3. Relative Index Register (RIR) Format

The RIR is used in relative addressing and is loaded under executive control by an Enter RIR (7766) or the Enter IFR and RIR (7761) instruction. The 11 bits (06–16) are the high order bits of a 17-bit number for the R_i. The six low order bits are 0's. Bits 00–05 and 18–29 are not used and are usually written to 0's.

An interrupt or fault condition will deactivate the RIR. Memory references beginning with the start of the interrupt routine will not be biased by the RIR. The contents of the RIR are retained to be used by a Return Jump or an Enter Bx and Jump instruction, contained in the fixed address for the particular type of interrupt, to store a relative address. When this is completed, the RIR is cleared. During the time that no relative addressing is used in the interrupt routine, the Memory Select register (see 2.10) furnishes the two highest order bits to make up a 17-bit address and thereby selects the memory bank to be used.

The hardware locks out I/O interrupts from the time the Enter RIR instruction or Enter IFR and RIR instruction is read from memory until the RIR contains the new value. Therefore, an interrupt which occurs after an Enter RIR will capture a relative P register value relative to the new RIR.

The absolute memory address reference for an instruction is obtained by adding the relative address to the relative index (R_i) and is then loaded into the P register. The relative index may come from either the LL in the PLR or from the RIR. This addition is always a 17-bit addition and any carry at the highest order bit position is discarded. This makes possible generation of an absolute address equal to 000000 as shown in the following (using octal notation):

$$\begin{aligned} \bar{y} &= 77000 \\ R_i &= 301000 \\ \text{absolute address} &= 000000 \end{aligned}$$

This relative addressing makes possible "going around the end of memory," that is, a sequence of absolute addresses in a program might be . . . 377777, 000000, 000001 . . . When an effective address is to be captured as in an interrupt routine, it is obtained by subtracting the contents of the RIR from the contents of the P register. Obviously, such a procedure is useless when absolute addressing goes around the end of memory. In this case, all 0's are captured.

3.3.4. Buffer Control Register (BCR) Format

A buffer control register (BCR) is used to control I/O transfer operations between a portion of primary storage used as a buffer and the data peripheral equipment. A buffer control word (BCW) is loaded into a BCR as part of the I/O routines. Initially, the lower 17 bits of the BCW define the first address of a buffer; the upper 12 bits, the number of primary storage addresses allotted to the buffer. The address portion permits access to any primary storage address; the 12 bits of the address count portion limit a buffer to 4096_{10} addresses. As each data transfer between buffer and peripheral equipment takes place, the current address is increased by 1 and the address count is decreased by 1. When the address count is 0, the buffer is terminated and the current address will be one more than the last address used. Buffer control register is the term used for that memory location holding a BCW during ISI (internally specified index) I/O data transfers. The memory register for the BCW which governs an ESI (externally specified index) I/O data transfer is termed an ESI location. The formats of the BCW in the BCR and the ESI location are identical.

3.4. ARITHMETIC OPERANDS

Four types of operand formats are used (Figure 3–3), depending upon the type of arithmetic operand:

- Integer, single precision
- Integer, double precision
- Decimal
- Exponential

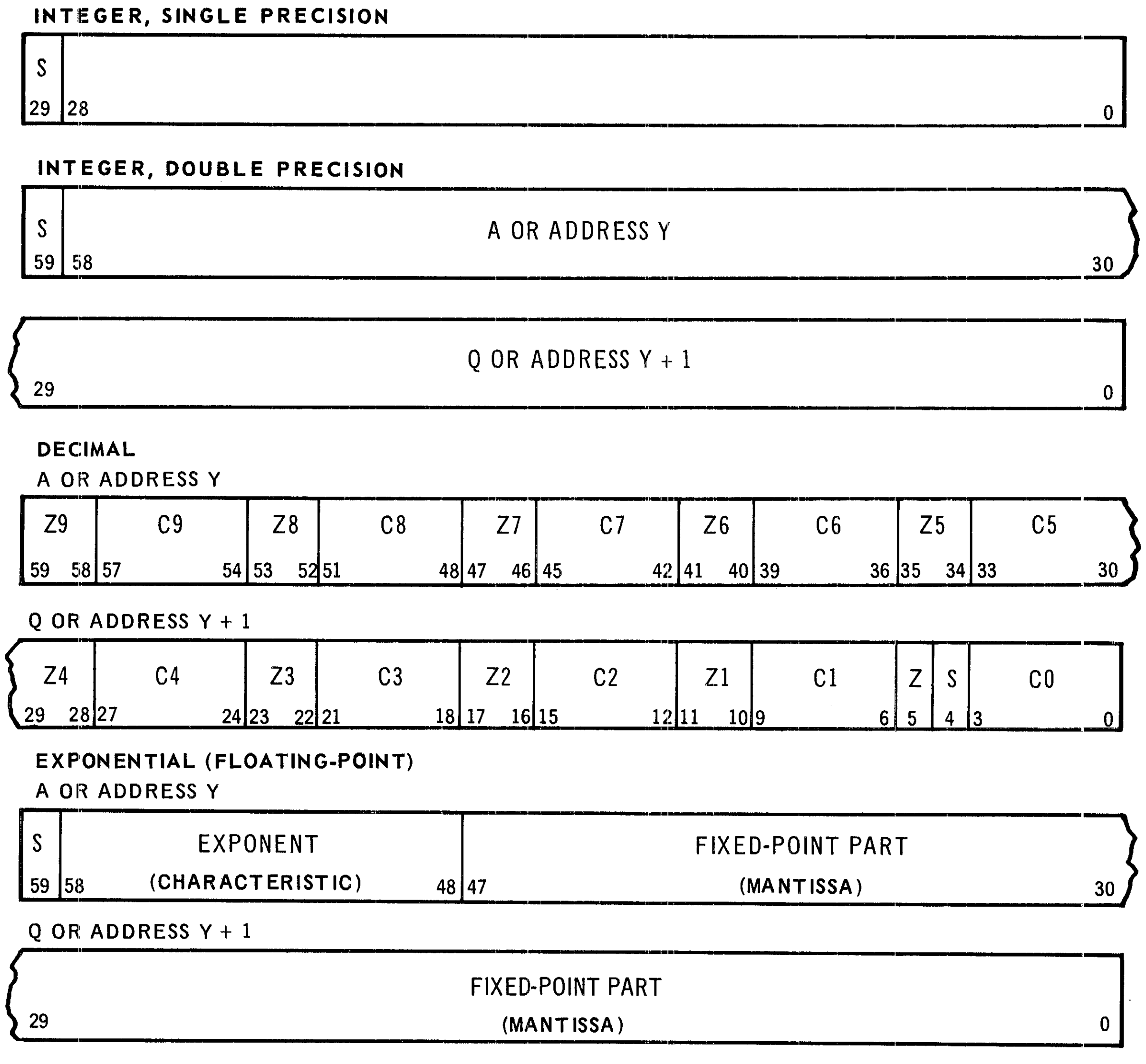


Figure 3–3. Arithmetic Operand Formats

3.4.1. Integer, Single Precision

The operand used in integer, single-precision arithmetic operations is a 30-bit word which may be stored in a register or memory location. The k designator permits the use of half-word operands from memory by filling in the vacated 15-bit positions, as required. Bit 29 indicates the sign of the operand: a 0 for positive, a 1 for negative.

Because ones complement arithmetic is used in the CPU, a negative number is the ones complement of the same positive number. A + 0 would be presented as all 0 bits, while a -0 would be presented as the ones complement of +0, which is all 1 bits (7777777777_8). The absolute value of a 30-bit operand may not exceed 3777777777_8 (which is 536870911_{10}); a half word, 16383_{10}.

3.4.2. Integer, Double Precision

Results to 20 octal digits are obtained through integer, double-precision operations. The sign and high order bits are stored in the accumulator (or at a primary storage address) with the low order bits stored in the Q register (or at the next successive primary storage address). Positive and negative operands are presented and stored in the same manner as for the single-precision counterpart.

3.4.3. Decimal

Decimal operands are used when inputs arrive as a succession of 10 six-bit characters in conformance with a code similar to the Fieldata code. The "Z" (zone) bits shown in the format are arbitrary and are determined by the code itself, playing no part in the arithmetic operation. The Z (zone) bits are unchanged by the arithmetic operation. However, the lowest order digit must indicate the sign in its fifth-bit: a 1 for positive, a 0 for negative. The C (character) fields, which actually represent the binary coded decimal (BCD) digit, must be encoded as shown in Table 3–3. No other encoding is acceptable for the BCD arithmetic instructions in the instruction repertoire. A positive decimal operand is exactly the same as the negative decimal operand (having the same absolute value) except for the sign bit.

DIGIT	CODING	DIGIT	CODING
0	0000	5	0101
1	0001	6	0110
2	0010	7	0111
3	0011	8	1000
4	0100	9	1001

Table 3–3. BCD Coding

3.4.4. Exponential (Floating-Point)

The exponential operand is used in those arithmetic operations where the operand is expressed as a fraction (fixed-point part) multiplied by 2^n, where n is the exponent in the format shown in Figure 3–3. The exponent is always an integer. Two successive memory addresses and/or the AQ register are required for floating-point operations. The sign bit represents the sign of the fixed-point part and therefore represents the sign of the floating-point number (a 0 for positive, a 1 for negative). The characteristic is biased by 2000_8. A positive exponent is added to this bias to determine its machine representation; a negative exponent, is subtracted. This bias eliminates the need for a sign but limits the value of the exponent to $1023_{10} \geq E \geq 1024_{10}$. The fixed-point part is usually "normalized" (shifted left until its highest order bit is a 1) so that it represents a fraction between 0.5 and 1.0. A negative floating-point number is represented as the 60-bit ones complement of the equivalent positive floating-point number. An exception is that the number 0 is represented as all 0 bits in floating-point format.

The octal-coded machine representation for different signed combinations of $0.75_{10} \times 2^3$ is:

0.75×2^3	2003 6000000000000000
-0.75×2^3	5774 1777777777777777
0.75×2^{-3}	1775 6000000000000000
-0.75×2^{-3}	6002 1777777777777777

4. INSTRUCTIONS

4.1. GENERAL

The instruction repertoire (Table B–1) of the central processor is composed of:

- Shift instructions
- Transfer instructions
- Arithmetic instructions
- Logical instructions
- Compare instructions
- Jump instructions
- Sequence-modifying instructions
- Input/output (I/O) instructions

Abbreviations and special symbols used in this text are defined in Table A–1. Table B–1 lists instructions by function and by function code. The class of each instruction (read, store, replace) is also indicated in Table B–1. The class of the instruction indicates the proper interpretation of the k designator in the instruction in order to derive the operand, Y. Unless otherwise specified in the detailed description of an instruction, the class of instruction is read class and is interpreted as shown in Tables 3–2 and 4–1. Unless the detailed description states otherwise, the j designator is interpreted as shown in the "normal" column of Table 3–1. An example of a read class instruction with normal j interpretation is an Enter A (11) instruction.

Shift instructions move the contents of a selected register to the right or left as many positions as specified. **Transfer instructions** move data between a primary storage location and a register. **Arithmetic instructions** perform arithmetic in four modes (see 3.4): (1) integer, single precision, (2) integer, double precision, (3) decimal, and (4) exponential. **Logical instructions** provide the means for operation upon and editing selected bits of a word. **Comparison instructions** are either algebraic or alphanumeric, comparing two words at a time. **Jump instructions** transfer control from the next sequential instruction of the program to any specified area within primary storage. **Sequence-modifying instructions** enable repetitions, conditional skips, programmed interrupts, and the use of programmed "electronic switches". The **I/O instructions** initiate I/O functions for transferring data and control signals.

Some instructions within the above classifications may be arbitrary. An instruction such as Replace Y + Q (34) is both an arithmetic and a transfer instruction, but since the more important function of the instruction is arithmetic, it is listed under arithmetic instructions. Another simplification used in this section is the effective address where, from a technical point of view, absolute address would seem appropriate. The reason for this is that a worker program operating under executive program control has no control over the relative index, R_i, which determines where in memory the program or partially executed program will be stored. The relative index is used to assure most efficient use of primary storage and does not disturb the sequence of instructions and data in the worker program.

Application of the j designator (see Table 3–1) is described in detail, with an instruction where required. The general application of the k designator is shown in Table 3–2 and requires a knowledge of the class of instruction. The class of an instruction is shown in the repertoire of instructions, Appendix B, where necessary for k interpretation.

Most instructions are Read class instructions. In descriptions of the instructions in this section, the k designator determines the source of an operand, Y, as per Read class instructions (Figure 4–1) unless otherwise specified. A Read class instruction usually reads an operand from primary storage and retains the result in a register. A Store class instruction usually obtains the operand from a register and writes ("stores") the result in primary storage. The interpretation of the k designator for Store class instructions is presented with each Store class instruction in this section. A Replace class instruction is a combination of both Read and Store classes; an operand is read from primary storage, and unless the Replace class instruction is immediately preceded by a Repeat (70) instruction, is then replaced in primary storage by the result of the instruction. The operand, Y, is read from primary storage in accordance with the k designator for Read class instructions (Table 4–1). The result is then stored in primary storage in accordance with the rules outlined in Table 4–2. If the Replace class instruction is operating in the Repeat mode, the storage is governed by the rules outlined in 4.8. For Replace instructions, k designators of 0, 4, and 7 are not used.

k DESIGNATOR	OPERAND, Y
0*	Y_U is all 0's. Y_L is effective operand, $y+(B_b)$, or $\bar{y}$.
1	Y_U is all 0's. Y_L is lower half of contents at relative address, $(\bar{y})_L$
2	Y_U is all 0's. Y_L is upper half of contents at relative address, $(\bar{y})_U$
3	Y is contents at relative address, $(\bar{y})$.
4	Y_U is sign-filled-$\bar{y}_{14}$. Y_L is $\bar{y}$.
5	Y_U is sign-filled-$(\bar{y})_{14}$. Y_L is $(\bar{y})_L$.
6	Y_U is sign-filled-$(\bar{y})_{29}$. Y_L is $(\bar{y})_U$.
7	Y is contents of accumulator, (A).

*For k = 0 and (B_b) = 17 bits, Y_L = 17 bits and Y_U = 13 bits

Table 4–1. Operand Designation for Read Class Instructions

k DESIGNATOR	STORAGE OF Y
1 or 5	$(A)_L \longrightarrow (\bar{y})_L$ with $(\bar{y})_U$ unchanged
2 or 6	$(A)_L \longrightarrow (\bar{y})_U$ with $(\bar{y})_L$ unchanged
3	$(A) \longrightarrow (\bar{y})$

Note: 1. k designators of 0, 4, and 7 not used.

2. Storage of Y is modified if this instruction is preceded by a Repeat (70) instruction.

Table 4–2. Transfer of Y to Primary Storage for Replace Class Instructions

4.2. SHIFT INSTRUCTIONS

Shift instructions move the contents of a selected register either to the right or left by as many positions as required. If the instruction calls for a right shift, all bits shifted out of the register are lost and all vacated positions are sign-filled. The term "logical right shift" is reserved for right shifts with zero-fill. A left shift is a circular shift. All bits shifted out of the register at the left are returned, in turn, at the right to fill the vacated positions. Except for the Scale Factor Shift (7730) instruction, the number of shifts (the "shift count") is the six-bit binary number which is the lowest order six bits of the operand, Y. A shift count greater than 59_{10} may not be used. The other bits of the operand are not used in the instruction.

RIGHT SHIFT Q (01)

Shift contents of Q register to the right the number of positions specified by the shift count. If the shift count is greater than 28_{10} and less than 60_{10}, the Q register will be filled with the original sign bit (bit 29).

RIGHT SHIFT A (02)

Shift contents of the accumulator to the right the number of positions specified by the shift count. If the shift count is greater than 28_{10} and less than 60_{10}, the accumulator will be filled with the original sign bit.

RIGHT SHIFT AQ (03)

Shift contents of the AQ register to the right the number of positions specified by the shift count. If the shift count is 59_{10}, the AQ register will be filled with the original sign bit (bit 59); if less than 59_{10} and greater than 28_{10} , the accumulator will be filled with the original sign bit.

LEFT SHIFT Q (05)

Shift contents of the Q register to the left the number of positions specified by the shift count. If the shift count is 30_{10}, the final state of the Q register will be the same as its initial state.

LEFT SHIFT A (06)

Shift contents of the accumulator to the left the number of positions specified by the shift count. If the shift count is 30_{10}, the final state of the accumulator will be the same as its initial state.

LEFT SHIFT AQ (07)

Shift contents of the AQ register to the left the number of positions specified by the shift count. If the shift count is 30_{10}, the contents of accumulator and Q register will be interchanged.

LOGICAL RIGHT SHIFT Q (7751)

Shift contents of the Q register to the right by the number of positions specified in the shift count with zero-fill.

LOGICAL RIGHT SHIFT A (7755)

Shift contents of the A register to the right by the number of positions specified in the shift count with zero-fill.

LOGICAL RIGHT SHIFT AQ (7756)

Shift contents of the AQ register to the right by the number of positions specified in the shift count with zero-fill.

SCALE FACTOR SHIFT (7730)

Shift contents of the accumulator to the left until the two highest order bits are different, and record the number of shifts required in the Q register. When all bits in the accumulator are equal, a shift count of 28_{10} is recorded and the accumulator remains unchanged.

4.3. TRANSFER INSTRUCTIONS

Transfer instructions are used to move data within the CPU. All transfers are non-destructive in that the original source of data remains unchanged except in Replace class instructions. Transfers may consist of 60, 30, 15 bits, or in character packing and unpacking, 6 bits, as determined by the k designator or the instruction itself. In addition, transfers involving index registers may consist of 17 bits if the 17-bit index register mode is activated by the IFR. Normally, for worker programs, the 15-bit index register mode is activated. Use of the j designator is shown in Table 3–1. The operand, Y, is described in Table 4–1, except where otherwise specified.

ENTER Q (10)

Transfer the operand, Y, to the Q register.

ENTER A (11)

Transfer Y to the accumulator. If the k designator is 7, the accumulator remains unchanged.

ENTER B_j (12)

Transfer Y to the active (executive or worker) index register (1–7) specified by the j designator (Table 3–1). This instruction may not be used immediately following an Enter IFR (7761) or Enter RIR (7766) instruction. Since the j designator specifies an index register, it cannot be used to program a skip. The combination of j and k designators conditions operation as shown in the following tabulation (Table 4–3).

j DESIGNATOR	k DESIGNATOR	RESULT
j = 4, 5, 6, or 7	0	17-bit $\bar{y} \longrightarrow B_j$
	1	15-bit $(\bar{y})_L \longrightarrow B_j$ with zero-fill at high end of B_j
	2	15-bit $(\bar{y})_U \longrightarrow B_j$ with zero-fill at high end of B_j
	3	17-bit $(\bar{y})_{0-16} \longrightarrow B_j$
	4	15-bit $\bar{y} \longrightarrow B_j$ with sign-extension
	5	15-bit $(\bar{y})_L \longrightarrow B_j$ with sign-extension
	6	15-bit $(\bar{y})_U \longrightarrow B_j$ with sign-extension
	7	17-bit $(A)_{0-16} \longrightarrow B_j$
j = 1, 2, or 3	0	15-bit $\bar{y} \longrightarrow B_j$
	1	15-bit $(\bar{y})_L \longrightarrow B_j$
	2	15-bit $(\bar{y})_U \longrightarrow B_j$
	3	15-bit $(\bar{y})_{0-14} \longrightarrow B_j$
	4	15-bit $\bar{y} \longrightarrow B_j$
	5	15-bit $(\bar{y})_L \longrightarrow B_j$
	6	15-bit $(\bar{y})_U \longrightarrow B_j$
	7	15-bit $(A)_{0-14} \longrightarrow B_j$
j = 0	NO OPERATION	

Table 4–3. Enter Bj Instruction

STORE Q (14)

Store contents of Q register (or its ones complement) at Y. The k designator determines Y and the portion of the Q register contents to be stored, as follows:

- k is 0: $CP(Q) \longrightarrow Q$
- k is 1: $(Q)_L \longrightarrow (\bar{y})_L$, $(\bar{y})_U$ remains undisturbed
- k is 2: $(Q)_L \longrightarrow (\bar{y})_U$, $(\bar{y})_L$ remains undisturbed
- k is 3: $(Q) \longrightarrow (\bar{y})$
- k is 4: $(Q) \longrightarrow (A)$
- k is 5: $CP(Q)_L \longrightarrow (\bar{y})_L$, $(\bar{y})_U$ remains undisturbed
- k is 6: $CP(Q)_L \longrightarrow (\bar{y})_U$, $(\bar{y})_L$ remains undisturbed
- k is 7: $CP(Q) \longrightarrow (\bar{y})$

STORE A (15)

Store the contents of the accumulator (or its ones complement) at Y. The k designator determines Y and the portion of the accumulator contents to be stored as follows:

- k is 0: $(A) \longrightarrow (Q)$
- k is 1: $(A)_L \longrightarrow (\bar{y})_L$, $(\bar{y})_U$ remains undisturbed
- k is 2: $(A)_L \longrightarrow (\bar{y})_U$, $(\bar{y})_L$ remains undisturbed
- k is 3: $(A) \longrightarrow (\bar{y})$
- k is 4: $CP(A) \longrightarrow (A)$
- k is 5: $CP(A)_L \longrightarrow (\bar{y})_L$, $(\bar{y})_U$ remains undisturbed
- k is 6: $CP(A)_L \longrightarrow (\bar{y})_U$, $(\bar{y})_L$ remains undisturbed
- k is 7: $CP(A) \longrightarrow (\bar{y})$

STORE B_j (16)

Store the contents of an index register (1–7) at Y. The j designator indicates the index register chosen (Table 3–1). If the j designator is 0, and k is 3, all 0's will be stored at Y. If j is 0, and k is 7, all ones will be stored at Y. Before referring to the use of the k designator which determines the destination, a full-word (30-bit) transfer is necessary to store the contents of an active 17-bit Index register (refer to Index registers, Section 2). The b designator functions as usual and may result in a 17-bit effective address, $\bar{y}$. The k designator functions as follows:

- If k is 0: $(B_j) \longrightarrow (Q)_L$ – lowest 15 or 17-bit positions. Remainder of (Q) is filled with 0's.
- If k is 1: $(B_j) \longrightarrow (\bar{y})_L$ – lowest 17-bit positions. $(\bar{y})_U$ remains undisturbed.
- If k is 2: $(B_j) \longrightarrow (\bar{y})_U$ – highest 15-bit positions. $(\bar{y})_L$ remains undisturbed.
- If k is 3: $(B_j) \longrightarrow (\bar{y})_L$ – highest 15 or 17-bit positions. Remainder of $(\bar{y})$ is filled with 0's.
- If k is 4: $(B_j) \longrightarrow (A)_L$ – lowest 15 or 17 bits. Remainder of (A) is filled with 0's.
- If k is 5: $CP(B_j) \longrightarrow (\bar{y})_L$ – lowest 15 bits. Remaining 15 bits of $(\bar{y})$ are undisturbed.
- If k is 6: $CP(B_j) \longrightarrow (\bar{y})_U$ – highest 15 bits. Remaining 15 bits of $(\bar{y})$ are undisturbed.
- If k is 7: $CP(B_j) \longrightarrow (\bar{y})_L$ – lowest 15 or 17 bits. Remainder of $(\bar{y})$ is filled with bits similar to the highest order bit.

ENTER AQ (7721)

Transfer the 60-bit operand $(\bar{y})$ and $(\bar{y} + 1)$ to the AQ register. Transfer $(\bar{y})$ to A and $(\bar{y} + 1)$ to Q.

STORE AQ (7725)

Store the 60-bit contents of the AQ register at $(\bar{y})$ and $(\bar{y} + 1)$. Store (A) as $(\bar{y})$ and (Q) as $(\bar{y} + 1)$.

CHARACTER PACK LOWER (7731)

Transfer, into the accumulator, the six bits (00–05) in each of five successive primary storage locations so as to make up a full-word in the accumulator. $(\bar{y})_{00\text{-}05} \rightarrow (A)_{24\text{-}29}$, $(\bar{y} + 1)_{00\text{-}05} \rightarrow (A)_{18\text{-}23}$, $(\bar{y} + 2)_{00\text{-}05} \rightarrow (A)_{12\text{-}17}$, $(\bar{y} + 3)_{00\text{-}05} \rightarrow (A)_{06\text{-}11}$, $(\bar{y} + 4)_{00\text{-}05} \rightarrow (A)_{00\text{-}05}$. This instruction is most useful when data is received from peripheral equipment as successive five or six-bit characters in successive buffer (primary storage) locations. The successive characters are then read into the accumulator to form a 30-bit word.

CHARACTER PACK UPPER (7732)

Transfer, into the accumulator, the six bits (15–20) in each of five successive primary storage locations so as to make up a full-word in the accumulator. $(\bar{y})_{15\text{-}20} \rightarrow (A)_{24\text{-}29}$, $(\bar{y} + 1)_{15\text{-}20} \rightarrow (A)_{18\text{-}23}$, $(\bar{y} + 2)_{15\text{-}20} \rightarrow (A)_{12\text{-}17}$, $(\bar{y} + 3)_{15\text{-}20} \rightarrow (A)_{06\text{-}11}$, $(\bar{y} + 4)_{15\text{-}20} \rightarrow (A)_{00\text{-}05}$. This instruction is similar to the Character Pack Lower (7731) instruction except that the incoming five or six-bit characters must have been stored in the upper half of successive buffer locations.

CHARACTER UNPACK LOWER (7735)

Transfer, into bit positions 00–05 at each of five successive primary storage locations, the successive sets of six bits in the accumulator starting from the high end of the accumulator. $(A)_{24\text{-}29} \rightarrow (\bar{y})_{00\text{-}05}$, $(A)_{18\text{-}23} \rightarrow (\bar{y} + 1)_{00\text{-}05}$, $(A)_{12\text{-}17} \rightarrow (\bar{y} + 2)_{00\text{-}05}$, $(A)_{06\text{-}11} \rightarrow (\bar{y} + 3)_{00\text{-}05}$, $(A)_{00\text{-}05} \rightarrow (\bar{y} + 4)_{00\text{-}05}$. At each of the five successive primary storage locations, bits 06–14 are cleared to 0's. This instruction is used principally to break up a 30-bit word into successive sets of 5 six-bit characters and load these characters into the low end of five successive output buffer locations.

CHARACTER UNPACK UPPER (7736)

Transfer, into bit positions 15–20 at each of five successive primary storage locations, the successive sets of six bits in the accumulator starting from the high end of the accumulator. $(A)_{24\text{-}29} \rightarrow (\bar{y})_{15\text{-}20}$,$(A)_{18\text{-}23} \rightarrow (\bar{y} + 1)_{15\text{-}20}$, $(A)_{12\text{-}17} \rightarrow (\bar{y} + 2)_{15\text{-}20} \rightarrow$ $(A)_{06\text{-}11} \rightarrow (\bar{y} + 3)_{15\text{-}20}$, $(A)_{00\text{-}05} \rightarrow (\bar{y} + 4)_{15\text{-}20}$. At each of the successive primary storage locations, bits 21–29 are cleared to 0's. This instruction is similar to the Character Unpack Lower instruction, except that the data is located in the upper half of successive output buffer locations.

ENTER IFR AND RIR (7761)

Transfer the 30-bit word at the relative address to the IFR and bits 06–16 of the next address (after the effective address) to the RIR. The contents of the RIR become effective for execution of the next instruction; the contents of the IFR are used for the execution of the next instruction plus 1.

LOAD PLR (7762)

Transfer the 30-bit word at the relative address to the PLR. The format for this transferred word is shown in Figure 3–2. This instruction is intended for executive routines. The program storage limits are deactivated by an interrupt and activated by the Enter IFR instruction to set the limits for the worker program operating under control of the executive routine.

STORE IFR (7765)

Transfer the contents of the IFR to the relative address. This instruction is used principally in executive routines to store the contents of the IFR in primary storage before a new word is transferred to the IFR.

ENTER RIR (7766)

Transfer bits 06–16 of the contents at the relative address to the RIR. This instruction is used principally in executive routines. The P register will not be biased by the new RIR value until a Jump instruction to the worker program is executed. Relative addressing is not used in an interrupt routine. The interrupt routine must enter the RIR before the RIR can become effective in the following worker program.

ENTER B-WORKER (7771)

Transfer the lowest order bits at each of seven successive primary storage locations (starting from the relative address $\bar{y}$) to the seven worker index registers, in turn. $(\bar{y})_{00\text{-}14} \longrightarrow B1$, $(\bar{y}+1)_{00\text{-}14} \longrightarrow B2$, $(\bar{y}+2)_{00\text{-}14} \longrightarrow B3$, $(\bar{y}+3)_{00\text{-}16} \longrightarrow B4$, $(\bar{y}+4)_{00\text{-}16} \longrightarrow B5$, $(\bar{y}+5)_{00\text{-}16} \longrightarrow B6$, $(\bar{y}+6)_{00\text{-}16} \longrightarrow B7$.

STORE B-WORKER (7775)

Transfer the contents of the worker index registers, in turn, to the low order bit positions of seven successive primary storage locations, starting from $\bar{y}$. $B1 \longrightarrow (\bar{y})$, $B2 \longrightarrow (\bar{y}+1)$, etc. These transfers are 30-bit transfers with unused portions (either bits 15–29 for 15-bit registers or bits 17–29 for 17-bit registers) filled with 0's.

4.4. ARITHMETIC INSTRUCTIONS

All arithmetic operations in the CPU use the 60-bit parallel adder shown in the block diagram, Figure 2–2. Subtraction, multiplication, and division are performed by this basic adder. As a result of internal processing, the results will conform to the common algebraic rules for each arithmetic operation.

4.4.1. Integer Addition and Subtraction

The actual operations and examples of the addition and subtraction processes are described in Appendix B. The following characteristics of these operations are of interest to the programmer:

- The programmer must guard against overflow conditions. For single precision operations, the absolute value of the operands and results may not exceed 2^{29} - 1; for double precision operations, 2^{59} - 1. In the event of an overflow, the result will be incorrect. No indication is present for an overflow condition.

- A sum of negative zero cannot be generated unless both augend, (A), and addend, Y, are both negative zeros. A difference of negative zero cannot be generated unless the minuend, (A), is a negative zero and subtrahend, Y, is a positive zero. In all other cases involving negative zero, the same result will be obtained as if a positive zero were used.

4.4.2. Integer Multiplication and Division

Integer multiplication and division are performed as a series of addition and/or subtraction processes. The results of interest to the programmer can be summarized as follows:

- Multiplication results in the normal algebraic sign. Where the signs are like, the result will be positive; where the signs are different, the result will be negative. In division, except for division by zero, the quotient (which will be in the Q register) will have its sign algebraically determined; the remainder (in the accumulator) will have the same sign as the quotient.

- In multiplication, the entire product *will* be in the Q portion of the AQ register if bit position (28-n) of the multiplier is a sign bit, where n is the most significant bit position of the multiplicand. The most significant bit of a positive number is a 1; of a negative number, 0. The entire product *may* be in the Q portion if bit position (28-n) of the multiplier is the most significant bit. The entire product *will not* be contained in the Q portion if bit position (29-n) of the multiplier does not contain a sign bit.

- In division, the quotient is entered into the Q portion of the AQ register. An overflow *will not* occur if the dividend has no significant bits past bit position n + 28, where n is the most significant bit position of the divisor; overflow *will* occur if the dividend has a significant bit past bit position n + 29. An overflow *may* occur if the most significant bit of the dividend is in bit position n + 29. If *overflow* occurs, the quotient will appear as +0 (if divisor and dividend had like signs) or as -0 (if the signs were different). However, for j interpretation, Q will appear as -0 when there is an overflow.

A negative *zero* in the A or Q portion of the AQ register may have an adverse effect on further calculations. Negative zero will result from the following conditions:

- The dividend is an integral multiple of the divisor (within the limits of resolution), the signs of both are different, and both values are not 0 (+ or -). The quotient will be correct but A will be -0. For j-sensing, Q will appear as the absolute value of the quotient and A will be +0.

- The absolute value of the divisor is greater than the absolute value of the dividend, signs are different, and both are nonzero. In this case, Q will be -0 and A will be the complement of the absolute value of the dividend. For j-sensing, Q is +0 and A is the absolute value of the dividend.

Division by +0 or -0 has the following results:

- If a positive number is divided by +0, Q will contain -0 and the remainder in A will be the initial Q value. For j-sensing, the final Q and A are used.
- If a negative number is divided by +0, Q will be +0 and the remainder in A will be the initial Q value. For j-sensing, Q will be -0 and A will be the complement of initial Q.
- If a positive number is divided by -0, Q will be +0 and the remainder in A will be the complement of initial Q. For j-sensing, Q will be -0, and A will be initial Q.
- If a negative number is divided by -0, Q will be -0 and the remainder in A will be the complement of initial Q. For j-sensing, the final Q and A are used.

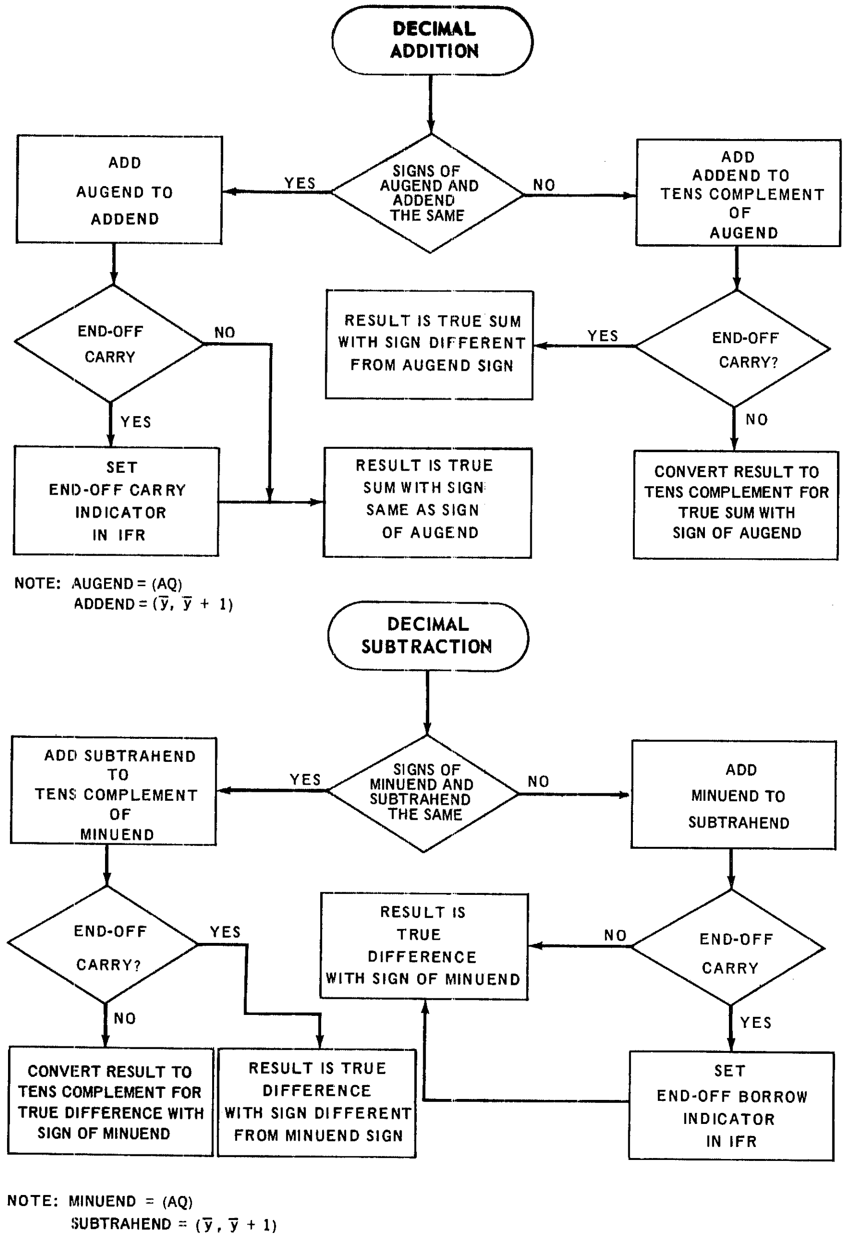

DECIMAL ADDITION
SIGNS OF AUGEND AND ADDEND THE SAME
YES
NO
ADD AUGEND TO ADDEND
ADD ADDEND TO TENS COMPLEMENT OF AUGEND
END-OFF CARRY
NO
YES
SET END-OFF CARRY INDICATOR IN IFR
RESULT IS TRUE SUM WITH SIGN SAME AS SIGN OF AUGEND
RESULT IS TRUE SUM WITH SIGN DIFFERENT FROM AUGEND SIGN
END-OFF CARRY?
YES
NO
CONVERT RESULT TO TENS COMPLEMENT FOR TRUE SUM WITH SIGN OF AUGEND
NOTE: AUGEND = (AQ)
ADDEND = (ȳ, ȳ + 1)
DECIMAL SUBTRACTION
SIGNS OF MINUEND AND SUBTRAHEND THE SAME
YES
NO
ADD SUBTRAHEND TO TENS COMPLEMENT OF MINUEND
ADD MINUEND TO SUBTRAHEND
END-OFF CARRY?
YES
NO
CONVERT RESULT TO TENS COMPLEMENT FOR TRUE DIFFERENCE WITH SIGN OF MINUEND
RESULT IS TRUE DIFFERENCE WITH SIGN DIFFERENT FROM MINUEND SIGN
RESULT IS TRUE DIFFERENCE WITH SIGN OF MINUEND
END-OFF CARRY
NO
YES
SET END-OFF BORROW INDICATOR IN IFR
NOTE: MINUEND = (AQ)
SUBTRAHEND = (ȳ, ȳ + 1)

Figure 4–1. Basic Decimal Arithmetic

4.4.3. Exponential (Floating-Point) Arithmetic

In exponential arithmetic, the following items are of interest to the programmer:

- The floating-point overflow interrupt will be generated in any part of an exponential operation where the exponent part of an operand exceeds 1023_{10} or when division by a floating-point ±0 is attempted.
- The floating-point underflow interrupt will be generated if a floating-point number has an exponent part less than -1023_{10}.

4.4.4. Decimal Arithmetic

Basic decimal operations (addition and subtraction) are performed as shown in the flowcharts of Figure 4–1. The following terms are used in the description of decimal arithmetic: the *end-off carry* is a carry generated by the addition at the highest order BCD digit positions, the *nines complement* of a BCD number is obtained by subtracting each BCD digit from 9, the *tens complement* of a BCD number is obtained by adding an arithmetic 1 to the nines complement (performing all carries).

The following examples illustrate the use of the flowchart:

(1) Addend = +1234567890
Augend = - 9123456789

Since the signs are different, the tens complement of the augend is used.

```
1234567890
0876543211
──────────
2111111101
```

Since there is no end-off carry, the tens complement of this result is used.

Sum = -7888888899, the sign being taken from the original augend.

(2) Minuend = -3062918521
Subtrahend = +9732802954

```
-2795721475   True result since signs are different

     1        End-off carry, retained in the IFR for use in
              multiprecision decimal operations
```

Multiprecision decimal addition and subtraction differ from basic decimal arithmetic in that: (1) whenever it is necessary to generate a complement, the nines complement is used instead of the tens complement; and (2) the end-off carry from the previous add or subtract is introduced as a carry into the least significant decimal digit position.

ADD A (20). (Integer, single precision)

Add the 30-bit operand, Y, to (A) and retain this sum as the final (A). The Q register remains unchanged. Wherever the j designator refers to (A), the final (A) is used.

SUBTRACT A (21). (Integer, single precision)

Subtract the 30-bit operand, Y, from (A) and retain this difference as the final (A). Where the k designator is 7, the final (A) will be +0. Where the j designator refers to (A), the final (A) is used.

MULTIPLY (22). (Integer, single precision)

Multiply (Q) by the 30-bit operand, Y, and retain this product as the final (AQ). The designator k of 7 may be used only if the initial (Q) is a positive number, otherwise, the product in (AQ) will be the ones complement of the true product. The least significant portion of the product is always stored in the Q portion of the AQ register. If the product (including sign) exceeds the 30-bit capacity of the Q register, an overflow condition exists. The j designator defines a skip condition (see Table 3–1). Note that if a skip occurs on j = 4, it *may* be necessary to obtain the sign of the product from (A).

DIVIDE (23). (Integer, single precision)

Divide (AQ) by 30-bit operand, Y, and retain the quotient as (Q) and the remainder as (A). A k designator of 7 may not be used. The j designator is interpreted as shown in Table 3–1, and may use the presence of a remainder to condition program sequence.

REPLACE A + Y (24). (Integer, single precision)

Add 30-bit operand, Y, to (A). Retain sum as (A) and replace operand as indicated in Figure 4–2. For the addition, the operand, Y, is determined by the k designator, as shown in Table 4–1. Wherever the j designator refers to (A), the final (A) is used.

REPLACE A - Y (25). (Integer, single precision)

Subtract the 30-bit operand, Y, from (A). Retain difference as (A) and store it as indicated in Table 4–2. For the subtraction, the 30-bit operand, Y, is determined by the k designator, as shown in Table 4–1. Wherever the j designator refers to (A), the final (A) is used.

ADD Q (26). (Integer, single precision)

Add 30-bit operand, Y, to (Q). Retain sum as (Q). Wherever the j designator refers to (Q), the final (Q) is used. Refer to Table 3–1 for j interpretation.

SUBTRACT Q (27). (Integer, single precision)

Subtract 30-bit operand, Y, from (Q). Retain difference as (Q). Wherever the j designator refers to (Q), the final (Q) is used. Refer to Table 3–1 for j interpretation.

ENTER Y + Q (30). (Integer, single precision)

Add the 30-bit operand, Y, to (Q) and retain the sum as (A). The contents of the Q register remain unchanged. Wherever the j designator refers to (A), the final (A) is used.

ENTER Y – Q (31). (Integer, single precision)

Subtract (Q) from the 30-bit operand, Y, and retain the difference as (A). The contents of the Q register remain unchanged. Wherever the j designator refers to (A), the final (A) is used.

STORE A + Q (32). (Integer, single precision)

Add the 30-bit (A) to the 30-bit (Q), retaining the result as the final (A) and (except when the k designator is 0 or 4) storing it in primary storage. Except for the k designator of 0, the Q register remains unchanged. The k designator governs storage of the sum as follows:

k = 0:	Retain the sum also as the final (Q)
k = 1:	$(A)_L \longrightarrow (\bar{y})_L$ with $(\bar{y})_U$ remaining unchanged
k = 2:	$(A)_L \longrightarrow (\bar{y})_U$ with $(\bar{y})_L$ remaining unchanged
k = 3:	$(A) \longrightarrow (\bar{y})$
k = 4:	The sum is retained only as the final (A)
k = 5:	$CP(A)_L \longrightarrow (\bar{y})_L$ with $(\bar{y})_U$ remaining unchanged
k = 6:	$CP(A)_L \longrightarrow (\bar{y})_U$ with $(\bar{y})_L$ remaining unchanged
k = 7:	$CP(A) \longrightarrow (\bar{y})$

Wherever the j and k designators refer to (Q) and (A), the final (Q) and (A) are used.

STORE A – Q (33). (Integer, single precision)

Subtract the 30-bit (Q) from the 30-bit (A), retaining the difference as the final (A) and (except when the k designator is a 0 or 4) storing it in primary storage. The j and k designators are used in exactly the same manner as used in the Store A + Q (32) instruction previously described, except that the difference is used instead of the sum.

REPLACE Y + Q (34). (Integer, single precision)

Add the 30-bit operand, Y, to (Q), retaining this sum as (A) and store it in primary storage. Table 4–1 describes the 30-bit operand, Y, and Table 4–2 describes the replacement operation. The Q register remains unchanged. The k designators of 0, 4, or 7 may not be used.

REPLACE Y – Q (35). (Integer, single precision)

Subtract the 30-bit (Q) from the 30-bit operand Y, retaining this difference as (A) and store it in primary storage. Table 4–1 describes the 30-bit operand, Y, and Table 4–2 describes the storage operation. The Q register remains unchanged. The k designators of 0, 4, or 7 may not be used.

REPLACE Y + 1 (36). (Integer, single precision)

Add an arithmetic 1 to the 30-bit operand, Y, retaining this sum as (A) and store it in primary storage. Table 4–1 describes the 30-bit operand, Y, and Table 4–2 describes the storage operation. The k designators of 0, 4, or 7 may not be used.

REPLACE Y – 1 (37). (Integer, single precision)

Subtract an arithmetic 1 from the 30-bit operand, Y, retaining this sum as (A) and store it in primary storage. Table 4–1 describes the 30-bit operand, Y, and Table 4–2 describes the storage operation. The k designators of 0, 4, or 7 may not be used.

ADD AQ (7722). (Integer, double precision)

Add the signed 60-bit ($\bar{y}$, $\bar{y}$ + 1) to the signed 60-bit (AQ) and retain the 60-bit sum as the final (AQ).

COMPLEMENT AQ (7724). (Integer, double precision)

Convert the 60-bit (AQ) to its ones complement (changing binary 1's to 0's and 0's to 1's) and retain the complement as the final (AQ).

SUBTRACT AQ (7726). (Integer, double precision)

Subtract the signed 60-bit ($\bar{y}$, $\bar{y}$ + 1) from the signed 60-bit (AQ) and retain the 60-bit difference as the final (AQ). The contents of $\bar{y}$ + 1 and the Q register are the least significant portions; the contents of $\bar{y}$ and the accumulator are the most significant portions.

FLOATING-POINT ADD (7701). (Exponential)

Add the signed floating-point number in ($\bar{y}$, $\bar{y}$ + 1) to the signed floating-point number (AQ), and retain the normalized floating-point sum as the final (AQ). Floating-point operands need not be normalized for this instruction. Operation is checked for floating-point overflow and underflow.

FLOATING-POINT SUBTRACT (7702). (Exponential)

Subtract the signed floating-point number in ($\bar{y}$, $\bar{y}$ + 1) from the signed floating-point (AQ), and retain the normalized difference as the final (AQ). Floating-point operands need not be normalized for this instruction. Operation is checked for floating-point overflow and underflow.

FLOATING-POINT MULTIPLY (7703). (Exponential)

Multiply the signed floating-point (AQ) by the signed floating-point number in ($\bar{y}$, $\bar{y}$ + 1) and retain this product as the final (AQ). This product, (AQ), will be normalized and correct only if the original operands were normalized. Operation is checked for floating-point overflow and underflow.

FLOATING-POINT DIVIDE (7705). (Exponential)

Divide the signed, normalized floating-point (AQ) by the signed, normalized floating-point number in ($\bar{y}$, $\bar{y}$ + 1) and store the signed, normalized floating-point quotient as the final (AQ). Any remainder will be discarded. Operation is checked for floating-point overflow and underflow.

FLOATING-POINT PACK (7706). (Exponential)

Pack the biased exponent in $(\bar{y})_{00-10}$ with the unnormalized fixed-point part, (AQ), to make up a floating-point operand. The 11 bits from $(\bar{y})$ are treated as a positive number. The final floating-point operand thus formed as (AQ) is normalized and signed with its exponent adjusted for normalization.

FLOATING-POINT UNPACK (7707). (Exponential)

Store the absolute magnitude of the exponent in AQ as $(\bar{y})_{00-10}$ and fill $(\bar{y})_{11-14}$ with 0's. Extend the sign of the fixed-point part in (AQ) to fill the 11-bit portion allotted to the exponent.

DECIMAL TEST AQ (7710). (Decimal)

Test the BCD number (AQ) for sign, overflow, and/or comparison of signs as indicated in Table 4–4. Wherever a binary 1 is present in the instruction word, a test is made. If any, or all, of the test conditions are satisfied, the next sequential instruction is skipped. Bits 11–17 of the instruction word are not used.

BIT POSITION	CONDITION
0	Skip if decimal overflow designator is set in IFR.
1	Skip if decimal overflow designator is clear in IFR.
2	Skip if sign is +, sign test only.
3	Skip if number is 0, disregarding sign.
4	Skip if sign is -, sign test only.
5	Skip if sixth decimal digit is unequal to 0.
6	Skip if seventh decimal digit is unequal to 0.
7	Skip if eighth decimal digit is unequal to 0.
8	Skip if ninth decimal digit is unequal to 0.
9	Skip if tenth decimal digit is unequal to 0.
10	Skip if (AQ) $\neq$ 0 (neglect sign).

NOTE: Bit positions 5–9 may be used by the programmer to detect a result that exceeds the normal data field length.

Table 4–4. Decimal Test

DECIMAL ADD (7711). (Fixed-point zoned BCD double precision)

Add the zoned and signed 10-digit (AQ) to the zoned and signed 10-digit $(\bar{y}, \bar{y} + 1)$. The BCD sum, the sign of the sum, and the zone bits originally in (AQ) will be retained as the final 10-digit (AQ). This addition is performed as shown in Figure 4-1.

DECIMAL SUBTRACT (7712). (Fixed-point zoned BCD double precision)

Subtract the zoned and signed 10-digit $(\bar{y}, \bar{y} + 1)$ from the zoned and signed 10-digit (AQ). The BCD difference, the sign of the difference, and the zone bits originally in (AQ) will be retained as the final 10-digit (AQ). This subtraction is performed as shown in Figure 4–1.

DECIMAL COMPLEMENT AQ (7714). **(Fixed-point zoned BCD double precision)**

Convert the 10-digit (AQ) to its decimal complement. Retain the original zone bits and sign. If $\bar{y}$ is odd, the decimal complement is the nines complement (each digit is replaced by its difference from 9). If $\bar{y}$ is even, the decimal complement is the tens complement (the complete nines complement plus 1).

DECIMAL ADD WITH CARRY (7715). **(Decimal, multiprecision)**

Add the zoned and signed 10-digit (AQ) together with any end-off carry from the previous BCD addition to the zoned and signed 10-digit ($\bar{y}$, $\bar{y}$ + 1). The BCD sum, the sign of the sum, and the zone bits originally in (AQ) will be retained as the final 10-digit (AQ). This addition is similar to the addition shown in Table 4–1 except that nines complements are used in place of tens complements.

DECIMAL SUBTRACT WITH BORROW (7716). **(Decimal, multiprecision)**

Subtract the zoned and signed 10-digit ($\bar{y}$, $\bar{y}$ + 1), together with any end-off borrow from a previous BCD subtraction, from the zoned and signed 10-digit (AQ). The BCD difference, its sign, and the zone bits originally in (AQ) will be retained as the final (AQ). This subtraction is similar to the subtraction shown in Table 4–1 except that the overflow is introduced as a carry into the least significant digit position during the addition performed directly after the sign comparison. In addition, where Figure 4–1 uses tens complements, this instruction uses nines complements.

DECIMAL CONVERT LOWER (7733). **(Decimal)***

Convert the five-digit decimal number made up of the lowest BCD-coded digit (bits 0 through 3) in each of five successive memory locations (starting from y) to binary form and store the result at the low end of the AQ register. The maximum binary number that can be stored in the AQ register by this transformation is 2^{34} -1 ($17{,}179{,}869{,}183_{10}$). Therefore, precautions must be taken prior to using this instruction so this value is never exceeded. Assuming the AQ register was initially cleared, 99999_{10} is the maximum number that can be converted and stored for one execution of the instruction. However, two successive instructions could be used since the maximum number thus converted and stored would be 9,999,999,999 and this is less than the maximum permissible number.

DECIMAL CONVERT UPPER (7734). **(Decimal)***

This instruction operates exactly as does the Decimal Convert Lower previously described, except that the five- digit decimal number to be converted and stored is taken from bits 15 through 18 at the five successive memory locations.

* The nonsignificant bits in Y through Y + 4 of the half-words to be converted (bits 6 to 14 for 7733 and bits 21 to 29 for 7734) must be 0 prior to execution of the 7733 or 7734 instruction. The Character Unpack instruction will properly format the data to be corrected.

4.5. LOGICAL INSTRUCTIONS

Logical instructions provide the programmer with the means of operating upon specific bits of a word. These logical operations are the logical product (LP), the selective set, selective clear, selective complement, and the selective substitute. Except for the selective complement, two words are involved in a logical operation. The logical operation is performed upon the bits in the same corresponding bit positions of each of the words to form the resulting word. For all j interpretations which use the contents of a register to determine a skip, the final state of the register is always used.

The logical product is generally used for "masking" (lifting the selected bits of a word and using 0 bits for unselected positions). This is accomplished by placing 1's in the mask to select bits and 0's for the other bits. Wherever there is a 1 in the mask, the corresponding bit of the operand will appear in the logical product. Wherever there is a 0 in the mask, a 0 will appear in the logical product. Thus, the logical product corresponds to the AND function – the logical product is a 1 when the mask *and* the operand are both 1's; otherwise, it is a 0. The following example illustrates the logical product:

```
Mask                    111  000  001  010  011  100  101  110  111  000
Operand                 010  100  110  000  001  011  110  111  101  100
                        ------------------------------------------------
LP                      010  000  000  000  001  000  100  110  101  000
```

The selective set is used to force 1's into selected bits of the accumulator. Where-ever there is a 1 in the operand, a 1 is forced into the accumulator. If the accumulator bit is already a 1, it remains undisturbed. Wherever there is a 0 in the operand, the accumulator bit remains undisturbed. Thus, the selective set corresponds to the inclusive OR function: the result is a 1 if the accumulator bit is a 1 *or* the operands bit is a 1, or both. The following example illustrates operation of the selective set.

```
Operand                 010  100  110  000  001  011  110  111  101  100
Accumulator (initial)   111  000  001  101  011  100  101  110  111  000
                        ------------------------------------------------
Accumulator (final)     111  100  111  101  011  111  111  111  111  100
```

The selective clear forces 0's into selected bits of the accumulator. Wherever there is a 1 in the operand, a 0 will be forced into the accumulator. If the accumulator bit is a 0, it remains undisturbed. Wherever there is a 0 in the operand, the accumulator bit remains undisturbed. The selective clear can also be regarded as a modified masking operation. Wherever there is a 0 in the operand (mask), the corresponding bit of the accumulator is lifted and placed in the final result. The following example illustrates operation of the selective clear:

```
Operand                 010  100  110  000  001  011  110  111  101  100
Accumulator (initial)   111  000  001  101  011  100  101  110  111  000
                        ------------------------------------------------
Accumulator (final)     101  000  001  101  010  100  001  000  010  000
```

The selective complement operates upon selected bits of the accumulator. Whereever there is a 1 in the operand, the accumulator bit is ones-complemented. The following example illustrates operation of the selective complement.

Operand	010 100 110 000 001 011 110 111 101 100
Accumulator (initial)	111 000 001 101 011 100 101 110 111 000
Accumulator (final)	101 100 111 101 010 111 011 001 010 100

The selective substitute replaces selected bits in the accumulator with the corresponding bit of the operand. Selection is performed by the Q register – for each 1 bit in the Q register, the substitution is made. The following example illustrates operation of the selective substitute:

Q register	101 010 000 111 100 011 110 001 110 011
Operand	010 100 110 000 001 011 110 111 101 100
Accumulator (initial)	111 000 001 101 011 100 101 110 111 000
Accumulator (final)	010 000 001 000 011 111 111 111 101 000

ENTER LOGICAL PRODUCT (40)

In the accumulator, form and retain the logical product of the operand, Y, and the contents of the Q register (Q). If the number of 1's in (A) is even, a j designator of 2 will cause a skip; if odd, a j designator of 3 will cause a skip (Table 3–2).

ADD LOGICAL PRODUCT (41)

Add the logical product of Y and (Q) to the contents of the accumulator and retain this sum in the accumulator. The contents of the Q register are not changed by this instruction.

SUBTRACT LOGICAL PRODUCT (42)

Subtract the logical product of (Q) and Y from the contents of the accumulator and store this difference in the accumulator. The Q register remains unchanged.

REPLACE LOGICAL PRODUCT (44)

Form the logical product of Q and Y in the accumulator and store the logical product in the location from which Y was obtained. The j designator is a skip indicator which may, when j = 2 or 3, indicate a skip depending upon the number (odd or even) of binary 1's in the logical product (Table 3–2). The operand Y is derived as shown in Table 4–1 and the logical product is stored as shown in Table 4–2. The Q register remains unchanged by this instruction.

REPLACE A + LOGICAL PRODUCT (45)

Add the logical product of Y and (Q) to (A). Retain this sum as (A) and store it at Y. The operand, Y, is derived as shown in Table 4–1, and the sum is stored in primary storage as shown in Table 4–2.

REPLACE A - LOGICAL PRODUCT (46)

Subtract the logical product of Y and (Q) from (A). Retain this sum as (A) and store it in primary storage as indicated by the k designator. The operand,Y, is derived as shown in Table 4–1, and the difference is stored in primary storage as shown in Table 4–2.

STORE LOGICAL PRODUCT (47)

Store the logical product of (A) and (Q) at Y, as follows:

k = 0:	$LP \rightarrow (Q)$ with (A) undisturbed
k = 1:	$(LP)_L \rightarrow (\bar{y})_L$ with (Q), (A), and $(\bar{y})_U$ undisturbed
k = 2:	$(LP)_L \rightarrow (\bar{y})_U$ with (Q), (A), and $(\bar{y})_L$ undisturbed
k = 3:	$LP \rightarrow (\bar{y})$ with (Q) and (A) undisturbed
k = 4:	$LP \rightarrow A$, with (Q) undisturbed. The b and y designators are not used.
k = 5:	CP $(LP)_L \rightarrow (\bar{y})_L$ with (Q), (A), and $(\bar{y})_U$ undisturbed
k = 6:	CP $(LP)_L \rightarrow (\bar{y})_U$ with (Q), (A), and $(\bar{y})_L$ undisturbed
k = 7:	Not used

SELECTIVE SET (50)

Force binary 1's into (A) wherever there is a binary 1 in the operand Y, leaving the other bits of (A) undisturbed. A k designator of 7 may not be used.

SELECTIVE COMPLEMENT (51)

For every binary 1 in the operand Y, complement the corresponding bit in (A). For k = 7, (A) will become + 0.

SELECTIVE CLEAR (52)

For every binary 1 in the operand Y, force a binary 0 into the corresponding bit position of (A). A k designator of 7 may not be used.

SELECTIVE SUBSTITUTE (53)

Wherever (Q) has a binary 1, replace the corresponding bit in (A) with the corresponding bit of the operand, Y. A k designator of 7 may not be used with this instruction.

REPLACE SELECTIVE SET (54)

Wherever the operand, Y, has a binary 1, set the corresponding bit in (A) to a binary 1, leaving the other bits of (A) unchanged. Retain the result as (A) and store it in primary storage. The operand, Y, is derived as shown in Table 4–1, and the result is stored in primary storage as shown in Table 4–2.

REPLACE SELECTIVE COMPLEMENT (55)

Wherever the operand, Y, has a binary 1, complement the corresponding bit in (A), leaving the other bits of (A) unchanged. Retain the result as (A) and store it in primary storage. The operand, Y, is derived as shown in Table 4–1, and the result is stored as shown in Table 4–2.

REPLACE SELECTIVE CLEAR (56)

Wherever the operand, Y, has a binary 1, clear the corresponding bit of (A) to binary 0, leaving the other bits of (A) unchanged. Retain the result as (A) and store it in primary storage. The operand, Y, is derived as described in Table 4–1, and the result is stored as shown in Table 4–2.

REPLACE SELECTIVE SUBSTITUTE (57)

Wherever (Q) has a binary 1, transfer the corresponding bit of Y to the corresponding position in (A), leaving the other bits of (A) unchanged. Retain the result as (A) and store it in primary storage. The operand, Y, is derived as shown in Table 4–1, and the result is stored as shown in Table 4–2.

4.6. COMPARISON INSTRUCTIONS

Comparisons may be performed, within a narrow range, for most read class instructions by means of the j designator to determine a skip condition (see Table 3–1). The following comparison instructions greatly extend the range of comparisons. Comparisons may be either alphanumeric or arithmetic, the difference being, that alphanumeric instructions perform absolute comparisons, treating the highest order bits as part of the absolute values being compared. Arithmetic comparisons are algebraic comparisons which recognize the sign bit so that a positive number is recognized as being greater than a negative number. Thus, in an alphanumeric comparison, -0 (all 1's) is greater than +0 (all 0's), but in an arithmetic comparison, -0 is less than +0 since a binary 1 is always greater than a binary 0 in alphanumeric comparisons. Alphanumeric comparisons are especially useful in sorting and collating programs where both numbers and characters are involved.

MASKED ALPHANUMERIC EQUAL (7753)

Wherever there is a binary 1 in (Q), compare the corresponding bits of ($\bar{y}$) and (A), using 0's in the unselected bits of (A) and ($\bar{y}$). If the masked accumulator word is equal to the masked primary storage word, skip the next sequential instruction; if unequal, execute the next sequential instruction.

MASKED ALPHANUMERIC LESS (7757)

Using (Q) as the mask, compare the masked (A) with the masked ($\bar{y}$). If the masked (A) is less, skip the next sequential instruction; if not, execute the next sequential instruction.

COMPARE (04)

Compare the signed Y with the signed (A) and/or (Q) and skip the next instruction as determined by the j designator (Table 3–1). This comparison can be used on fixed-point or floating-point binary operands, but may not be used for operands in the zoned BCD mode.

COMPARE MASKED (43)

Subtract the logical product of Y and (Q) from (A) and skip the next sequential instruction if the condition specified by the j designator (Table 3–1) is satisfied. Both (Q) and (A) remain unchanged by this instruction.

COMPARE AQ EQUAL (7723)

Compare the signed (AQ) with ($\bar{y}$), ($\bar{y}$+1) and skip if equal; if unequal, perform the next sequential instruction. The sign and most significant portion of the double-precision words will be in (A) and ($\bar{y}$). The AQ register and the primary storage locations remain unchanged. This instruction is not intended for zoned BCD operands.

COMPARE AQ LESS (7727)

Compare the signed (AQ) with ($\bar{y}$), ($\bar{y}$+1) and skip if (AQ) is less. If (AQ) = ($\bar{y}$), ($\bar{y}$+1), perform the next sequential instruction. The sign and most significant portion of the double-precision words will be in (A) and ($\bar{y}$). The AQ register and the primary storage locations remain unchanged. This instruction is not intended for zoned BCD operands.

DECIMAL COMPARE EQUAL (7713)

Compare the signed 10-digit zoned BCD number of (AQ) with the signed 10-digit zoned BCD number in ($\bar{y}$), ($\bar{y}$+1) and skip if equal; if unequal, perform the next sequential instruction. All zone bits (but not the sign bit) are ignored in the comparison. The AQ register and the primary storage locations remain unchanged.

DECIMAL COMPARE LESS (7717)

Compare the signed 10-digit zoned BCD number of (AQ) with the signed 10-digit zoned BCD number in ($\bar{y}$), ($\bar{y}$+1) and skip if the number in (AQ) is less; if equal, perform the next sequential instruction. All zone bits, but not the sign bit, are ignored in the comparison. The AQ register and the primary storage locations remain unchanged.

4.7. JUMP INSTRUCTIONS

A Jump instruction transfers control of the program from the next sequential instruction to the instruction at a programmed address, provided the conditions for the jump are satisfied. Jump instructions conditioned by I/0 operations are described in the subsequent paragraph on I/0 instructions; this paragraph covers arithmetic and manual jumps, and those jumps which implement interrupt and executive routines by capture of P register values.

In Jump instructions, the operand, Y, is the address of the next instruction to be executed or the address of a memory location which, in turn, contains the address of the next instruction to be executed if jump conditions are satisfied. The address is determined by the k designator and is the lower 15 or 17 bits of the operand shown in Table 4–1. The j designator (Table 3–1) sets up the requirements for the jump.

JUMP – ARITHMETIC (60)

If the jump conditions specified by the j designator (Table 3–1) are satisfied, execute instruction at address Y; otherwise, execute next sequential instruction. Conditions for this jump are determined by (A) or (Q). If the j designator is 0 or 1, the interrupt lockout will be cleared. The address is the lower 15 or 17 bits of the operand shown in Table 4–1.

JUMP – MANUAL (61)

If jump conditions, as determined by manually controlled switches and the j designator (Table 3–1) are satisfied, jump to address Y; otherwise, execute next sequential instruction. The instruction will cause a jump and stop if j values of 4, 5, 6, or 7 are used. This will cause the CPU to halt with the P register set to Y. If j is 0, an unconditional jump to address Y will be executed. The address, Y, is the lower 15 or 17 bits of the operand shown in Table 4–1.

RETURN JUMP – ARITHMETIC (64)

If jump conditions are satisfied, as determined by (Q) or (A), store the relative address of the next sequential instruction (P-RIR) in the lower half of address Y and jump to address Y + 1 for the next instruction to be executed; otherwise, execute the next sequential instruction. The address Y is the lower 15 bits of the operand shown in Table 4–1. If j = 0 or 1, the interrupt lockout will be set (see Interrupts, Section 2). The address that is stored at Y is relative address (P-RIR). If this relative address is 17 bits, the two highest-order bits will be lost. (To store a 17-bit relative address, an Enter B and Jump instruction – 7744, 7745, 7746, or 7747 – must be used.) This instruction is usually the instruction stored at a fixed address, accessed by an interrupt.

A typical use of this instruction in an interrupt routine is shown in Figure 4–2.

(1) After execution of the second instruction in the main program, an interrupt sends the program to a Return Jump-Arithmetic instruction.

(2) The Return Jump-Arithmetic instruction stores the relative address of the third instruction of the main program at $\bar{y}$ and jumps to $\bar{y}+1$ for the next instruction.

(3) The interrupt subroutine is executed, starting from address $\bar{y}+1$, and then a Jump-Manual (61) instruction is executed.

(4) The Jump-Manual instruction sends the interrupt routine to $(\bar{y})_L$ for the address of the next instruction to be executed.

(5) This relative address is the relative address of the third instruction of the main program, and thus the main program is resumed after the interrupt.

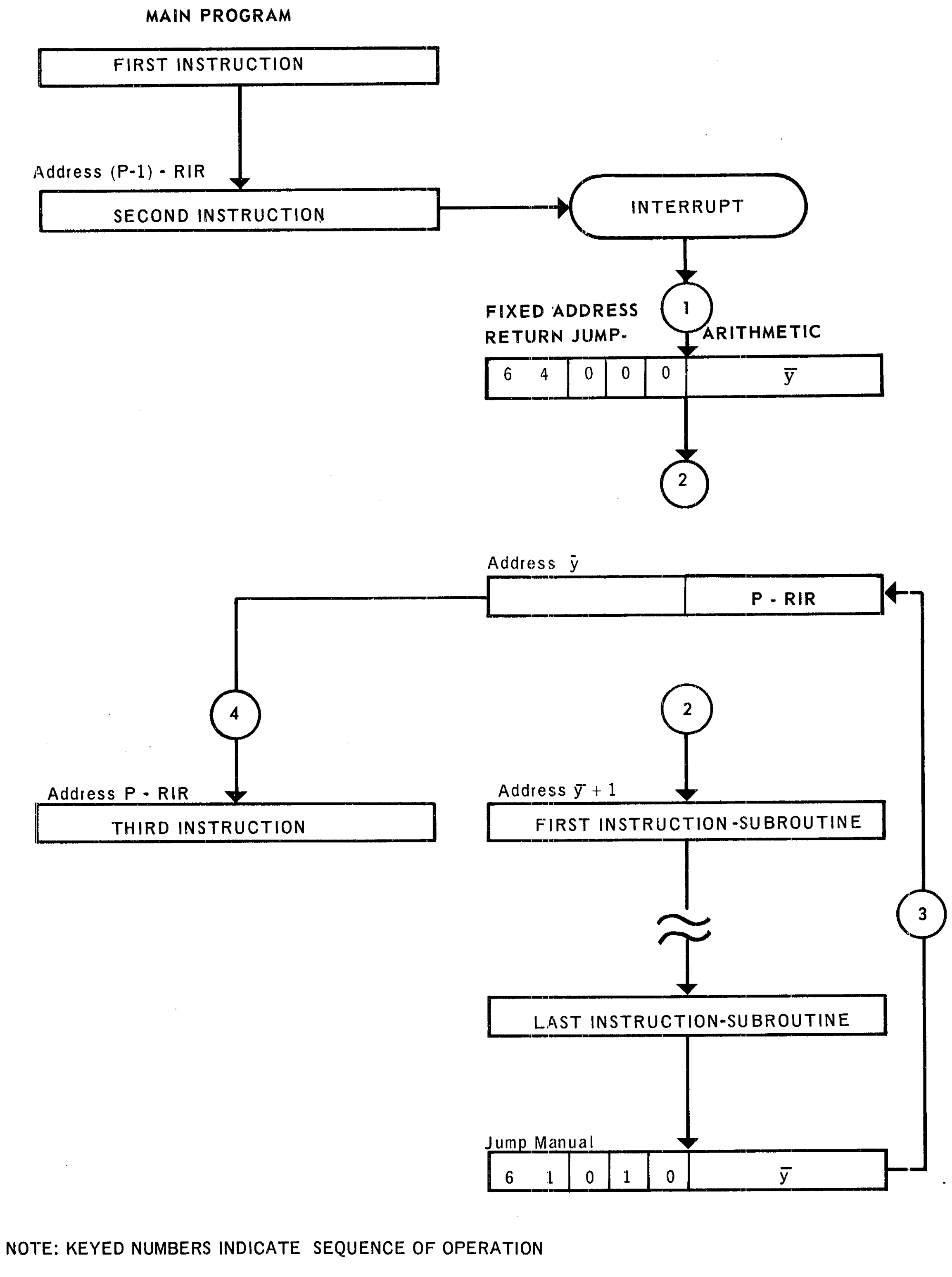


Figure 4–2. Operation of Typical Return Jump Instruction

RETURN JUMP – MANUAL (65)

If jump conditions are satisfied, as determined by the j designator and settings of manual switches (Table 3–1), store the relative address of the next sequential instruction at the lower half of address Y and jump to address Y + 1 for the next instruction. If no jump occurs, execute the instruction immediately following the Return Jump-Manual instruction. Where a stop is to be executed, as determined by the j designator, the address of the next sequential instruction will be stored at address Y. When the program is again started, it will start with the instruction at address Y + 1. As in the Return Jump-Arithmetic (64) instruction, a 15-bit relative address (P-RIR) is stored, and the same precautions should be observed for a 17-bit relative address (P-RIR). The address Y is the lower 15 bits of the operand shown in Table 4–1.

EXECUTE REMOTE (7737)

Execute the instruction located at memory address $\bar{y}$. If the instruction in $\bar{y}$ is not a jump or skip instruction, return to the instruction immediately following the Execute Remote instruction. If the instruction in $\bar{y}$ is a Jump instruction, execute the jump. If the instruction in $\bar{y}$ indicates a skip, skip the instruction immediately following the Execute Remote instruction and execute the next sequential instruction.

ENTER B0 AND JUMP (7740)

Jump, unconditionally, to $\bar{y}$.

ENTER B1 AND JUMP (7741)

Transfer P-RIR to B1 and jump to $\bar{y}$ (using original contents of B_b). This instruction can only be used for a 15-bit (P-RIR). If P-RIR is 16–17 significant bits, the highest order bits will be lost.

ENTER B2 AND JUMP (7742)

Transfer P-RIR to B2 and jump to $\bar{y}$ (using original contents of B_b). This instruction can only be used for a 15-bit (P-RIR). If P-RIR is 16–17 significant bits, the highest order bits will be lost.

ENTER B3 AND JUMP (7743)

Transfer P-RIR to B3 and jump to $\bar{y}$ (using original contents of B_b). This instruction can only be used for a 15-bit (P-RIR). If P-RIR is 16–17 significant bits, the highest order bits will be lost.

ENTER B4 AND JUMP (7744)

Transfer P-RIR to B4 and jump to $\bar{y}$ (using original contents of B_b). This instruction will enter a 17-bit (P-RIR) regardless of 15-bit or 17-bit activation by the IFR and is used in place of a Return Jump instruction.

ENTER B5 AND JUMP (7745)

Transfer P-RIR to B5 and jump to $\bar{y}$ (using original contents of B_b). This instruction will enter a 17-bit (P-RIR) regardless of 15-bit or 17-bit activation by the IFR and is used in place of a Return Jump instruction.

ENTER B6 AND JUMP (7746)

Transfer P-RIR to B6 and jump to $\bar{y}$ (using the original contents of B_b). This instruction will enter a 17-bit (P-RIR) regardless of 15-bit or 17-bit activation by the IFR and is used in place of a Return Jump instruction.

ENTER B7 AND JUMP (7747)

Transfer P-RIR to B7 and jump to $\bar{y}$ (using original contents of B_b). This instruction will enter a 17-bit (P-RIR) regardless of 15-bit or 17-bit activation by the IFR and is used in place of a Return Jump instruction.

4.8. SEQUENCE - MODIFYING INSTRUCTIONS

Sequence-modifying instructions can cause repeated execution of an instruction a specified number of times, and can cause a skip or a jump to another portion of the program.

REPEAT (70)

Execute the next sequential instruction Y (the repeat count) times successively and then proceed to the instruction immediately after the repeated instruction. The repeat count is the lower 15 or 17 bits of the operand defined by the k designator per Table 4–1, and may have a maximum of 17 significant bits. Just before the first execution, Y is stored in B7 as a 17-bit number, regardless of 15-bit or 17-bit activation by the IFR. If a 15-bit number is transferred, three 0's will be added at the high order portion. With each execution of the repeated instruction, the number stored in B7 is decremented by 1. If this number is initially 0, or becomes 0 because of the repeated executions, the program proceeds to the second instruction after the Repeat instruction. Any repeated incrementation or decrementation of the instruction being repeated due to interpretation of the j designator are stored in an intermediate register so that the instruction being repeated remains unchanged at its original address. The Repeat instruction may not immediately follow an Enter IFR (7765) or Enter RIR (7766) instruction.

There are several instructions for which a repeat is not valid. When any of these instructions is preceded by a Repeat instruction, it will be executed only once if the repeat count is initially unequal to 0 and the program will advance to the next sequential instruction; if the repeat count is initially 0, the instruction will be skipped. The forbidden instructions are:

- Illegal Instruction (00 or 7700)
- Enter B_j (12)
- Send External Function (13)
- Store Channel (17)
- All Jump, Buffer Activation and Termination (60 through 76)
- All 77 instructions

The j designator of the Repeat instruction modifies the operand, Y, of the instruction being repeated, as shown in Table 4–5. .

j	OPERATION
0 or 4*	Y remains unchanged with each execution of the instruction.
1 or 5*	Y is increased by one after each execution.
2 or 6*	Y is decreased by one after each execution.
3 or 7*	Y is increased by (B_b) after each execution.

Designators of 4, 5, 6, or 7 are reserved for Replace class instructions. The result is stored at address $Y+(B_6)$ where Y is the operand in effect when the instruction is executed.

Table 4–5. Interpretation of j Designator for Repeat Instruction

A Repeat instruction may be interrupted by either a Fault or I/O interrupt. If execution of the repeated instruction is interrupted, the value in the P register is decreased by 1 so that the value is the address of the repeated instruction. This value can be captured by a Return Jump instruction or by an Enter B_X and Jump instruction. The f1 field of the IFR contains either an address or operand (see 4.3). If an I/0 interrupt halts the Repeat instruction and field f1 of the IFR holds a relative address, the Repeat can be reinitiated. When the interrupt occurs, the value in the P register must be stored first, then the contents of the IFR must be stored. At the end of the interrupt routine, the sequence must be Enter IFR followed by a Jump to the value captured at the beginning of the routine. If a Fault interrupt halts the Repeat execution, the Repeat cannot be reinitiated. In this case, field f1 of the IFR holds an absolute address for use in error diagnosis. Bit 29 in the IFR must be cleared before the Enter IFR is executed at the end of the fault routine. When a repeat operation is interrupted by an I/0 interrupt, the count value at the point of interrupt is retained in B_7 for reinitiation of the repeat operation and must not be modified during the interrupt routine if correct reinitiation of the repeat operation is to occur.

B SKIP ON B_j (71)

If the operand, Y, is not equal to the contents of the index register specified by the j designator (B_j), add 1 to (B_j), and execute the next sequential instruction; if equal to (B_j), clear (B_j), and skip the next sequential instruction. The number of bits in the index register (B_j) is determined by the j designator and the bit mode indicated by the IFR (see Register Formats, Section 3). However, if the k designator is 4, 5, or 6, the B value for the comparison will be the lowest 15 bits in the index register with the 15th bit extended through bit 29, regardless of the IFR setting. The k designator, which determined Y, determines the bit length of Y to be used in the comparison as follows:

k = 0, 3, or 7:	The bit length of Y (starting from the lower end) will be matched to the bit length of (B_j). If $(IFR)_{26}$ is a binary 1, and j = 4, 5, 6, or 7, Y will be 17 bits; if neither condition exists, Y will be 15 bits.

k = 4, 5, or 6:	Y will be 30 bits, with bits 15–29 an extension of bit 14.
k = 1 or 2:	Y will be 15 bits. In this case, the programmer must ensure that (B_j) contains only 15 significant bits, or that j is a 1, 2, or 3, or that $(IFR)_{26}$ is a binary 0.

Since the j designator is used to specify the index register to be used for the comparison, it cannot be used for indicating a skip condition, except where j = 0 (see Table 3–1). If j = 0 and k = 0, skip the next sequential instruction. This instruction may not immediately follow an Enter IFR (7765) or Enter RIR (7766) instruction.

B JUMP ON B_j (72)

If $(B_j) \neq +0$, subtract 1 from its contents and jump to Y for the next instruction; if $(B_j) = +0$, or if the j designator is a 0, execute the next sequential instruction. The significant number of bits in (B_j) is determined by $(IFR)_{26}$, the j designator (Table 3–1), and the actual number in (B_j). This instruction may not immediately follow an Enter IFR (7765) or Enter RIR (7766) instruction.

TEST AND SET (7752)

Test the bit 14 at $\bar{y}$. If this bit is a 0, set bits 0 through 14 to 1 and proceed to the next sequential instruction; if the bit is a 1, generate the Test and Set interrupt which leads to an interrupt routine at a fixed address. This bit is a programmable "electronic switch" which can be turned either on or off, with its state conditioning further program action.

EXECUTIVE RETURN (7754)

Interrupt the program to a fixed address for an executive routine (see Executive Control, 2.5).

4.9. INPUT/OUTPUT INSTRUCTIONS

Input/output instructions cause data and control signals to be transferred between the CPU and peripheral equipment through I/O channels controlled by a multilevel priority scale and through buffers assigned to the CPU according to the mode (ESI or ISI) of the transfer (see 2.4 for details of I/O operations). The instructions activate and terminate buffers and channels, and perform various functions for keeping the system informed of data availability, program status, and other activities.

The following typical examples illustrate use of the I/O instructions with the UNISERVO VI C Magnetic Tape Subsystem (a detailed description of the instructions is furnished after the three following examples). A description of the required function codes and status words is presented in the manuals for the individual I/O devices.

Sample 1. Rewind unit 7 on channel 12 with interrupt. This requires two instructions: the Enter Channel Select register (7773) and the Send External Function (13) instructions. (In most cases, an Activate Output Buffer Without Monitor (74) with a $\hat{j}$ designator of 1 can be substituted for the Send External Function (13) instruction.)

Enter Channel Select Register: 7773027654

The lowest order five bits at address 27654 are 01100, corresponding to a decimal 12 or octal 14.

Send External Function: 1303012345

The function code at address 12345 would appear as: 0000300007

Instead of the Send External Function instruction, an Activate Output Buffer Without Monitor (74) could have been substituted as follows:

Activate Output Buffer Without Monitor: 7407034567

(Since four bits are allotted to $\hat{j}$ and two to $\hat{k}$, the combination of $\hat{j} = 1$ and $\hat{k} = 3$ would appear as octal-coded 07, as shown above.) The buffer control word at address 34567 (loaded into the BCR) would be: 0001012345 and the function code at address 12345 would be sent.

Sample 2. Write on logical unit 11, channel 5 with interrupt. Three instructions are required: the Enter CSR, the Send External Function, and the Activate Output Buffer With Monitor (76).

Enter Channel Select Register: 7773034567

Send External Function: 1303001010

The function code at address 01010 would be 0000500011.

Activate Output Buffer With Monitor: 7603001020

The buffer control word at address 01020 would be loaded into the BCR and the successive transfers would take place until the buffer was emptied.

Sample 3. Read forward 377_8 words from logical unit 6, channel 6, starting from buffer address 20000, with interrupt. Three instructions are required: the Enter CSR, the Activate Input Buffer Without Monitor (73), and the Send External Function.

Enter Channel Select Register: 7773023456

Activate Input Buffer Without Monitor: 7303001022

The input BCR for channel 6 would be loaded with: 0377020000

Send External Function: 1303001016

The function code at address 01016 would contain: 0000600006.

When the buffer is filled with the 377_8 words, an interrupt will be generated within the central processor. The BCR will contain: 0000020400.

NOTE: In the above examples, the Output Data Buffer was activated after the EXFCT for output while the Input Data Buffer was activated before the EXFCT for input. This convention must be followed.

SEND EXTERNAL FUNCTION (13)

Send the one function word $(\bar{y})$ together with the External Function signal on the channel specified by the CSR. No Output Data Request from the peripheral device need be present for the function word to be sent. The $\hat{j}$ designator must be 0 (therefore no skip condition can be programmed in this instruction) and the $\hat{k}$ designator must be 3, so that a 30-bit transfer can take place. If $\hat{k} \neq 3$, no transfer can occur. If $\hat{k} \neq 0$, the operator's console audio signal is enabled if $\bar{y}_2 = 1$ or disabled if $\bar{y}_3 = 1$. (The instruction uses $y + B_b$ bits if $\hat{k} = 0$.) If $\hat{k} = 1$ or 2, no operation results. Note that execution of this instruction is governed by I/O function priority.

STORE CHANNEL (17)

Store, as $(\bar{y})$, the status word on the input data lines which accompanies the External Interrupt signal on the channel specified by the Interrupt Address Storage register. After the status word is stored in primary storage, send the Input Data Acknowledge signal to the input device. The $\hat{j}$ designator must be 0, and the $\hat{k}$ designator must be 3. If $\hat{k} = 0$, 1, or 2, no operation results. This instruction should be executed as soon as feasible after the External Interrupt. Note that execution of this instruction is governed by I/O function priority.

JUMP ON ACTIVE INPUT BUFFER (62)

Jump to Y for the address of the next instruction if the channel specified by the CSR is in an active input buffer state. A channel is in an active input buffer state if the input BCR associated with the channel is active. The $\hat{j}$ designator must be 0. The $\hat{k}$ designator is interpreted as follows:

$\hat{k} = 0$:	Y is $\bar{y}$
$\hat{k} = 1$ or 3:	Y is $(\bar{y})_L$
$\hat{k} = 2$:	Y is $(\bar{y})_U$

JUMP ON ACTIVE OUTPUT BUFFER (63)

Jump to Y for the address of the next instruction if the channel specified by the CSR is in an active output buffer state. A channel is in an active output buffer state if an output BCR associated with the channel is active. The $\hat{j}$ designator must be 0. The $\hat{k}$ designator is interpreted as follows:

$\hat{k} = 0$:	Y is $\bar{y}$
$\hat{k} = 1$ or 3:	Y is $(\bar{y})_L$
$\hat{k} = 2$:	Y is $(\bar{y})_U$

TERMINATE INPUT BUFFER (66)

Terminate the active input buffer state of the channel specified by the CSR. Both $\hat{j}$ and $\hat{k}$ designators must be 0. The b and y designators have no effect in this instruction. Any input transfer in effect at the time of this instruction will be completed but further input transfers on this channel will be inhibited. No monitor interrupt will be generated, even though an input buffer may have been activated with monitor, and the buffer filled.

TERMINATE OUTPUT BUFFER (67)

Terminate the active output buffer state of the channel specified by the CSR. Both $\hat{j}$ and $\hat{k}$ designators are normally 0. The b and y designators have no effect in this instruction. Any output transfer currently in effect will be completed but further output transfers will be inhibited on this channel. No monitor interrupt will be generated, even though an output buffer on the channel may have been activated with monitor, and the buffer empties by the current transfer. A 67 instruction with $\hat{k}$ = 3 is a No Op in the 494 mode; in the 490 mode it is similar to the 7772 instruction.

ACTIVATE INPUT BUFFER WITHOUT MONITOR (73)

Activate an input buffer on the channel specified by the CSR. Because the input buffer is activated without monitor, no interrupt will be generated when the buffer is filled, even though the buffer is deactivated when the word count becomes 0. Therefore, this instruction is normally restricted to ISI input, since the ESI input operation requires an interrupt to store the ESI location address when the buffer is filled. The $\hat{j}$ designator is not used and is normally 0. In the ISI mode, the $\hat{k}$ designator is normally 3 which causes transfer of a BCW into the input BCR for the channel when the input channel is activated. A $\hat{k}$ designator of 0 is used only for ESI operation but no BCW is moved. In this case, the b and y designators are not used. (The BCW must have been loaded into the ESI location by a prior instruction.) This manner of ESI input operation requires that the peripheral equipment send an External Interrupt signal for storage of the ESI location address when the buffer is filled. Possible variations for ISI operation are: a $\hat{k}$ designator of 1, $(\bar{y})_{00-14}$ are transferred to bit positions 00–14 of the input BCR; a $\hat{k}$ designator of 2, $(\bar{y})_{15-29}$ are transferred to bit positions 15–29 of the input BCR.

ACTIVATE OUTPUT OR EXTERNAL FUNCTION BUFFER WITHOUT MONITOR (74)

Activate an output or External Function buffer on the channel specified by the CSR. Because the buffer is activated without monitor, no interrupt is generated when the buffer is deactivated (word count is 0) as the buffer is emptied. For ISI operation: an odd-valued $\hat{j}$ designator indicates an External Function buffer; an even-valued $\hat{j}$ designator, a data buffer. If an External Function buffer, an External Function signal is sent with the information on the data output lines; if a data buffer, the Output Data Acknowledge (ODA) signal accompanies the information on the data lines. For ESI operation, a $\hat{j}$ designator of 0 is required thereby preventing transmission of ESI function codes with this instruction. This instruction is different from the Send External Function instruction (13) in that an ODR is required before the first function code can be sent and may be used for sending a chain of function codes. In the ISI mode, a $\hat{k}$ designator of 3 is normally used to transfer a BCW from $\bar{y}$ to the output BCR. In the ESI mode, a $\hat{k}$ designator of 0 is required. No BCW is moved in this case. Possible variations for ISI operation are: a $\hat{k}$ designator of 1, $(\bar{y})_{00-14}$ is transferred to bit positions 00–14 of the output BCR; a $\hat{k}$ designator of 2, $(\bar{y})_{15-29}$ is transferred to bit positions 15–29 of the output BCR.

ACTIVATE INPUT BUFFER WITH MONITOR (75)

This instruction is similar to the Activate Input Buffer Without Monitor except that when the buffer is filled and terminated (the address count in the BCR is 0), a monitor interrupt will be generated and control will be transferred to an interrupt routine for further processing, and therefore, it can be used in the ESI mode.

ACTIVATE OUTPUT BUFFER WITH MONITOR (76)

This instruction is similar to the Activate Output Buffer Without Monitor except that when the buffer is emptied and terminated (the address count in the BCR is 0), a monitor interrupt will be generated upon receipt of an ODR signal and control will be transferred to an interrupt routine for further processing.

INITIATE SYNCHRONIZING INTERRUPT (7770)

Send an interrupt signal to either of two collocated central processors working together. If bit 10 of the y designator is a 1, send the interrupt on Synchronizing Interrupt #0 line (see Figure 2–1); if bit 1 of y + b is a 1, send the interrupt on Synchronizing Interrupt #1 line.

STORE CHANNEL NUMBER (7772)

This instruction is valid only within an interrupt subroutine. When this instruction is used in the Buffer Control Word or I/O Data Parity Error subroutine, the channel number will be stored at $\bar{y}$ from the Parity Error Channel Storage register (PECSR). When this instruction is used in an External Interrupt subroutine, the channel number will be stored from the Interrupt Address Storage register (IASR). If the instruction is used outside of an interrupt subroutine, a value of 37_8 will be stored in Y_a.

ENTER CHANNEL SELECT REGISTER (7773)

Transfer the lowest order five bits of $(\bar{y})$ to the Channel Select register (CSR).

APPENDIX A. ABBREVIATIONS AND SYMBOLS

A table of abbreviations and symbols used in this manual throughout the text is shown in Table A-1.

ABBREVIATION OR SYMBOL	MEANING
A	The 30-bit accumulator register
AQ	The 60-bit AQ register – a combination of A and Q registers
B	Index register
b	An indicator in an instruction word, referring to an index register
BCD	Binary-coded decimal number
BCR	Buffer Control register
BCW	Buffer Control Word
CP (x)	The ones complement of the quantity within the parentheses
CPU	Central Processor Unit
CSR	Channel Select register
DEC	Decimal
EI	External Interrupt
ESI	Externally specified index mode of channel operation, generally used with multiplexed communication equipment
EX	Exponent portion of a floating-point number which consists of an exponent and a fixed-point part
EXFCT	External Function
EXRN	Executive Return
FP	Fixed-point part of a floating-point number
IASR	Interrupt Address Storage register
IDA	Input Data Acknowledge
IDR	Input Data Request
I/O	Input/Output
ISI	Internally specified index mode of channel operation, generally used with standard peripheral equipment such as drums, card readers, etc.
IFR	Internal function register
j or $\hat{j}$	An instruction sequence modification indicator within an instruction word, generally specifying the conditions for a skip ($\hat{j}$ used in I/O instruction word)
k or $\hat{k}$	An operand modification indicator within an instruction word, generally defining the source and/or destination of the operand in the instruction ($\hat{k}$ used in I/O instruction word)
LP(M and N)	The logical product of (M) and (N); the nth bit is a binary 1 if $(M)_n$ and $(N)_n$ are both binary 1's.
MSR	Memory Select register
NI	Next Sequential Instruction
ODA	Output Data Acknowledge
ODR	Output Data Request
P	Program register, containing the address of either the current or the next instruction
PECSR	Parity Error Channel Storage register
PLR	Program Lock-In register
Q	Quotient register
Rd	Read class instruction
R_i	Relative index
RIR	Relative index register
R_p	Replace class instruction
St	Store class instruction
y	The low order 15 bits of an instruction word
$\bar{y}$	The sum of y and the contents of the index register specified by b. Depending upon the other contents of the instruction word, $\bar{y}$ may be a directly used effective operand or a relative address (containing the operand of the instruction).
Y	The operand of an instruction, from whatever source derived. This operand is determined by the instruction, $\bar{y}$, and/or the k designator of the instruction.
(z)	The contents of the register or memory location denoted by z.
$(z)_i$	The initial contents of z (before execution of instruction)
$(z)_f$	The final contents of z (after execution of instruction)
$(z)_n$	The nth bit of the contents of z, starting from 0 at the right or least significant position
$\longrightarrow$	Direction of data transfer

Table A–1. Abbreviations and Symbols

APPENDIX B. REPERTOIRE OF INSTRUCTIONS

A listing of the repertoire of instructions for the UNIVAC 494 Real-Time System is provided in Table B-1.

TYPE	FUNCTION CODE (OCTAL)	INSTRUCTION	CLASS	OPERATION
SHIFT	01	Right Shift Q	Rd	Shift Q register right with sign-fill } by $Y_{00\text{-}05}$ positions
	02	Right Shift A	Rd	Shift A right with sign-fill
	03	Right Shift AQ	Rd	Shift AQ right with sign-fill
	05	Left Shift Q	Rd	Circular-shift Q to left
	06	Left Shift A	Rd	Circular-shift A to left
	07	Left Shift AQ	Rd	Circular-shift AQ to left
	7751	Logical Right Shift Q	–	Shift Q to right with zero-fill
	7755	Logical Right Shift A	–	Shift A to right with zero-fill
	7756	Logical Right Shift AQ	–	Shift AQ to right with zero-fill
	7730	Scale Factor Shift	–	Circular-shift A to left until bit 29 ≠ bit 28
TRANSFER	10	Enter Q	Rd	$Y \longrightarrow Q$
	11	Enter A	Rd	$Y \longrightarrow A$
	12	Enter B_j	Rd	$Y \longrightarrow B_j$; if j = 0, no operation
	14	Store Q	St	$Q \longrightarrow Y$; if k = 0, CP(Q) $\longrightarrow Q$
	15	Store A	St	$A \longrightarrow Y$; if k = 4, CP(A) $\longrightarrow A$
	16	Store B_j	St	$(B_j) \longrightarrow Y$; if j = 0, 0's $\longrightarrow Y$
	7721	Enter AQ	–	$(\bar{y}) \longrightarrow A$ and $(\bar{y}+1) \longrightarrow Q$
	7725	Store AQ	–	$A \longrightarrow (\bar{y})$ and $Q \longrightarrow (\bar{y}+1)$
	7731	Character Pack Lower	–	$(\bar{y})00\text{-}05 \longrightarrow A_{24\text{-}29}$, $(\bar{y}+1)_{00\text{-}05} \longrightarrow A_{18\text{-}23}$, $(\bar{y}+2)_{00\text{-}05} \longrightarrow A_{12\text{-}17}$, $(\bar{y}+3)_{00\text{-}05} \longrightarrow A_{06\text{-}11}$, $(\bar{y}+4)_{00\text{-}05} \longrightarrow A_{00\text{-}05}$
	7732	Character Pack Upper	–	$(\bar{y})_{15\text{-}20} \longrightarrow A_{24\text{-}29}$, $(\bar{y}+1)_{15\text{-}20} \longrightarrow A_{18\text{-}23}$, $(\bar{y}+2)_{15\text{-}20} \longrightarrow A_{12\text{-}17}$, $(\bar{y}+3)_{15\text{-}20} \longrightarrow A_{06\text{-}11}$, $(\bar{y}+4)_{15\text{-}20} \longrightarrow A_{00\text{-}05}$
	7735	Character Unpack Lower	–	$A_{24\text{-}29} \longrightarrow (\bar{y})_{00\text{-}05}$, $A_{18\text{-}23} \longrightarrow (\bar{y}+1)_{00\text{-}05}$, $A_{12\text{-}17} \longrightarrow (\bar{y}+2)_{00\text{-}05}$, $A_{06\text{-}11} \longrightarrow (\bar{y}+3)_{00\text{-}05}$, $A_{00\text{-}05} \longrightarrow (\bar{y}+4)_{00\text{-}05}$
	7736	Character Unpack Upper	–	$A_{24\text{-}29} \longrightarrow (\bar{y})_{15\text{-}20}$, $A_{18\text{-}23} \longrightarrow (\bar{y}+1)_{15\text{-}20}$, $A_{12\text{-}17} \longrightarrow (\bar{y}+2)_{15\text{-}20}$, $A_{06\text{-}11} \longrightarrow (\bar{y}+3)_{15\text{-}20}$, $A_{00\text{-}05} \longrightarrow (\bar{y}+4)_{15\text{-}20}$
	7761	Enter IFR and RIR	–	$(\bar{y}) \longrightarrow$ IFR, $(\bar{y}+1)_{06\text{-}16} \longrightarrow$ RIR (executive mode only)
	7762	Load Program Lock-In Register	–	$(\bar{y}) \longrightarrow$ Program Lock-In Register (executive mode only)
	7765	Store IFR	–	IFR $\longrightarrow (\bar{y})$ (executive mode only)
	7766	Enter RIR	–	$(\bar{y})_{06\text{-}17} \longrightarrow$ RIR (executive mode only)
	7771	Enter B-Worker	–	$(\bar{y})_L \longrightarrow B1$, $(\bar{y}+1)_L \longrightarrow B2$, $(\bar{y}+2)_L \longrightarrow B3$, $(\bar{y}+3)_L \longrightarrow B4$, $(\bar{y}+4)_L \longrightarrow B5$, $(\bar{y}+5)_L \longrightarrow B6$, $(\bar{y}+6)_L \longrightarrow B7$
	7775	Store B-Worker	–	$B1 \longrightarrow (\bar{y})_L$, $B2 \longrightarrow (\bar{y}+1)_L$, $B3 \longrightarrow (\bar{y}+2)_L$, $B4 \longrightarrow (\bar{y}+3)_L$, $\longrightarrow B5 \longrightarrow (\bar{y}+4)_L$, $B6 \longrightarrow (\bar{y}+5)_L$, $B7 \longrightarrow (\bar{y}+6)_L$

Table B–1. Repertoire of Instructions
(Part 1 of 5)

TYPE			FUNCTION CODE (OCTAL)	INSTRUCTION	CLASS	OPERATION
ARITHMETIC	INTEGER	SINGLE PRECISION	20	Add A	Rd	$Y + A_i \longrightarrow A_f$
			21	Subtract A	Rd	$Y - A_i \longrightarrow A_f$
			22	Multiply	Rd	$(Q)(Y) \longrightarrow AQ$
			23	Divide	Rd	$(AQ)_i / Y \longrightarrow (AQ)_f$; Quotient $\longrightarrow$ Q, Remainder $\longrightarrow$ A
			24	Replace A + Y	Rp	$A_i + Y_i \longrightarrow A_f$ and Y_f
			25	Replace A - Y	Rp	$A_i - Y_i \longrightarrow A_f$ and Y_f
			26	Add Q	Rd	$Q_i + Y \longrightarrow Q_f$
			27	Subtract Q	Rd	$Q_i - Y \longrightarrow Q_f$
			30	Enter Y + Q	Rd	$Y + Q \longrightarrow A$
			31	Enter Y - Q	Rd	$Y - Q \longrightarrow A$
			32	Store A + Q	St	$A_i + Q \longrightarrow A_f$ and Y
			33	Store A - Q	St	$A_i - Q \longrightarrow A_f$ and Y
			34	Replace Y + Q	Rp	$Y_i + Q \longrightarrow Y_f$ and A
			35	Replace Y - Q	Rp	$Y_i - Q \longrightarrow Y_f$ and A
			36	Replace Y + 1	Rp	$Y_i + 1 \longrightarrow Y_f$ and A
			37	Replace Y - 1	Rp	$Y_i - 1 \longrightarrow Y_f$ and A
		DOUBLE PRE-CISION	7722	Add AQ	–	$(AQ)_i + (\bar{y}, \bar{y}+1) \longrightarrow (AQ)_f$
			7724	Complement AQ	–	$CP(AQ)_i \longrightarrow (AQ)_f$
			7726	Subtract AQ	–	$(AQ)_i - (\bar{y}, \bar{y}+1) \longrightarrow (AQ)_f$
	EXPONENTIAL		7701	Floating-Point Add	–	$(AQ)_i + (\bar{y}, \bar{y}+1) \longrightarrow (AQ)_f$
			7702	Floating-Point Subtract	–	$(AQ)_i - (\bar{y}, \bar{y}+1) \longrightarrow (AQ)_f$
			7703	Floating-Point Multiply	–	$(AQ)_i (\bar{y}, \bar{y}+1) \longrightarrow (AQ)_f$
			7705	Floating-Point Divide	–	$(AQ)_i / (\bar{y}, \bar{y}+1) \longrightarrow (AQ)_f$ with remainder discarded
			7706	Floating-Point Pack	–	Y_{EX} and $(AQ)_{FXP} \longrightarrow (AQ)_f$ as floating-point number
			7707	Floating-Point Unpack	–	$(AQ)_{EX} \longrightarrow Y$
	DECIMAL		7710	Decimal Test	–	Skip as indicated by BCD test
			7711	Decimal Add	–	$(AQ)_i + (\bar{y}, \bar{y}+1) \longrightarrow (AQ)_f$
			7712	Decimal Subtract	–	$(AQ)_i - (\bar{y}, \bar{y}+1) \longrightarrow (AQ)_f$
			7714	Decimal Complement AQ	–	$\bar{y}$ even, tens $CP(AQ)_j \longrightarrow (AQ)_f$; y odd, nines $CP(AQ)_j \longrightarrow (AQ)_f$
			7715	Decimal Add With Carry	–	$(AQ)_i + (\bar{y}, \bar{y}+1)$ + previous carry $\longrightarrow (AQ)_f$
			7716	Decimal Subtract With Borrow	–	$(AQ)_i - (\bar{y}, \bar{y}+1)$ - previous borrow $\longrightarrow (AQ)_f$
			7733	Decimal Convert Lower	–	$(\bar{y})_{0-3}, (\bar{y}+1)_{0-3}, \ldots, (\bar{y}+4)_{0-3}$ BCD $\longrightarrow AQ_{BINARY}$
			7734	Decimal Convert Upper	–	$(\bar{y})_{15-18}, (\bar{y}+1)_{15-18}, \ldots, (\bar{y}+4)_{15-18}$ BCD $\longrightarrow AQ_{BINARY}$

Table B–1. Repertoire of Instructions (Part 2 of 5)

TYPE	FUNCTION CODE (OCTAL)	INSTRUCTION	CLASS	OPERATION
LOGICAL	40	Enter Logical Product	Rd	$LP(Y.Q) \rightarrow A$
	41	Add Logical Product	Rd	$A + LP(Y.Q) \rightarrow A$
	42	Subtract Logical Product	Rd	$A - LP(Y.Q) \rightarrow A$
	44	Replace Logical Product	Rp	$LP(Y.Q) \rightarrow$ Y and A
	45	Replace A + Logical Product	Rp	$A + LP(Y.Q) \rightarrow$ Y and A
	46	Replace A - Logical Product	Rp	$A - LP(Y.Q) \rightarrow$ Y and A
	47	Store Logical Product	St	$LP(A.Q) \rightarrow Y$
	50	Selective Set	Rd	Set $(A)_n$ to 1 where $(Y)_n = 1$
	51	Selective Complement	Rd	Ones-complement $(A)_n$ where $(Y)_n = 1$
	52	Selective Clear	Rd	Clear $(A)_n$ to 0 where $(Y)_n = 1$
	53	Selective Substitute	Rd	$(Y)_n \rightarrow (A)_n$ where $(Q)_n = 1$
	54	Replace Selective Set	Rp	Set $A_n = 1$ where $Y_n = 1; \rightarrow (Y)_f$ and $(A)_f$
	55	Replace Selective Complement	Rp	Ones-complement $(A)_n$ where $(Y)_n = 1; \rightarrow Y_f$ and $(A)_f$
	56	Replace Selective Clear	Rp	Clear A_n to 0 where $Y_n = 1; \rightarrow Y_f$ and $(A)_f$
	57	Replace Selective Substitute	Rp	$Y_n \rightarrow A_n$ where $Q_n = 1; \rightarrow Y_f$ and $(A)_f$
COMPARE	04	Compare	Rd	Compare Y to A and /or Q to determine skip (algebraic)
	43	Compare Masked	Rd	Compare A to LP(Y.Q) to determine skip (algebraic)
	7713	Decimal Compare Equal	–	Skip NI if $(AQ)_{BCD} = (\overline{y}, \overline{y}+1)_{BCD}$, neglecting zones (algebraic)
	7717	Decimal Compare Less	–	Skip NI if $(AQ)_{BCD} < (\overline{y}, \overline{y}+1)_{BCD}$, neglecting zones (algebraic)
	7723	Compare AQ Equal	–	Skip NI if $AQ = (\overline{y}, \overline{y}+1)$ (algebraic)
	7727	Compare AQ Less	–	Skip NI if $AQ < (\overline{y}, \overline{y}+1)$ (algebraic)
	7753	Masked Alphanumeric Equal	–	Skip NI if LP(A and Q) = LP(Y and Q) (alphanumeric sign not recognized)
	7757	Masked Alphanumeric Less	–	Skip NI if LP(A and Q) < LP(Y and Q) (alphanumeric sign not recognized)

Table B-1. Repertoire of Instructions
(Part 3 of 5)

TYPE	FUNCTION CODE (OCTAL)	INSTRUCTION	CLASS	OPERATION
JUMP	60	Jump-Arithmetic	Rd	$Y \longrightarrow P$; if j = 0 or 1 release interrupt lockout (RIL)
	61	Jump-Manual	Rd	$Y \longrightarrow P$; dependent upon keys
	64	Return Jump-Arithmetic	Rd	$(P - RIR) \longrightarrow Y_L$ and jump to Y + 1 ; if j = 0 or 1 set interrupt lockout (SIL)
	65	Return Jump-Manual	Rd	$(P - RIR) \longrightarrow Y_L$ and jump to Y + 1 ; dependent upon keys
	7737	Execute Remote	–	Execute instruction at y and then return to P + 1 (conditional)
	7740	Unconditional Jump	–	Jump to Y
	7741	Enter B1 and Jump	–	$(P - RIR) \longrightarrow B1$ and jump to Y
	7742	Enter B2 and Jump	–	$(P - RIR) \longrightarrow B2$ and jump to Y
	7743	Enter B3 and Jump	–	$(P - RIR) \longrightarrow B3$ and jump to Y
	7744	Enter B4 and Jump	–	$(P - RIR) \longrightarrow B4$ and jump to Y
	7745	Enter B5 and Jump	–	$(P - RIR) \longrightarrow B5$ and jump to Y
	7746	Enter B6 and Jump	–	$(P - RIR) \longrightarrow B6$ and jump to Y
	7747	Enter B7 and Jump	–	$(P - RIR) \longrightarrow B7$ and jump to Y
SEQUENCE - MODIFYING	70	Repeat	Rd	Repeat NI (Y) times
	71	B Skip on B_j	Rd	If $(B_j) = Y$, Skip NI and clear B_j; if $(B_j) \neq Y$, advance (B_j) and execute NI
	72	B Jump on B_j	Rd	Jump to Y if $(B_j) \neq +0$, $(B_j)_f = (B_j)_i - 1$; if $(B_j) = +0$, execute NI
	7752	Test and Set	–	If switch is set, interrupt; if not set, then set and execute NI
	7754	Executive Return	–	Interrupt

Table B–1. Repertoire of Instructions (Part 4 of 5)

TYPE	FUNCTION CODE (OCTAL)	INSTRUCTION	CLASS	OPERATION
INPUT/OUTPUT	13	Send External Function	Rd	$(\bar{y}) \longrightarrow$ external equipment with External Function signal
	17	Store Channel	St	Status word $\longrightarrow$ Y
	62	Jump On Active Input Buffer	Rd	Y $\longrightarrow$ P if Ch_j active
	63	Jump On Active Output Buffer	Rd	Y $\longrightarrow$ P if Ch_j active
	66	Terminate Input Buffer	Rd	Terminate buffer
	67	Terminate Output Buffer	Rd	Terminate buffer
	73	Activate Input Buffer Without Monitor	Rd	Activate buffer
	74	Activate Output Buffer Without Monitor	Rd	Activate buffer
	75	Activate Input Buffer With Monitor	Rd	Activate buffer
	76	Activate Output Buffer With Monitor	Rd	Activate buffer
	7770	Initiate Synchronizing Interrupt	–	Send interrupt (Executive mode only)
	7772	Store Channel Number	–	IASR or PECSR $\longrightarrow$ Y (Executive mode only)
	7773	Enter Channel Select Register	–	Y $\longrightarrow$ (Channel Select Register) (Executive mode only)

*Table B–1. Repertoire of Instructions
(Part 5 of 5)*

APPENDIX C. INTEGER ADDITION AND SUBTRACTION

The following description of integer addition and subtraction in the UNIVAC 494 CPU is included for interest to the programmer.

C.1. INTEGER ADDITION

The adder uses subtractive-type logic internally accepting operands (A) and Y directly for addition. The result is formed by two half-subtracts. The first half-subtract results in a ones-complemented difference and (possibly) a borrow which is propagated until it is "satisfied" by a stage where $(A)_n = Y_n$. A borrow is generated only at a stage where $(A)_n = Y_n$ is equal to 0. If the borrow is not satisfied or if a borrow is generated at the highest order bit positions, the borrow becomes an end-around borrow and is propagated from the lowest order bit positions until satisfied. If it cannot be satisfied, it is propagated around back to the point where it was generated. The following truth table describes generation of the complemented difference and generation of a borrow at each bit position for the first half-subtract:

$(A)_n$	Y_n	COMPLEMENTED DIFFERENCE	BORROW
0	0	1	1
0	1	0	0
1	0	0	0
1	1	1	0

The second half-subtract subtracts all borrows from the result of the first half-subtract. It is a bit-by-bit subtract, generating only a difference (no borrow). The following examples illustrate the operation of the addition process (using nine-bit numbers with highest order bit reserved for sign):

```
1. (A) = +343_8    (A) 0 1 1   1 0 0   0 1 1
    Y  = -115_8     Y  1 1 0   1 1 0   0 1 0
         ------        ---------------------
         +226_8        0 1 0   1 0 1   1 1 0    Complemented difference
                               1<-1- 1<----     Propagated borrows
                       ---------------------
                       0 1 0   0 1 0   1 1 0    Second half-subtract and
                                                  final sum

2. (A) = -334_8         1 0 0 1 0 0   0 1 1
    Y  = +262_8         0 1 0 1 1 0   0 1 0
         ------        ---------------------
         -052_8         0 0 1 1 0 1   1 1 0
                      +--1-1-  1<-1- 1<--  1<-1-+
                      |--------------------------|
                  Sum | 1 1 1 0 1 0   1 0 1     |
                      +--------------------------+
```

C.2. INTEGER SUBTRACTION

Integer subtraction is similar to integer addition except that the operand Y is first ones-complemented before being inserted into the adder in place of Y. The same truth table governs operation except that $CP(Y)_n$ is used in place of Y_n. The following examples illustrate operation of the subtraction process:

1.	(A) = +000	(A) =	0 0 0	0 0 0	0 0 0	
	Y = +000	CP(Y) =	1 1 1	1 1 1	1 1 1	
	+000		0 0 0	0 0 0	0 0 0	First half-subtract (No borrows)
			0 0 0	0 0 0	0 0 0	Difference
2.	(A) = $+343_8$	(A) =	0 1 1	1 0 0	0 1 1	
	Y = $+115_8$	CP(Y) =	1 1 0	1 1 0	0 1 0	
	$+226_8$		0 1 0	1 0 1	1 1 0	First half-subtract
				1 1 1		Borrows
			0 1 0	0 1 0	1 1 0	Difference
3.	(A) = -251_8	(A) =	1 0 1	0 1 0	1 1 0	
	Y = -241_8	CP(Y) =	0 1 0	1 0 0	0 0 1	
	-010_8		0 0 0	0 0 1	0 0 0	First half-subtract
			~~1 1 1~~	~~1 1~~ 1 ←	~~1 1 1~~	Borrows
			1 1 1	1 0 1	1 1 1	Difference

APPENDIX D. EXECUTION TIME OF INSTRUCTIONS

Table D-1, which follows, furnishes the execution time of each instruction in each of the two operating modes (odd-even addressing and continuous addressing) together with any conditions which may modify the speed of execution.

FUNCTION CODE	MODIFYING CONDITIONS	ALTERNATE BANK TIME	SAME BANK TIME		
			k=0,4	k=7	k=1,3,5,6
01		750	750	750	1500
02	(j≠4,5)/(j=4,5)	750/964	750/964	750/964	1500/1714
03		964	964	964	1714
04	(j≠4,5)/(j=4,5 and first test determines skip or no skip)	750/857	750/857	750/857	1500/1607
	(j=4,5 and both tests required to determine skip or no skip	964	964	964	1714
05	(j≠2,3)/(j=2,3)	750/857	750/857	750/857	1500/1607
06	(j≠6,7)/(j=6,7)	750/857	750/857	750/857	1500/1607
07		1071	1071	1071	1821
10,11		750	750	750	1500
12	B_j=0 or B_j≠(B_b of NI)	750	750	750	1500
	B_j≠0 and B_j=(B_b of NI)	857	857	857	1500
13	$\hat{k}$=3/$\hat{k}$≠3	2889 min./750	2889 min./750	2889 min./750	2889 min./750
14, 15, 16		750	750	1500	1500
17		2889 min.	2889 min.	2889 min.	2889 min.
20,21	(j≠6,7)/(j=6,7)	750/857	750/857	750/857	1500/1607
22,23		7277	7277	7277	8027
24,25		1500	1500	1500	2250
26,27,30,31	(j≠6,7)/(j=6,7)	750/857	750/857	750/857	1500/1607
32,33		1500	1500	–	2250
34,35,36,37		1500	1500	1500	2250
40	(j≠2,3)/(j=2,3)	750/1071	750/1071	750/1071	1500/1820
41	(j≠6,7)/(j=6,7)	750/857	750/857	750/857	1500/1607
42		964	964	964	1714
43	(j=0,1,2,3)/(j=4,5,6,7)	750/964	750/964	750/964	1500/1714
44,45		1500	1500	1500	2250
46		1607	1607	1607	4357
47		750	750	–	1500
50,51,52,53		750	750	750	1500
54,55,56,57		1500	1500	1500	2250
60,61	Jump satisfied: k=0,4/≠0,4	750/1500	750	1500	1500
	Jump not satisfied	750	1500	750	1500

NOTE: All time in nanoseconds.

Table D-1. Instruction Execution Time (Part 1 of 3)

FUNCTION CODE	MODIFYING CONDITIONS	ALTERNATE BANK TIME	SAME BANK TIME		
			k=0,4	k=7	k=1,3,5,6
62,63	Ch$_j$ active: $\hat{k}$=0/≠0	750/1500	750(no $\hat{k}$=4)	no k=7	1500 (no $\hat{k}$=5,6)
	Ch$_j$ inactive	750	1500		1500
64,65	Jump satisfied: k=0,4/≠0,4	1500/2250	1500	2250	2250
	Jump not satisfied	750	1500	750	1500
66,67		750	750	750	750
70	k=0,4,7/k≠0,4,7	1285/1500	1285	1285	1500
71		750	750	750	1500
72	Jump satisfied: k=0,4/≠0,4	750/1500	750	1500	1500
	Jump not satisfied	750	1500	750	1500
73,74,75,76	$\hat{k}$=0/$\hat{k}$=3	750/1500	750 (no $\hat{k}$=4)	no $\hat{k}$=7	2250 ($\hat{k}$=3 only)
7701,7702	Sum=±0/sum≠±0	2356/2998		2991/3641	
7703		12093		12843	
7705	|mantissa|AQ ≥ |mant.| Y,Y+1	12200		12950	
	|mant.|AQ < |mant.| Y,Y+1	12414		13164	
	|Y,Y+1| = 0	1607		2357	
7706	(AQ)=±0/(AQ)≠±0	857/1500		1607/2250	
7707,7710		750		1500	
7711	Like signs/unlike signs	2035/2249		2785/2999	
7712	Unlike signs/like signs	2035/2249		2785/2999	
7713		1500		2250	
7714		1285		1500	
7715	Like signs OR AQ < |Y,Y+1|	2035		2785	
	Unlike signs AND AQ ≥ |Y,Y+1|	2249		2999	
7716	Unlike signs OR AQ < (Y,Y+1)	2035		2785	
	Like signs AND AQ ≥ (Y,Y+1)	2249		2999	

NOTE: All time in nanoseconds.

Table D–1. Instruction Execution Time (Part 2 of 3)

FUNCTION CODE	MODIFYING CONDITIONS	ALTERNATE BANK TIME	SAME BANK TIME
7717,7721, 7722,7723		1500	2250
7724		750	1500
7726,7727		1500	2250
7730		857	1607
7731,7732, 7733,7734, 7735,7736		3750	4500
7737,7740, 7741,7742, 7743,7744, 7745,7746, 7747		750	750
7751		750	1500
7752		1821	1821
7753		750	1500
7754		1285	1285
7755		750	1500
7756		857	1607
7757		750	1500
7761		1500	2250
7762,7765,7766		750	1500
7770		750	750
7771		5250	6000
7772,7773		750	1500
7775		5250	6000

NOTE: All time in nanoseconds.

Table D-1. Instruction Execution Time (Part 3 of 3)

UNIVAC

SPERRY RAND

UP-4049 Rev. 2